ETA SYSTEMS

ETA10 System Reference Manual

ETA10 Computer System

PUB-1005

ETA SYSTEMS

ETA10 System Reference Manual

PUB–1005
Revision A
December 4, 1987

The items listed below are referenced in this document and are names, products, or trademarks associated with the following companies:

Apollo is a trademark of Apollo Computer Corporation.
DOMAIN is a trademark of Apollo Computer Corporation.
CDC is a trademark of Control Data Corporation.
CYBER 180 is a product of Control Data Corporation.
CYBER 205 is a product of Control Data Corporation.
ETA is a trademark of ETA Systems, Incorporated.
ETHERNET is a trademark of Xerox Corporation.
Loosely Coupled Network (LCN) is a product of Control Data Corporation.
Multi–Host Network is a product of ETA Systems, Incorporated.
Network Access Device (NAD) is a product of Control Data Corporation.
Open Interconnection Network is a product of ETA Systems, Incorporated.
VAX is a trademark of Digital Equipment Corporation.

Disclaimer:

ETA™ Systems, Incorporated reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should consult ETA Systems to determine whether any such changes have been made. (At non-U.S. installations, the reader should consult the local marketing representative). This manual may not be reproduced and is intended for the exclusive use of ETA Systems' customers.

The terms and conditions governing the sale of ETA Systems hardware products and the licensing and use of ETA Systems software consist solely of those set forth in the written contracts between ETA Systems and its customers. No statement contained in this publication, including statements regarding capacity, suitability for use, or performance of products, shall be considered a warranty by ETA Systems for any purpose or give rise to any liability of ETA Systems.

In no event will ETA Systems be liable for any incidental, indirect, special, or consequential damages (including lost profits) arising out of or relating to this publication or the information contained in it, even if ETA Systems has been advised, knew, or should have known of the possibility of such damages.

The copyright laws prohibit the copying of this manual without the written consent of ETA Systems. Copying, under the law, includes translating into another language or format.

Prepared by: ETA Systems, Incorporated
Technical Communication Dept.
1450 Energy Park Drive
St. Paul, MN 55108

Revision Record

Documents that have a numeric revision code, such as 01 or 03, are still in the draft stage. They are not yet approved for release. Documents carrying an alphabetic code, such as A or C, are complete, finished documents. They are approved for release.

Document Revision	Software Version	Date
Revision A	Rel. 1, Vers. 1.0	December 4, 1987

Changes in This Revision

Compared to draft versions previously distributed, this revision of the *ETA10 System Reference Manual* includes new material on the operating system kernel in chapter 4, revisions of chapters 7 and 8, a glossary, and an index. It is meant to accompany Release 1, Version 1.0 of EOS, the ETA10 operating system.

Table of Contents

The Operating System Kernel **Chapter 4**

Mainframe Components Chapter 5

The I/O Subsystem **Chapter 6**

Networks **Chapter 7**

About This Document . . .

Purpose

The *ETA10 System Reference Manual* is a detailed overview of the ETA10's architecture and major features. It is not meant to repeat specific, operations-oriented information from other ETA Systems reference documents. The level of complexity is appropriate for system analysts who want comprehensive descriptions of hardware components and operating system functions. It is suitable for training courses providing general orientation on the ETA10 system. Readers wanting support for hands-on use of the system should refer to "Related Documents" later in this section.

This edition (Revision A) describes Release 1, Version 1.0 of EOS, the ETA10 operating system, and its VSOS user environment.

Intended Audience

This manual is intended for system analysts and experienced users as well as for administrators, trainers, students, and system evaluators.

How This Document Is Arranged

This document contains ten chapters, a glossary, and an index. A general table of contents at the beginning of the manual is augmented by detailed tables of contents at the start of each chapter.

Chapter 1: Introduction to System Architecture provides a brief overview of the ETA10 computer system: its architecture, hardware and software components, and model range. The reader is directed to more detailed discussions of each subject later in the manual.

Chapter 2: The User Environment describes VSOS, programming environment of the ETA10 operating system EOS.

Chapter 3: Resource Management describes system configuration, resource allocation, and management of users and user sessions.

Chapter 4: Operating System Kernel the monitor, and supervisors, distributed collection of features that make up the kernel of EOS Release 1, Version 1.0.

Chapter 5: The Mainframe Components describes the ETA10's CPU architecture and instruction set, system memories, interfaces, and transfer mechanisms.

Chapter 6: The Input/Output Subsystem describes I/O interface architecture, I/O unit (IOU) hardware and software, as well as the disk storage system, channels, and interfaces.

Chapter 7: Networks describes two networks supported by the ETA10: a local area network and a network supporting communications with remote hosts.

Chapter 8: System Peripherals describes the ETA10's disk storage peripherals.

Chapter 9: Service Unit Features and Functions describes the hardware components, support software, applications, file system, network, and operator environment of the service unit (SU).

Chapter 10: Power and Cooling Systems describes the cryogenic, refrigeration, and power systems.

The final component of the manual includes a glossary, a bibliography, and an index.

How to Use This Document

The general table of contents covers major subjects. More detailed tables of contents at the beginning of each chapter include lists of figures and tables. A glossary and an index appear at the end of the manual.

Conventions Used in This Document

An attempt has been made in this manual to expand the acronyms used in developmental documentation; e.g., "CPM" becomes "CP memory" or "central processor memory", "IOU" becomes "I/O unit", and so on. However, developers' acronyms are retained in diagrams and descriptions where they are most suitable.

Related Documents

The following ETA Systems publications are useful supplements to the System Reference Manual. The System Overview is a good introduction, and the manuals on VSOS and support tools give information (especially command and parameter descriptions) in greater detail than in this manual, which is meant as a conceptual rather than a hands-on guide.

PUB–1006	*ETA–10 System Overview*
PUB–1118	*Support Tools: Utilities, Debugger, B.E.S.T.*
PUB–1119	*Support Tools: Diagnostics*
PUB–1051	*VSOS Environment Reference Manual: Concepts and Commands*
PUB–1084	*VSOS Environment Reference Manual: System Interface Library Calls*

CHAPTER 1

Chapter 1

Introduction to the System Architecture

Chapter Contents

Chapter 1

Introduction to the System Architecture

In This Chapter . . .

Chapter 1 serves as the introductory chapter in this reference manual for the ETA10 supercomputer. As a system reference, the manual provides a functional description of the system, describing the hardware and software both as discrete components and as parts of an integrated system.

Section 1 introduces the two basic models of the ETA10 system and their major components. Section 2 is an architectural overview of the components, and a guide to the chapters that describe them. Section 3 discusses the range of ETA10 models and their cooling and redundancy options.

Section 1: Overview of the System Components

- Guide to chapter contents
- Introduction to super-cooled and air-cooled systems

Section 2: System Architecture

- Overview of computational, memory, I/O, and network architectures
- References to chapter contents

Section 3: ETA10 Model Range

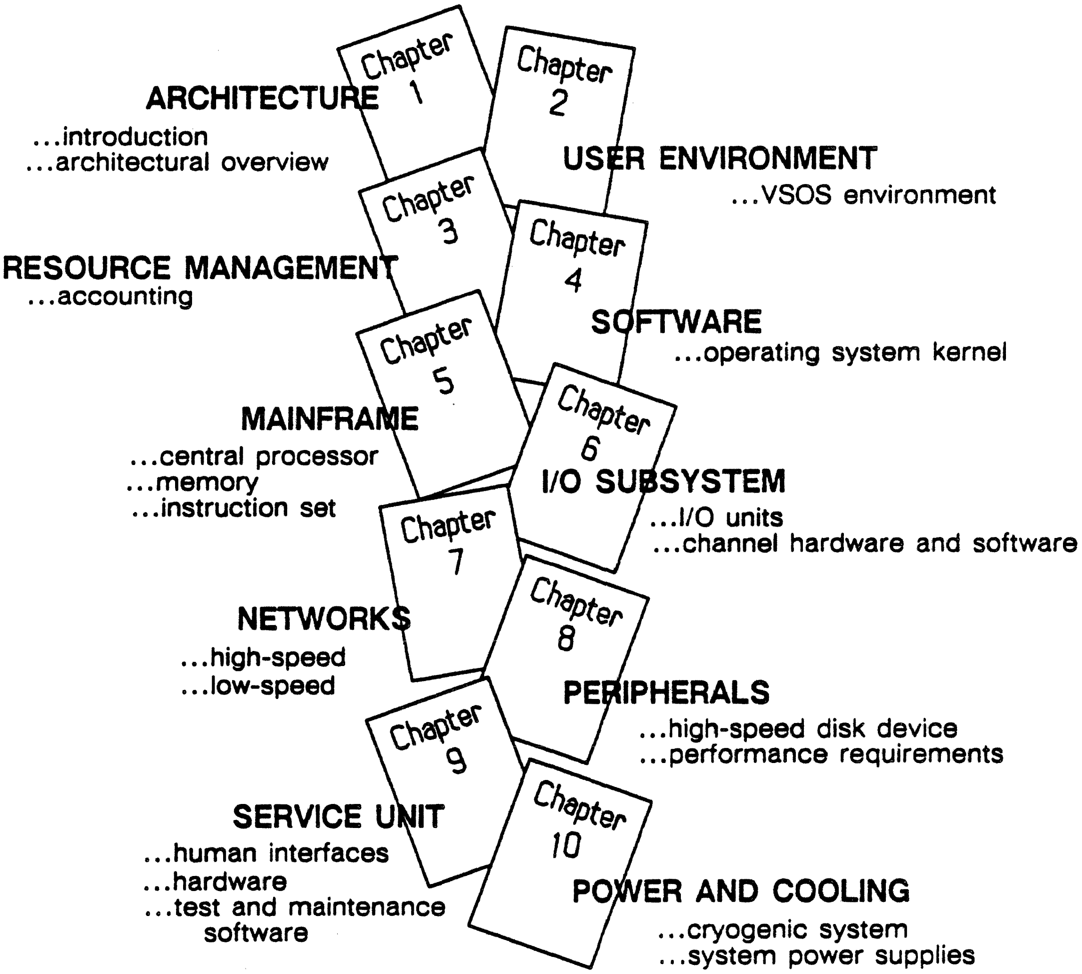
- Cooling options for model range
- Redundancy options for model range

Section 1: Overview of the System Components

An ETA10 system is comprised of the following components:

- one to eight central processing units
- 4-million word central processor memories, a large shared memory, and a communication buffer
- one to eighteen input/output units
- service unit for system maintenance and control
- power and cooling support appropriate for model
- distributed set of operating software
- set of user environment and application software

Guide to the Chapter Contents in this Manual...



Super-cooled Systems

Here is a computer-room view of a super-cooled ETA10 system:

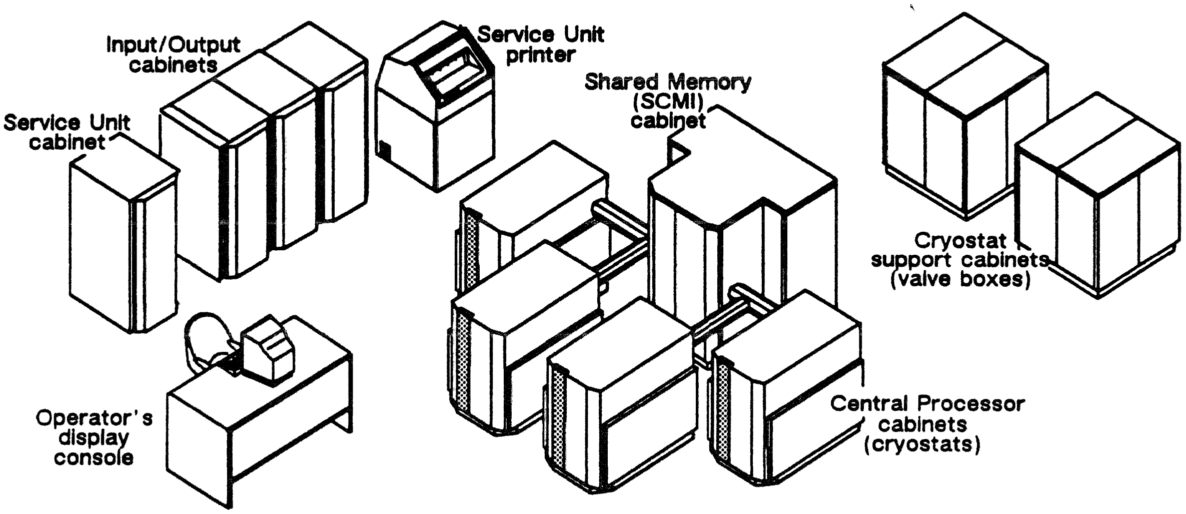
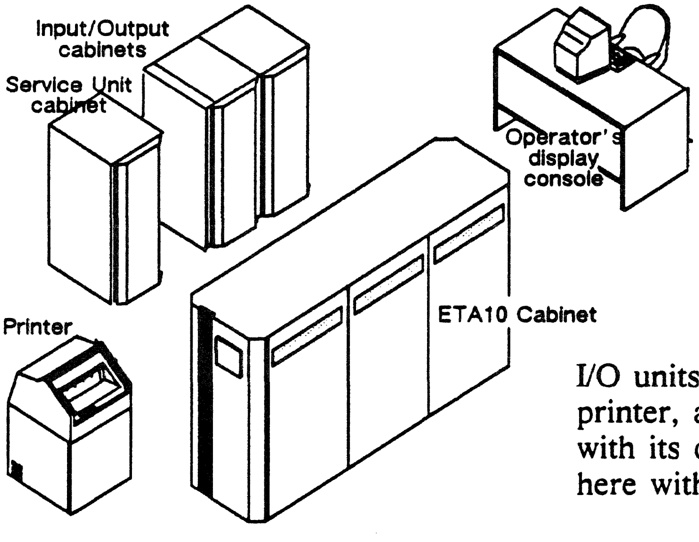

Input/Output cabinets
Service Unit printer
Shared Memory (SCMI) cabinet
Service Unit cabinet
Cryostat support cabinets (valve boxes)
Operator's display console
Central Processor cabinets (cryostats)

Figure 1–1. Computer-room view of super-cooled ETA10 system with four to eight processors.

In the center are the super-cooled central processors in their cryogenic housings, the cryostats. Since one or two processors may be contained in each cryostat, this can be a four-to-eight processor system. The central processing units directly access the shared memory housed in the tall cabinet behind and connected to the cryostats.

Cryogenic support cabinets provide mechanical controls for the cryostats, and are installed nearby. The I/O units and the service unit components are individually housed in identical cabinets. Also shown is an operator console with its display workstation; there may be several of these workstations networked with the service unit.

Air-cooled Systems



In contrast, the computer room-view of an air-cooled ETA10 is markedly different. One or two air-cooled central procesors, the memory, and the system interfaces are all packaged within a single cabinet.

I/O units, the service unit cabinet, printer, and an operator's console with its display workstation are shown here with the air-cooled ETA10 cabinet.

Figure 1–2. Computer-room view of a single processor air-cooled ETA10 system.

Although the models differ externally, internal views of both air-cooled and super-cooled ETA10 systems are identical. The architectures are the same: fast central processors, a large virtually addressed local memory, large second-level shared memory, wide bandwidth I/O transfers, and a distributed operating system. This manual describes the super-cooled ETA10 models.

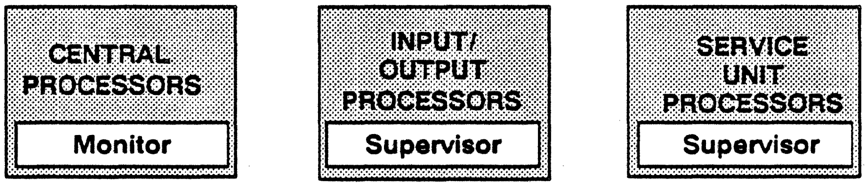
Operating Software

The ETA10 operating system, EOS, has three components:

- processor-specific core software
- system-wide kernel features
- applications software supported by the first two categories

Processor-Specific Monitor and Supervisors

Each type of system processor has its own software to provide its basic, low-level functions: the mainframe central processor has monitor software that runs in monitor mode. The 68020 processors in both the service unit and the I/O processors have individualized supervisor software that runs in supervisor mode.



EOS Kernel

The kernel level of EOS is distributed across the entire hardware system. Portions of the kernel coordinate activities among system processors, and manage the memory hierarchy to balance demands of the computational engine with wide bandwidth I/O from peripheral disk and network systems. Kernel features coordinate among processors to manage memories, I/O capabilities, and system maintenance/communication functions.

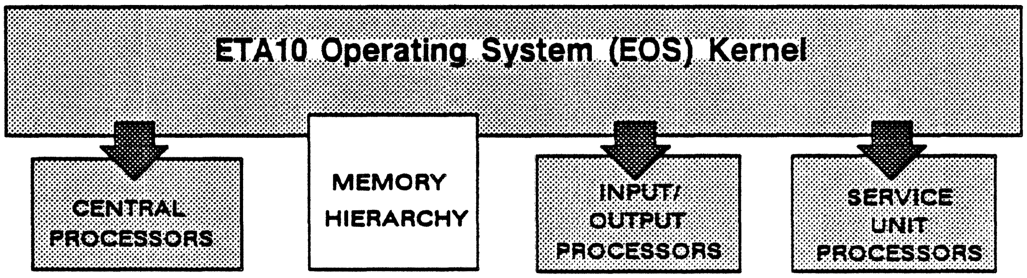


Figure 1–3. The EOS kernel is distributed across system hardware.

Kernel features perform a wide range of functions. The logical file system provides implicit and explicit I/O. Process management controls the multitude of processes active on each central processor. Domain management software provides efficient access to system operating code and routines without resorting to process switching. Interprocess communication uses the communication buffer to create and control semaphores that coordinate activities among central processors. Remote procedure calls enable the three types of system processors to communicate with each other. Memory managers coordinate the multi-level hierarchy of system memories. Another set of features provide system scheduling, monitoring, accounting, and control functions.

Applications Software

The applications software is supported by the first two categories. It includes the VSOS user environment, system and special libraries, compilers and debuggers, system utilities, and development and vectorizing tools.

Section 2: System Architecture

The ETA10 implements a balanced architecture characterized by parallelism throughout its high-speed processing, addressing, and data transfer operations. The multi-pipelined computational engine is balanced by extremely fast I/O, and by a nearly unlimited virtual memory space to support the multitude of requests coming from high-speed disks and multi-user local area networks.

The ETA10 is a fully distributed processing machine; work is spread throughout the system in such a way that central processors execute, I/O processors move data, the service unit monitors performance, all done simultaneously. Central processors dedicated to execution are balanced by independent I/O and service unit processors that perform other necessary but non-computational system work. The addition of the communication buffer enables the central processors to efficiently share and multiprocess data as well as to manage complex system resources.

Architectural Overview

At the heart of the ETA10 system is a high bandwidth shared memory that satisfies the data demands of vector processors and high-speed I/O transfers to disks and networks.

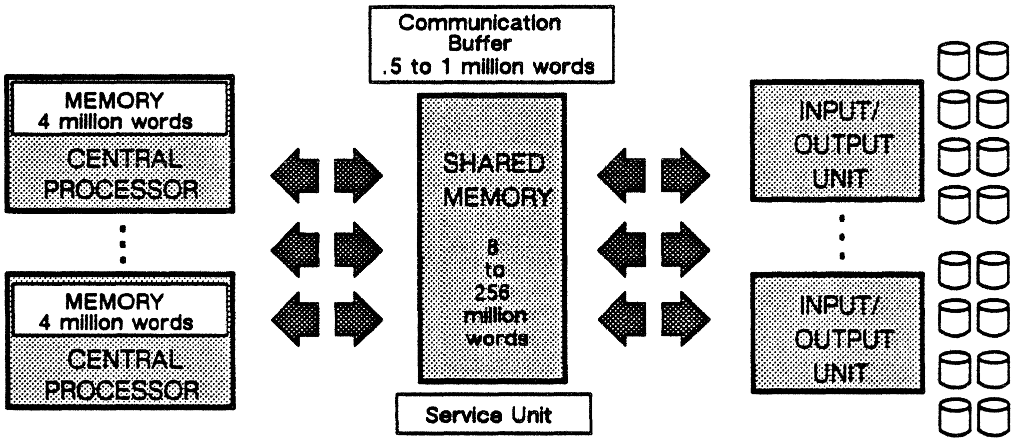


Figure 1-4. System components of the ETA10.

Each central processor has both a scalar unit and a double-pipelined vector unit that supply the computational power in the system. The data needs of these very fast processors are met by four million words of virtually addressed memory and ample system bandwidths. Shared memory makes direct transfer to the local central processor memory. The shared memory allows large blocks of data to be moved efficiently to central processor memories. I/O bandwidths allow data transfers to/from peripherals at high rates.

Hardware and Software Integration

Within the ETA10, hardware and operating system components execute programs concurrently, supported by large memory bandwidths and parallel processing capabilities. Software maximizes hardware resources by keeping several user programs in shared memory at the same time, rapidly switching the processors between programs as priority and data availability dictate. Hardware and software combine to provide parallel protection schemes: protection between programs, protection between processes, and protection within processes.

The operating system plays a major role in managing memory and controlling memory contention. The operating system must distribute data over the large shared memory so as to make the data quickly and readily accessible when requested.

Chapter 2 describes the VSOS user environment. Chapter 4 covers the main part – the kernel – of EOS (the ETA10 Operating System). System resource management is discussed in Chapter 3.

Computational Architecture

Central processor operations overlap with shared memory data transfers. Once the transfers from shared memory are started, data moves to the processor at one 64-bit word per clock cycle. Transfers can be made at this rate per cycle to five processors at once or, in redundant systems, to all eight processors.

Central processor memory moves eight 64-bit words per clock into the processor. The central processor has a scalar unit and twin vector pipelines that operate totally in parallel, the scalar unit producing one result per clock cycle, the vector unit producing two 64-bit results or four 32-bit results per cycle.

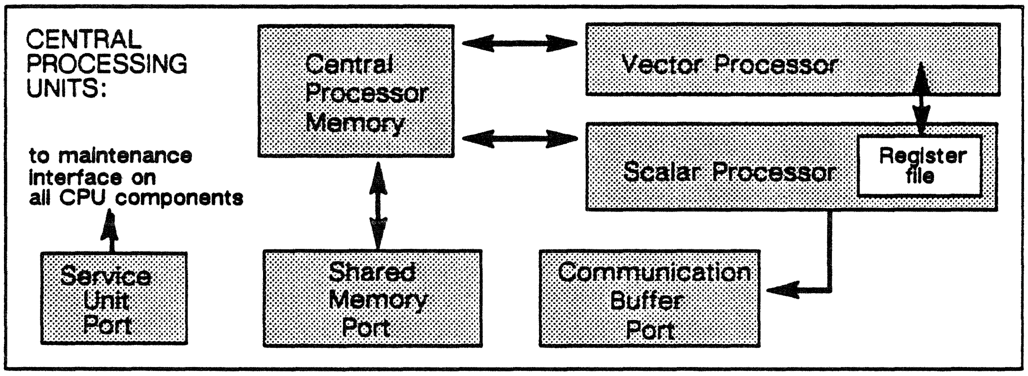


Figure 1–5. Components in the ETA10 central processing unit.

Vector operations can be overlapped, allowing a vector operation to be starting while another is completing. This overlap improves the performance of short vector operations.

Machine Arithmetic

A twos complement binary number system is used in ETA10 arithmetic operations. Floating point operations use twos complement arithmetic in formats for both full and half words. Full words have 64 bits, half words have 32. The benefit of using 32-bit words is that two pieces of data can be put in the same space; also, a vector processor processes twice as many 32-bit operands in the same amount of time. In the 48-bit coefficient of a full 64-bit word, numbers are precise to 14 decimal digits; in the 24-bit coefficient of a 32-bit half word, numbers are precise to about 7 decimal digits.

Machine arithmetic is described in Chapter 5, in the "Central Processor Architecture" section.

The Instruction Set

The ETA10 architecture uses a large instruction set designed primarily for vector operations. Both the ETA10 instruction set and processor hardware are vector-based; they are not merely extended to provide vector operations. The ETA10 instruction set derives historically from Iverson's *A Programming Language* (APL). The significance of APL is that vectors are inherent to the language rather than additions to it.

The ETA10 instruction set is model independent, and is compatible with Control Data Corporation's CYBER 205 instruction set. It has 256 function codes, 40 of which are unused. The richness of this set is extended by sub-functions that give an instruction additional refinement. Special shared memory instructions manage the queues used in memory-to-memory transfers between central processor memory and shared memory. A group of communication buffer instructions enable the central processors to communicate and share data.

The instruction set is described in Chapter 5, in the "ETA10 Instruction Set" section.

Memory Architecture

The ETA10 central processor supports a virtual memory system with virtual address space of 2 trillion words. In this system, users request memory rather than manage it; large jobs can be staged without loss in throughput and without involving the user. The system includes very large hierarchical memories. Fewer memory conflicts occur with a hierarchical structure than with systems depending upon a central memory. A hierarchical structure supports large numbers of small jobs as well as very large jobs. Explicit file I/O is also available.

Memory management software is described in Chapter 4, "Operating System Kernel". Memory hardware is described in Chapter 5, in the "System Memories" section.

Memory Hierarchy

At the top end of the hierarchy, speed and ease of memory access are greatest while memory size is smallest. The central processor has a 10 nanosecond access time for the register file; a central processor waits one second for data to be transferred in from an I/O unit. Although memory size increases at the other end of the hierarchy, access time also increases.

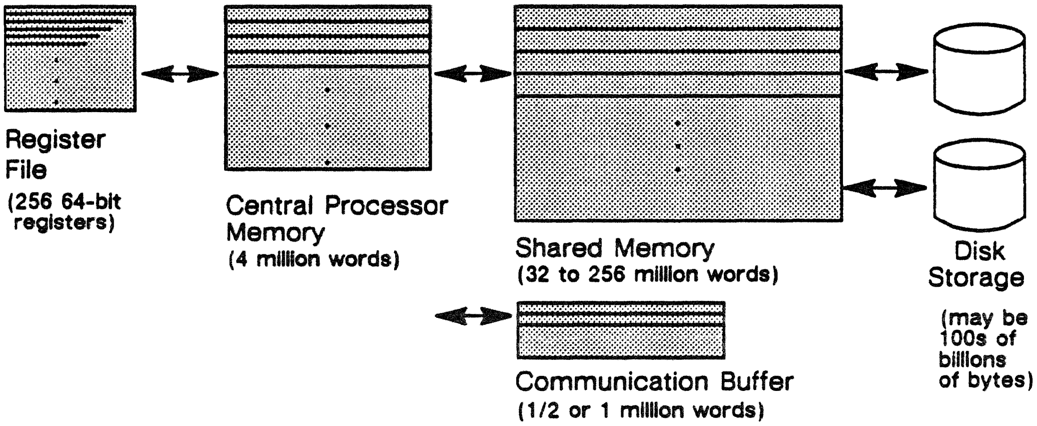


Figure 1-6. The ETA10 memory hierarchy.

Shared Memory

All system processors have access to shared memory. Shared memory serves as the repository for work as it comes into the system, executes, and leaves the system. When programs are multiprocessed, different pages of data are moved from shared memory and sent to different processors. The central processors communicate between themselves using shared memory and the communication buffer to pass messages back and forth to coordinate program execution. The I/O subsystem accesses shared memory to send or receive requests that bring work into the system and that move data out to the disks or networks.

Input/Output Architecture

An I/O unit functions in the system as a set of independent, multifunctional channels, able to access both shared memory and the communication buffer. Within each I/O unit, several types of channels interleave high speed transmissions from disk drives and network interfaces.

The software architecture distributes most of the I/O functionality within I/O processor-based code. Memory management software resident on the central processors makes and confirms I/O requests. Software required to interface network user programs for processing also resides on the central processors; other network protocol and message packaging software resides on the I/O processors.

I/O hardware and software are described in Chapter 6, "I/O Subsystem". Disk characteristics and requirements are described in Chapter 8, "Peripherals".

Network Architecture

Both a high-speed and a low-speed network are implemented in the ETA10 system architecture.

Low-Speed Network

The first network is workstation-based, and reflects the growing trend for supercomputing "on demand." This is the Open Interconnection Network, a local area network that serves to connect users into the system via a standard mode of access. This network allows users to take advantage of the ETA10's ability to efficiently handle a great many small, interactive sessions. The ETA10 interface to this network provides system access for a large number of network users and ensures reasonable performance and response time. The peak transfer rate across the Ethernet exceeds 1 million bits per second.

High–Speed Network

The second network is mainframe-based, and reflects the typical use of supercomputers for processing huge amounts of data stored on large systems. This is the Multi-Host Network, a loosely coupled network that connects mainframes rather than users to the ETA10. High-performance mainframes and peripherals are connected to the system over a trunk that achieves a peak transfer rate of approximately 6 million bits per second. The higher performance is required because the work processed via this type of network is large-batch oriented.

Chapter 7, "Networks", describes both types of networks and their interfaces to the ETA10 system.

Section 3: The ETA10 Model Range

The ETA10 series of supercomputers include two air-cooled models and two super-cooled models. The P and Q series are air-cooled and achieve peak 32-bit performance ranges of 375 to 950 MFLOPS. The E and G series are super-cooled and achieve peak 32-bit performance ranges of 850 to 10,200 MFLOPS. (Peak 64-bit performance is one half the 32-bit performance.)

All ETA10 models are architecturally identical. All models use the same central processor, instruction set, and primary hardware components; all models run under the same operating software and applications set. Performance increments are gained by adding central processors and pluggable memory cards, and by super-cooling central processors.

This chart compares characteristics of the ETA10 models:

ETA10 Model:	Number of CPUs:	Cycle Time: (ns)	Million Words of Shared Memory:	Cooling Method:
P	1-2	24	8-64	AIR
Q	1-2	19	8-64	AIR
E	1-4	10.5	32-128	SUPER
G	2-8	7	64-256	SUPER

Although ETA10 models are architecturally identical, cooling and redundancy options provide ranges of system performance and availability during failure and maintenance.

Cooling Options

Cooling options provide significant performance enhancements for the central processor boards. CMOS semiconductor technology is used in the high density chips that populate the central processor boards. Because CMOS chips use relatively little power and dissipate very little heat, they need only be cooled with ambient air to function normally. The ETA10 P and Q models are cooled in this manner.

However, by substantially lowering the environmental temperature, logic gate delays are greatly reduced and double the processing capability of the board. Central processor boards in ETA10 G and E models are housed in cryostats where they are super-cooled in baths of liquid nitrogen. Super-cooled systems offer very exceptional levels of performance; air-cooled systems offer efficiency in installation and lower costs of operation.

Redundancy Options

System redundancy ensures the system is available during component failure or maintenance. Availability is used as a measure of the system's ability to function and to perform during failure or maintenance.

Redundancy means that the user sees no change in the system during instances of failure or maintenance. Redundant design in the ETA10 is apparent in several ways:

- multiple components that can independently perform the same function. Normally these components extend system capability; but they can substitute for other components as needed. For example, the multiple central processors.
- redundant components that are included as insurance; they may or may not extend system capability. For example, the parallel power supplies.
- redundant connections between components that are another form of system insurance

The upper portion of this diagram shows the number of components in each category included in a typical non-redundant model E system. The numbers refer to how many of the components are in the system.

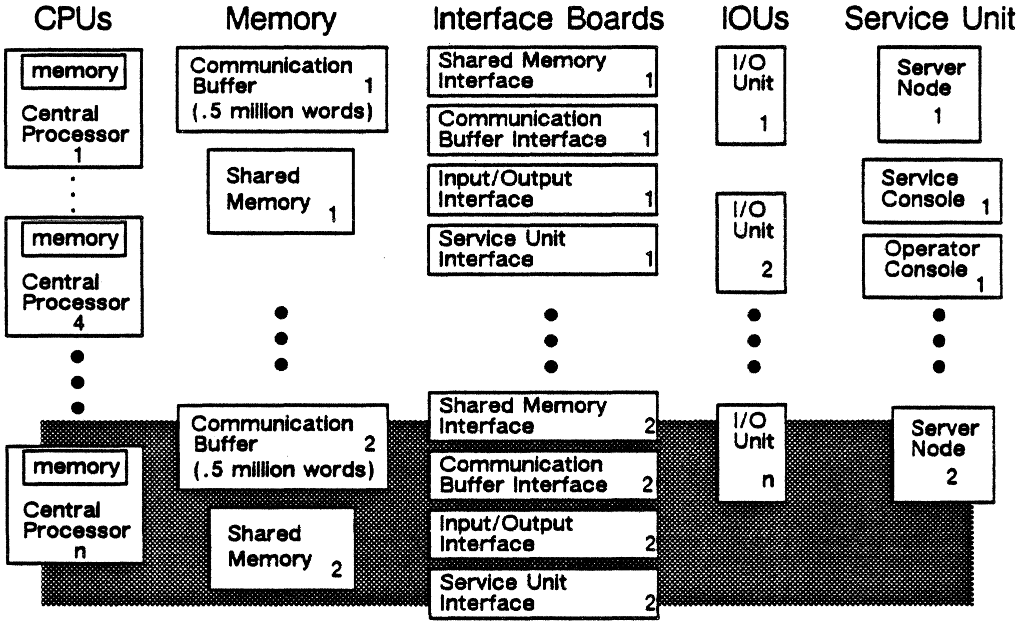


Figure 1–7. Redundancy options in the ETA10 super-cooled systems.

The components within the shaded area at the bottom are those added in each category to provide redundancy in a model E system.

Multiple Components

Both redundant and non-redundant models have multiple central processors and I/O units: these units function independently and readily substitute for a like unit removed from the system configuration. Multiple processors are required to meet system performance needs, but they also serve to ensure system availability.

Redundant Components

The diagram above shows the components required to make the system redundant, absolutely ensuring system availability. Required components are a shared memory, a communication buffer, a set of the major system interface boards, and a second service unit server node.

When there are two units of shared memory and communication buffer, each fully functions as a component "half". For example, IOUs that port into one unit of the shared memory are also fully able to access the second or redundant unit of shared memory. Redundant interface boards may enhance system performance as well as provide system insurance. A second I/O interface board doubles the number of I/O ports into shared memory; a second shared memory interface board doubles the number of central processor access slots to shared memory. A redundant system includes a second server node in the service unit to maintain parallel connections to the system maintenance interface.

Redundant Connections

Some components are dually connected. Each IOU is connected by a pair of serial input/output lines to each service unit server node. Peripheral devices and networks are usually dually connected to two IOUs to maintain connections even in the event of failures.

CHAPTER 2

Chapter 2

The User Environment

Chapter Contents

Contents (continued)

List of Figures

List of Tables

Chapter 2

The User Environment

In This Chapter

Chapter 2 introduces you to the Virtual Storage Operating System (VSOS) environment available on the ETA system.

- The Environment Model
- Overview of the ETA VSOS Environment
- Functional Description of VSOS
- The VSOS Command Shell

The Environment Model

Introductory Description

EOS is a general purpose operating system that is designed to support multiple views or presentations of its functions. EOS comprises a kernel, a set of interface libraries, a programming environment, and support tools. The kernel provides basic system functions to the user: file management, I/O, memory management, and process control. It manages system resources, and controls system hardware and software.

User applications running on the ETA10 make use of kernel functions by means of a programming environment that maps on to the underlying operating system transparently to the user. The programming environment, a layer of software built on top of the kernel, handles the necessary interaction between the user and the kernel's services. The environment enables you to take advantage of the machine's capabilities using a language and context that is familiar.

Figure 2-1 shows the layers of software on the ETA10: the basic kernel functions, an interface allowing access to these functions, and the ETA10 user environment, which accesses the functions by way of the interface.

All the necessary pieces for program development and execution - libraries, editors, compilers, debuggers and other tools, are accessible from the programming environment. Users have access to a common set of languages and tools, such as FORTRAN 200, ETA Debug, and the loader. These products are defined independently of the environment, but use some of the environmental support for context. For example, ETA Debug uses the file search algorithm of the active environment to locate files.

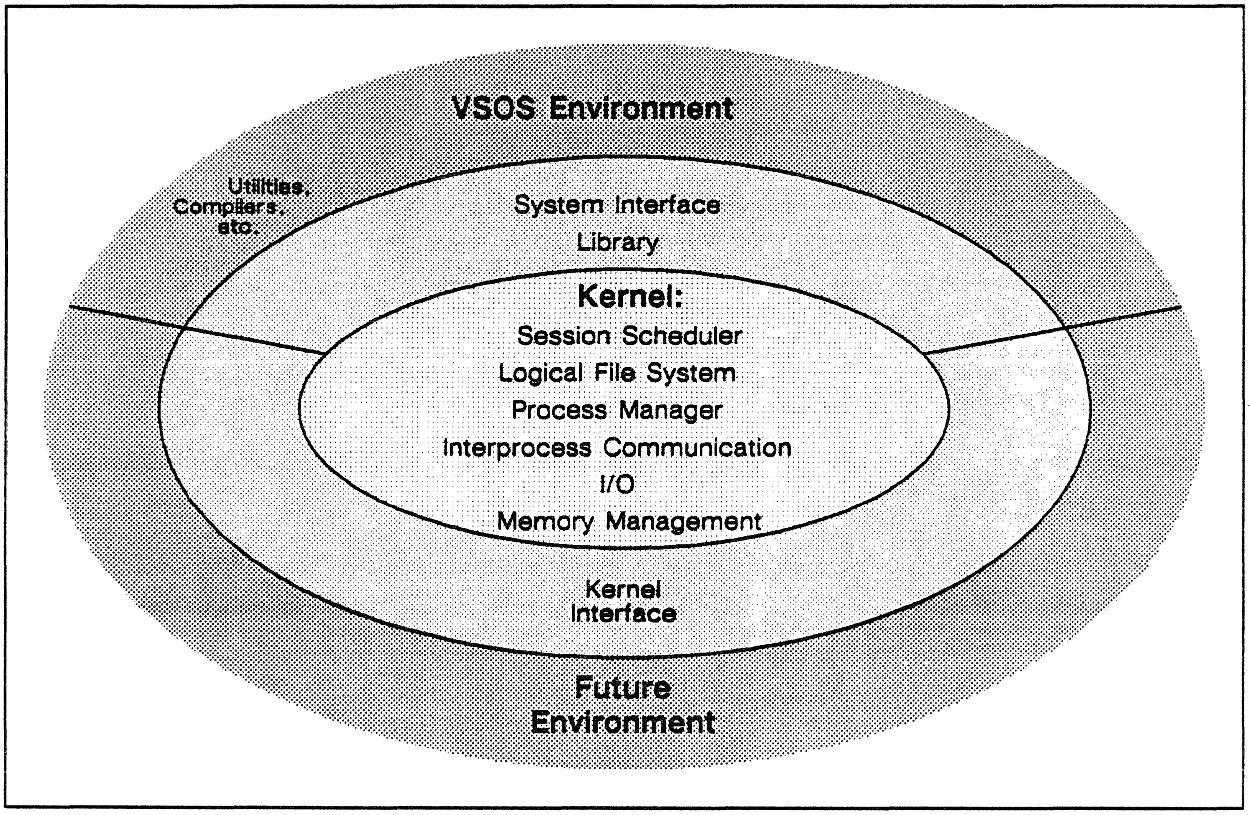


Figure 2-1. The environment model.

An environment provides basic features to its users. A *command language* allows you to initiate utility programs, compilers, user applications, and to specify parameters controlling their execution. The *command shell* interprets the command language and causes the appropriate commands to be executed in a session. *Libraries* provide the interfaces between a user program and the operating system, allowing programs to communicate with the kernel using subroutines or system calls. ETA Systems provides *application packages* for many subject areas, including biotechnology, computational fluid dynamics, and structural analysis.

The VSOS User Environment

ETA Systems supports the Virtual Storage Operating System (VSOS) command set, the user interface of the CYBER 205, as its first programming environment. There are some differences in the ETA10 version. CYBER 205 features which have no analogue in ETA computer systems have been deleted. Some utilities have been replaced by new ones with equivalent or new functionality. New features have been added. Richer interactive abilities are provided and the user interface has been made more powerful. The main differences between CYBER 205 VSOS and the ETA10 system are discussed later in this chapter.

Calling a kernel function from an environment

Applications running in an environment call the kernel through an interface which is a collection of routines. For example, VSOS applications call **System Interface Library** (SIL) routines, which communicate with the kernel to complete both I/O and non-I/O functions. The interface used by an environment is specific to that environment.

Figure 2-2 shows a request to the operating system from an application running in the VSOS Environment. The request is to prepare a new process for execution. The application issues a SIL subroutine call, Q5INIT (initialize a controllee), to a process management function in the kernel, requesting that it run a process, and passing information about the process that is to run.

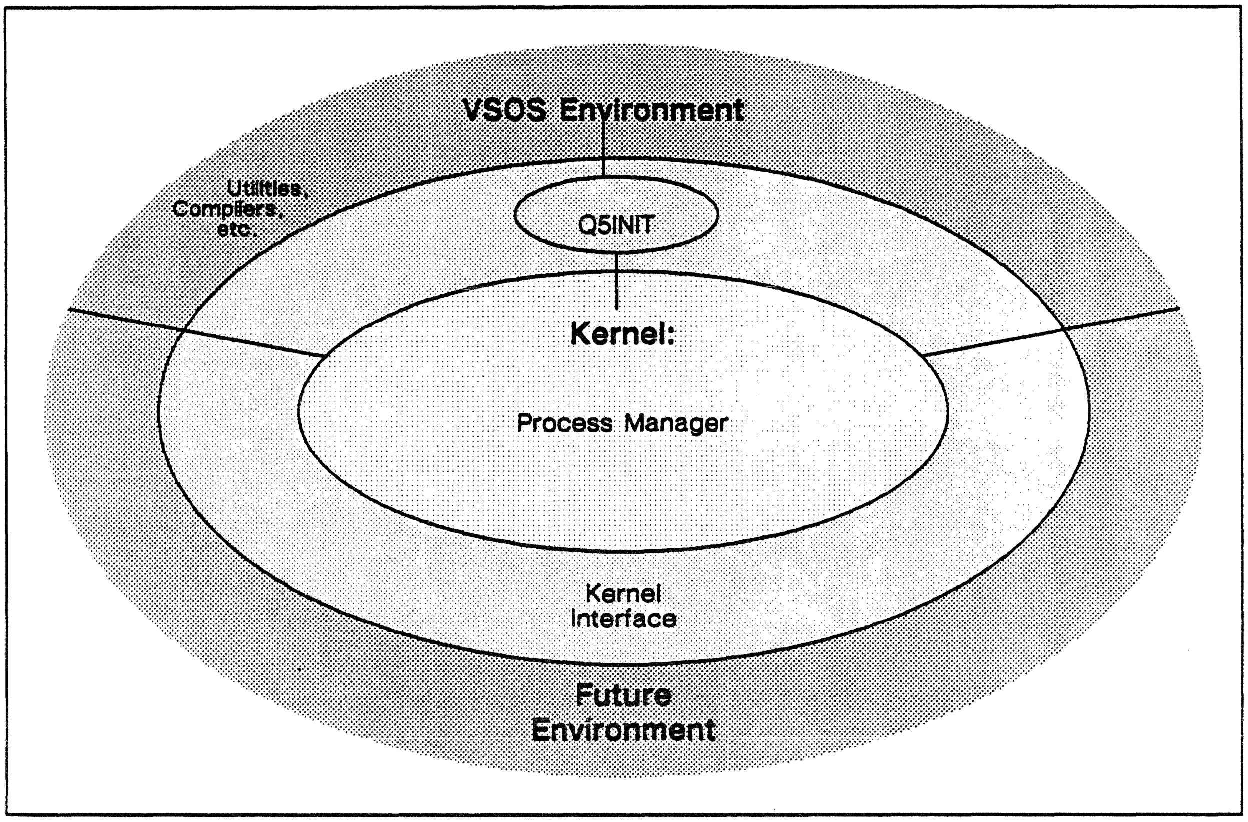


Figure 2–2. Calling a kernel function from an environment.

Elements in the Environment

Operating system components fall into two groups in relation to user environments: those dependent on an environment, and those independent of it. Environment–independent tools serve as common resources.

Environment–independent components

Kernel functions. These are accessed by user environments, but their structure is not dependent on an environment's functional requirements.

A common set of languages and tools. These include the FORTRAN 200 compiler, ETA Debug, and the loader.

The object text and executable file formats.

File representation and file limits. These include the maximum number of records in a file, length of records, the maximum number of file objects, the length of the data that can be transmitted during I/O, the maximum number of outstanding I/O operations allowed, and the maximum number of files that can be open at the same time.

File access permissions. The ETA file system allows access to a file based on a state ID, which contains the user name, the user's group ID, and the domain name. The state ID is checked against valid permission types for that state to decide if the user can perform file operations. The access types exposed depend on the environment.

User validation and privileges, including system security, access to the system and user objects, and validation during the user login procedure. These are controlled by the kernel features.

File attributes. Intrinsic attributes necessary to define a file include the physical file identifier, the owner user name, and the file's device class identifier. Record management attributes define such elements as record type, maximum record length, and padding characters. Accounting and statistical attributes describe the file's creation date and time, the account ID, information about when the file was last modified, and so on. Resource limitation attributes include the file's device class and the current file length.

Environment-dependent components

The interface to kernel functions. In the VSOS environment, these consist of subroutine calls that request activity and return results.

Libraries containing routines tailored to a particular environment, such as the System Interface Library for VSOS.

Utilities.

The command processor and command formats.

User validation. This is conducted at login time to determine a user's access rights to the selected environment. A default environment is activated for the user who does not select an environment at login time.

The process management model. Processes in a session run under the control of the environment in which they are activated. Inheritance information is passed on to new processes in the session. This contains data about directories known to the user environment, as well as other needed information.

File permissions, which allow users to reference files. An environmentally determined access state allows users to access a file. A state ID, containing the user name, the user's group ID, and the domain name, is checked against the type of file operation requested by the user and access permission granted accordingly. File attributes which are dependent on the user's environment include the type of file access - sequential or direct. Although the system does not define a file type as having sequential or direct access, a user environment does make that distinction.

File system model. The environment presents a view of the underlying file system that is tailored to its user.

Management of Users and User Sessions

Anyone privileged to log in to the system is a user, but users differ in the extent to which they are privileged to access files, system resources, and other users. At the top of the pyramid is the *system administrator*, who has almost unlimited access to the system. The system administrator grants system access to other users and oversees their privileges. Depending on the site, there may be additional site administrators under the authority of the system administrator. A *project administrator* manages privileges for all the users on a project, while an *account administrator* does the same for the users associated with his/her accounts. System, project, and account administrators are all users in the sense that they have user IDs and passwords; in addition, however, they have special privileges that allow them to fulfill their roles in site administration. An ordinary user may be privileged only to log in and run a specific program, and may make changes only under his own user ID.

User Access to the System

Site administration controls user access through the *user registry*, the interface through which users are identified to the system. If the system cannot identify the user ID or password of someone logging in, the login is rejected.

The user registry is the record of all system users and privileges granted to them. Examples of these privileges include the permission to submit batch sessions and the permission to charge resource usage to a particular project and account. The user registry is covered in greater detail in chapter 3 in "The User Registry" section. The set of privileges for a user is called the *user profile*.

A user profile is a list of those system privileges and features available to a certain user; it is a subset of the user registry. The system administrator sets up user IDs, supplies them to the user registry, and creates the individual profiles that define user access to a subset of the available system features.

Each user profile contains a set of default session parameters which include default accounting information and default working-directory parameters that execute automatically when the user logs on or off.

Environment Processes

In an ETA10 environment, a process is the execution of the smallest unit of work that can be executed. An executing program is a process. Tasks in the VSOS environment map to the processes that the kernel works with on the ETA10 system. A series of processes constitutes a session. Processes run under both batch and interactive sessions, and are monitored and controlled on a session basis. Once a process is activated, it runs in its user's environment for the duration of the session. Users cannot switch environments in mid–session. For a discussion of VSOS task execution, refer to the *VSOS Environment Reference Manual: Concepts and Commands, PUB–1051.*

Each process is spawned serially. A spawned process can start other processes - one at a time. The starting process is the controller, the process it starts is its controllee. A series of two or more controllers and controllees forms a controllee chain. The command shell is the first process born in a session and is always a controller, at level 1. It may start a new process, or controllee, at level 2, which, acting as a controller, can spawn a controllee at level 3, and so on. After a process is spawned, the shell continually checks process management for the status of the spawned process, until the spawned process terminates. When the process has completed, the shell spawns the next process to execute.

The VSOS environment always spawns processes in hierarchical fashion. It never has multiple child processes running at the same time. However, use of a library of multitasking routines available in future releases makes a difference in the way child processes are spawned, as shown in figure 2–3. Multitasking allows a user's program, not the command shell, to spawn processes horizontally, producing multiple child processes running at the same level.

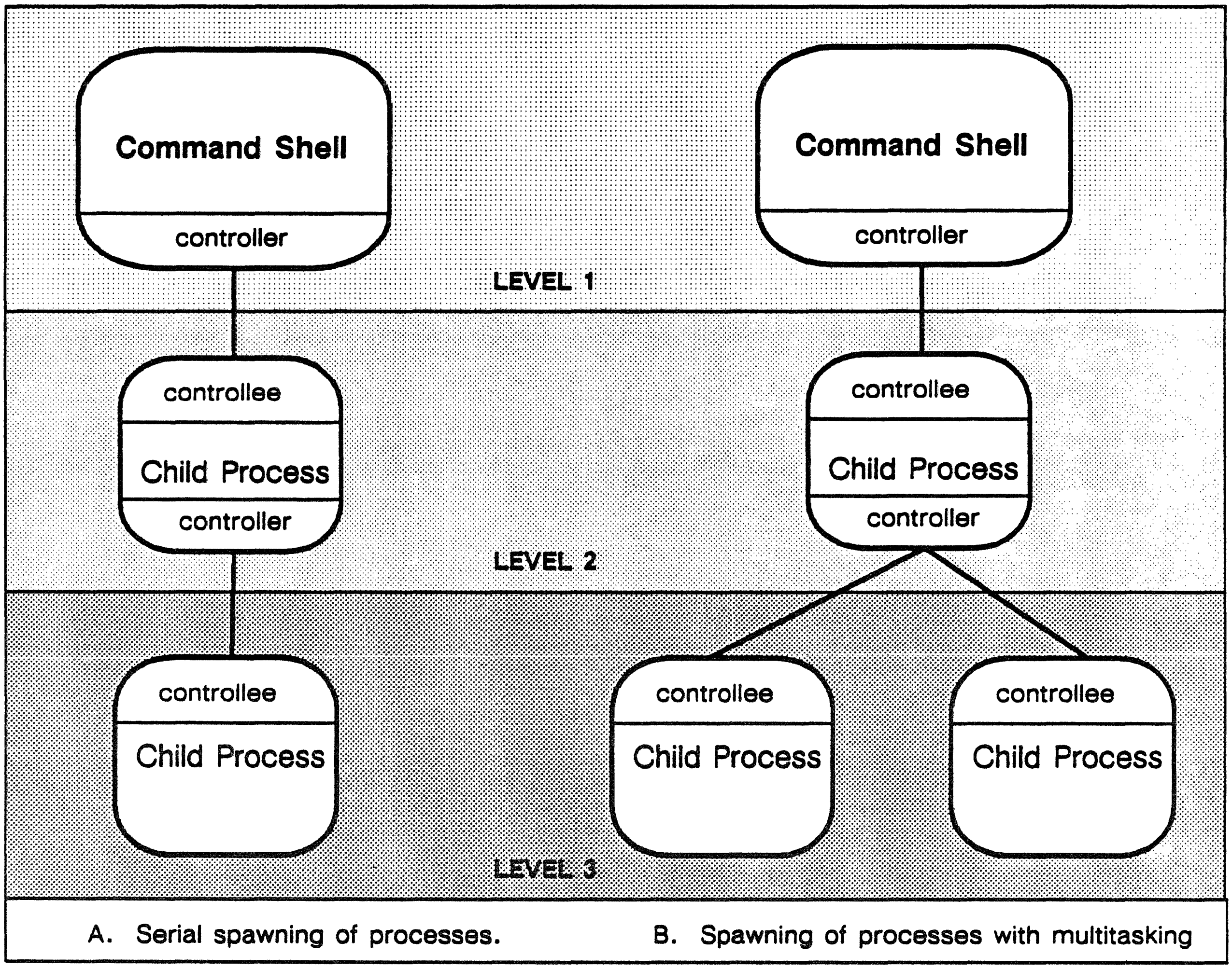


Figure 2–3. Spawning of processes in the VSOS environment.

User Sessions

On the ETA10, jobs and sessions are the same work unit, and the system manages batch jobs and interactive sessions in exactly the same way. ETA Systems uses the term *session* to refer to this work unit on the system.

Every session is the execution of a set of processes, whether the process name arrives at the command shell from a batch input file or directly from a user's keyboard.

- Every session runs under the control of the user environment. A user logged into a VSOS session remains in that environment till logging out; the user session runs under control of the VSOS shell that enables the user to enter commands and do work.
- Every session has a unique name, so that a user or operator can refer to all of the work being performed in it as a single unit.
- The operating system uses the session as an accounting and control envelope for a set of process executions.

Batch Sessions and Interactive Sessions

Although the ETA10 treats batch sessions and interactive sessions in the same way, there are some differences between these types of user sessions, as shown in figure 2-4. An interactive session usually involves a two-way exchange of prompts and responses. As an example, a user logs in, issues a command or commands, receives a response, and the user logs out. In a batch session, a file of commands is entered into the input queue and executed later. In the VSOS environment, a batch session can be submitted to the input queue from an interactive session.

The input and output from batch sessions are associated with files; the input and output from interactive sessions are associated with terminals. Local batch sessions terminate after the output file is transferred to the user's home directory, and interactive sessions terminate when the user logs off.

INTERACTIVE SESSIONS	BATCH SESSIONS - 2 START OPTIONS
START-UP:	START-UP:
• USER logs in, connection is established by Common Login Processor...	1. ENQUEUE command (local batch) via an interactive session... 2. ENQUEUE command (local batch) via a batch session...
• User session is queued as an interactive login...	• Session is created and queued to input queue...
• User environment and command shell established...	• When session is selected to execute, the user environment is established...
EXECUTION:	EXECUTION:
• User processes execute and complete...	• User commands are executed from file...
COMPLETION:	COMPLETION:
• User logs out...	• Output is returned to specified file ...
• Session terminates...	• Session terminates...

Figure 2–4. Comparison of interactive and batch sessions.

Session Start-Up

Although batch and interactive sessions arrive on the system in different ways, the two kinds of sessions are validated in the same way (figure 2-5). The user management feature continually runs on the system, managing system use and access. It prompts for the username and password on the login line, then user management turns the session over to global scheduler.

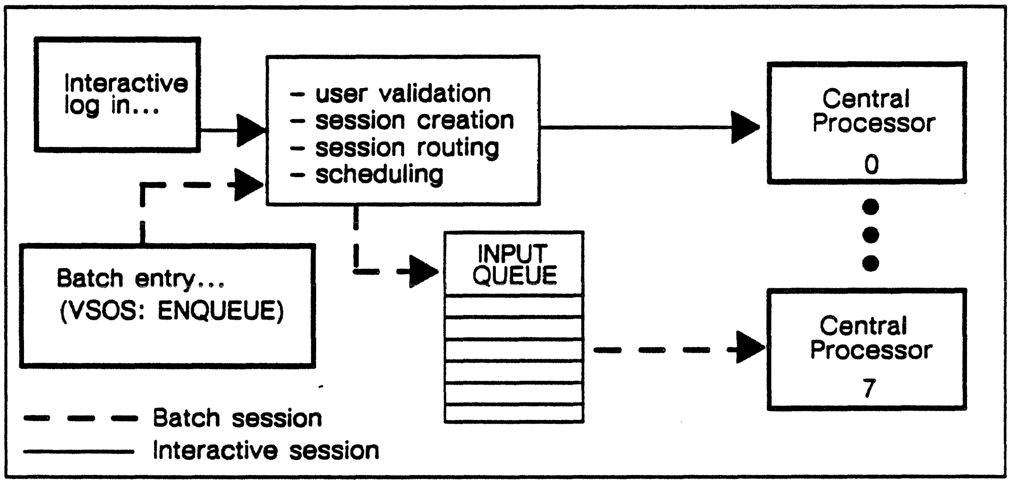


Figure 2-5. System entry of batch and interactive sessions.

Session Execution and Completion

Global scheduler stores the information about the session and the processes included in it, and creates the session shell - a process that starts and runs the session. Then Global scheduler moves the session into the session input queue. When it finds a central processor with enough memory available to start a new process, Global scheduler sends the processor one of the session's processes for execution. Associated with each process is a process table that includes the names of the executable files, the name of the shell, for example, as well as other information needed by processes making up the session.

When the last process has completed, the environment command shell sends a report to the system dayfile. If it is a batch session, the processes have written to the indicated output file or files.

The Login Process

The common login feature validates user logins; it is initialized during system start-up and runs continuously on the system. It has two functions: the first is to respond to users logging in and to interactively prompt for user ID and password, and the second is to validate the user ID, password, and privilege to log in and, when those items are valid, to create the appropriate session.

When the user has been validated, the common login feature calls global scheduler, which initiates the session by setting up the user environment. When the session is initialized, the first process may be started. Information from the user data block is used to manage the session, to send data to the accounting system, for example. Figure 2-6 illustrates the interactive login procedure.

A local batch session may be started during an interactive or batch session using the ENQUEUE command.

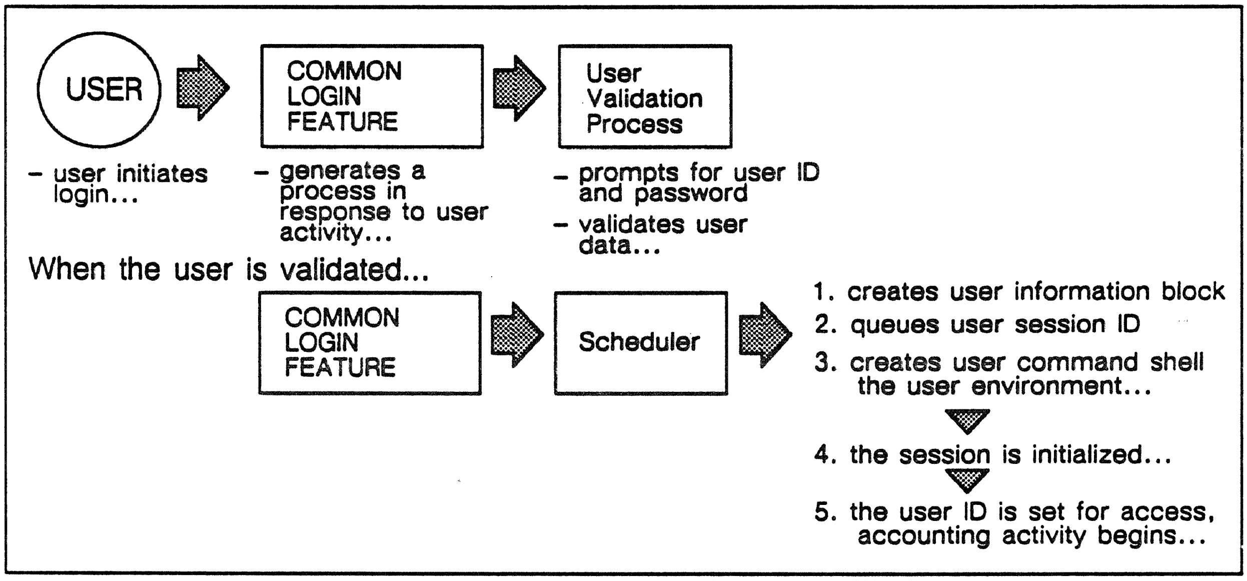

USER
COMMON LOGIN FEATURE
User Validation Process
- user initiates login...
- generates a process in response to user activity...
- prompts for user ID and password
- validates user data...
When the user is validated...
COMMON LOGIN FEATURE
Scheduler
1. creates user information block
2. queues user session ID
3. creates user command shell the user environment...
4. the session is initialized...
5. the user ID is set for access, accounting activity begins...

Figure 2–6. Interactive login process involving the common login feature and global scheduler.

Overview of the VSOS Environment

The VSOS Environment consists of a number of components:

- Files
- Libraries
- Language processors
- Utilities
- The System Interface Library
- The command shell

Files

In the VSOS Environment, a file is a data structure accessible to users and the system by name. The VSOS file system has a flat structure, meaning that you have only one directory containing all your permanent disk files. In the underlying system kernel, files are arranged in a hierarchical tree structure, where files are linked to the leaf nodes of a naming tree. The tree is defined and maintained using directories. Files are grouped by being linked to directories. Information about files, such as their attributes, is stored in the file catalog.

Figure 2–7 describes the ETA10 file directory system. Access is via a pathname, consisting of an ordered list of directory names, down to the file name. The kernel's file system structure (called the logical file system) is generalized enough to support the environment's file and I/O models. The VSOS environment automatically performs the necessary mapping of your files into the logical file system's tree structure.

Three data structures are supported by the ETA system:

- Records
- Groups
- Files

Within this logical file structure, a byte is the smallest unit of data managed by the system. A collection of bytes forms a record. A group, which is a collection of records, is the next higher structure, and a file is the highest level. These logical groups of data are also referred to as partitions. Data can be accessed by referencing partitions, using the partition number.

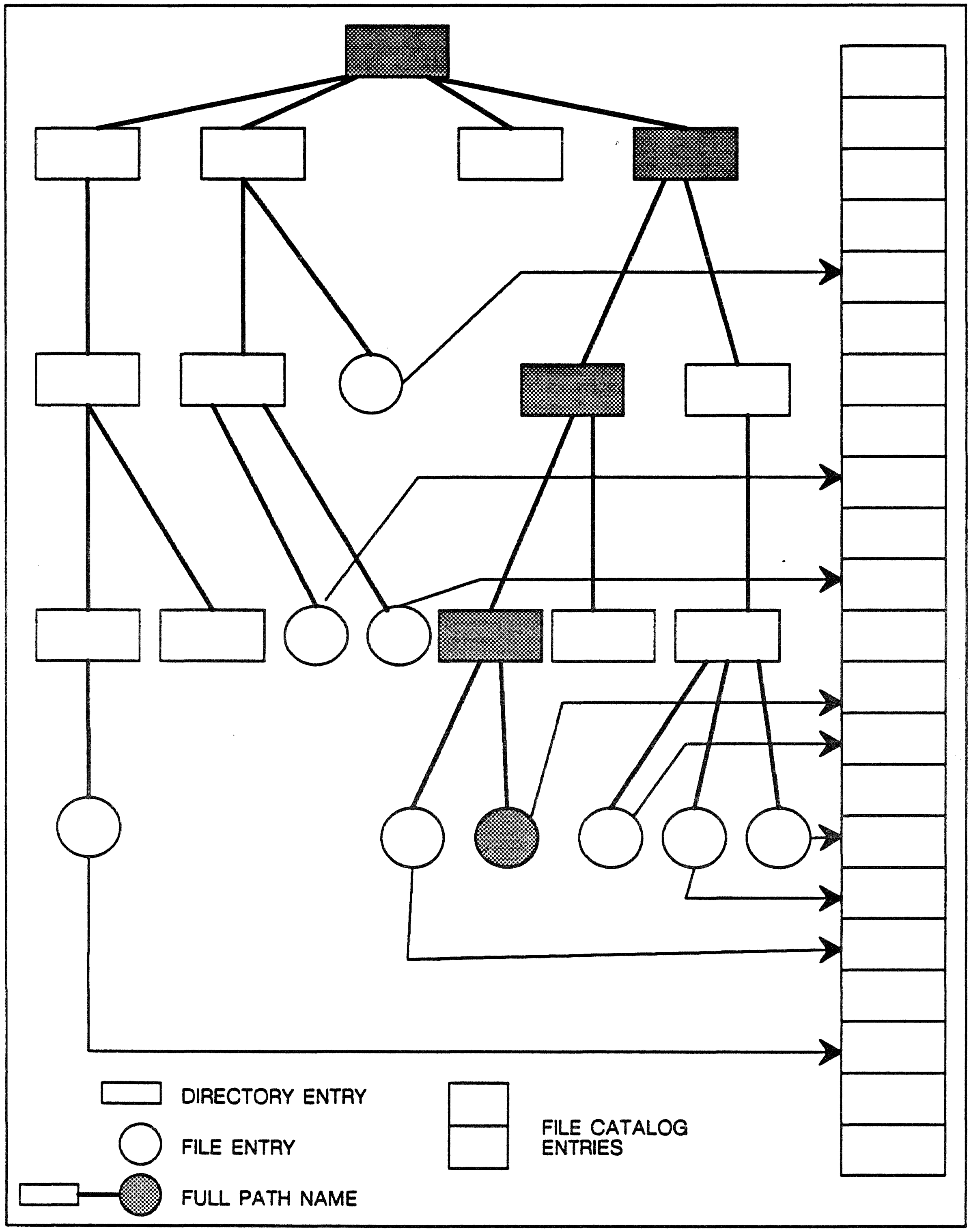


Figure 2–7. File directory/catalog manager file system.

File formats define the logical record structure of a file to the ETA kernel. The logical file system defines the logical structure of files according to four user-selectable record formats:

- ANSI fixed length (F) - records with a fixed, specifiable number of bytes that is set as a file attribute.
- Record mark delimited (R) - records of variable length delimited by a byte at the end of each.
- Undefined (U) - no records; the file is considered one continuous string of bytes.
- Control word delimited (V) - variable length records delimited by control words which mark record, group and file partitions. VSOS type (W) records are mapped to the file system's type (V) records.

File Attributes describe the characteristics of a file, such as its name, access permissions, and its device class. Attributes are set by the VSOS Environment when the file is created. You can assign the file attributes, or the system can assign default values. The attributes remain in effect for the life of the file, or until the user changes them explicitly.

File Types determine the way the system uses the files. A controllee file, also referred to as a virtual or virtual code file, is an executable file generated by LOAD. A process is the execution of a controllee file. Data files are any non-controllee files. Output files, which contain data to be processed by an output device, and object, or binary files, which provide the input to LOAD to generate executable controllee files are data files.

File Duration, the length of a file's existence on the system, depends on the file type. Temporary files exist only as long as the session continues. Permanent files are stored until explicitly deleted. Local files, which can be temporary or permanent, are immediately available to a task.

Figure 2-8 shows a portion of the logical file system tree pertinent to a single session. Because the kernel operating system supports only permanent files, the command shell creates the directory needed to simulate the behavior of local files (local_sess). *Sess* is the unique session ID. The local_sess directory is unique to a session. It contains files or links to files which behave as local files to the session, and becomes the working directory for all processes in the environment. Files in this directory can be attached, defined and returned. The shell also creates a directory (vsos_shell_dir_sess) to store its own files such as the dayfile, which are unique to the session and inaccessible to the user. Both directories are destroyed at the end of the session.

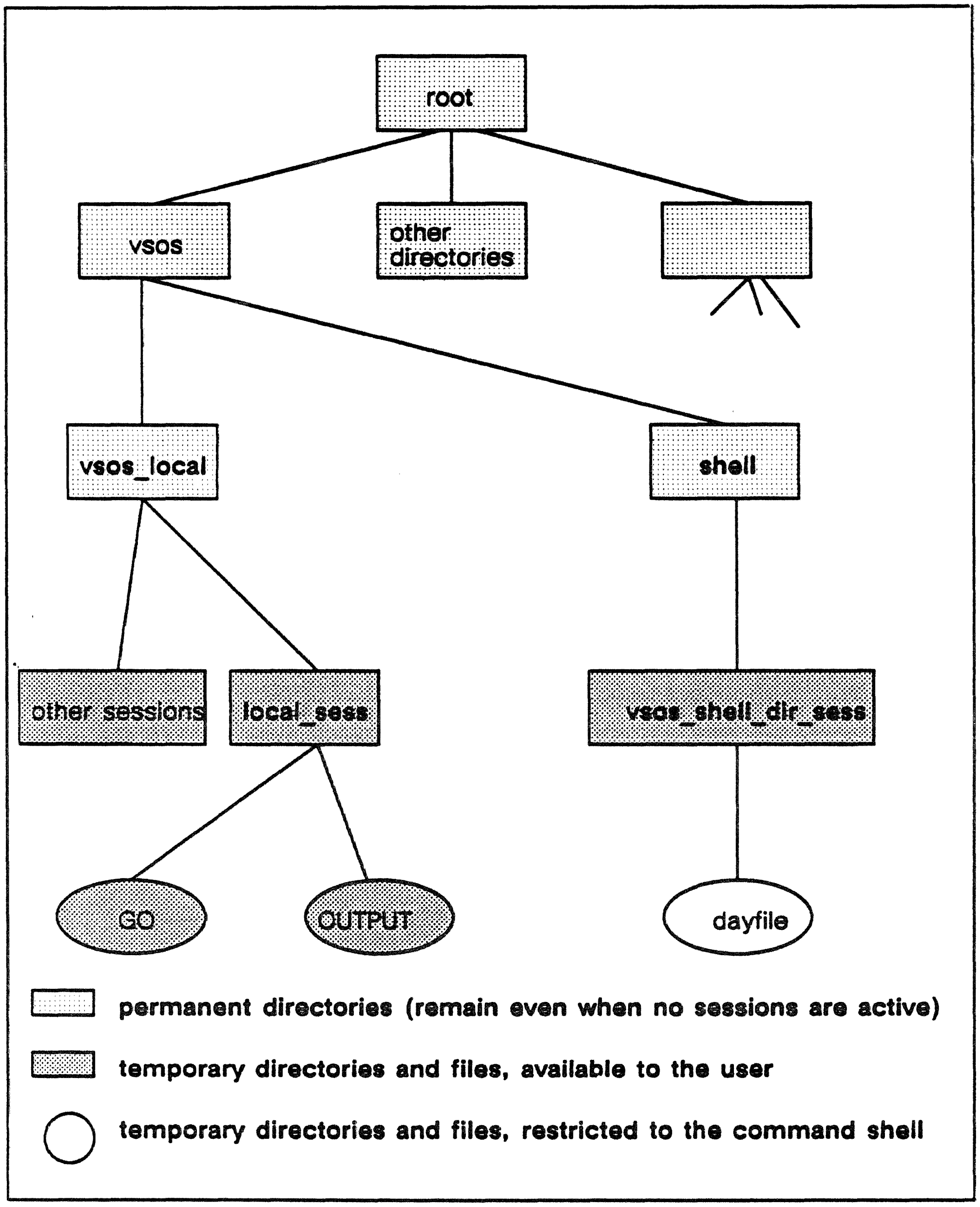

root
vsos
other directories
vsos_local
shell
other sessions
local_sess
vsos_shell_dir_sess
GO
OUTPUT
dayfile
permanent directories (remain even when no sessions are active)
temporary directories and files, available to the user
temporary directories and files, restricted to the command shell

Figure 2–8. The VSOS environment directories.

File Ownership may be by the VSOS environment (public), or by a user name (private). Utilities, commands, compilers, and general purpose routines are public files.

For more detailed information about VSOS environment file concepts, refer to the *VSOS Environment Reference Manual: Concepts and Commands, PUB–1051.*

Libraries

System libraries are available to you as an environment user. Some libraries contain routines that are specific to an environment, such as the VSOS SIL routines.

Other libraries are available to users in any environment. A library of multitasking routines available in future releases provides you with tools to perform multitasking operations such as task initialization, synchronization, and data sharing.

Standard libraries for language compilers are also available. In addition, you can create your own libraries within your environment. However, unlike system libraries, which are stored in system file space, user-created libraries are stored in the user's file space.

Language Processors

FORTRAN 200 is compatible with the FORTRAN CYBER 205 compiler, making it convenient to transport existing FORTRAN programs from the CYBER to the ETA10. ETA VAST is a vectorizing pre-compiler for the ETA10, intended for use in conjunction with the FORTRAN compiler.

Utilities

Several system utilities are available to the environment user. OLE, the object library editor, performs maintenance of object files. The LOAD utility links object files and libraries, and generates an executable file, also known as a controllee file. You can debug your programs interactively using ETA Debug, the symbolic debugger, which allows you to troubleshoot programs written in ETA product languages.

System Interface Library (SIL)

SIL consists of a number of subroutines that provide access to basic operating system features:

- File management
- Record I/O
- Message transmission
- Task initiation
- Task monitoring
- Task termination

Transparent routines translate a variable format SIL call into a fixed format acceptable to the CYBER Implementation Language (CYBIL), the language used for the development of the operating system software. Each library routine then:

- Parses the actual parameters
- Verifies parameter legality
- Makes the kernel calls necessary to accomplish the requested action.

Results from the kernel are translated into terms that a VSOS user is accustomed to and passed back to the calling application.

The SIL subroutine calls are detailed in the *VSOS Environment Reference Manual: Concepts and Commands (PUB-1084).*

Command Shell

The VSOS environment command shell serves three purposes:

1. It communicates with you, accepting your VSOS environment commands as input and returning responses as output. This communication may be interactive, or the command shell may be reading and writing disk files without your direct involvement.
2. It manages the processes of a batch or interactive session. It controls the initiation and sequencing of processes within a session.
3. It establishes those characteristics which cause a session to resemble a VSOS job, including the:

   Establishment of a VSOS-like file system

   Management of a session dayfile

   Disposition of specially named files

The Open Interconnection Network (OIN)

The Open Interconnection Network (OIN) is a local area network based on the Ethernet protocol. Users on the OIN can interface to the VSOS environment through the communications control system (CCS) to log in, enter commands, and transfer files.

VSOS/Kernel Interfaces

VSOS Environment users interact with operating system functions using *commands* and *System Interface Library (SIL) routines* to perform such activities as:

- Session and process management
- Memory management
- Handling of file system and I/O
- Program development and code file management
- Inter-process communication
- Retrieval of information about the system

Figure 2–9 illustrates the VSOS user interface to the kernel functions.

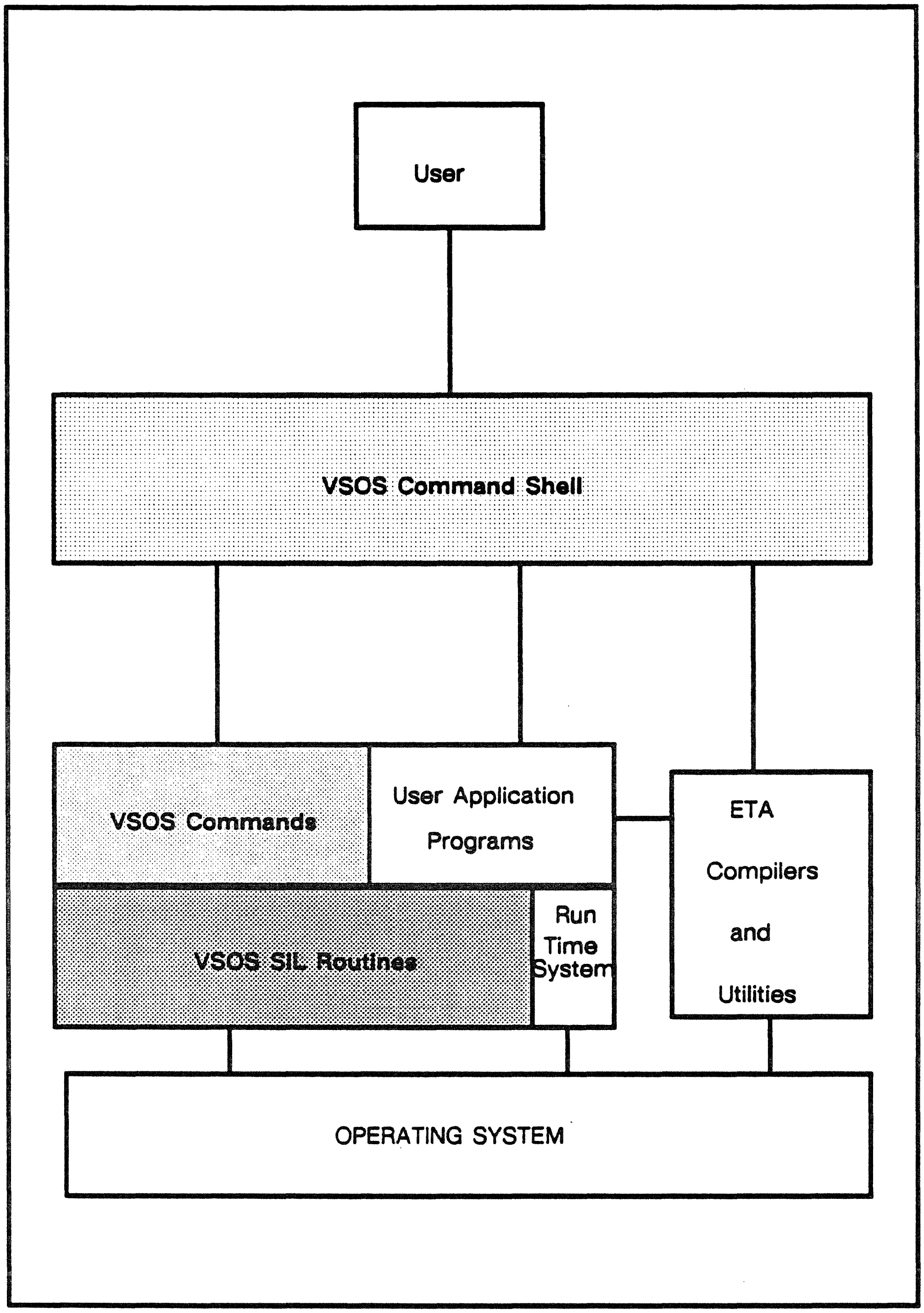


Figure 2–9. The user interfaces to the operating system.

The user issues commands through the command shell, either batch or interactively. Refer to the *VSOS Environment Reference Manual: Concepts and Commands, PUB-1051*, for detailed information about VSOS commands. Table 2-1 lists the VSOS environment commands.

Table 2-1. VSOS Command Summary.

Command type	Command Name	Description
Job Management	*	Send a comment to the dayfile
	BEGIN	Begin procedure
	COMMENT	Send a comment to the dayfile
	CONTINUE	Resume execution
	DAYFILE	Copy the dayfile
	ENQUEUE	Submit a job to a user environment
	EXIT	Set abnormal termination path
	PROC	Define procedure
	PROCEND	End procedure
	SET	Change job characteristics
	SETTERM	Set terminal attributes
	SUBMIT	Submit a file to a queue
	TV	Set threshold value
System access	BYE	End user environment interactive session
	PASSWORD	Change user password
	USER	Provide user validation information
File Management	ATTACH	Attach permanent files
	COMPARE	Compare file contents
	COPY	Copy a file
	COPYL	Copy logical partitions
	DEFINE	Define a permanent file
	FILES	List file information
	GIVE	Transfer file ownership
	LISTAC	List access permission set
	MFGIVE	Transfer a file to a remote host
	MFLINK	Permanent file transfer using LCN/RHF
	MFQUEUE	Explicit file routing
	MFTAKE	Accept a file transfer from a remote host
	PERMIT	Change access permission set
	PURGE	Destroy permanent files
	REQUEST	Create a temporary file
	RETURN	Evict local files or detach permanent files
	REWIND	Rewind a file
	SKIP	Reposition a file
	SWITCH	Change file attributes
Code File Management	LOAD	Create a controllee file
	OLE	Object library editor
Compiler	FTN200	FORTRAN 200 compiler
Debugger	EDB	ETA System debugger

Application programs executing in the VSOS Environment issue SIL calls to subroutines that communicate with kernel functions. Refer to the *VSOS Environment Reference Manual: System Interface Library Calls, PUB-1084*, for detailed information about SIL calls.

Table 2-2 lists the SIL non-I/O calls, which allow a process to exchange information with the operating system.

Table 2-2. SIL non-I/O routines.

Call type	**SIL call**	**Call description**
Process Requirements	Q5MEMORY Q5RECALL	Allocate a static stack Suspend process execution
Controllee Chains	Q5GETCTS Q5GETMCE Q5GETMCR Q5INIT Q5SNDMCE Q5SNDMCR Q5SNDMDF Q5SNDMJC Q5SNDMJS Q5TERM Q5TERMCE	Get controllee's termination status Get message from a controllee Get message from a controller Initialize, or initialize and start a controllee chain Send message to a controllee Send message to a controller Send message to a dayfile Send message to a job controller Send message to the system Terminate a process and its controllee chain Disconnect a controllee
System and Process Information	Q5CPUTIM Q5DCDMSC Q5DCDPFI Q5GETACT Q5GETTL Q5GETTN Q5GETUID Q5TIME	Get the CPU time the process has used Get and decode miscellaneous system information Decode permanent file information Get the system resources the process has used Get the process' time limit Get the process' characteristics Get the user name under which the process runs Get the system date and time
Permanent File Indices Information	Q5LFIHIR Q5LFIPRI Q5LFIPUB	List attached permanent or local file index entries List private file index entries List public file index entries

Table 2-3 lists the SIL I/O calls, which allow a process to perform file I/O functions.

Table 2-3. SIL I/O routines.

Call type	SIL call	Call description
Permanent File Access	Q5ATTACH	Attach a permanent file
	Q5CHANGE	Change file attributes
	Q5DEFINE	Define a permanent file
	Q5GIVE	Give file ownership to another user
	Q5PURGE	Purge a permanent file
	Q5RETURN	Return a permanent file
Local File Access	Q5CHANGE	Change file attributes
	Q5RETURN	Return a local file
	Q5RQUEST	Create or access a local file
Public files	Q5GIVE	Give file ownership
FIT Manipulation	Q5GENFIT	Generate a File Information Table
	Q5GETFIT	Retrieve contents of FIT
	Q5RETFIT	Return a FIT
	Q5SETFIT	Set File Information Table fields
I/O Preparation	Q5CLOSE	Close a file for I/O
	Q5FLUSH	Flush file buffers
	Q5GETFIL	Open, or create and open a file
	Q5MAPIN	Map virtual space
	Q5MAPOUT	Map out virtual space
	Q5OPEN	Open a file for I/O
Explicit I/O	Q5CHECK	Check I/O request status
	Q5ENDPAR	Write partition delimiter
	Q5GETN	Read complete partition
	Q5GETP	Read partial partition
	Q5PUTN	Write complete partition
	Q5PUTP	Write partial partition
	Q5READ	Read block of data
	Q5WRITE	Write block of data
File Positioning	Q5REWIND	Rewind file
	Q5SKIP	Skip file partition
Miscellaneous	Q5PERMIT	Change access permission set
	Q5REDUCE	Reduce file space

How ETA VSOS differs from CDC VSOS

Differences between the ETA10 computer system and the CYBER 205 have resulted in some changes in ETA VSOS. CYBER 205 features with no analogue in ETA systems have been deleted. Features performing similar functions, but which execute differently on the ETA10 are renamed. New features have been added to the ETA system. The major differences are described below.

Pool files are no longer supported in the VSOS environment. The CDC CYBER 205 had numeric user identifiers and pool files were identified by an alphanumeric name. On the ETA system, user IDs are all alphanumeric. In order to achieve the same effect as pool files, users can ATTACH all files belonging to another username if they have access rights. The site may specify a file directory as the system pool replacement. This directory will be included in the command search path.

Tape files are not currently supported in the VSOS environment.

File security levels are not supported in the VSOS environment.

The current file position is preserved across tasks executing in the VSOS environment, with the result that when a file is opened, it may not always be at the beginning-of-file. However, the user can issue a command requesting the file system to rewind the file between tasks.

The interactive interface to the VSOS environment has a session dayfile. The command shell creates the dayfile as a temporary shell file to hold messages from executing processes, copies of commands entered by the user, and status information. Output is directed to the user's terminal. At the end of the session, the command shell destroys the dayfile, unless the user has saved it as a permanent file.

Device classes in the VSOS environment replace the CYBER 205 pack name designations. Device classes are logical partitions of the ETA system's physical disks.

The VSOS environment does not support connected files.

The EDB ETA system debugger is available to the VSOS environment user.

Many VSOS version 2.2 system tables no longer exist in the ETA10 system. Some of these are the Miscellaneous Table, the Minus Page, the Pack File Index, and Device Statistics Table. Some have analogues in the ETA10 system, others do not. Therefore, SIL calls requesting table information are supported to a limited extent; if the information requested by a SIL call is available from the operating system, it is supplied to the caller. No calls which returned a complete copy of a CYBER 205 system table are supported on the ETA10.

Other differences between the ETA VSOS environment and VSOS Version 2.2 can be divided into File Concept, Device Characteristics, Process Management, and User Interface categories.

File Concept Differences

In the VSOS environment, users can specify a letter or number for the first character of a file name. VSOS version 2.2 required that file names begin with a letter.

A VSOS environment file name may begin with the characters 'Q5' through 'Q9'. These characters were reserved for operating system files in version 2.2.

The ETA10 paging file replaces CYBER 205 drop files.

Output is written sequentially to the file OUTPUT in the VSOS environment, unless directed elsewhere.

Output files are treated as a special case of data files.

The CYBER 205 restrictions on print files and their families do not apply on the ETA10 system.

The ETA VSOS environment does not create and use scratch files.

The limit of 256 permanent file per user ID in version 2.2 no longer exists.

The termination character of the record mark delimited (R) record format cannot be specified as an installation parameter.

Device Characteristics

A device set defines a physical set of disk drives, and is only applicable to disk drives.

A device class, which is a logical class of device, has been added to the VSOS environment. It has a 24-character field length, instead of the two hexadecimal digits used for the VSOS version 2.2 device set.

Tape operations are not currently supported on the ETA10.

Process Management

A controllee chain in the VSOS environment may have more than nine levels, which was a version 2.2 restriction.

In the VSOS environment, a controllee chain is serially created by one task initiating the next in the chain. Initializing a multiple task chain at one time is not supported. Dynamic linking of controllee chains is not supported. The ETA system does not permit processes to be restarted.

The minus page and bound import and export maps are replaced by various process and register packages.

The LOGIN command is not needed in an interactive login. Instead, the system prompts for a user ID at login time.

The ETA VSOS environment user ID may be up to 31 characters in length, and the password may also have up to 31 characters. In VSOS version 2.2, the limits were 6 digits for the user ID and 8 characters for the password.

Batch processes are handled by the same command shell handling interactive processes, instead of by a batch processor. There are fewer distinctions between batch and interactive processing.

Interactive access to the ETA10 system is through the Open Interconnection Network.

There is no support for file archiving on remote hosts.

User Interface Differences

Although most commands and SIL calls used in VSOS version 2.2 are supported in the ETA VSOS environment, some are not, and there are some new ones. Some parameters on commands and SIL calls which were implemented on the CYBER 205 are no longer available because of the different ETA10 implementation. For example, a pack name cannot be specified on a DEFINE command.

Refer to PUB–1084, *VSOS Environment Reference Manual: Concepts and Commands,* and PUB–1051, *VSOS Environment Reference Manual: System Interface Library Calls* for detailed information about commands and SIL calls on the ETA10. A list of commands and SIL calls that are no longer supported is supplied below.

The following CYBER 205 commands are not supported:

AUDIT	List permanent file information
BLANK	Blank label a tape volume
CHARGE	Assign account and project numbers
DEBUG	Debug a program
DMAP	Provide information on location of permanent file segments

DUMP	Dump a drop file
DUMPF	Copy permanent files to archive storage
EDITPUB	Add or destroy a public file
LABEL	Supply label information for a tape file
LOADPF	Reload files from permanent storage
LOOK	Dump virtual space
NORERUN	Set norerun status
PACCESS	Grant pool access
PATTACH	Attach a pool
PCREATE	Create a pool
PDELETE	Remove a pool
PDESTROY	Destroy a pool
PDETACH	Detach an attached pool
PFILES	List pool information
Q	List job status
RERUN	Set rerun status
RESOURCE	Set job limits
SLGEN	Generate a shared library
SUMMARY	Provide resource usage information
TASKATT	Alter a controllee attribute

The following SIL routines are not supported:

Q5ADVISE	Inform system of task's virtual space requirements
Q5CHECKB	Check if buffer I/O is complete
Q5CLIOER	Clear tape I/O error
Q5DCDDST	Get information for the disk status table
Q5DCDPLB	Decode the pack label
Q5DESBIF	Destroy the batch input file
Q5DISAMI	Disable message interrupt processing
Q5DISATI	Disable abnormal termination control
Q5DMPACT	Dump the cumulative accounting file
Q5ENAMI	Enable message interrupt processing
Q5ENATI	Enable abnormal termination control
Q5GETB	Read buffer record
Q5GETIIP	Copy an interrupted task's invisible package
Q5GETIRF	Copy a task's register file
Q5GETMOP	Get message sent by the system operator
Q5GETMPG	Copy an interrupted task's minus page information
Q5GETPFI	Copy the label and permanent file indices from a pack
Q5INITCH	Initialize a controllee chain
Q5LABEL	Create or access a local file in multiple set
Q5LFIPOL	Copy the pool file index entries
Q5LSTBUT	Copy the Bank Update table
Q5LSTSTB	Copy the statistics buffer
Q5LSTTCB	Copy the time card buffer
Q5PATACH	Attach a pool
Q5PCREAT	Create a pool
Q5PDESTR	Remove a pool
Q5PDTACH	Return a pool

Q5PGRACC	Grant access to a pool
Q5POOLS	List pools
Q5PREACC	Remove an account
Q5PUSERL	List users with access to a pool
Q5PUTB	Write buffer record
Q5REELSW	Continue processing with next tape volume
Q5REPREV	Enable or disable user reprieve processing
Q5RFI	Return control from an interrupt routine
Q5ROUTE	Route a file
Q5RUNBIF	Rerun a task if the system fails
Q5SNDMOP	Send a message to the system operator
Q5SNDSTR	Start a controllee execution
Q5VRACC	Change a task's accounting rate

Functional Description of VSOS

The VSOS File System

The file directory/catalog manager (FDCM) feature of the underlying operating system defines a tree-structured file system. In this file system, files are linked to leaf nodes of the naming tree. Calls are provided to manipulate the tree structure and file descriptions. Trees are defined and maintained by use of file directories. Files are grouped by being linked to directories. Directories and files are identified and located by means of pathnames, which are ordered lists of directory names, with each succeeding directory linked to its predecessor.

VSOS presents a flat file system in which each user has only one directory containing all of his or her permanent disk files. A user's own files are inaccessible until explicitly activated through a VSOS ATTACH command, which names the files to be used in the session and makes them 'local'.

VSOS resolves references to files by searching:

1. All local files, including attached permanent files and temporary files to which you have access.
2. All files in the system pool selected by the site.

   (Pools and pool files are not supported in the VSOS environment. Many sites, however, depend on VSOS environment pools for integration of modified controllees. Those sites may select a directory name to use as a pool through a system configuration option).

3. Permanent files in the public file directory (the public file directory is selected by the site).

In the VSOS environment, these searches are accomplished by mapping the VSOS file system onto the kernel's logical file system.

When a new user account is created by system administration, a home directory with the name /USR_root/USR/*user_name* is created as shown in Figure 2-10. The directory will contain what VSOS will consider to be private permanent files.

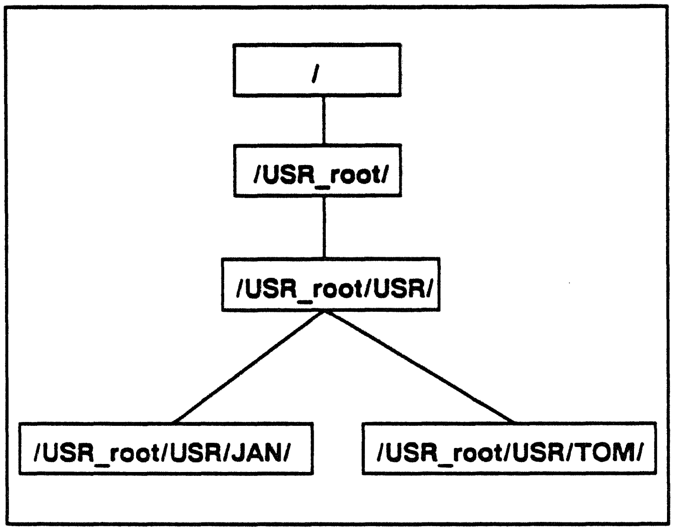


Figure 2–10. Directories for newly–created users.

The identifier /USR_root/USR/*user_name* is used for the home directory for two reasons:

- It allows a VSOS environment user to locate the home directories of other users knowing only the user name
- It provides a convenient way to share files.

Links to files in the /USR_root/USR/*user_name* directory are established in working directory /LOCAL_*sess*/ as files are DEFINEd and ATTACHed.

The links are destroyed as the files are RETURNed.The directory /LOCAL_*sess*/ also contains temporary files as they are REQUESTed.

The command shell creates and manages the directory /LOCAL_*sess*/, keeping it in existence only while a session is active. The *sess* field is a unique session identification number, used to separate the local files of one session from those of another.

Figure 2-11 shows the contents of the /USR/ and /LOCAL_*sess*/ directories before a file is created, after a file is created, after a file is DEFINEd, and after the job or session terminates.

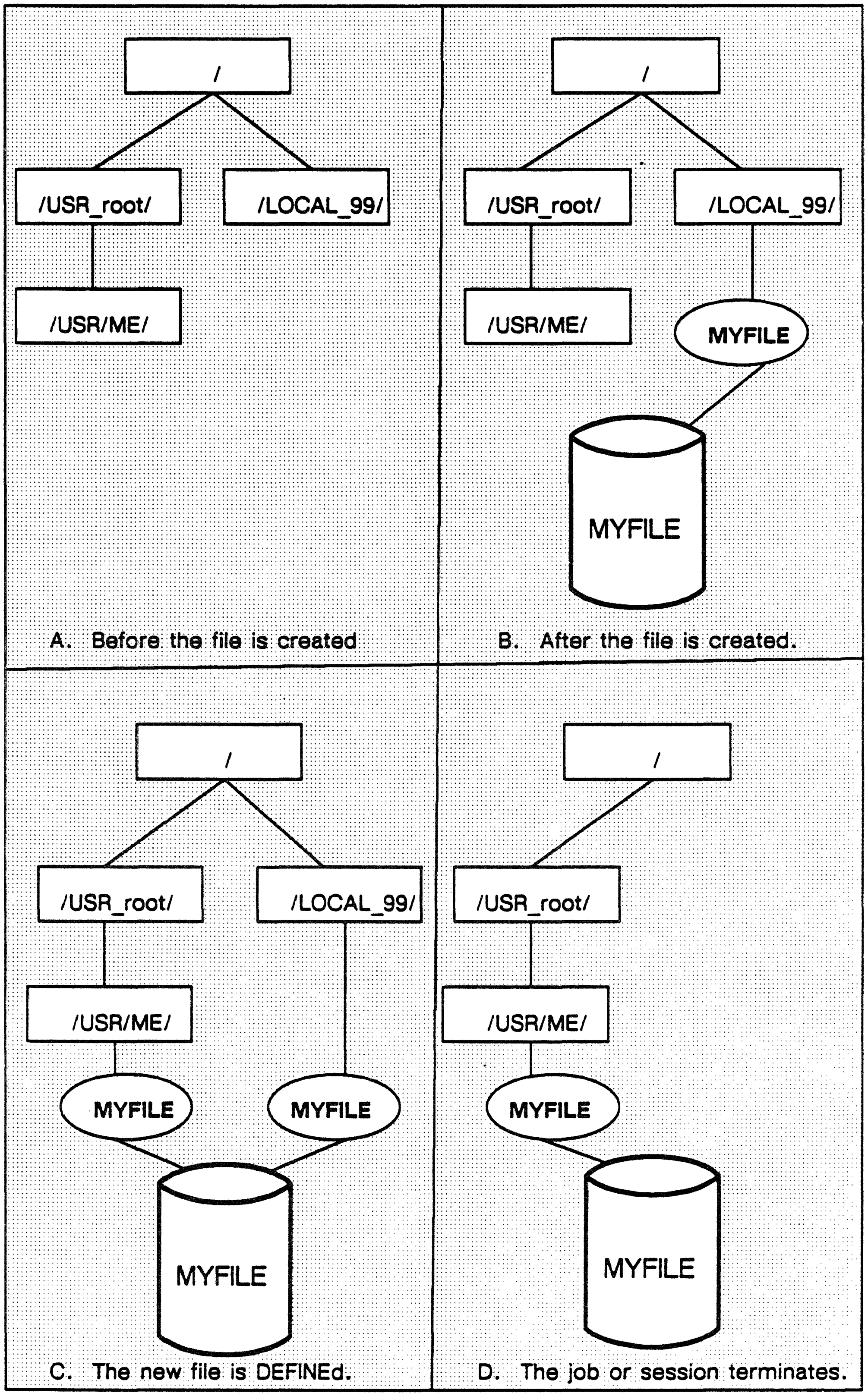


Figure 2–11. VSOS File management.

The VSOS Tables

Because most of the CYBER 205 VSOS Tables do not exist in the ETA10 operating system, the SIL routines which access certain tables are no longer required. When the system table does exist on the ETA10, the relevant SIL routine is supported in the VSOS environment. When the table does not exist, but the requested information can be retrieved from elsewhere in the system, SIL calls to the kernel obtain the data and return it to the calling program.

The File Information Table

The File Information Table (FIT) is a VSOS table used to describe the characteristics of a file. CYBER VSOS allows a FIT to be created and its contents specified before the file is actually created. This pre-specification of file attributes is not a feature of the operating system file management routines, and so must be simulated in the VSOS environment. To use a file in the VSOS Environment, the application program calls a SIL routine to create an entry in the FIT describing the file, specifying its attributes, and establishing an association between the logical filename and a number.

The actual implementation of the FIT is not visible to the caller, and no assumptions can be made about the structure of a FIT entry. The FIT is the only CYBER VSOS data structure simulated in the VSOS environment, and it exists only while the file is active or open for I/O. References to other tables are either translated into calls to the kernel or not supported.

The Command Shell

Communicating With the User

You specify actions in the VSOS environment by issuing commands which the command shell receives and interprets. There are a number of different ways for you to communicate with the ETA10, which requires that the command shell be able to read commands from several different sources. Several of these sources, and their paths to the VSOS environment, are specified in figure 2-12.

If the session is of *local batch* origin, the commands are read from a disk file using the record manager (RM). When it starts the session, global scheduler instructs the command shell where to find the input file. Output is returned to a disk file.

If the session is of *interactive* origin (i.e. you are logged into the ETA10 through the OIN), the commands are read from the terminal through the communication control subsystem (CCS). global scheduler tells the command shell how to identify the OIN when it starts the session. Output is returned to the terminal.

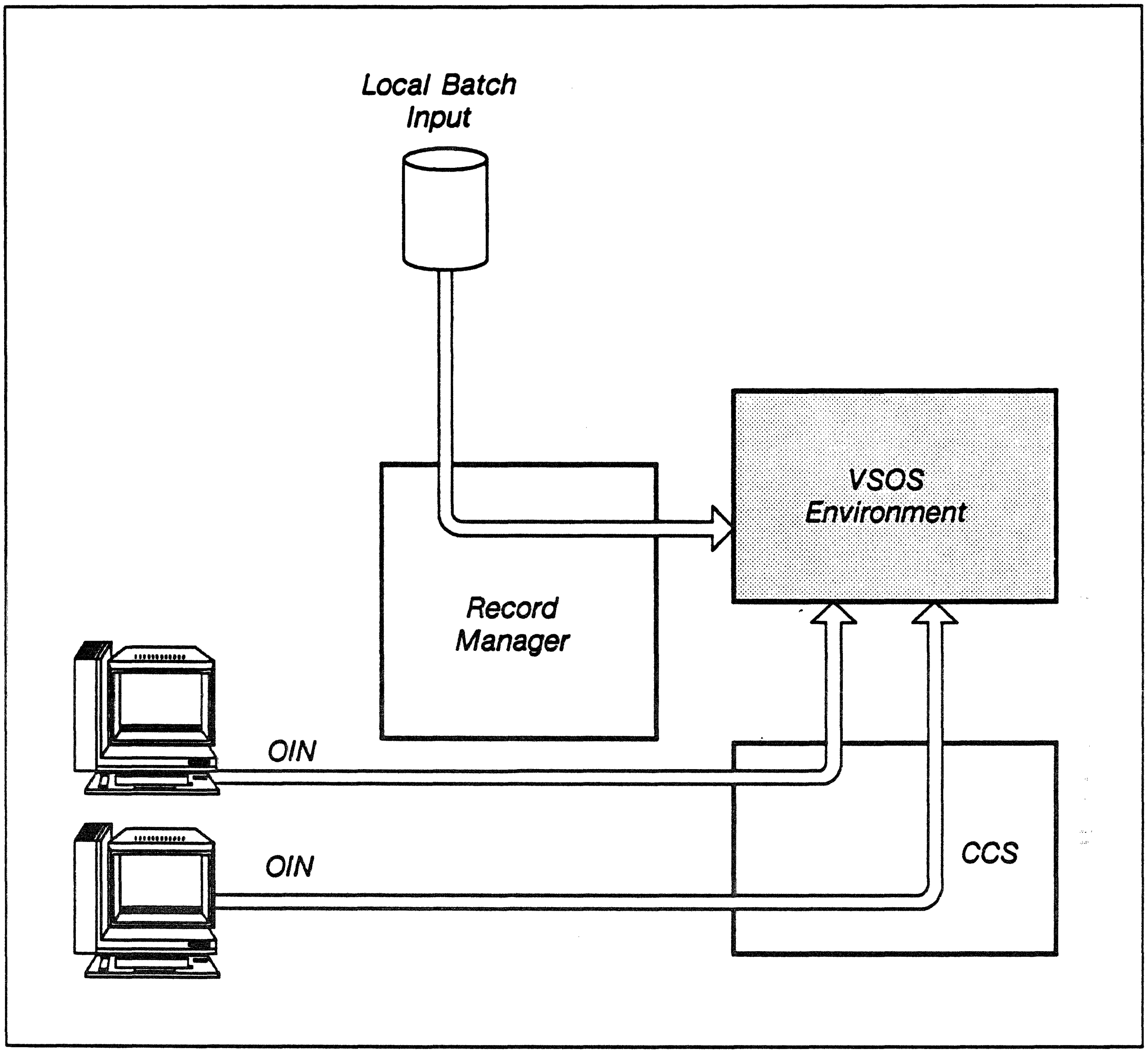


Figure 2–12. Pathways to the VSOS environment.

Managing Processes

VSOS environment commands consist of a keyword and a list of optional parameters. The command shell parses each command to discover its keyword and takes one of two courses:

1. Some command keywords are recognized by the command shell as specifying things the shell itself should do. The command shell completes parsing the command, takes the required action, and resumes reading the command stream.
2. Most commands are the names of executable object files. The command shell searches for the file in three directories: private files (both local and attached permanent), the site-selected system pool, and public files.

When the file is found, the command shell calls the Process Create function within process management to run the program.

The command shell is a serial processor and never initiates a new process until the preceding one has completed, waiting while its child process (the controllee) executes. Process management notifies the shell when the child has completed.

Establishing the VSOS Environment

The VSOS command shell is instrumental in giving a session VSOS-like characteristics. The command shell:

- Creates the directory needed to simulate the behavior of local files (the ETA10 kernel supports only permanent files)
- Establishes an interprocess communication path using the remote procedure calls feature to simulate the controller/controllee message passing of CYBER VSOS
- Creates files characteristic of a VSOS session:

  The dayfile

  The INPUT file

  The batch routing files

The command shell creates a working directory, unique to the session, that contains files or links to files which are to behave as local files. This directory becomes the directory for child processes. A file is attached by creating a link from this working directory to the home directory, returned by deleting this path, and defined by creating a link from the home directory to the working directory.

The two directories, and files attached from the home directory to the working directory, are illustrated in figure 2-13.

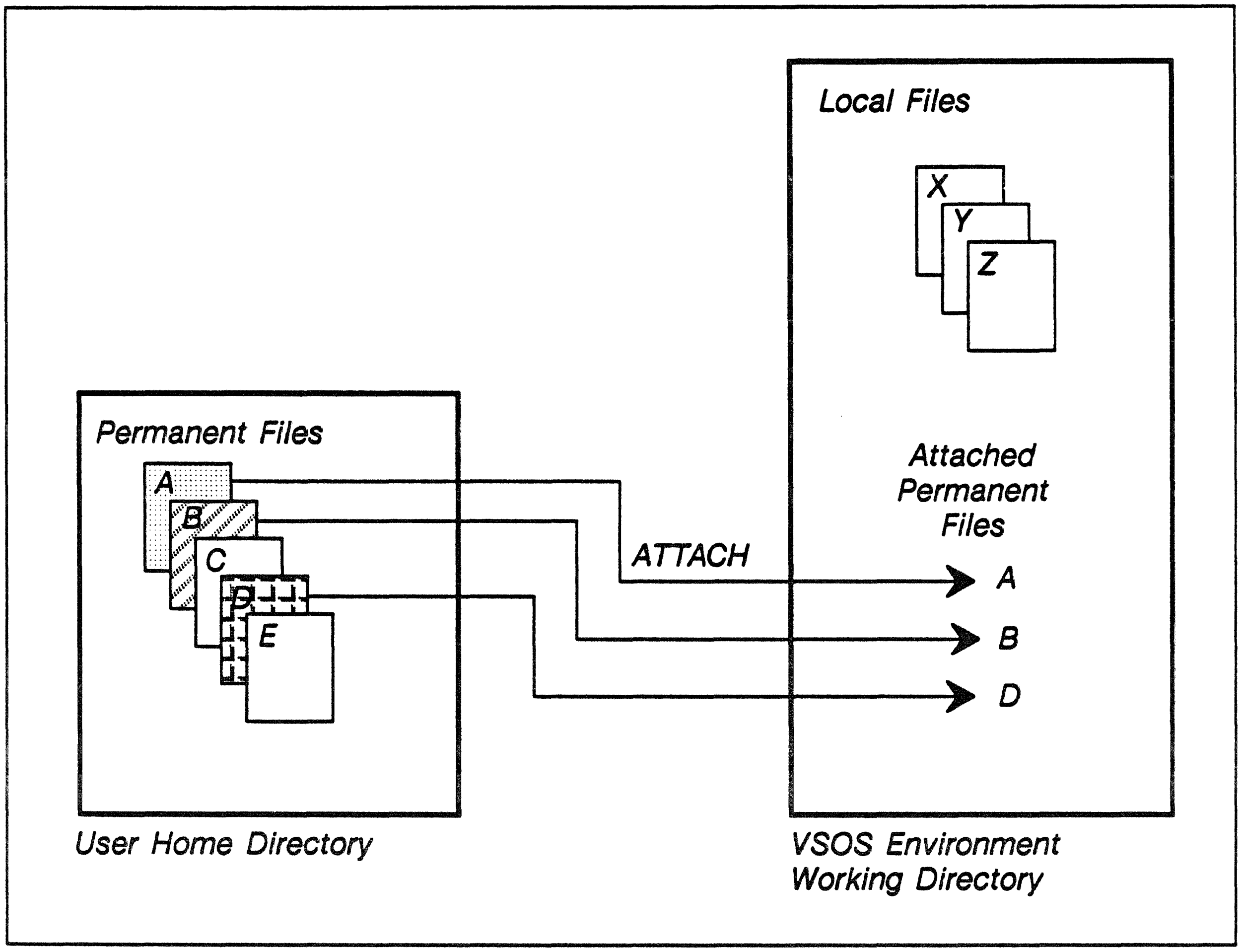


Figure 2–13. VSOS environment working directory with links to the user's home directory.

Interprocess communication (passing messages between the command shell and its child processes) is done through the IPC remote procedure call feature, which enables processes to pass messages to each other using mailboxes.

At initialization, as shown in Figure 2-14, the command shell creates a mailbox that it will use to receive messages from other processes in the session. The mailbox name is created using the shell's process ID. A message passing protocol allows various message dispositions to be identified along with message text.

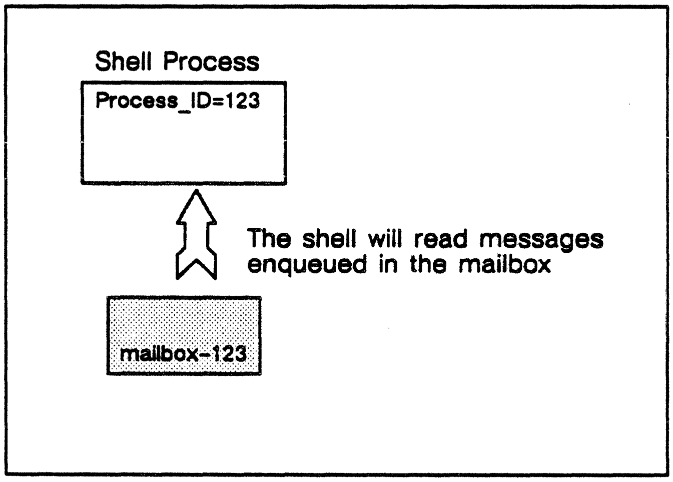


Figure 2-14. The shell creates (exports) a mailbox.

During command processing, when the shell determines that it must create a process to execute a command, it calls process management and passes along inheritance information which the process will need, including the mailbox IDs of the parent's mailbox and the shell's mailbox (figure 2-15).

In figure 2-15, the mailbox ID of the shell and the mailbox ID of the parent of Process 456 are the same, because in this case, the shell happens to be the parent of this process.

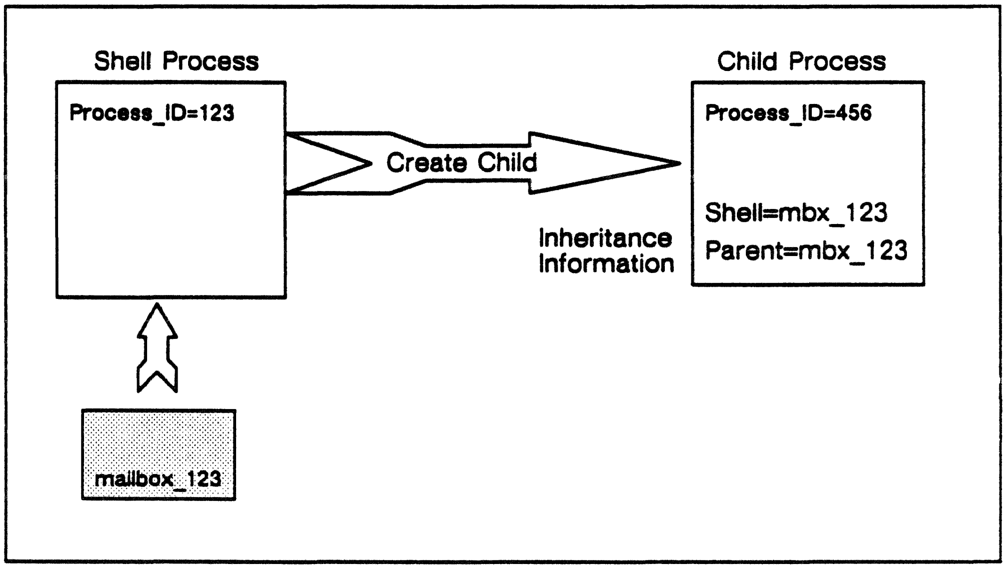


Figure 2-15. The shell creates a process, passing inheritance information.

As part of the initialization of the VSOS user environment in the newly created process, the mailbox (mailbox_456) from which the new process will receive messages is created, as shown in Figure 2-16.

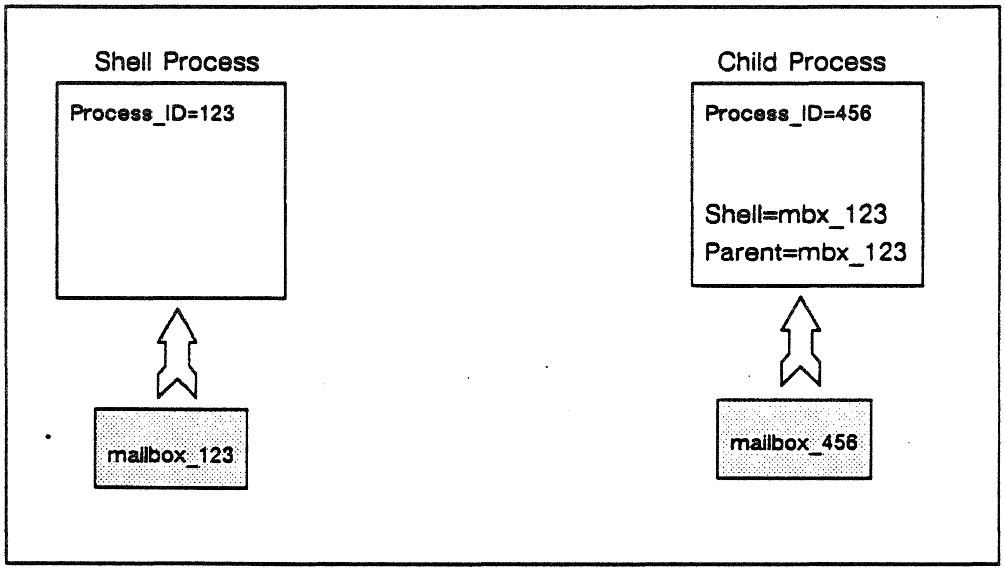


Figure 2-16. The newly created process exports (creates) its own mailbox.

The new process connects to (imports) the shell's mailbox so it can send messages to the shell, and sends its own mailbox ID to the shell, so the shell can send messages to it. (figure 2-17). It knows the shell's mailbox ID from the inheritance information which was passed to it.

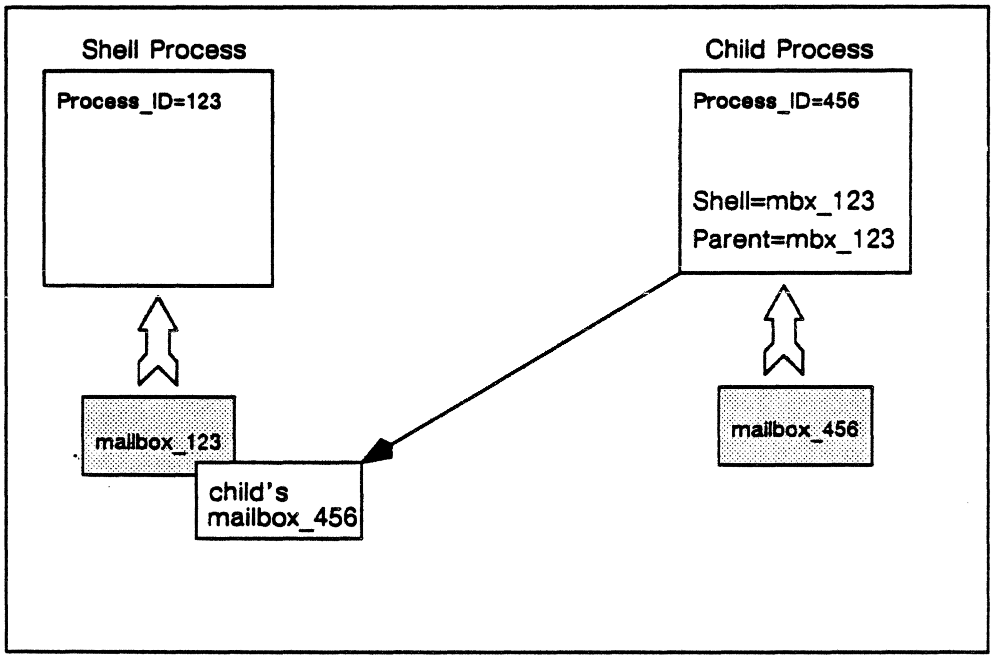


Figure 2-17. The shell is informed of the mailbox ID of the new process.

When the shell receives the message from the new process, it can connect to the child's mailbox and send messages to it. If a process needs to send a message to the shell, it simply writes the message to the shell's mailbox, with the appropriate disposition identified, as detailed in Figure 2-18. In this diagram, Child Process 2 sends a dayfile message to the shell's mailbox.

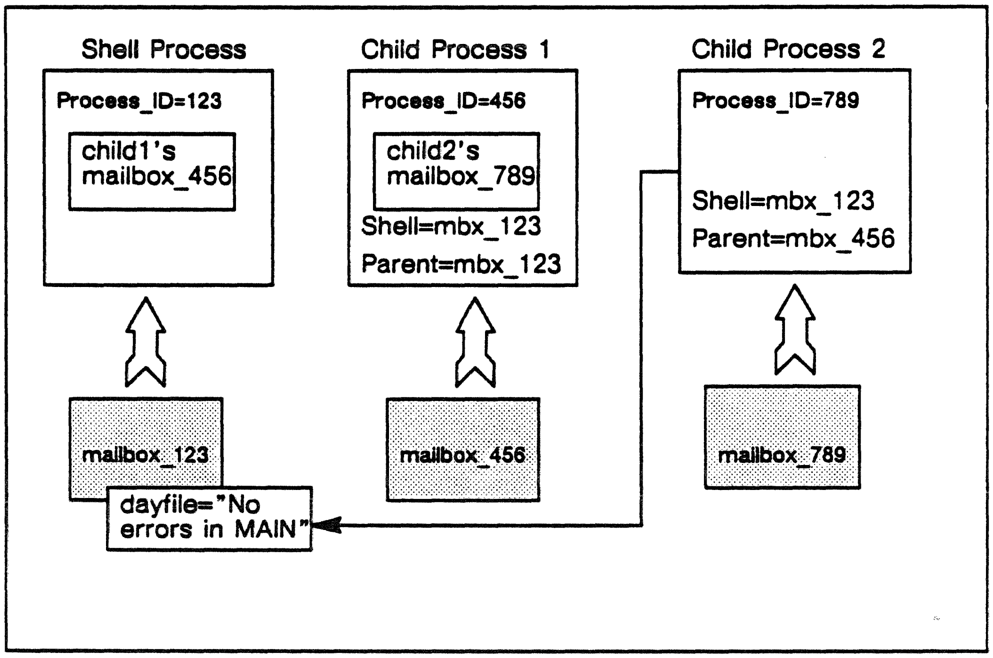


Figure 2-18. A process sends a dayfile message to the shell's mailbox.

The command shell creates the files that are characteristic of VSOS (INPUT and the dayfile), and ensures that these files and the user-created OUTPUT file get the required special processing.

For example, file INPUT, in a local batch session, is the file of commands of input data that make up the session.

The command shell is also responsible for creating, reading, writing, and disposing of the session's private dayfile. This is a transcript of the session, containing images of all your commands and a record of the results of each. You have no direct access to the dayfile, but may send messages to it, or read it using the DAYFILE command.

The OUTPUT file is treated differently for different session origins. In local batch, output may be sent to a file. In an interactive process, output is sent to the terminal.

Security and Protection

A process' address space is divided into two *domains*. The system domain, which supports the operating system, protects its routines and data structures from unauthorized access or modification. The application domain supports VSOS and its libraries and tools, as well as user processes, but it cannot access the address space of another domain unless it is explicitly shared. For a detailed discussion of domain protection, refer to chapter 4.

The primary protection for user *files* is through the establishment and verification of user file-access permissions. Users may not access a file without specific permission.

Files are grouped by being linked to a directory. Users are permitted to create directories which are accorded the same permissions given to the current directory, and they can link files to directories on the basis of the access permissions granted. Each directory and file pathname is owned by a single username. This username (the file owner) and the account associated with it may use the resources reserved for that file.

Users restrict or share access to their file pathnames and directories by using access permissions. Access is granted by username, account identifier, and project identifier. A user may give a file special access permissions that are valid when the file is executed. When a file pathname is created without an access permission list, a default access permission is automatically fixed to the pathname. Site administration defines the default access permission.

When a user begins to create a file in a directory, the file system first verifies that the user has permission to create a file there. During the creation process, the user (now a file owner) assigns it a set of access permissions complying with the conventions of the user environment and the site. As the file system receives requests to open, read or write to files, it verifies each type of access request against the requester's access permissions before granting access. Although users control access to their files, the file system itself always reserves a user management permission able to override the owner's access permission.

Files defined by the VSOS command shell are protected by the logical file system. Files private to the session are protected by being in a directory that exists for that session alone. Only processes belonging to the session process cluster can access these files.

The system does not currently limit file size. Allocation of space to files is dynamic, and the system automatically extends a file while it is being written as long as there is space for it to grow on its logical device.

The file system uses shared memory as a disk cache, protecting the files it moves into and out of it. As needed, the files are mapped for read/write or read-only into a central processor memory. Within a central processor memory, two user processes cannot access the same section of memory at the same time.

The VSOS Command Shell

Global Scheduler calls the command shell when a user logs into the VSOS Environment. The processes belonging to the user's session, including the command shell, are logically grouped into a process cluster by Global Scheduler, with the characteristic specified of INHERITANCE=TRUE, so that child processes of members automatically belong to the cluster.

The command shell is activated and passed the session origin type (interactive or batch), and the local file ID of the interactive connection, or the pathname of the batch input file for a batch session.

The processing of a VSOS environment session is illustrated in figure 2-19.

After the shell is and initialized, it accepts a command entered by the user. Depending on the command, the shell either calls process management to create a process to execute the command, or performs some internal task. When Command Processing completes and there are no more commands waiting, any resulting output is handled and the shell terminates.

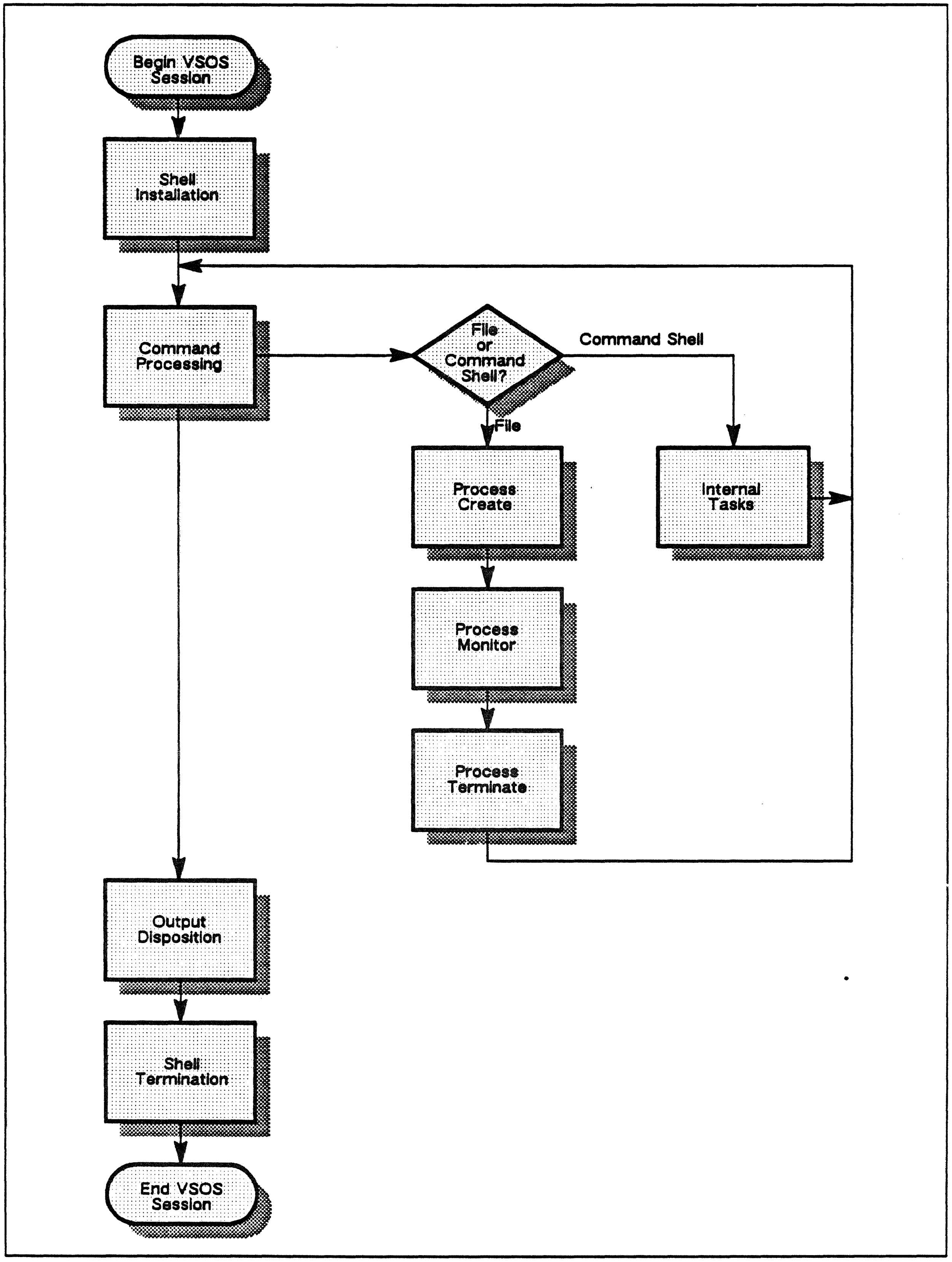


Figure 2–19. Processing of a VSOS environment session.

Shell Initialization

When called to execute a command, the command shell performs an initialization procedure:

1. Get the shell's process ID from process management.
2. Create and open the session dayfile.
3. Determine whether the session origin is batch or interactive.
4. Create a mailbox for use in communicating with child processes in the session.
5. Create an input path to read commands. For an interactive session, the input path is a CCS terminal connection. For a batch session, the input path is a path to the batch input file.
6. Find the names and global file IDs of the system pool and public file directories.
7. Identify the user who owns this session, and the user's home directory.
8. Create the local file directory for the session.
9. If the session is *batch*, create a local file INPUT which is an alias, or indirect identifier, for the batch input file.
10. Exit to Command Processing.

The shell initialization procedure is illustrated in figure 2-20.

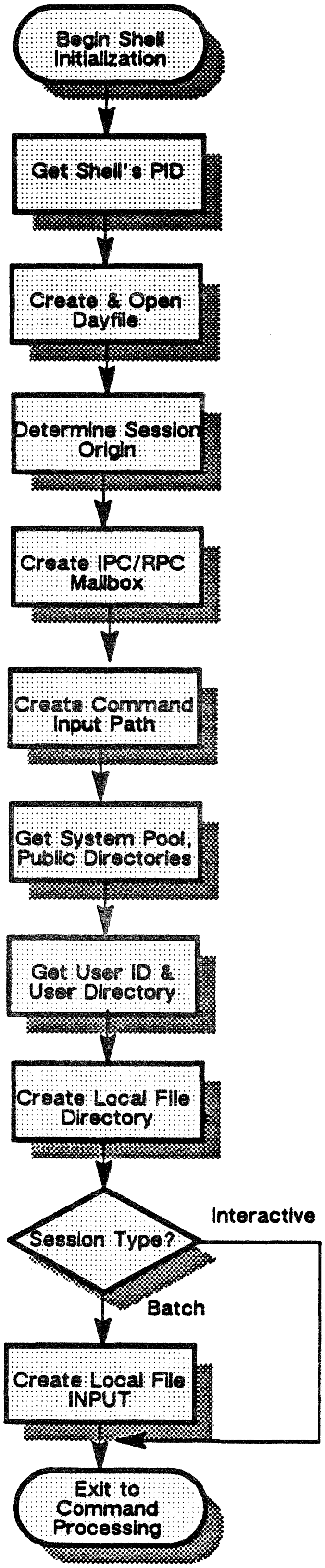

Begin Shell Initialization
Get Shell's PID
Create & Open Dayfile
Determine Session Origin
Create IPC/RPC Mailbox
Create Command Input Path
Get System Pool, Public Directories
Get User ID & User Directory
Create Local File Directory
Session Type?
Interactive
Batch
Create Local File INPUT
Exit to Command Processing

Figure 2–20. Shell Initialization flow chart.

Command Processing

Once the path to standard input is open, Command Processing can begin. In an interactive session, the command shell prompts for your command. The procedure is:

1. Read one line from the input path (batch input file path or interactive CCS terminal connection).
2. Copy the command line to the dayfile.
3. Parse the command to extract the keyword. If the command can not be successfully parsed, report the error to the system dayfile and the terminal, and exit to Command Shell Termination (batch) or go to step 7 (interactive).
4. Using the keyword as a filename, search the local, pool, and public directories. If the search is unsuccessful, look for the keyword in the command shell table of internal task names.
5. If the command keyword names a file, exit to Process Create.
6. If the command keyword names a command shell task, exit to Internal Tasks.
7. If the batch input file is at end-of-file, or if the interactive user has disconnected from VSOS, exit to Output Disposition. Else go to step 1.

The Command Processing procedure is illustrated in figure 2-21.

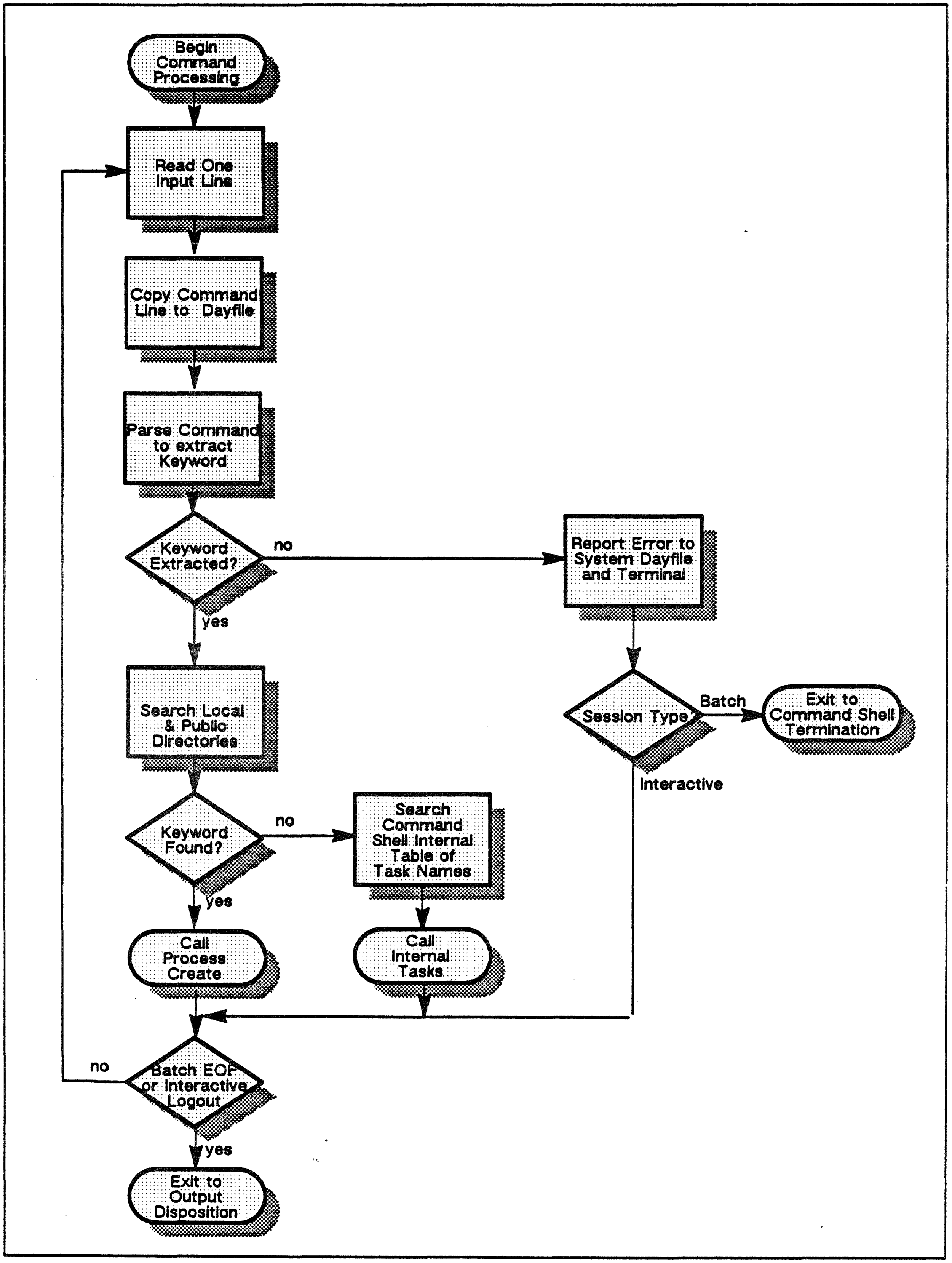

Begin Command Processing
Read One Input Line
Copy Command Line to Dayfile
Parse Command to extract Keyword
Keyword Extracted?
no
yes
Report Error to System Dayfile and Terminal
Session Type?
Batch
Exit to Command Shell Termination
Interactive
Search Local & Public Directories
Keyword Found?
no
yes
Search Command Shell Internal Table of Task Names
Call Process Create
Call Internal Tasks
Batch EOF or Interactive Logout
no
yes
Exit to Output Disposition

Figure 2–21. Command Processing flow chart.

Internal Tasks

When Command Processing has identified an operation for the command shell itself to handle, the Internal Tasks procedure is called. The commands that the VSOS command shell interprets as internal tasks are:

- * or COMMENT
- DAYFILE
- EXIT
- PASSWORD
- SET
- SUMMARY
- TV

The Internal Tasks procedure is illustrated in figure 2-22.When the requested operation completes, Internal Tasks exits to Command Processing.

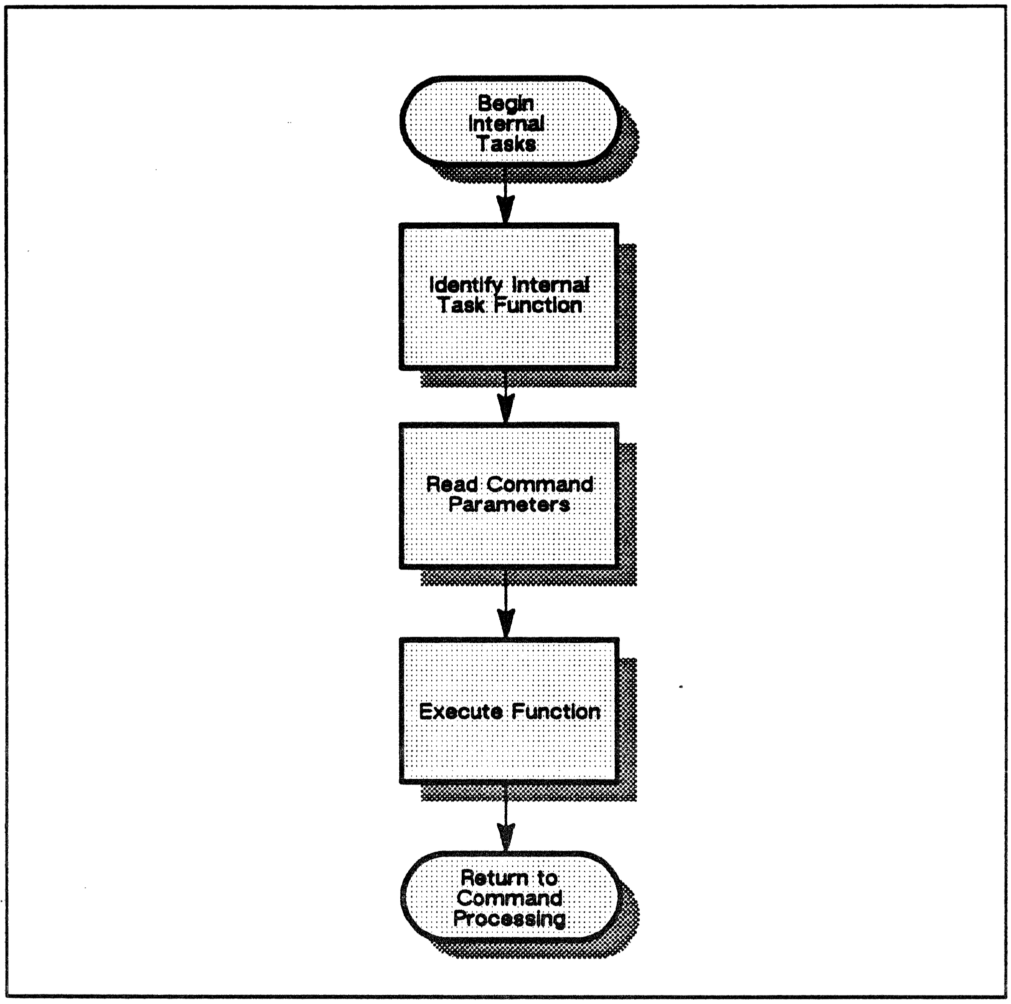


Figure 2–22. Internal Tasks flow chart.

Process Create

When Command Processing identifies a command that calls for a file to be executed, it calls process management to create a new process. The name of the executable file is passed to process management. The parameters of the command are available as a character string.

Process management creates the process from the named file and assigns it a unique ID. Inheritance information for the process is retrieved from the command shell and other system features. The new process and its associated tables are initialized.

If the process was successfully created and initialized, it is scheduled to run on a selected CPU, and begins execution. If an error occurred during this stage, process management reports it to the command shell, which logs it to the dayfile and the terminal of an interactive user, then exits to Shell Termination (batch) or Command Processing (interactive).

Process Create exits to Process Monitor, which waits until the child process has terminated. Because the command shell is a serial processor, it cannot accept another command until the currently executing process finishes.

The Process Create procedure is illustrated in figure 2-23.

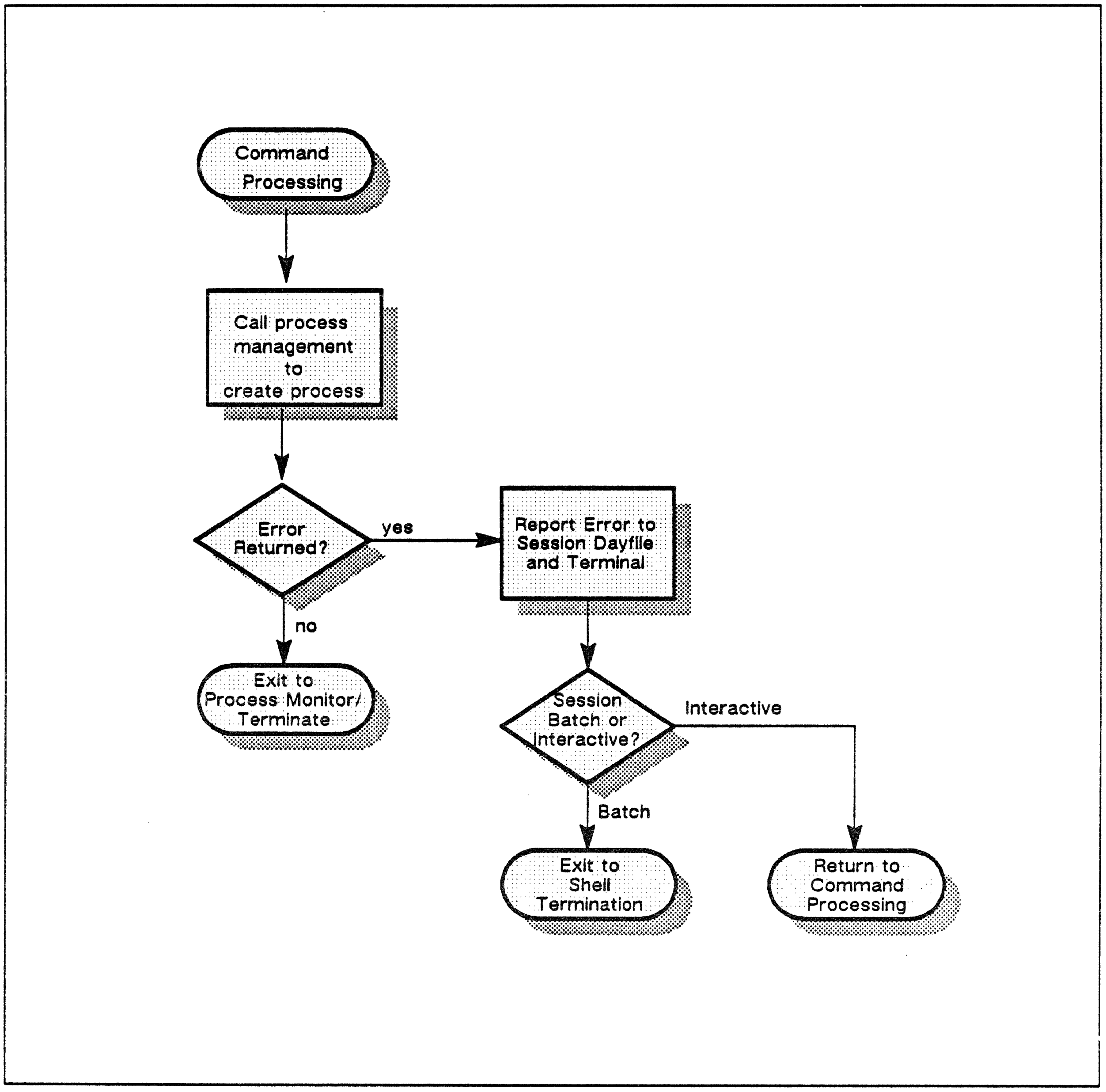

Command Processing
Call process management to create process
Error Returned?
yes
Report Error to Session Dayfile and Terminal
no
Exit to Process Monitor/ Terminate
Session Batch or Interactive?
Interactive
Batch
Exit to Shell Termination
Return to Command Processing

Figure 2–23. Process Create flow chart.

Process Monitor / Process Terminate

When Process Create has created a child process which starts executing, Process Monitor/Terminate waits for it to complete. When the appropriate software notification is received by the shell from process management, the termination status of the child process is known and is passed to Command Processing.

The Process Monitor/Terminate procedure is illustrated in figure 2-24.

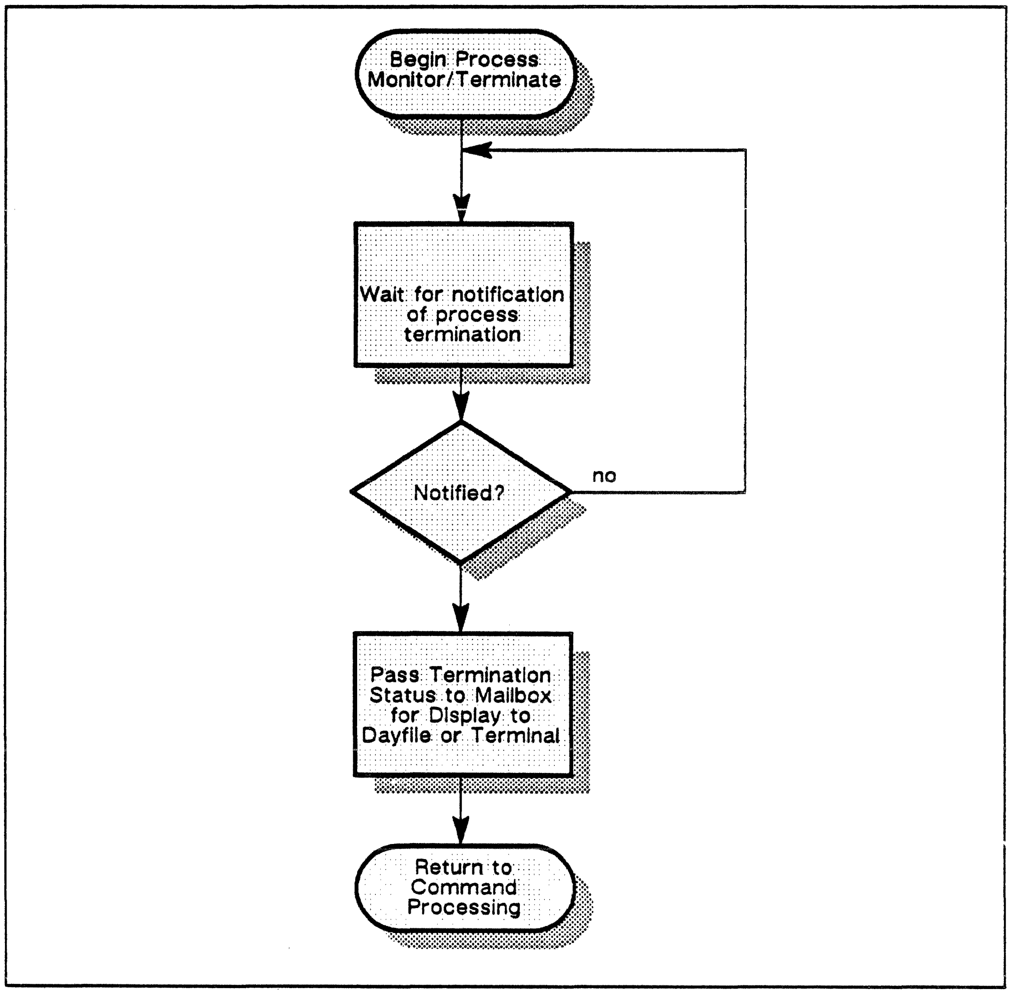

Begin Process Monitor/Terminate
Wait for notification of process termination
Notified?
no
Pass Termination Status to Mailbox for Display to Dayfile or Terminal
Return to Command Processing

Figure 2-24. Process Monitor/Terminate flow chart.

Output Disposition

When there are no more commands to be processed in the session, either because an interactive user has disconnected from his or her terminal, or because all the commands from an input file have been read, Output Disposition is entered by the command shell, illustrated in figure 2-25. It is also entered when any error fatal to the session is encountered. The shell writes session statistics to the session dayfile and exiťs to Shell Termination.

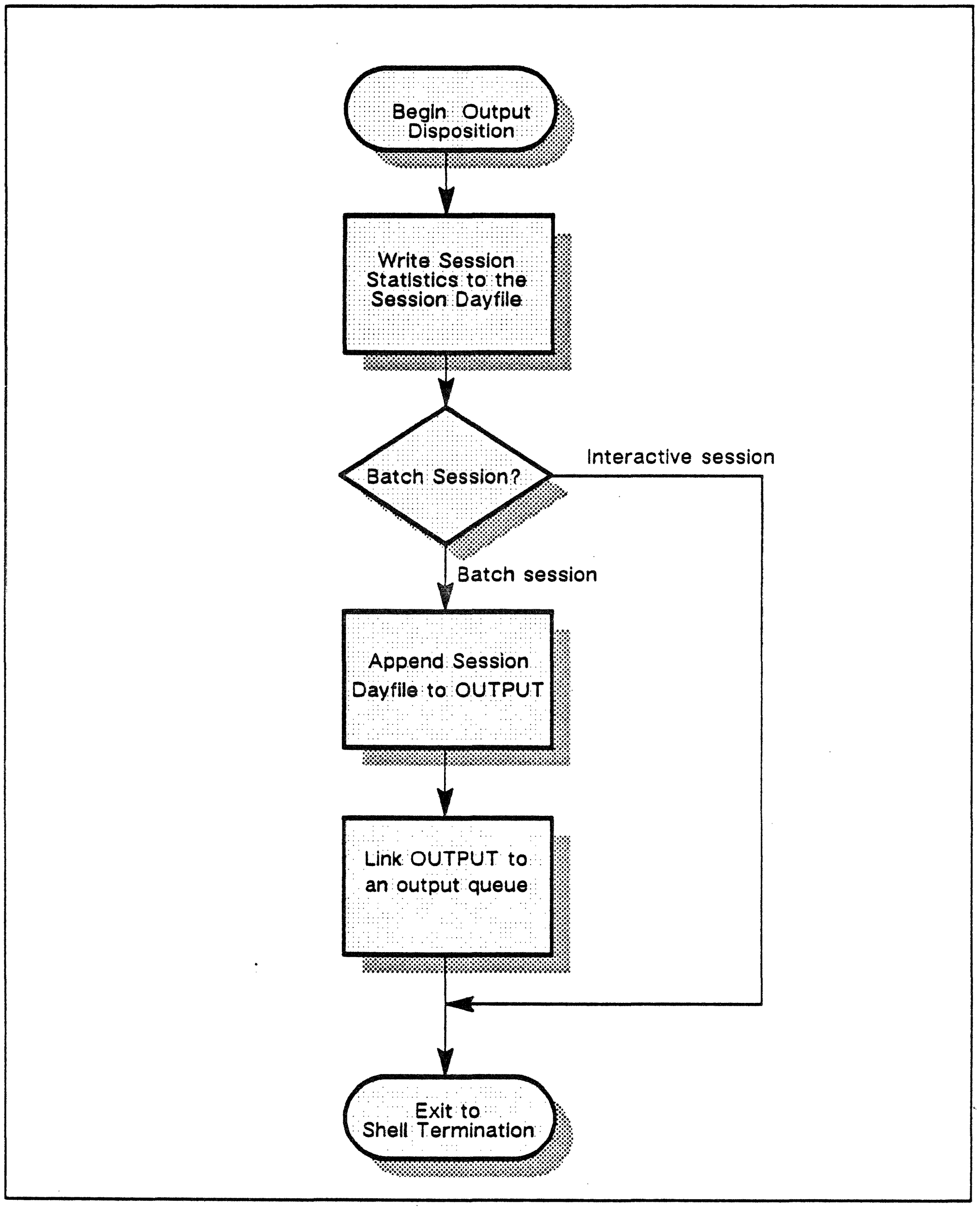

Begin Output Disposition
Write Session Statistics to the Session Dayfile
Batch Session?
Interactive session
Batch session
Append Session Dayfile to OUTPUT
Link OUTPUT to an output queue
Exit to Shell Termination

Figure 2-25. Output Disposition flow chart.

Shell Termination

When Output Disposition has completed, Shell Termination is called. The procedure is:

1. Destroy local files and the local file directory.
2. Delete command shell files and shell directory.
3. Terminate all shell children.
4. Call process management to terminate the command shell.

The Shell Termination procedure is illustrated in figure 2-26.

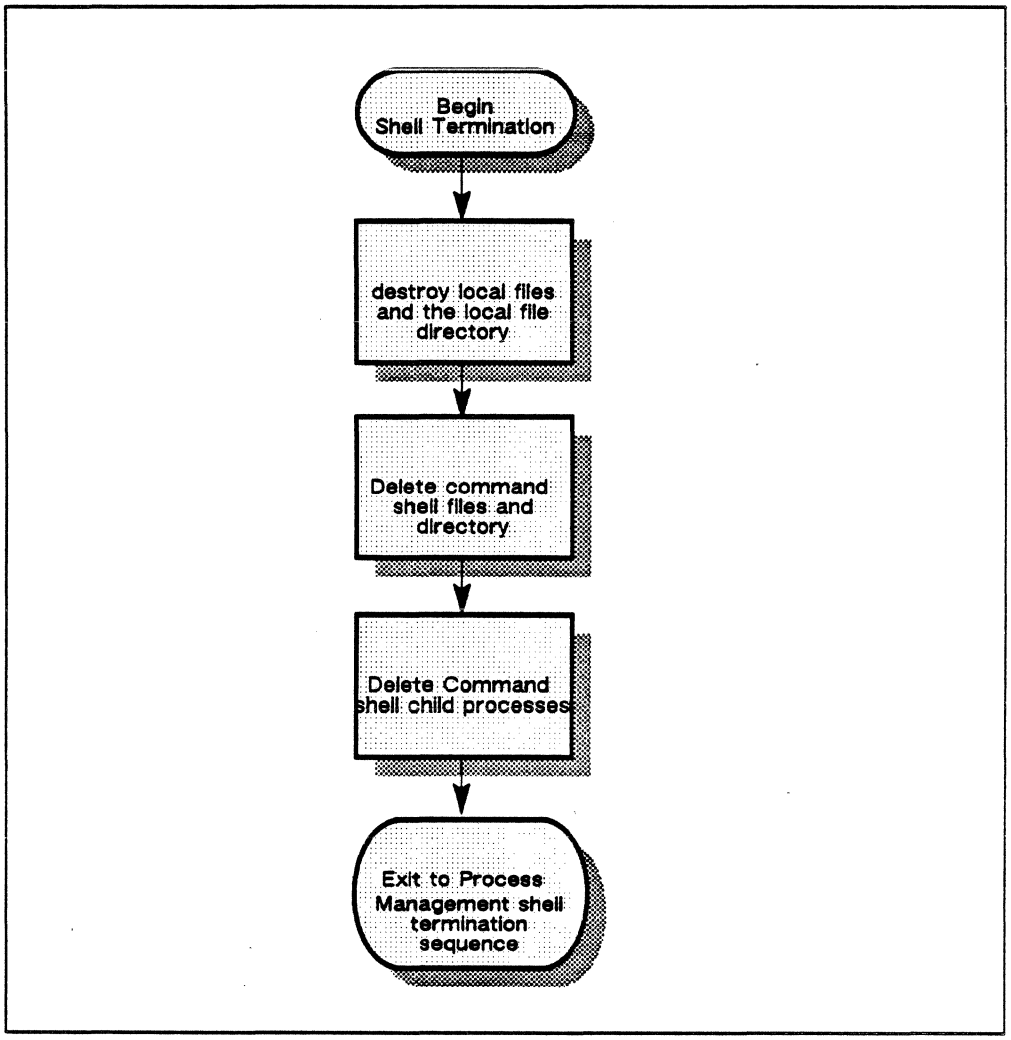

Begin
Shell Termination
destroy local files and the local file directory
Delete command shell files and directory
Delete Command shell child processes
Exit to Process Management shell termination sequence

Figure 2-26. Shell Termination flow chart.

CHAPTER 3

Chapter 3

Resource Management

Chapter Contents

List of Figures

Chapter 3

Resource Management

In This Chapter . . .

Chapter 3 discusses how the central processors, the memory hierarchy, and the I/O subsystem are managed to fully utilize the capabilities of the ETA10 and its resources.

- Introduction to Resource Management
  - a discussion of dynamic and static management
- Section 1: Resource Management
  - how system resources, user sessions, and the system configuration are managed
- Section 2: Resource Allocation and Accounting
  - how the accounting system tracks resource usage
  - how the accounting data is accessed

Introduction

ETA10 resources are internally managed by software managers that coordinate resources such as memories, multiple processors, and the input/output subsystem. Externally, the ETA10 is managed by a variety of people - analysts, administrators, operators - who direct and monitor resources, optimizing performance for each site.

Dynamic and Static Resource Management

The concepts of dynamic and static resource management enter into the management of resources. Static resource management involves the reservation of resources prior to use. For example, the imposition of a central processor time limit allowed a user session is an example of statically managing the central processor resource. In the ETA10 system, certain resources for a user's session are reserved prior to actual use and some may be reserved for the duration of a session. Users are often required to declare their needs for statically managed resources.

Dynamic management of resources involves the allocation (and deallocation) of resources as they are needed; as a result, dynamic management may obtain more efficient use of the system. As an example, unneeded memory can be reallocated to another process. In the ETA10, the accounting and control of resource usage occurs on a dynamic basis.

Resource Limits for Users

Limits are put upon users' access to system resources. Site administration defines a user's permission to access resources and limits the amount of permitted resources available to each user. Configuration limits such as number of processors and amount of memory may also impose certain restrictions on user access to the system. Users may also encounter external limits such as the availability of disk space in the user's assigned logical device.

Section 1: ETA10 Resource Management

This section describes how the following system resources are internally managed and coordinated with other system resources:

- central processor memory
- shared memory
- communication buffer memory
- central processors
- I/O subsystem

Each hardware resource is managed by one or more independent software features called resource managers. In addition, other software coordinates among the resource managers. System administrators and analysts are a second set of managers who optimize system performance and resource utilization in response to the needs of sites and users.

Shared System Data

Parallel operation of system functions and some hardware support for shared information enable the system to use several types of data sharing. For some operations, several system features may need to read the same data. As shown in figure 3-1, the system allows features executing in different processors to share data. They may also share code in the same central processor.

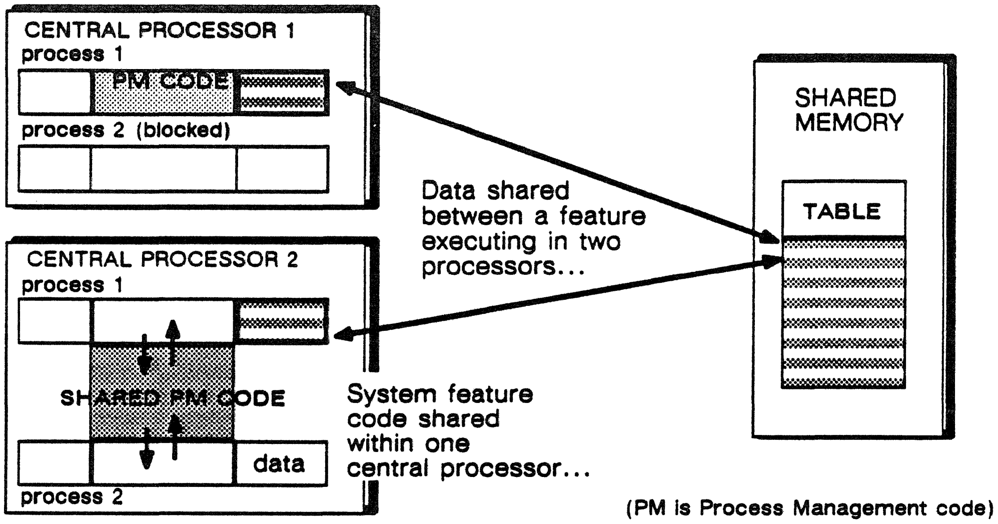

CENTRAL PROCESSOR 1
process 1
PM CODE
process 2 (blocked)
CENTRAL PROCESSOR 2
process 1
SHARED PM CODE
data
process 2
Data shared between a feature executing in two processors...
System feature code shared within one central processor...
SHARED MEMORY
TABLE
(PM is Process Management code)

Figure 3-1. Sharing of system data and code among features.

Ease of access and amount of data to be shared determine where the data resides. Larger amounts of data, say 100 words or more, are stored in shared memory. Smaller increments of shared data that require rapid access are stored in the communication buffer.

Reliability also influences where shared data is stored. When system operation requires redundant sets of memory-resident shared data, the two data sets are located either in different units of the same memory or in different types of memory. The system configuration table is accessed in shared memory by the operating system, but copies of the table are also kept on the service unit network for maintenance and operations access.

Shared System Code

System code is shared to a great extent among processes on the same processor. Code protection mechanisms known as domains allow multiple users and system tasks to shared code as shown in figure 3–1.

Management of the System Memories

The ETA10 has three system memories: central processor memory, shared memory, and communication buffer. Each has different characteristics and functions. Each memory is managed by software, using a scheme that optimizes the memory's primary function.

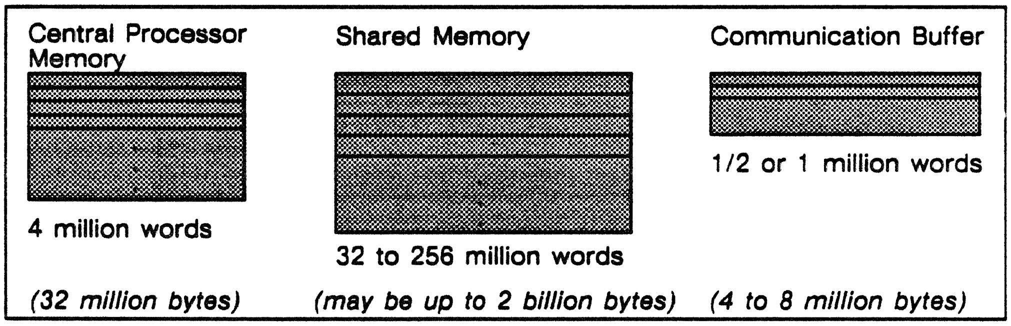


Figure 3–2. The system memories.

A central processor has unique access to its local 4 million word memory. The function of central processor memory is to provide a fast access, good-sized memory very close to each central processor. As the system's largest memory resource, shared memory ranges from 32 to 256 million words; it serves the entire system. Shared memory is available to all system processors and functions as a staging memory for the central processors. The communication buffer is used to store small amounts of data shared among the central processors. Communication buffer enables communication of information between processes running on any of the system processors.

Each system memory has an independent manager that controls access, shares resources, and performs data transfers. Although memories are managed individually, competition for memory space is coordinated. The various memory managers interact with one another to locate and transfer data within the memory hierarchy. For example, to resolve a page fault, the central processor memory manager initiates a call to the shared memory manager to locate and transfer the required pages.

Refer to chapter 4 for descriptions of memory software managers and to chapter 5 for descriptions of memory hardware.

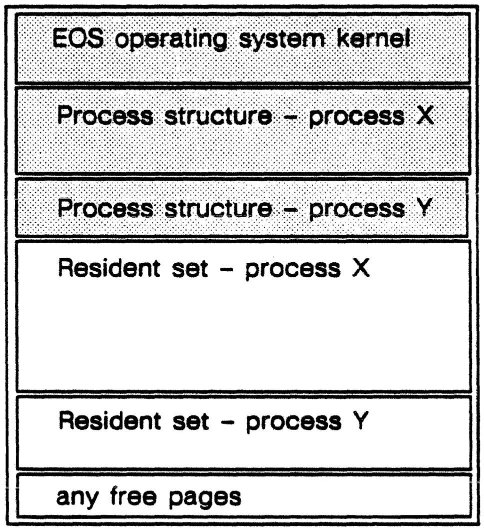
Central Processor Memory Management

A central processor memory serves one central processor and has a direct port to shared memory via its interface. The memory is independently managed by central processor memory manager software. This manager allocates and deallocates pages of central processor memory, translates virtual addresses to physical addresses, and processes page faults. Central processor memory pages are 2K words, or 2048 bytes.

Allocation of Central Processor Memory

The four million 64-bit word space in each central processor memory is allocated in pages and is divided as shown in the diagram below.

Central Processor Memory :


EOS operating system kernel
Process structure - process X
Process structure - process Y
Resident set - process X
Resident set - process Y
any free pages

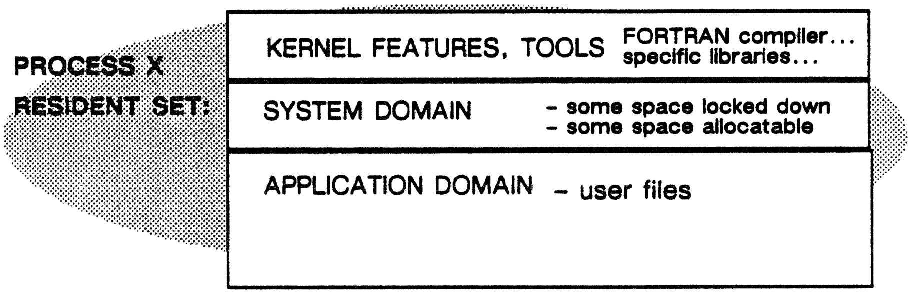
- A locked-down area is reserved for parts of the operating system kernel.
- A locked-down area is reserved for each active process's process structure: its process package, register and page tables, and the process descriptor block.
- All processes initiate and execute within their resident set. This is the total amount of memory allocated to a process at any time.
- Typically there are no free pages.

One locked-down area of central processor memory is reserved for parts of the operating system kernel. A copy of the operating system is kept in each central processor; as a process needs a certain feature such as a library or a compiler, a copy of the feature moves to the process's resident set. When the process finishes with the code, that space can be reallocated to this or another process.

There is usually little free space in central processor memory. When process X needs additional space for a compilation, that amount of space is deallocated from process Y's least recently used space and is reallocated to process X. In this way, central processor memory is kept fully allocated.

Resident Set

A resident set is the total amount of memory allocated to a process at any one time. Pages of central processor memory are allocated on demand, using a variation of a least recently used page scheme.



The size of a resident set fluctuates as the process executes. As shown above, several components comprise the resident set:

- specific features of the kernel required by the process, these are pieces like a FORTRAN run-time library.
- system domain data base; this is a list of the subsections of operating system code that are not locked and that may be allocated to meet the space requirements of the process.

Page Sizes

Central processor pages are 2K words (2048 words).

Managing Central Processor Memory for Processes

Additional pages of central processor memory may be allocated to a process when they are needed. When the page faults for a process increase, then central processor memory manager allocates additional resident working space to the process to minimize page faulting.

Page Faults

A page fault occurs when a process requires a page of data not found in central processor memory. A page fault causes an interrupt to the paging feature, requesting it to locate the page and move it into central processor memory from shared memory or from the disk. Central processor memory manager blocks the process while the page containing the data is located and moved into central processor memory. The central processor can execute other processes while the page fault is being satisfied.

The Paging Process and Page Faults

The paging feature performs basic page retrieval functions. When a process references a virtual address, hardware searches the central processor memory's space table. If the address is not found, the hardware generates an interrupt that the pager processes. As shown in figure 3-3, a page fault occurs when a referenced virtual address is not in central processor memory.

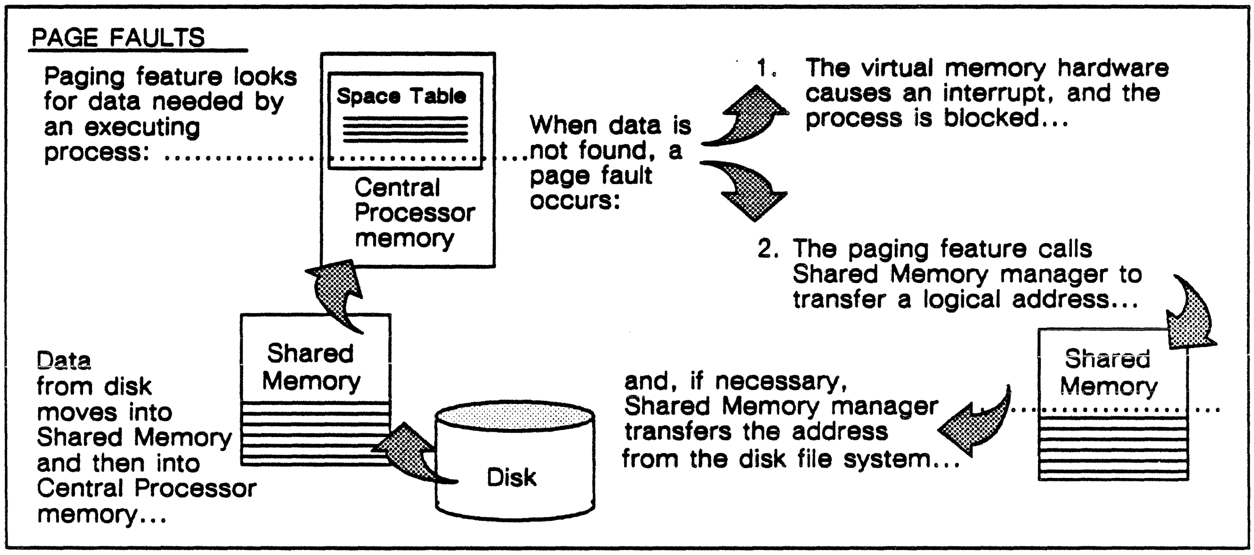

PAGE FAULTS
Paging feature looks for data needed by an executing process: ....
Space Table
Central Processor memory
When data is ...not found, a page fault occurs:
1. The virtual memory hardware causes an interrupt, and the process is blocked...
2. The paging feature calls Shared Memory manager to transfer a logical address...
Shared Memory
and, if necessary, Shared Memory manager transfers the address from the disk file system...
Disk
Shared Memory
Data from disk moves into Shared Memory and then into Central Processor memory...

Figure 3-3. Sequence of events involved in resolving a page fault.

To resolve the page fault, the pager translates the virtual address to a logical file address so that shared memory manager can transfer the file. Then the pager calls shared memory manager to make the transfer. When the data is not in shared memory, the data is moved from the disk into shared memory. To satisfy the page fault, the required page is moved into central processor memory, and the blocked process is then rescheduled to execute.

Excessive Paging Rate – Thrashing

Page faults increase when the available central processor memory becomes over-committed due to the demands of processes. At this point, there is a possibility in the central processor of thrashing. Thrashing occurs when the system spends more time handling page faults than doing productive work.

Shared Memory Management

Shared memory is used as a staging memory for the central processors. All allocation of shared memory is managed by the shared memory manager. Shared memory is allocated in blocks of 2K words (2048 bytes). Features of shared memory manager software reside on central processors, I/O processors, and service unit processors. When an executing process needs to store or retrieve data from a file in shared memory, the file system opens the file and notifies the shared memory manager. The shared memory manager determines the location of the data and performs the requested read or write operations. If the file is not in shared memory, the shared memory manager requests the disk file system to move the data in from the disk to shared memory.

As shown in figure 3–4, shared memory is accessed through its interface by the central processors through the central processor ports, and by the service unit and I/O units through the I/O interface (IOI) ports.

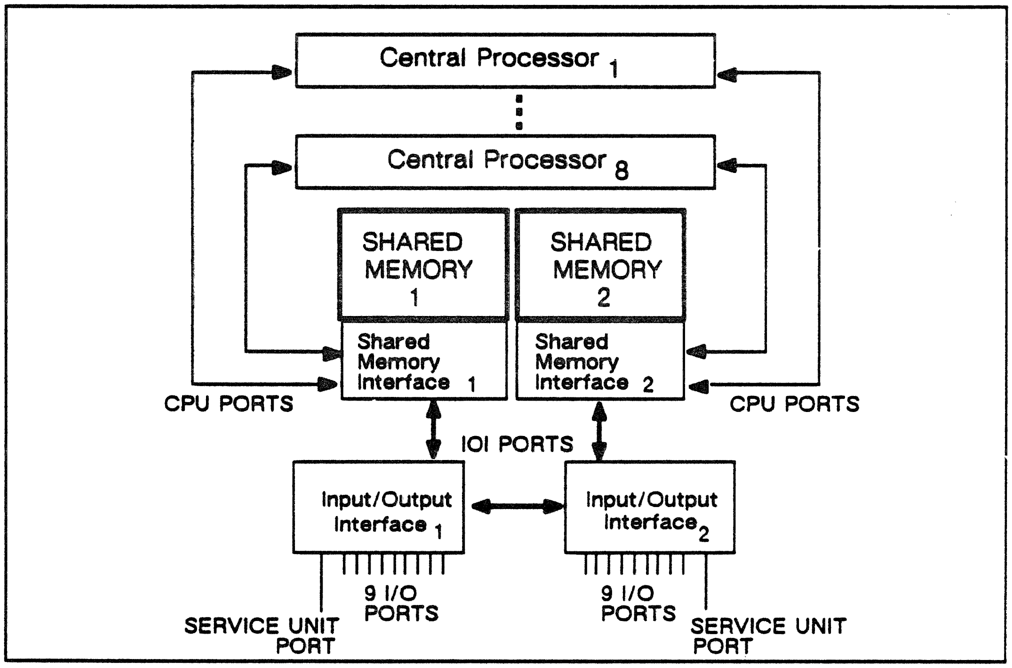


Figure 3–4. Central processor, service unit, and I/O access to shared memory in a redundant ETA10 with two units of shared memory.

Managing Shared Memory for Users

Users do not request space in shared memory. User files are cached in shared memory by the system and transferred as needed. The shared memory manager keeps only one copy of a file in shared memory.

Managing Shared Memory for System Use

Some system features maintain system data and system tables in shared memory using files and shared memory objects. Shared memory objects reside in shared memory, but are not managed by shared memory's manager.

As an example, the system configuration table is a file residing in shared memory that is referenced by many operating system features. It is managed by the system configuration control feature through which other system features indirectly access the system configuration table.

Although the communication buffer stores small amounts of data for system use, shared memory is used to hold larger amounts of data to be shared among the system processors. When the remote procedure call feature is used to send messages among system processors, the message is stored in shared memory since it is accessible by all the processors. The remote procedure call software uses communication buffer to provide the control functions associated with message passing and mailboxes used to send and receive messages.

Communication Buffer Memory Management

All central processors and system processors communicate using communication buffer, as shown below.

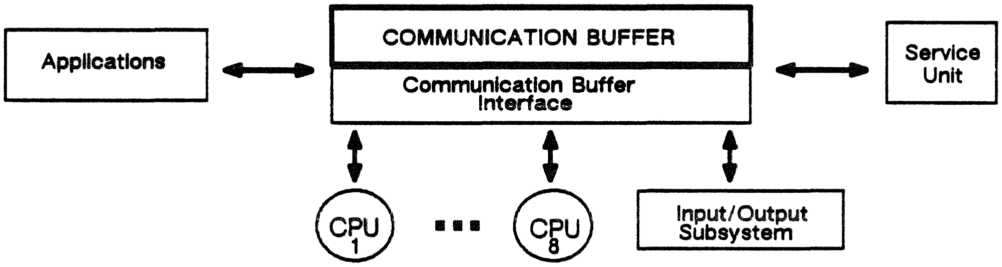


Figure 3-5. Processors/components with access to the communication buffer.

Operating system features store and share small amounts of system data in the communication buffer since it is more readily accessed than shared memory. When the amount of shared data is large, it is stored in shared memory. The communication buffer allows processes executing on different processors to share data; because there are multiple processors, more than one process may need to access the tables in the communication buffer at the same time. The service unit uses the communication buffer interface to access the maintenance interface for monitoring and diagnostic purposes.

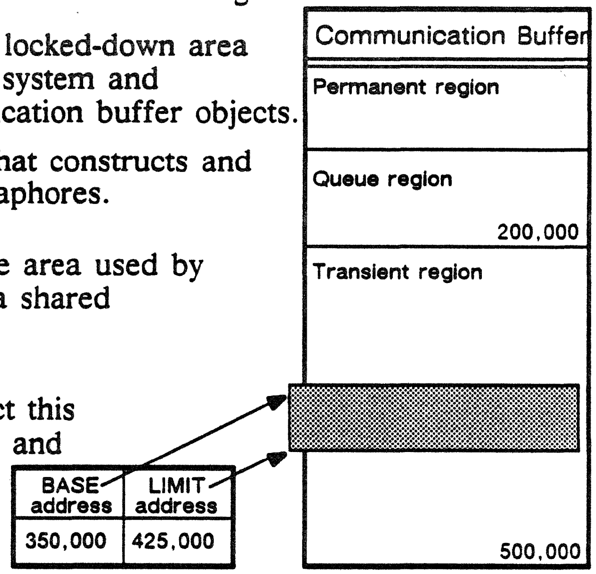
Communication buffer has three fixed-size regions:

Permanent region - this is a locked-down area used for system and communication buffer objects.

Queue region - stores data that constructs and controls semaphores.

Transient region - relocatable area used by applications and to store data shared by user processes.

Base/limit access pairs protect this memory by defining the area and types of access granted to each user process accessing communication buffer.



The communication buffer manager allocates memory in variably sized blocks to system callers. Users may indirectly access the communication buffer when using some applications.

System Data Storage

The communication buffer stores and maintains system data required by all central processors:

- shared memory block table (describes shared memory contents)
- additional configuration information not stored in shared memory
- global file information for system tables residing in communication buffer and shared memory
- remote procedure call tables (a feature that allows messages to be passed between processes on different processors; using remote procedure calls, all system processors communicate with each other)
- semaphore assignment data (simple semaphores provide access control over a single resource, usually a table in central processor memory; extended semaphores provide access control over multiple resources)

Management of Processors

The current software release supports the management of central processors as separate entities. User sessions are started at nodes directed to one processor. As shown in figure 3-6, all processors function independently, accessing the ranks of shared memory and communication buffer reserved for each. Each I/O unit is dedicated to a single processor. Disks connected to an I/O unit constitute the logical disk devices for that central processor.

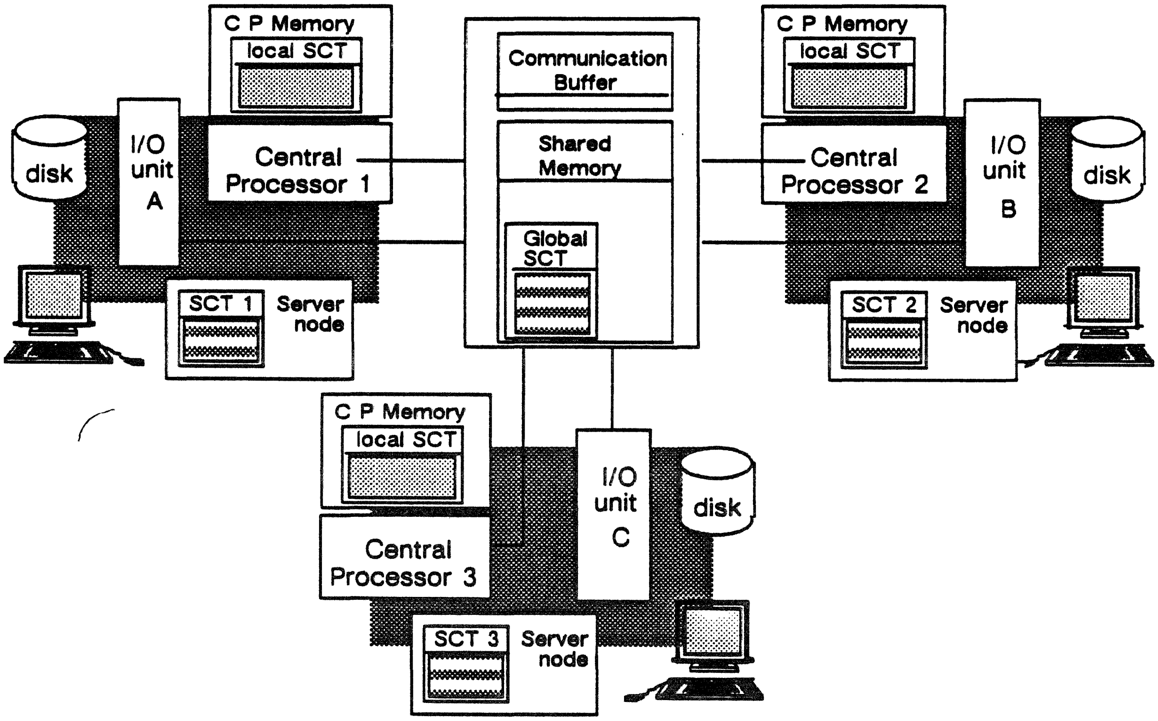


Figure 3-6. Independently functioning central processors.

Each central processor keeps a copy of the operating system in its own memory. The global scheduling feature is resident on each processor and manages only the sessions sent to that processor.

Note that there is a global system configuration table residing in shared memory, it is required in order to initialize and start up the system. A local copy reflecting a single central processor and its I/O units, disks, and other components is stored and referenced in each processor's memory. A copy of the local system configuration table for each processor is also kept in the server node of each service unit for maintenance and modifications.

Management of the Input/Output Subsystem

The I/O units use shared memory as a cache for user files. Each I/O unit has its own port to shared memory where all the I/O ports can transfer concurrently into special I/O buffers.

In addition to providing a rapid transfer of data to and from the mainframe, the I/O subsystem is the major manager of disk space. The I/O subsystem off-loads I/O processing activity from the central processors and allocates sectors on the disk storage units. As shown in figure 3–7, disk physical-file system software translates physical file addresses used by the operating system to sectors on the disk drives.

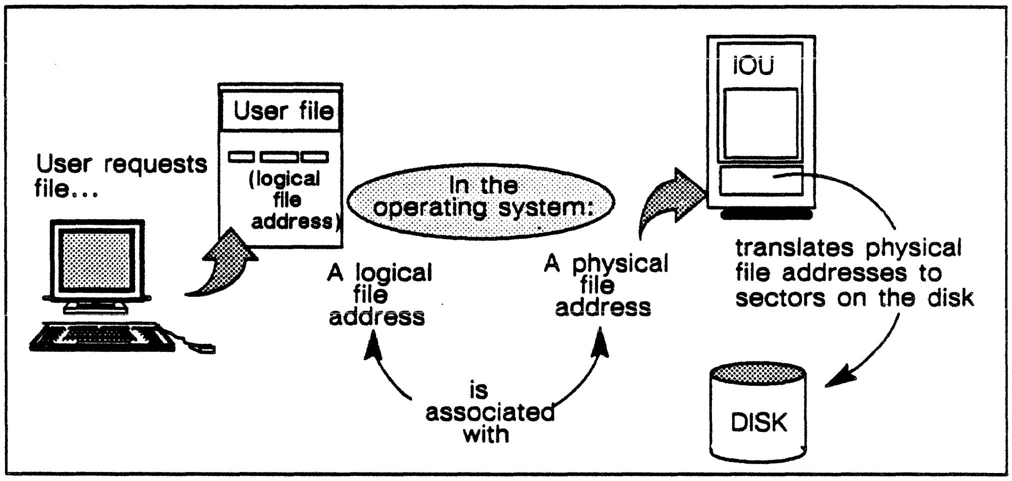


Figure 3–7. I/O file request and address translation process.

Logical Devices and Device Classes

Administrators define logical devices and device classes for the disk storage system. Portions of up to four disk drives are grouped to form a logical device. Logical devices belong to a device class or classes. Device classes represent a logical device(s) reserved for specific uses: large files, operating system files, a specific person or project, or some other class of user. The disk physical-file system assigns files to a logical device based upon the user's assigned device class as well as available space in the logical device(s) belonging to the device class.

File Size

The allocation of space to files is dynamic. Currently, the system does not limit file size. The system automatically extends a file while it is being written as long as there is space for it to grow on its logical device. If a file from a large batch session overflows its allocated logical device, the logical file system can re-start the write if there is a second logical device in the same device class.

Management of the System Configuration

The system configuration table contains a record of all the configurable hardware and software units in a particular ETA10 system and maintains the system configuration information. This includes the state and owner of all configurable units, software versions to be loaded into processors, and initialization and tuning parameters. A set of service unit hardware and software configuration displays are used to monitor the system configuration and to dynamically modify the table.

The System Configuration Table

A special program builds the initial system configuration table (SCT) during hardware checkout. This initial table is loaded as a file in the service unit, and during system initialization, a copy called the global SCT is loaded into shared memory. All information necessary to initialize or autoload the system must be contained in the global SCT. A copy of this table is kept in each central processor memory and is referenced by operating system features to obtain information such as the status of memory ranks.

Service unit server nodes also keep a copy of the system configuration table which is dynamically updated. System configuration control software in the service unit maintains changes to the table and enables it to be externalized via the hardware and software configuration displays. When a component is configured in or out of the system, this service unit version of the system configuration table is updated to reflect the new configuration, a new copy is sent to the shared memory to replace the prior configuration record, and the ETA10 is then autoloaded.

Contents of the System Configuration Table

This table contains all pertinent configuration information for the ETA10 system. This includes all system hardware and software components and all site configurable and tuning parameters. System configuration control software has sole access to read and write to the table. The system configuration table is composed of a variable number of blocks that are tailored to the type of unit each block describes. For example, blocks that describe hardware components have a different structure than blocks that describe software.

The primary purpose of this table is to record the current status of the configurable units in the system, including:

- hardware unit definitions and interrelationships
- hardware unit status and attributes

- software server/driver status and communication path identifications
- software unit versions and attributes
- central processor parameters
- logical disk configuration and allocation information
- network configuration parameters and statistics
- site modifiable system parameters such as defaults, limits, thresholds, and tuning parameters

Internal Configuration Software

The software feature that maintains the system configuration table is called system configuration control. This feature records changes in the table and returns requested data to other system features. A second feature, system configuration management, performs several configuration functions:

- receives status changes from monitoring software
- coordinates changes in the configuration status of all hardware and software units, forwards changes to system configuration control software
- communicates status changes to other system resource managers
- enables operators to manually change status of configurable units

Section 2: Resource Allocation and Accounting

The allocation and accounting of resources is through the system accounting structure. System resources are directly allocated to site accounts. Each account has one or more projects; user names are assigned at the project level. Using the accounting structure, site administrative staff define the system accounting units allocated to each account and project.

The accounting system collects data on the usage of central processor time by executing processes.

Resource Tracking and Accounting

This diagram shows a set of hardware resources (top, left) that are assigned to various software resource managers. As processes execute and use resources, the resource managers track the usage and send resource usage reports to the accounting system interface.

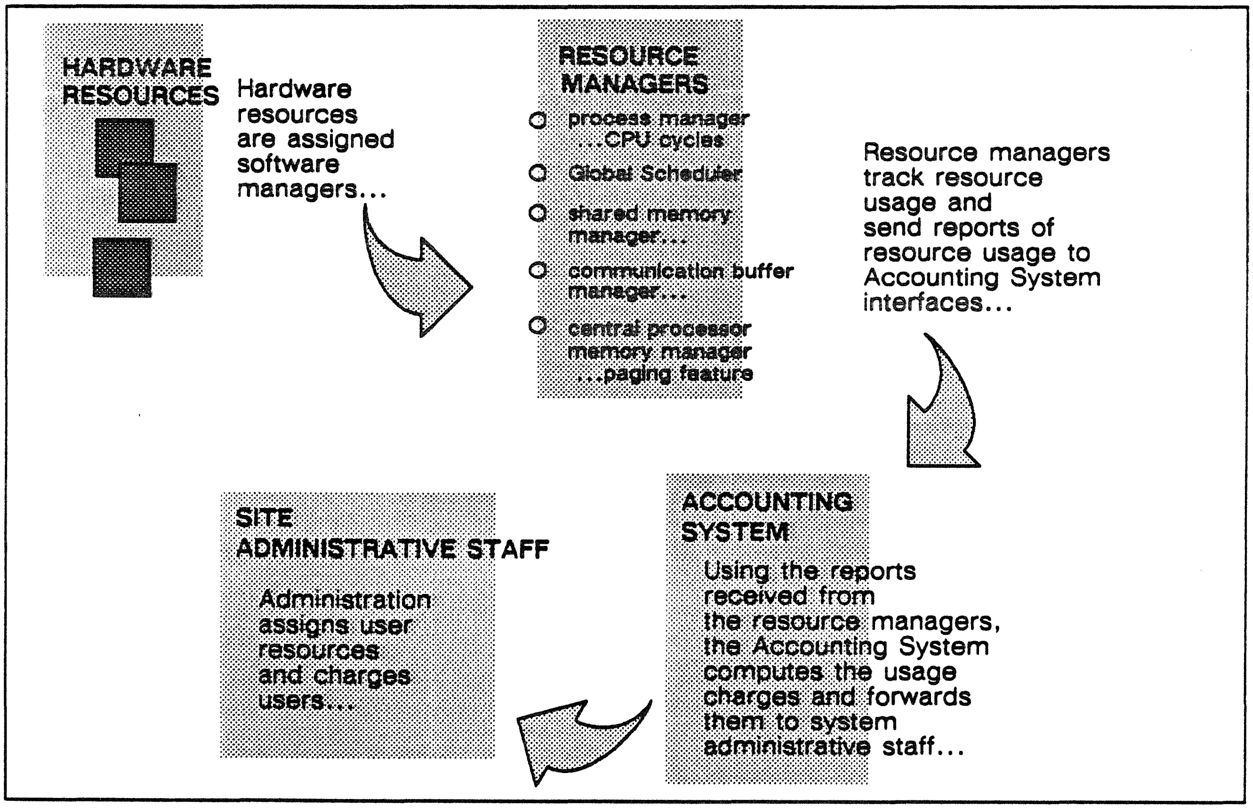

HARDWARE RESOURCES
Hardware resources are assigned software managers...
RESOURCE MANAGERS
○ process manager ...CPU cycles
○ Global Scheduler
○ shared memory manager...
○ communication buffer manager...
○ central processor memory manager ...paging feature
Resource managers track resource usage and send reports of resource usage to Accounting System interfaces...
ACCOUNTING SYSTEM
Using the reports received from the resource managers, the Accounting System computes the usage charges and forwards them to system administrative staff...
SITE ADMINISTRATIVE STAFF
Administration assigns user resources and charges users...

Figure 3–8. The relationship between resources, the accounting system, and site administrative staff.

Accounting System

The primary purpose of the accounting system is to collect statistics that reflect the resources used by each session and process in order to track resource usage.

The manager of each hardware resource reports the usage of that resource to a collection point accessed by the accounting system. Whether a session or process is stopped or fully executes, the accounting system totals the resource usage statistics for the session or process. The usage totals are translated into charge units that site administrative staff can convert into amounts billed to the account, project, and user.

Accounting System Functions

The accounting system has several functions:

- it collects and accumulates resource usage data
- it builds and maintains a global accounting file
- it calculates charge units to be used by site accounting
- it applies charge units to accounts

Accounting File

The accounting file contains records of:

- user code execution time in number of central processor cycles
- user code execution time in microseconds
- number of vector processor cycles used in session
- system execution time in microseconds (use of operating system for file creation and retrieval, process initialization, and so forth)

The accounting system records the resource usage at the process level and accumulates/translates the usage to the session level. These records are kept in the accounting file. The accounting file is a printable ASCII record, a maximum of 1024 bytes in length.

Access to Accounting System Data

Site administrative staff access the information in the accounting system by logging onto the ETA10 in an interactive session, calling up the accounting file, and printing it out. This is the ASCII version of the accounting file, and it is formatted as a text file. But it may be copied and edited like any file, and may be printed out.

Tracking Resource Usage

As resource managers coordinate and manage the hardware, they also record resource usage to the accounting system interfaces. This enables the accounting system to track usage information at session and process levels.

Tracking Levels

The accounting system uses a file maintained in the account/project registry to validate users, projects, and accounts.

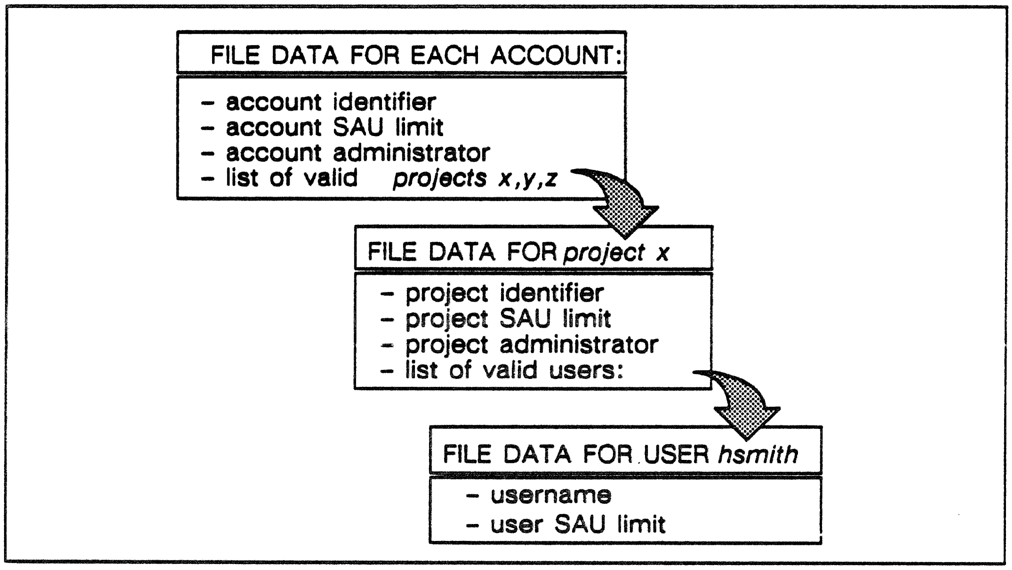


Figure 3–9 . Account and project registry file data that tracks and accrues charges for users, projects, and accounts.

Projects are defined and managed within accounts. Resources are assigned to projects and are recorded as system accounting unit (SAU) limits. Users are registered under each project to which they are allowed to charge. A project's resource usage and accounting information is reported per individual user session.

The accounting system maintains a running total of system accounting units (SAUs) for each active process. System accounting units are increments of processor time or system time.

Each session has a data block that includes a project and user name to enable the accounting system to charge system accounting units to users as well as to projects. The charges accruing to a set of users can be combined to determine charges per project.

Accounting at the Session Level

As the global scheduler feature moves a session from the input queue to the execution queue, it starts the accounting report mechanism. The report mechanism writes a record in the session accounting catalog for the new session. The catalog resides in shared memory. As the session executes, global scheduler reports resource usage data for the session to its record in the session accounting catalog. The accounting system uses the data in the accounting catalog to maintain a running total of system accounting units (SAUs) for each active session. System accounting units are so many increments of processor time or system time.

After the session fully executes or is stopped, its accounting record is sent from the session accounting catalog out to the global accounting file that resides on disk. Process statistics are added up to determine the session's accumulated usage for each type of system resource. In this way, the accounting system determines the system accounting units to be charged to this session. The site is free to use these charge units in their particular billing scheme.

Accounting at the Process Level

When the process manager feature initiates a process, it starts the accounting report mechanism. The report mechanism writes a record for the new process in the process accounting catalog. As the process executes, the process manager reports resource usage data for the process to its record in the process accounting catalog. The accounting catalog resides in central processor memory and is a record of resource usage for processes that run in that central processor. When a process is initialized to run, the process manager calls the accounting system to start. The process manager suspends accounting activity if the process is blocked, and terminates accounting activity when the process completes.

The accounting system uses the data in the accounting catalog to maintain a running total of system accounting units (SAUs) for each active process. System accounting units are so many increments of processor time or system time.

After the process fully executes or is stopped, its accounting record is sent out from the process accounting catalog to the global accounting file that resides on disk. The same accounting record is added to the record of the session to which the process belongs. Using the process's accumulated system accounting units now in the accounting file, the accounting system determines the system accounting units (SAUs) to be charged to this process. The accounting system combines the data from a session's processes into the session's final record in the accounting file.

System Accounting Units (SAUs)

A system accounting unit is the charging unit. All users charges and billing are based upon the system accounting unit as shown in this diagram:

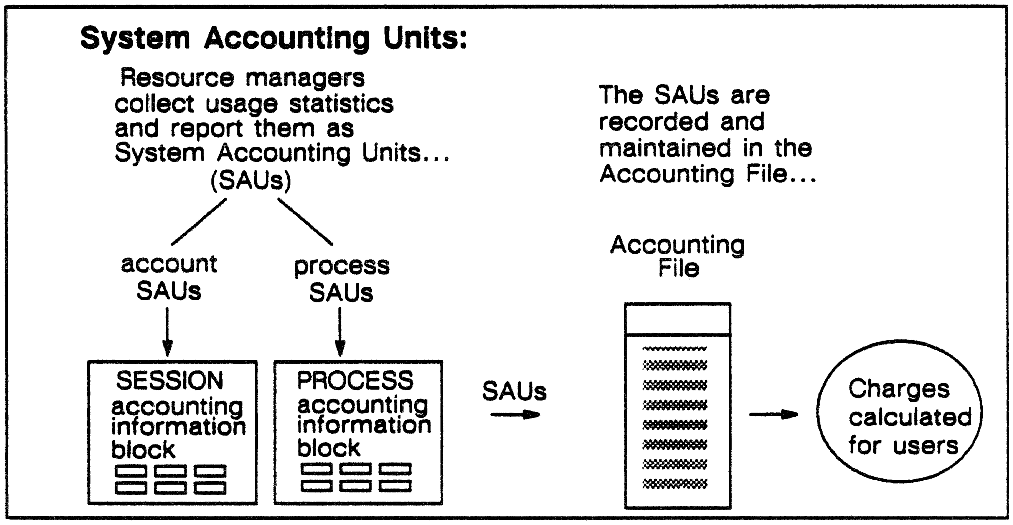

System Accounting Units:
Resource managers collect usage statistics and report them as System Accounting Units... (SAUs)
account SAUs
process SAUs
SESSION accounting information block
PROCESS accounting information block
SAUs
The SAUs are recorded and maintained in the Accounting File...
Accounting File
Charges calculated for users

Figure 3-10. How the accounting system collects, stores, and uses SAUs.

To obtain a billing unit, the site combines a charge unit or a system accounting unit with a premium that may reflect session priority, time of day, or special software.

The User Registry

This section describes the structure of the user registry, the types of information it contains, and how it is used.

The user registry resides as files on the system disk and is accessed through an administrator utility. The registry is a system database of user identification and privileges.

End user information consists of descriptions of user privileges and the specific user identification required for each user to log on. The registry maintains an entry for each user that enables the resource managers to verify specific resource limits as well as to validate users for a project.

Access to the Registry

Internally, the user registry is accessed by resource managers as they manage processes and sessions. This access is to verify resource and user privileges. The common login processor feature accesses the registry to validate usernames, passwords, and access permissions.

Externally, the user registry is accessed by site administrative staff to enter and delete users, or to modify user information. Users may view their own information, but they usually do not have the privilege to change it except for their password. They may not view other user's information.

Both administrative utilities are designed to be used from network terminals via interactive sessions with the ETA10. Most of the maintenance data associated with these utilities are stored and maintained on disk files.

User Management Utility

This utility is used to build and modify the user registry. It provides the system interface for users and their resource usage and limits. System administrators use this routine to specify and maintain user validation parameters and limits. This utility allows the administrator to create templates for a variety of user types and to apply the template to a user to build a specific user profile. A user may use this utility to look at the information about himself or herself only; special privileges are required to look at other users. When the system is started up, this utility creates a minimal user registry that the site can build from unless some version of a registry is part of the software loaded during installation.

User Information

The user information includes a considerable list of user attributes including permissions and records of user activity. A template for a typical user can be used to build individual user profiles. Partial listings of user privileges and user records follow:

Privileges to:

- log on batch and interactive
- create accounts
- change passwords
- look at other user's information
- change other user's information

Records of:

- password
- password start and end dates
- user name
- default account, project identifiers
- modify, look, change privileges
- password expiration date
- directory pathnames
- time of last interactive log-on

User Validation by the Common Login Processor

The common login processor runs continually on the system. Two of its functions are, first, to respond to users logging on and interactively prompt for user name and password, and, second, to validate the username, password, and privilege to log on. It uses the information in the user registry to validate users. When the user is validated, the common login processor creates the interactive connection for the user.

The Account and Project Registry

This section describes the structure of the account and project registry, the type of information it contains, and how it is used.

The account and project registry is accessed through an administrator utility and resides as a set of files in the system disk. The registry is a database of three types of accounting information: accounts, projects, and users.

In this registry, administrators identify all permitted accounts, define an account's maximum resource usage, and list the projects that belong in each account. The projects in each account are also identified and assigned maximum resource usages, and a list of users may be assigned to each project.

The account and project registry provides the description of the relationships between users, projects, and accounts. Administrative staff use the Account Registry utility to enter and delete accounts and projects, and to modify resources allocated to accounts and projects.

Account Registry Utility

The Account Registry utility is used to create and maintain a site's account and project registry. This utility is called from network terminals via interactive sessions with the ETA10. Most of the accounting data associated with the utility is stored and maintained on disk files.

CHAPTER 4

Chapter 4

Operating System Kernel

Chapter Contents

Contents (continued)

Contents (continued)

Contents (continued)

Contents (continued)

Contents (continued)

Contents (continued)

Contents (continued)

Contents (continued)

Contents (continued)

Contents (continued)

List of Figures

Contents (continued)

Chapter 4

Operating System Kernel

In This Chapter . . .

This chapter describes the features of the computer's operating system, and how they interact to perform computing tasks. It is divided into ten parts:

1. Introduction
2. Underlying Operations
3. Processes on the ETA10
4. The Logical File System:
   - File Directory/Catalog Manager
   - File Support Module
   - Record Manager
5. The Memory Managers:
   - Central Processor Memory Manager
   - Shared Memory Manager
   - Communication Buffer Management
6. The Process Managers:
   - Global Scheduler
   - Process Management
   - Remote Procedure Calls
7. The Domain Manager:
   - Domain Management
8. The System Managers:
   - System Configuration Control
   - System Monitor
   - System Logging and Analysis

9. System Entry:
   - User Management

Section 1: Introduction

The operating system of the ETA10 series of computer systems, is a multiple CPU, multitasking system distributed across the hardware as a series of *features*. Features are either duplicated in different parts of hardware, or are distributed across the hardware in a series of pieces.

The operating system consists of three distinct layers of software hierarchy: kernel, environment, and process layer.

The kernel layer contains the lowest level of operating system software. They perform the functions of memory management, process management and communication, and implement system services such as the logical file system, user management, and accounting.

Above this layer, and depending heavily on it, is the environment layer. This layer is what you as a user will perceive as the operating system. You can not directly interact with the kernel. The familiar commands of the CYBER 205 VSOS version 2.2 operating system are implemented in this layer.

For a system administrator or other special user, there is a second environment layer containing the service unit operating system. The service unit operating system may only be accessed from a service unit node. It is used to monitor and control conditions in the computer system.

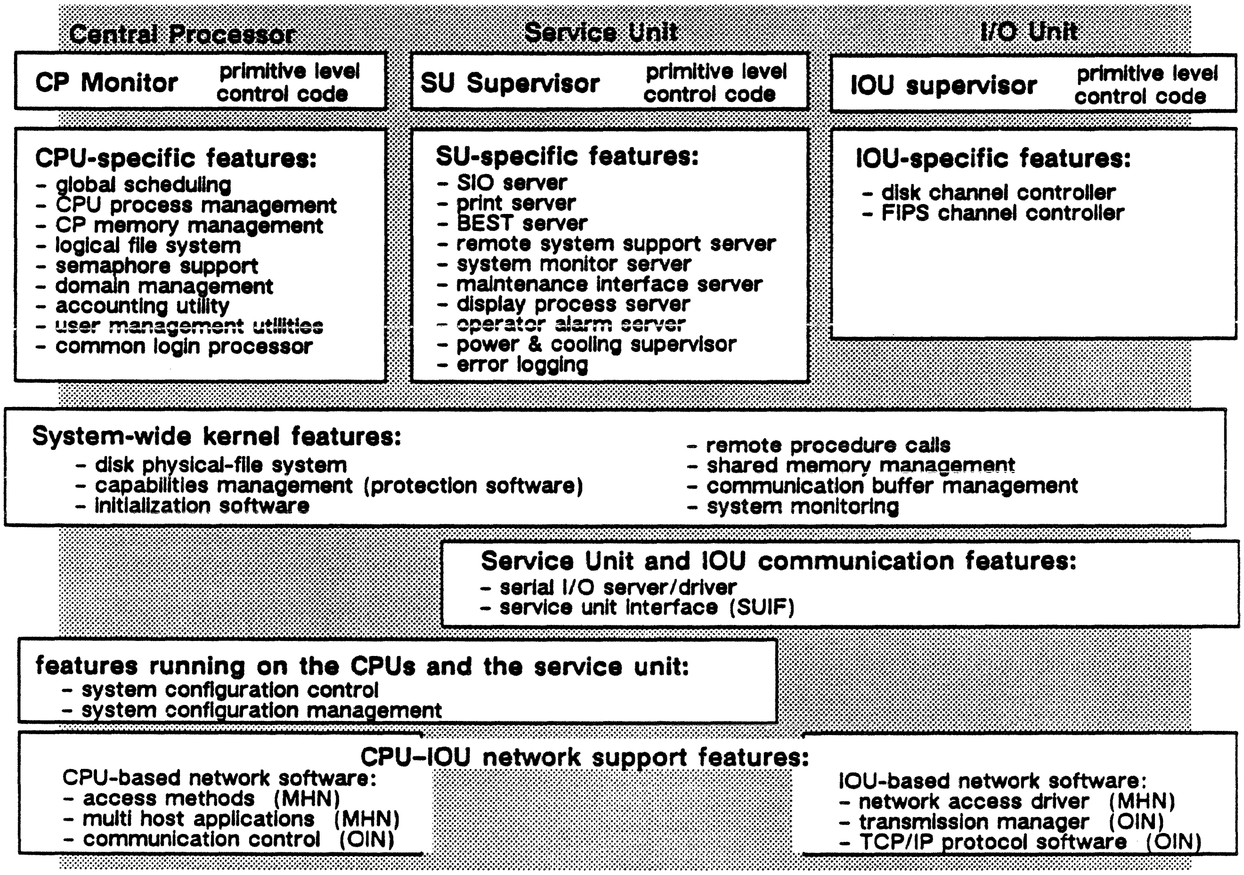
The uppermost layer of the operating system is the process layer. This layer consists of the programs and applications that are run from the environment layer, such as a utility or a FORTRAN 200 program.

For the service unit operating system, the process layer includes the applications and diagnostics used to monitor and diagnose the computer system hardware and software.

Distribution of the Kernel Features

The kernel features are distributed across the computer system hardware, residing on all three classes of processing units as shown in figure 4–1.

Kernel features:



Central Processor | Service Unit | I/O Unit

CP Monitor primitive level control code

SU Supervisor primitive level control code

IOU supervisor primitive level control code

CPU-specific features:
- global scheduling
- CPU process management
- CP memory management
- logical file system
- semaphore support
- domain management
- accounting utility
- user management utilities
- common login processor

SU-specific features:
- SIO server
- print server
- BEST server
- remote system support server
- system monitor server
- maintenance interface server
- display process server
- operator alarm server
- power & cooling supervisor
- error logging

IOU-specific features:
- disk channel controller
- FIPS channel controller

System-wide kernel features:
- disk physical-file system
- capabilities management (protection software)
- initialization software
- remote procedure calls
- shared memory management
- communication buffer management
- system monitoring

Service Unit and IOU communication features:
- serial I/O server/driver
- service unit interface (SUIF)

features running on the CPUs and the service unit:
- system configuration control
- system configuration management

CPU–IOU network support features:

CPU-based network software:
- access methods (MHN)
- multi host applications (MHN)
- communication control (OIN)

IOU-based network software:
- network access driver (MHN)
- transmission manager (OIN)
- TCP/IP protocol software (OIN)

Figure 4–1 . Kernel features and feature locations.

Features such as global scheduler and CP memory manager are confined to each central processor. Their functions are specific to CPU activity. Other features such as the disk file system are used by all three processor types, and thus have their pieces distributed across the entire system.

Central Processor Kernel Features

The central processor holds those kernel features needed to execute tasks, including the initialization, execution, input/output, and termination phases. It also holds the features which allow it to communicate with the other processors in the system.

The primary central processor feature is the CP memory manager (CPMM). It organizes and accounts for the central processor memory, and is responsible for its allocation to processes and features. The features reside in a reserved area allocated by CP memory manager at initialization.

Below CP memory manager are global scheduler and process management. They schedule and control tasks within the CPU. As a result of their activity, logical file support, accounting, or semaphore support may be called.

If a scheduled task is a user login, the common login processor may be called, followed by user management. If the login is from a remote connection, one of the network software features may be called.

It should be noted that the CPU may operate in either of two modes: *monitor* and *job* mode. In monitor mode the CPU is running its primitive level control code. The CPU is unaware of anything beyond its limited monitor instruction set and the task it has been called to perform. Monitor mode is entered by an interrupt to the CPU in job mode. Job mode, however, is the normal user mode of the system. In job mode, the feature processes may be active and the CPU is set up to execute user tasks.

Service Unit Kernel Features

The service unit holds those features needed to query the ETA10 hardware and diagnose any problems that may be encountered. It also holds the service unit operating system (SU_OS). The service unit operating system is an interface between the native operating system of the service unit host, the user, and the ETA10 operating system kernel.

Residing on the service unit operating system are the many service unit features. Most of these are concerned with the diagnosis and maintenance of the ETA10, as well as the monitoring and displaying of normal operating status.

Additionally, to allow the service unit access to the information it needs, the shared memory and CB manager features are present in the service unit. The disk physical-file system (DPFS) feature is provided so that the service unit may monitor and retrieve disk information from the I/O unit.

Another part of the service unit's function involves configuration. The service unit works closely with the CPU to perform system configuration. The two processor types share the system configuration control (SCC) feature. System configuration control has as its responsibility the system configuration table (SCT), which holds the actual configuration information. A master copy of the system configuration table resides in the service unit. At system initialization the table is copied to and used by the individual CPUs.

The service unit and the I/O unit (IOU) processor share the serial I/O server/driver and service unit interface (SUIF) features. These features allow the service unit and I/O unit to communicate with each other over the serial line that connects them.

I/O Unit Kernel Features

The I/O unit is responsible for all data movement into and out of the computer system, including the disk physical file system and the networks.

The service unit supervisor consists of primitive level control code that directs the MC68020 I/O unit processors. Each of these processors is also executing in its native supervisor mode. Running above the supervisor and making calls to it are the I/O unit features.

One responsibility of the I/O unit is the disk file system. The disk physical file system and disk channel controllers are features of the I/O unit.

Another responsibility is network communication. The I/O unit shares network communication features with the CPU processors.

On the system side, the I/O unit shares the communication buffer and central processor memory manager features with the rest of the system. The I/O unit shares the system monitoring feature with the rest of the system. It also shares a serial I/O line and its supporting software with the service unit.

The Kernel Operating System Feature Groups

The operating system kernel level may be considered as a number of features arranged in groups, each group representing features that have a common or interrelated purpose. Several of these groups are introduced here. The groups and their features are described in detail beginning with section 4 of this chapter.

The Logical File System

The file directory/catalog manager (FDCM), file support module (FSM) and record manager (RM) are the three elements of the logical-file system (LFS).

File directory/catalog manager oversees the tables where the file catalog for the system is kept. Each directory, subdirectory, and file is represented as a node. A file's position in the directory tree is represented by its position with respect to the nodes surrounding it. A chain of nodes forms a path name. Functions are provided that allow callers to create and delete nodes, thus creating and destroying the directories and files they represent.

File directory/catalog manager supplies directory information to all tasks that request it, including any user FILES commands coming from the VSOS environment.

The file support module opens and closes files. It also manages the tables that maintain file states for tasks. These tables indicate whether the file is open or closed, file positioning, whether the file is shared or not. Other file support module tables hold attributes associated with files in the VSOS environment.

The file support module also provides the interface between the logical file system and requests coming from other kernel features and the I/O unit processors.

The record manager is the single feature through which explicit I/O with the file system is performed. The concepts normally associated with files - record format and blocking type - are implemented in the record manager.

The record manager is not used for any type of implicit I/O.

The Memory Managers

The CP memory manager (CPMM), SM manager (SMM), and CB manager (CBM) are the three memory managers.

CP memory manager controls the allocation of the memory that immediately surrounds each CPU. It allocates and deallocates memory pages for processes. It reserves the area used by other kernel features. It is also the communication link between the CPU and the outside world, interacting with the SM manager and logical file system for saving and retrieving data.

SM manager controls the large independent memory used by all the CPUs and features for general-purpose, in-system storage. SM manager allocates memory pages for processes until all memory has been allocated. If further memory is needed, it begins to page out least-recently used pages of memory in coordination with the logical-file system until all requests are satisfied.

CB management controls the fast memory of the communication buffer. This buffer is used almost exclusively for the passing of messages and control information (including semaphores) between processes and/or processors. It is not used for general storage.

The Process Managers

Global scheduler and process management are the two process managers.

Tasks are received by global scheduler and assigned to execute on a CPU. No non-monitor activity executes in the CPU without first passing through global scheduler.

Process management oversees a process throughout its lifetime. When a request for process creation is received, process management creates the table entries and information necessary for the process to execute. It creates and initializes the process and notifies global scheduler.

When the process begins executing, process management works with the CP memory manager to ensure the proper resources are present. When execution is complete, process manager is responsible for removing the process and all its entries in system tables from the system.

The Domain Manager

Domain management (DM) is the domain manager. It sets up the system and user domains, and maintains the security of the system kernel from unauthorized access.

The System Managers

System configuration control (SCC), system monitor (SMTR), and the system logging and analysis feature(SLA) are the system managers.

System configuration control is responsible for the system configuration table (SCT). This table holds the detailed configuration information for both system hardware and software.

The master copy of the system configuration table always exists in the service unit. At system initialization, individual copies of the table are made in each CPU. The CPUs then execute based on the information contained in their local system configuration table.

System monitor is the feature that receives error and status information from all the other system features. It provides periodic verification of hardware and software operation.

The system logging and analysis feature receives error information from system monitor and places it in a database accessible from the service unit.

System Entry

User management (UM) is the system entry feature responsible for validating user access to the system.

The common login processor (CLP) is a part of user management. The common login processor presents the prompt you see each time you log in. User management also maintains the files which hold your default login parameters and the user registry, which contains your privileges and limits within the computer system.

Section 2: Underlying Operations

Certain essential functions of the operating system involve a low level interaction between system software and the hardware. These include the most important functions of the monitor - *interrupt handling* and *process switching* - as well as the operations of the *virtual address mechanism.*

Process switches involve the saving and restoring of different types of registers by hardware. The vital restart information for an inactive process is kept in sets of *locked-down process objects*, process objects kept at a fixed address in CP memory rather than in virtual memory. The same locked-down process objects exist in CP memory for every process from the time it is initialized on the CPU to the time it is terminated.

This section describes:

- Monitor operations.
- The virtual address mechanism's use of the associative registers and the space table in CP memory.
- The nature of the locked down process objects themselves, the process queues set up by process management and maintained by monitor, and the critical servers that reside on each CPU.
- The exchanges of process information between registers and process objects that accompany process switches and domain changes.
- The domain change, which allows a process to use system code without going through a process switch.

CPU Monitor

A monitor runs on each CPU in a special, non-interruptable mode - monitor mode. It accesses central processor memory directly by physical address.

All other processes, both applications and system servers, run in *job mode*, which is interruptable. Their access to CP physical memory is indirect, by way of the virtual address mechanism.

A single instruction, Exit Force, toggles between monitor and job modes. It is the only exit from monitor mode to job mode, so that monitor operations conclude with an explicit Exit Force instruction. In job mode, either an explicit Exit Force or an interrupt causes an exchange from job to monitor mode - from execution of a process to execution of the monitor.

Monitor's Functions

The monitor performs some basic functions:

Interrupt handling - When an interrupt occurs, hardware automatically exits from job to monitor mode, and monitor scans the interrupt register to see the source of the interrupt, then reschedules processes accordingly. Interrupts include:

- Access interrupt
- Transfer request block (TRB) completion interrupt
- Illegal instruction interrupt
- Exit Force interrupt (executed in Job mode).

For a more complete description of interrupts, see chapter 5, "Interrupt Register".

History - Monitor can keep a time-stamped log of all interrupts, messages, and executions of processes in the CPU.

Idle loop - If monitor has no processes to run, it loops in monitor mode, watching the interrupt register and checking server mailboxes in the communication buffer.

The monitor communicates with the kernel and with other CPUs through mailboxes in the communication buffer and shared data structures (including process objects) in CP memory. These data structures must be locked down for the benefit of the monitor, which addresses contiguous physical memory (contiguous virtual pages are not necessarily contiguous in physical memory). The kernel locks them at the time of process initialization (carried out for each process by process management's Process Initialize function) and then passes their physical addresses to the monitor.

Process switching - The monitor handles the specific sequence in which processes will run by moving processes around on the process queues. In carrying out this task, monitor responds both to specific indications in the interrupt register and to messages from process management (received through a shared table).

Interlock regions - Monitor guarantees completion of *interlock-region code* that operates on SM management objects in CP memory.

Process Queues

There are four process queues: the *ready*, *blocked*, *recovery*, and *termination* queues. Monitor puts processes scheduled to run onto the ready queue, where they get their turn on a first-in, first-out basis. After the monitor completes its business, executes an Exit Force, and toggles the system to job mode, the process at the head of the ready queue is put in the Running state - that is, begins using the central processor.

Process management coordinates the features which build a locked-down process object for each process, then notifies monitor to link the process's descriptor block to the **Ready queue**. Processes on a queue are linked through their process descriptor blocks by a double-linked list.

Processes are put on the **Blocked queue** to:

- Wait for I/O completion. When the transfer is complete, they are woken up by the completion server, which continuously polls the status of transfer request block queues and the disk physical-file system.
- Wait for message in a mailbox.
- Wait for a semaphore.
- Wait on a page fault.
- Wait for a time limit. With the exception of a process waiting on a page fault, processes on the blocked queue are subject to a time limit.

Monitor watches constantly for expiration of the time limit or a request from process management to change the status of blocked processes.

Monitor moves active processes to the **Recovery queue** in case of:

- An illegal instruction.
- A CB access violation. Use of a CB address that exceeds the bounds of the base and limit addresses contained in the four access pairs of the current domain.
- A memory access violation. Use of a virtual address out of a process's address space. This violation is first processed as a page fault, and the process is moved to the Blocked queue. When pager finds the address to be out of bounds, pager calls monitor to move the process to the Recovery queue.

In Release 1.0, the next step for processes on the Recovery queue is the **Termination queue**. After a process enters the Recovery queue, the process management server changes the execution address in its process package to a special termination routine, and puts it on the Ready queue. The process runs a last time, closing all its files and putting itself on the Termination queue. PM server finds it on the Termination queue, unlocks its process objects, and asks monitor to take it off the Process queues altogether.

Virtual Address Mechanism

Virtual addressing is carried out by coordinated hardware and software operations. On the hardware side, a set of 16 associative registers holds 16 associative words. Each word serves as an index between a virtual page identifier and the physical address of a page located somewhere in the four million words of CP memory. In effect, the associative registers mirror the top 16 words of a longer table – the *space table* – a system table locked down and maintained in CP memory by the kernel. The space table contains a set of associative words representing all of a process's pages that are stored in CP memory. It consists of at least 16 words (the number held by the associative registers) and an end-of-table marker.

Hardware searches for a virtual page address by checking first the associative registers, then the space table. When the requested page is found, its associative word is moved into the associative registers and the words below it are rippled down toward the slot it came from. If the page is not found, an access violation interrupt is generated denoting a page fault, and the search is turned over to the system page fault handler, the pager.

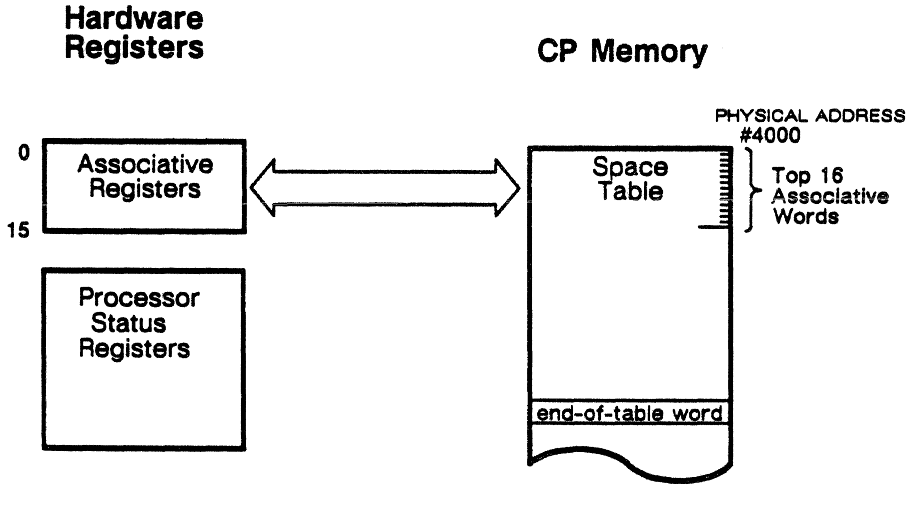


Figure 4–2: Virtual address mechanism.

Besides serving as an index between virtual and physical addresses, an associative word contains a usage-code field that is updated when the page is read or written. The usage code enables the virtual address mechanism to identify unused pages for replacement and to "clean" pages that have been modified by copying them back to shared memory before before replacing their page frame in physical memory with another page.

The associative word also contains a lock which protects it from unauthorized access. When the virtual address mechanism attempts to access a page through its associative word, it checks the lock against 12 keys held in the *processor status registers*, a diverse set of registers that hold information about the state of the CPU as well as the currently running process. If no match is found, access to the page is denied and an access violation interrupt generated. If a match is found, the addressing mechanism checks the proposed access type against a three-bit access violation code (read, read/execute, read/write) associated with each key.

The associative word contains four significant fields:

- *Usage*. Indicates the size of the page (small or large) and, if it has been accessed, whether it was read, or read and written to.
- *Lock*. Must be matched by one of the 12 keys in a domain key set for access to the page.
- *Virtual page address*. Address of the page in virtual memory.
- *Physical address*. Address of page in physical CP memory. The low-order bits of this address are an offset into the physical page.

Usage, lock, and virtual page address are set up by pager at the time of a page fault. Page size (a bit in the usage field), lock, and virtual memory address come from process memory maps and the invisible package, two of the locked-down process objects. The physical page address is supplied by pager at the time a page joins the resident set in the CPU. Virtual and physical addresses are linked in the page's associative word.

A more detailed description of virtual memory operations appears later in this chapter in "Page Fault Processing."

Locked–Down Process Objects

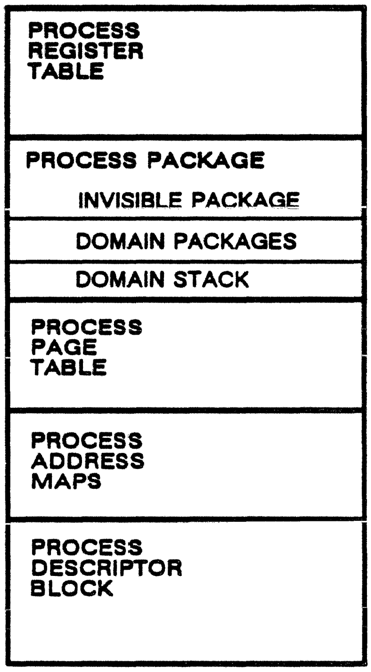


Figure 4–3: Process Object Group

The monitor sees only individual processes (not process clusters), and it sees each process as the data incorporated in its locked–down process objects. One element of the process structure the register table, which is loaded into the CPU register file when the process is activated and updated when the process is deactivated. Other elements are a page table, a process package, a memory map, and a process descriptor block (PDB).

Process management coordinates the creation of a process's locked down process objects during process initiation. Once they are locked down in CP memory, process management passes their physical addresses to monitor.

Process Register Table

The process register table contains a saved set of the process's general register. They are loaded into the central processor's register file when the process is activated.

Register Usage

EOS defines the CPU's 256 general-purpose registers as follows:

3 *Machine registers.* Contain machine zero and two addresses used by the data flag branch manager.

17 *Temporary registers.* For the execution of short subroutine modules.

6 *Global registers.* Contain the constants #20, #1A, #1, the parameter descriptor for calls, and two registers for simple or complex values returned by called functions.

8 *Environment registers.* Contain various pointers and links, and the return address from calls.

222 *Temporary and Working registers.*

Process Page Table

The process page table contains a copy of the space table at the time the process was blocked. (Except that pages of the system library have been excluded). It retains the order and usage information for the process's associative words.

Process Package

The process package contains critical restart information. It is made up of three components: an "invisible" package, a set of domain packages (two in EOS release 1, version 1.0), and a domain stack.

Invisible Package

The invisible package contains a saved set of the processor status registers containing vital restart information for the process:

- Program execution address.
- Keys and access rights (AVCs) for current domain.
- Four CB base/limit access pairs (BLAPs) for current domain.
- Access interrupt address.
- Vector pipe status and intermediate results.
- Instrumentation counters and select codes.
- Other process restart information for hardware.

When a process enters the Running state, hardware loads its invisible package into the processor status registers at the same time the general registers are loaded from the process register table.

Domain Packages

Each process on EOS 1.0 has two domain packages: one for the system domain and one for the application domain. There is space in the process package for 128 domain packages. A domain package is an abbreviated invisible package. It includes the program execution address, keys and BLAPs of the domain, instrumentation counters, and a breakpoint address.

Domain Stack

The domain stack is made up of an ordered stack of *stacked domain packages*. A stacked domain package contains an even more abbreviated version of an invisible package than a domain package. A forward domain change increases the stack by one package, a backward domain change decreases it by one. A stacked domain package includes program execution address, current instruction, a partial set of keys, and a stack index.

Process Address Map

The process address map links virtual page addresses with the position of the pages in the executable file and the paging file. The paging file is a scratch file holding pages that have been altered during the current run of the process. When a page fault occurs, pager looks in the memory map, fetches the appropriate page out of one of these files, and puts it into a CP memory frame. It then creates an associative word by associating the page's virtual address to the CP memory frame's physical address.

Process Descriptor Block

- Process's inheritance information.
- Process attributes.
- Scheduling parameters (monitor attributes).
- Message buffers used for communication between process management and the monitor.

Mode Exchanges

In exchanges between monitor and job modes, current registers are saved, and new ones are loaded. Information held in hardware registers that handle virtual address, process execution address, and the security of the system domain is saved and replaced. Then, mode and interrupt-enable are toggled, and control branches to the next instruction.

Hardware executes four kinds of mode exchanges:

A *Half Exchange*, performed after a Master Clear, loads the initial monitor registers into the register file.

An *Initial Exchange* to job mode initializes a process's locked-down process objects.

An *Interrupt Exchange* to monitor mode is performed when any bit is set in the interrupt register.

A *Return from Interrupt Exchange* to job mode restarts the execution of interrupted process code.

The interrupt exchange and return are the most typical examples of mode exchanges. They involve transferring critical process information between the register file and other hardware registers and the locked-down process objects. An interrupt exchange and return constitute a **process switch**.

A process switch involves two exchanges: an exchange from process A to monitor, then from monitor to Process B. Each exchange involves saving and restoring the hardware registers.

JOB MODE	MONITOR MODE	JOB MODE
Process A ==>	Monitor ==>	Process B

The initial interrupt (say an access violation interrupt when a virtual page can't be found in CP memory) stops the process and switches to monitor. Process registers and other information must be saved and monitor's registers loaded. Then monitor deals with the page fault by putting pager on the ready queue, where it lines up like any other process to be run on a round-robin basis. Monitor next issues an Exit Force. Its registers are saved and the registers and other information information of the process at the head of the Ready queue are loaded into hardware.

Interrupt Exchange: Job to Monitor

An interrupt causes a full exchange of register information and a switch from job to monitor mode. Figure 4–4 shows five elements of the exchange between hardware and memory:

1. Hardware saves the process's register file to the process register table.
2. Hardware saves the process status registers to the process's invisible package.
3. Hardware loads the monitor's registers from the monitor register package at CP memory addresses # 0 - 4000.
4. Monitor copies the process's associative registers to the top of the space table at memory address #4000.
5. Monitor then sorts the space table, saves part to the process page table and part to the system library page table (library page table not shown).

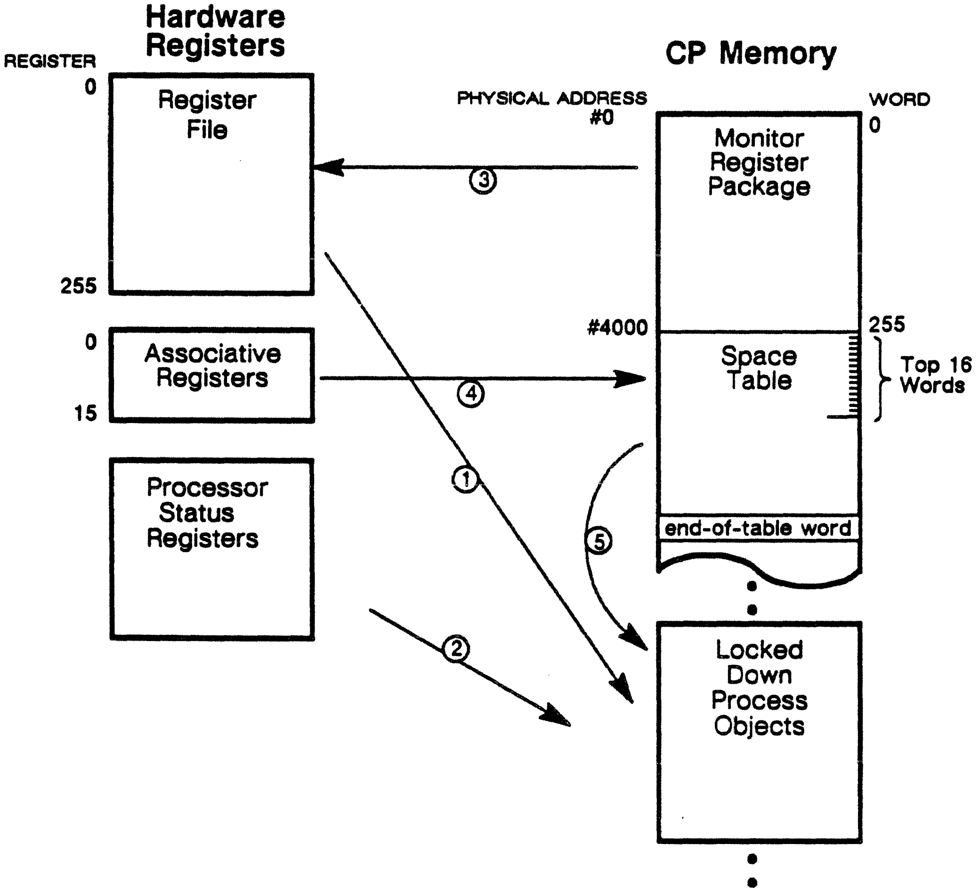


Figure 4–4: Job to Monitor Mode Exchange.

Return from Interrupt: Monitor to Job

The exchange from monitor to job modes is almost the reverse of the previous, as you can see in figure 4–5:

1. Monitor loads a process page table into the space table.
2. Monitor copies the top 16 entries of the new space table into the associative registers, then issues an Exit Force.
3. Hardware saves monitor's current registers to the monitor register package.
4. Hardware loads the incoming process's register table into the register file.
5. Hardware loads the process's invisible package into the processor status registers.

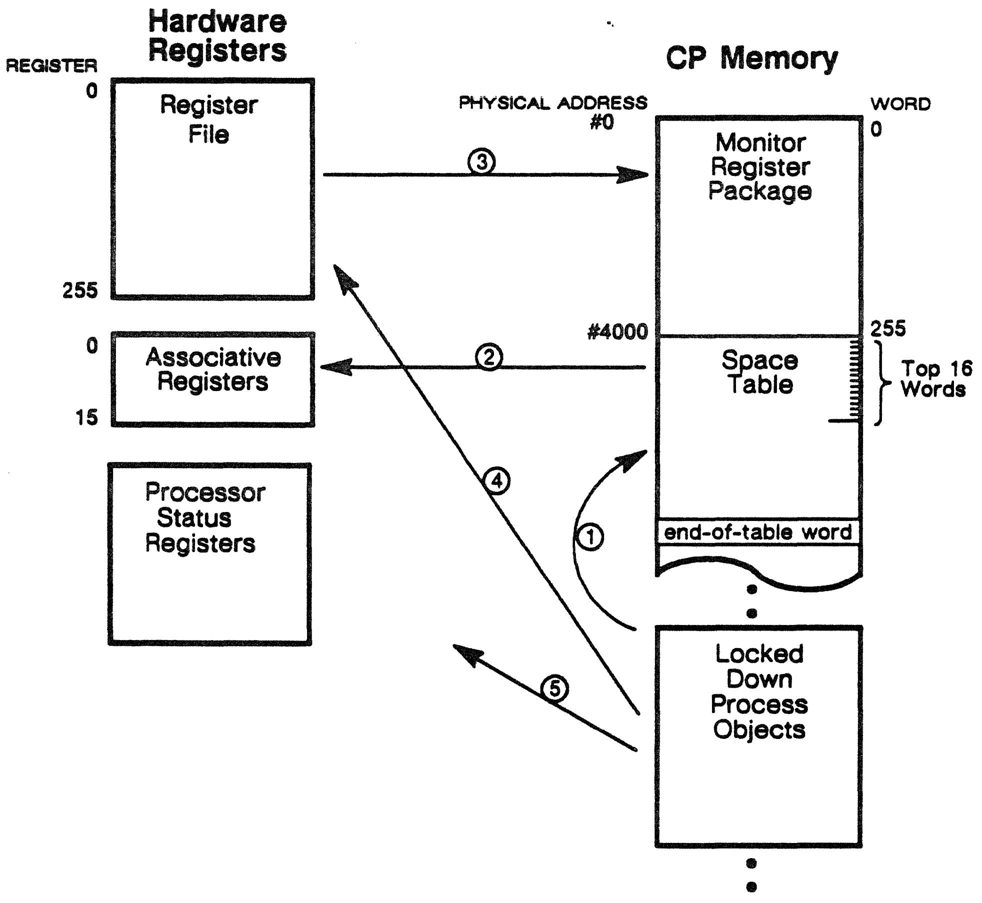


Figure 4–5: Monitor to Job Mode Exchange.

Domain Change

A *domain change* allows a process to access system facilities by branching to protected routines in a shared library while remaining in job mode. It uses the key-and-lock matching of the virtual address mechanism as a protective device. Hardware cannot access virtual pages for which it does not hold a key. A process can move from one domain to another without incurring the overhead of a process switch -- meaning that process switches are reserved for starting user applications or system servers.

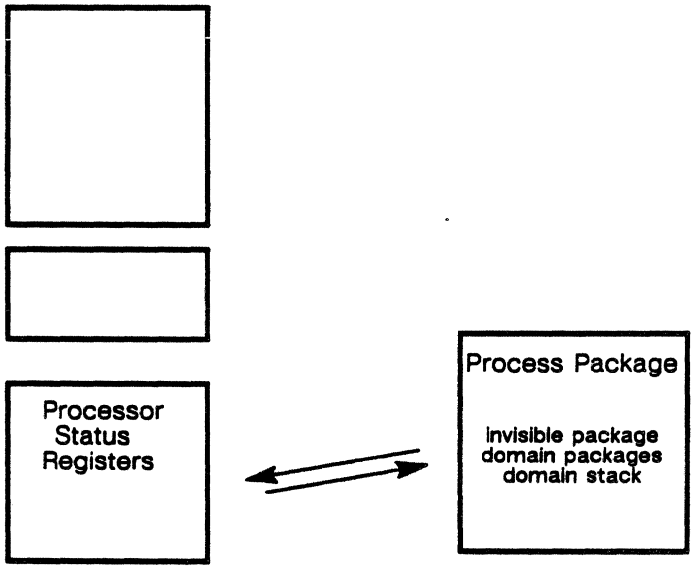


Figure 4-6: A forward domain change, with exchange of execution address, keys, access violation codes (AVCs), BLAPs and other information between process package and process status registers. Register file and page tables remain in place.

In job mode, hardware accepts virtual addresses from software and converts them into physical addresses. As part of the conversion, hardware matches one of 12 keys owned by the current domain and kept in special registers with the lock field in the associative word containing a target address. *The current domain is the set of virtual memory pages unlocked by hardware's current set of keys.* Hardware cannot access virtual pages for which it does not hold a key.

A hardware *forward domain change* instruction results in the loading of new execution address, keys, AVCs, CB base/limit pairs (BLAPs), and other information about the new domain from the process package, changing the set of virtual pages which the hardware can address. Similar information about the old domain is put on the domain stack as an abbreviated domain package (which is itself an abbreviated invisible package).

A *backward domain change* restores the stacked domain information, returning to the prior domain. No process switch has taken place.

Hardware and operating system architecture can support up to 128 domains. EOS release 1, version 1.0 supports two: a *system domain* and an *application domain*. The system domain is a set of shared subroutines available to processes. It includes all of the CPU job mode kernel.

Critical System Servers

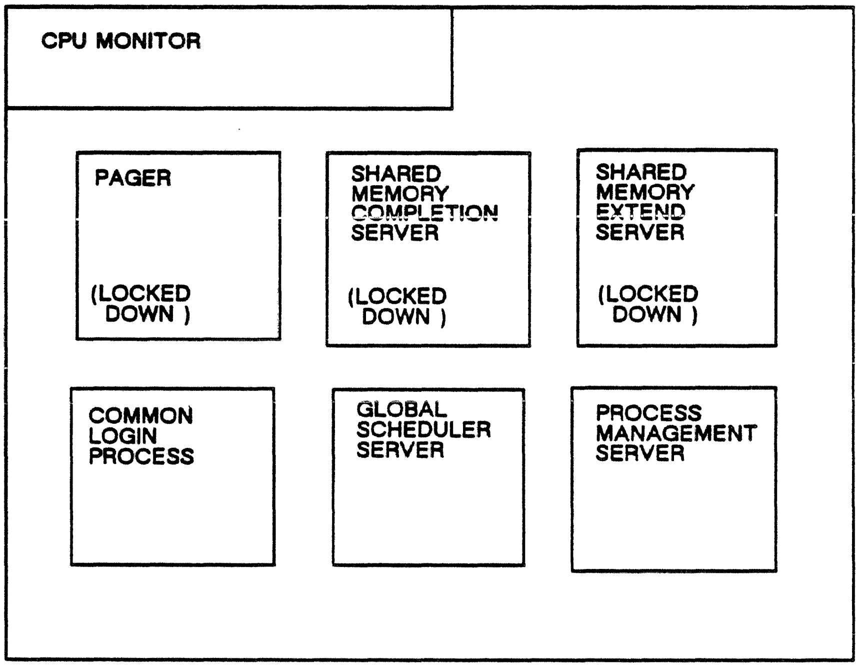


Figure 4–7: Critical System Servers.

Five system server processes are of critical importance and exist on every CPU:

- Pager, the page fault server.
- The completion server, which manages the interface logic that moves blocks of data between shared memory and the CPU.
- The shared memory extend server.
- The process management server.
- The global scheduler server.
- The common login process.

Pager, the completion server and the SM extend server are locked down in central processor memory. Monitor schedules pager as a direct response to interrupts. In EOS release 1, version 1.0, completion server does its own polling and is scheduled by process management. The activities of pager and the completion server are described in the next section of this chapter.

Section 3: Processes on the ETA10

This discussion follows the path of a command that is entered into the system by a logged-in user, accepted by the system, and from which a process is created that will execute in one of the system's central processing units. The following sections detail the interaction between different features of the operating system (including process management, domain management, global scheduler, and central processor memory manager) to create the process, execute it, and handle its termination.

Definition of a Process

A process in the ETA10 system can be thought of as an association between an executable file, a process ID, and process management objects which hold the information the system needs to manage the process. The executable file contains the process' loaded code as well as a header holding process information such as the domains it can access. The process ID is a system-wide unique ID assigned by the operating system to the process during its life. The process management objects include the process descriptor block (PDB), register table, and process package.

Process Objects

There is one PDB for every process in the system. The PDB is a block of process-specific data which includes inheritance information, the process attributes, monitor's process attributes, and buffers.

The register table holds the registers of a process when it is not executing. It is created and initialized by process management, and the process' original register values read into it from the executable file header.

The process package consists of the invisible package, domain package, and the domain stack, all of which hold information about the domains accessible to the process. The invisible package has data about the executing domain, for example, the current program address. The domain package has hardware information about all the domains known to the process, and the domain stack is a stack for domain packages that were executing, and will be returned to later.

Tracing a Process Through the System

Figure 4–8 shows the different stages in the life of a process.

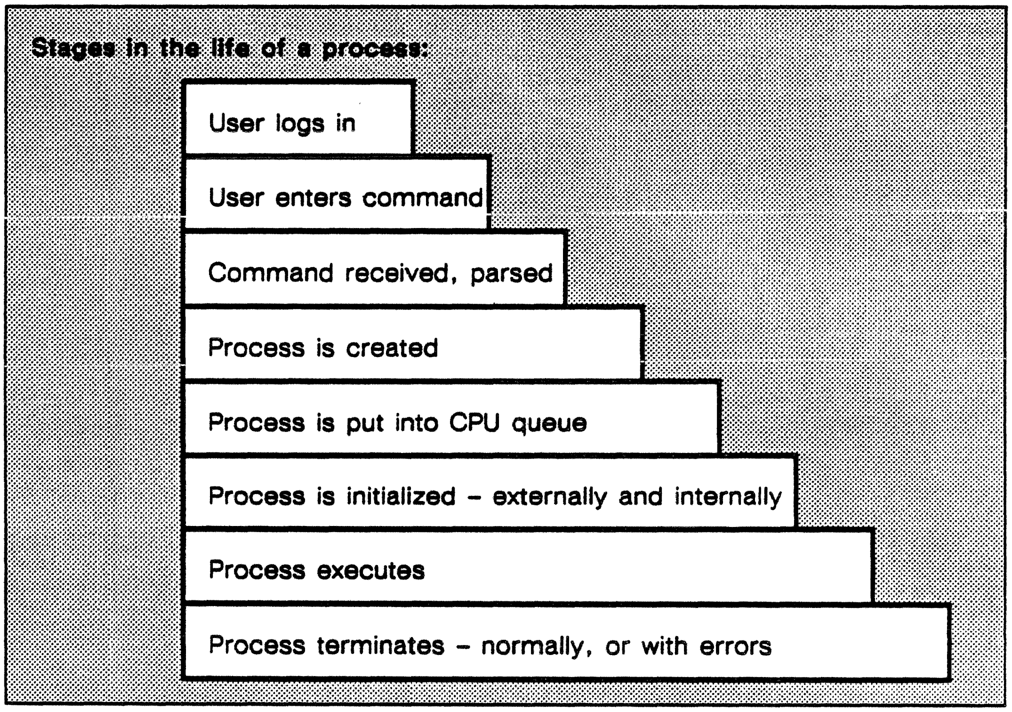

Stages in the life of a process:
User logs in
User enters command
Command received, parsed
Process is created
Process is put into CPU queue
Process is initialized – externally and internally
Process executes
Process terminates – normally, or with errors

Figure 4–8. Stages of a process in the ETA10 system.

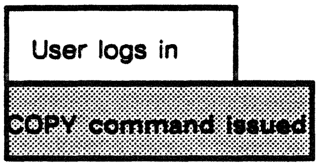
The eight stages described in figure 4–8 are detailed in succeeding sections.

User Login

A user logs into the system and enters a command which is received by the user environment command shell. One user shell is created by the common login processor for every login. After the command shell parses the command and verifies that it is valid, the Process Create function of process management is called to create a process for the command (executable file) to be executed.

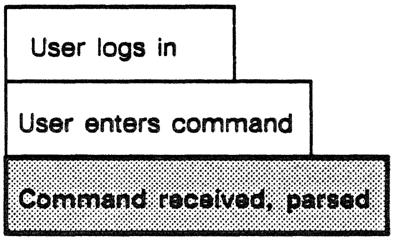
After the newly created process is initialized by process management, it begins executing. When the process terminates, process management performs the necessary termination functions, depending on whether the process terminated normally or with errors.

Command Entry



A logged-in user working in the VSOS environment can enter a command interactively or through a batch command file. The command is the name of an executable file which, when invoked, requests the operating system to perform some function. An example might be a user requesting to COPY one file to another.

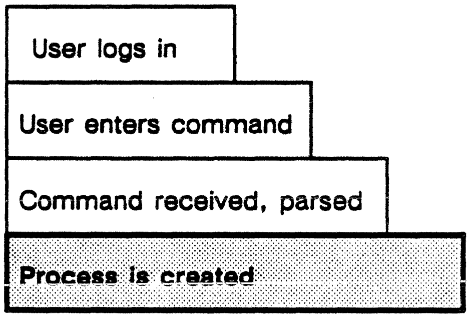
Command Acceptance



The entered command is received by the user environment command shell, which is responsible for managing the processes making up a session when a command is issued. (The command shell is itself a process that was created when the user logged on to the system). The command shell parses the command and searches the user's working directory, the site-identified pool directory, and the public file directory for the executable file corresponding to the command. When it finds the executable file, the command shell calls the Process Create function of process management, passing the name of the file to execute as a process.

Because the command shell initiates processes serially, it waits for termination of the process it is currently initiating before it accepts another command.

Process Creation



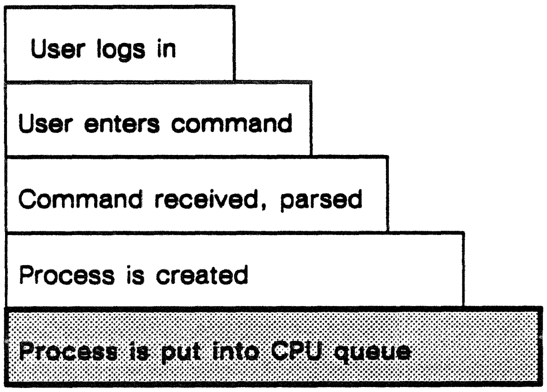
When Process Create is called by the command shell and given the ID of the file to execute, it first assigns a unique process ID to identify that process throughout the system. Inheritance information for the process is retrieved from the command shell, global scheduler, and user management features.

Each process has a paging file, a memory-addressable file where the process' modified pages not in CP memory will be stored during execution. Process Create calls the CP memory manager to create the paging file, and then temporarily stores the retrieved inheritance information there.

The process is added to the shared memory process catalog, which contains a list of all processes in the ETA10 system with their status, their executable file IDs, their paging file IDs, and a list of process clusters to which each process belongs. If the new process is a member of any cluster (a group of processes defined to the system as being related in some way), Process Create adds it to the cluster catalog.

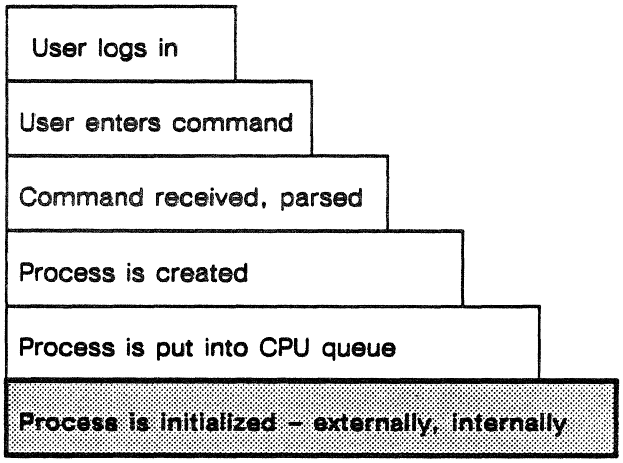
When the process is successfully created, Process Create places it in the *Initializing* state, and calls global scheduler; global scheduler sends it to the processor's Ready queue. If an error occurred and the process was not created, the paging file is destroyed and the error is reported back to the shell.

Global Scheduler Activity



Having been called by Process Create, global scheduler calls the process management server for the central processor on which the process will execute. The process management server begins external initialization of the process.

Process Initialization


User logs in
User enters command
Command received, parsed
Process is created
Process is put into CPU queue
Process is initialized – externally, internally

One process management server (PM server) runs on each central processor in an ETA10 system and handles all processes on that processor. When global scheduler calls the PM server to initialize a new process, external and internal initialization take place.

External Initialization

PM server first does some verification. It checks in the SM process catalog to confirm that the process exists and is in the *Initializing* state. The executable file header is checked for a transfer address, map table, register information, and initialization information. If any of this information is missing, initialization is exited with an error; otherwise, pointers to the various tables are set. The paging file is opened and the saved inheritance information is read.

A series of operations set up the process objects. First, the process image name, version, and transfer address are read from the initialization information in the executable file header. Process management calls system configuration control to get the process attribute block for this process image name from the system configuration table. An entry is made in the CP process catalog for the new process.

Next, PM server calls CP memory manager to initialize the CP memory manager tables necessary during the execution of the process. CP memory manager locks down memory for the register table, process descriptor block (PDB), and process package, and activates them, along with other required CP memory objects.

PM server then calls domain management to initialize the process package, specifying the process image. Domain management selects a process package template corresponding to the process image and copies it to a location specified by process management. The page size is inserted into the package.

CP memory manager is called again to initialize the paging file, the page table, and the process address maps from the executable file map information.

After the process objects are set up, PM server sets up the process descriptor block (PDB), register table, and process package. PM server moves the following information into the PDB: inheritance information from the paging file, process attributes, the original register information, monitor's attributes, some physical addresses and the linked transfer address. The transfer address for the process management startup routine is moved into the process package and the registers moved into the register table.

After the SM process catalog is updated, PM server issues an exit force instruction that moves the process to the *Ready* state and places it in the Ready queue. Process management begins executing in its startup routine and internal initialization begins.

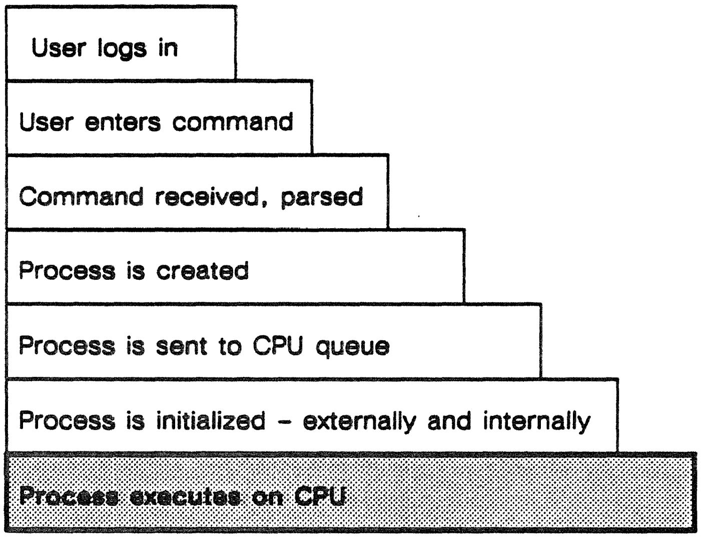
Internal Process Initialization

The internal process initialization cycles through the process initialization routines of the system features needed during process execution. These include shared memory, the logical file system, semaphore and mailbox handlers, communication buffer management, command shell, global scheduler, and user management. The process state is changed to *Ready* in the CP process catalog. All the necessary process structures and operating system features are now in place, and the process can start executing.

Process management calls domain management to set the true transfer address in the application domain registers for the forward domain call to the application. The application domain contains the transfer address of the process that is to execute. The original registers are loaded from the process descriptor block.

A forward domain change instruction is issued to transfer control to the application and the process starts executing at the linked transfer address.

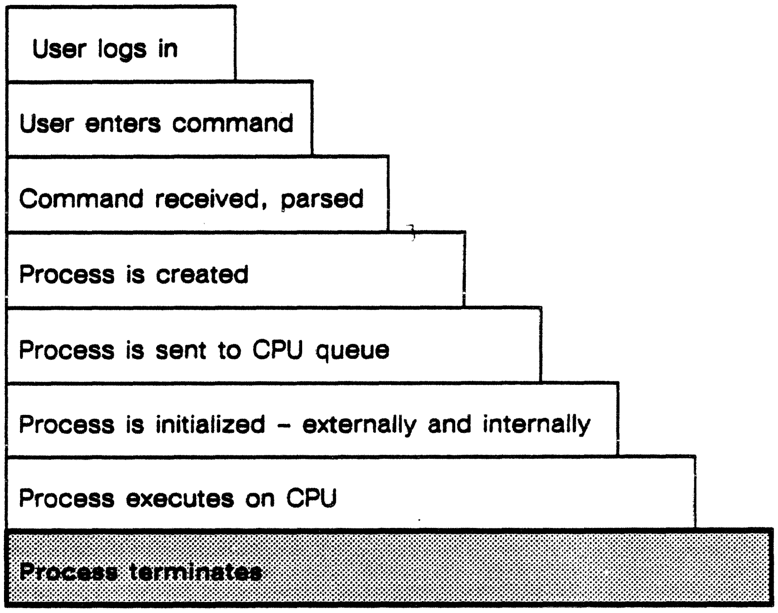
An Executing Process



While executing, the process may perform any number of operations, depending on the application. During its execution, the process might retrieve inheritance information from the process descriptor block. The operating system does resource accounting, such as tracking the time spent by the process in the application domain. The process can communicate with other processes on the central processor using IPC mailboxes.

An executing process may be temporarily placed in the *Blocked* state if a process management wait is requested, if the process is waiting on I/O completion, or while page faulting takes place (described in the following section). At some point, the process completes its operations and terminates.

Process Termination



An executing process terminates *normally* or *abnormally*. Normal termination occurs when the process completes its work and issues a process_terminate call to process management. Process management performs internal and external termination, as described in following paragraphs.

A program can also terminate normally by issuing the SIL call Q5TERM, specifying that a non-fatal or fatal error status be returned. (Refer to PUB_1084, *VSOS Environment Reference Manual: System Interface Library Calls* for details about Q5TERM). Process management performs normal termination as described in following sections.

Abnormal termination occurs when a runtime error happens that cannot be handled in the process. This type of error can occur in three ways: when a process has an access violation after faulting for a page for which it does not have access, when an illegal instruction is referenced, and when a communication buffer access violation occurs. When an access violation happens, pager calls process management to move the process to the Recovery queue to await disposition. Monitor does the same for an illegal instruction and CB access violation.

Process Recovery

The process was placed on the Recovery queue. PM server regularly checks the Termination and Recovery queues. When it finds a process on the Recovery queue, it will attempt to cleanly terminate that process.

PM server copies critical invisible package and register table information into the process' process descriptor block, and temporarily saves in local memory invisible package information about communication buffer base/limit access pair (BLAP) settings and instrumentation counters. It then calls domain management to copy the executing process' process package information to a new process package, and move the saved counters and BLAPs into it.

A flag is set in the process descriptor block to indicate the process is in *Recovery* state.

The transfer address is set to start the process executing in a process management routine, and monitor is requested via an exit force instruction to move the process to the Ready queue to begin executing with the new package and register table.

The reactivated process gets its recovery information from the PDB. It notifies the command shell with information about its termination that the shell will dump to the dayfile or to the terminal of an interactive user. A flag is set for PM server indicating the process has gone through recovery.

PM server is called to perform normal internal and external process termination.

Internal Termination

Process management sets the process state to 'terminating' in both the SM process catalog and the CP process catalog.

The operating system features that were called to execute their process initialization routines during initialization are now called to perform their process termination routines.

The termination reason is saved in the process descriptor block, and monitor is called via an exit force to move the process to the Termination queue, ready for final (external) termination.

As soon as internal termination is complete, the shell can accept a new command.

External Termination

The process can now be deleted from the system. Monitor is called to remove the process from the Termination queue. PM server removes the process from the process catalogs. If the process was a member of any cluster, it is removed from the cluster.

CP memory manager is called to terminate the process by freeing up the memory used by the process, returning any blocks, destroying the paging file, and deactivating the process from the CP memory objects.

Shared memory manager is called to remove references to the terminated process from its tables.

The process is deleted from the CP process catalog.

Page Fault Processing

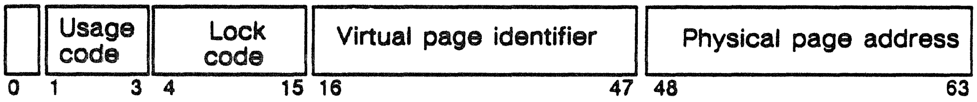
As a process executes, it requires access to read, write, and/or execute pages of system code and program data. This section describes how processes use virtual page addresses to access code and data in the ETA10's virtually addressed system.

Requests for data and code that resides in central processor memory are handled through the virtual-to-physical addressing mechanism. Hardware searches the process's virtual addresses listed in the space table and, when the matching virtual page is found, translate it to an absolute address in central processor memory. When a hardware search of the virtual addresses determines that the requested page is not in central processor memory, software features become involved in processing those page requests. After monitor receives a hardware interrupt signifying the requested page is not in the processor's page table, monitor begins the resolution of the page fault by calling pager. Pager is a feature of central processor memory manager that has the major role in satisfying page faults. Pager requests I/O transfers. to sat... Shared memory manager and its completion server coordinate to complete I/O transfer.

At any one time, I/O transfers for several page faults may be in progress.

Process's Page Table

When a process is created, pager builds a page table for the process that lists all the pages the process may access. The page table is a list of associative words that have this format:



The usage code indicates the page state – if it has been read or modified, for example. One bit in the usage code indicates page size. The lock code indicates to which domain or domains a page belongs. To access a page, a process must hold the key matching a page's lock code. (An attempt to access a page without a matching key results in the hardware issuing an access interrupt, which sends the process to the Recovery queue and termination.) The physical and virtual addresses comprise the rest of the associative word.

The process page tables for all processes active in the processor are stored in processor memory. When a process begins execution, monitor moves a copy of its page table into the processor memory space table. The space table is the link to the associative registers.

Hardware searches and reads the associative words as memory moves from the space table into the registers.

Address Map and Process Block

During process initialization, pager builds an address map and a process block. For a list of virtual addresses, the address map contains a mapping of a virtual address to a file address, and the name and the size of the file each address represents. The address map also has the key for each virtual address that is required to build an associative word for the address. The process block contains data about the process; the times it faults for a page, how many pages are allocated to it, accounting data, and so forth. Pager refers to these tables when a process faults for a page (to determine which page to replace) and when it terminates (to write out modified pages).

Virtual-to-Physical Addressing

Figure 4–9 shows a portion of central processor memory. Page tables for several processes occupy various locations, but the space table always occupies memory starting at address 4000. The page table for process A is shown being moved into the space table. At this time, the first 16 entries of the space table (also the first 16 entries of process A's page table) are moved into the registers.

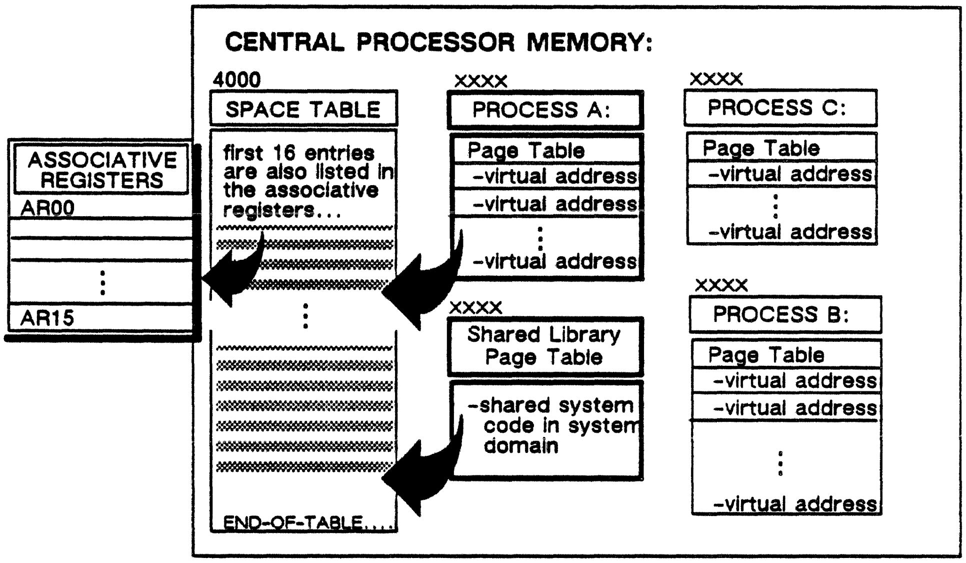


Figure 4–9. Creating a space table for process A.

A copy of the shared library page table is also moved into the space table. This table includes the operating system code, also known as the system domain. In this way, system domain code is shared among processes at the same time it is protected from illegal access or modification by a process. A process has legal access to all the pages in its page table and to all the pages in the shared library for which it holds domain keys. (Keys are explained earlier in the "Underlying Operations" section.)

As the hardware executes instructions, it picks up the virtual page references and reads the associative words in the associative registers, looking for a virtual match. As the matches are found, entries for the 16 most recent matches are maintained in the registers, and the least recently used or unused page entries accumulate at the end of the space table. If the match is not found in the registers, the next entries from the space table are moved, two at a time, into the registers to be read. When the match is found, the translation to a real address in memory is made and the access is granted.

Page Faulting

The hardware ripples through and searches first the associative registers, then the space table entries until it encounters an end-of-table usage code (see figure 4-9). When hardware encounters an end-of-table before it finds a match, it issues an access interrupt. The access interrupt causes an exchange from job mode to monitor mode. When an access interrupt occurs, monitor puts the process index from the process descriptor block into the Page Fault queue and moves the process to the Blocked queue. Monitor stores process A's space table entries back into its page table; maintaining the order of recently used pages, the entries are sorted back into page and shared library tables. Then monitor schedules pager into the Ready queue and begins to execute the next process on the Ready queue.

The Pager

Pager is one of the critical system servers that works with the monitor to manage work in the system; it is called by monitor to process page faults. When page faults require I/O transfers to shared memory, pager calls shared memory manager to manage these transfers. Pager is a central processor-resident process that executes in job mode.

When active, the pager process follows a loop of these activities:

- processing central processor memory requests for process objects
- processing page faults
- processing page fault I/O transfer completion

Processing Memory Requests

Starting its loop, pager checks the Page Fault queue. It removes the process from the queue and processes the page fault.

Processing Page Faults

First, pager takes the faulted-for virtual page address from the process's invisible package. Using this virtual address, pager searches the address map to obtain the logical file address of the file, and the length of the file. If an address map does not hold the faulted-for address, then pager declares a fault for an undefined address and the process is moved to the Recovery queue.

Pager locks down sufficient pages in memory to hold the page it is fetching so that they cannot be allocated to another process. Then pager builds a new page table entry for these locked-down pages using virtual page addresses from the invisible package as well as the key and page size from the address map. Pager calls shared memory manager and requests that pages from the file be read into processor memory. This request marks the end of pager's activity in actually processing page faults.

Processing I/O Completions

Pager moves into the third activity of its processing loop, and calls shared memory manager to see if the requested memory transfer has completed. If the shared memory transfer has completed, pager unlocks the newly-written page in processor memory and calls process manager to request that monitor move the process to the Ready queue.

No Completion If the transfer has not completed, there are two reasons. One is that the transfer from shared memory is too lengthy to be completed in the time pager allots; this time is usually sufficient for an average transfer from shared memory. The second reason may be that the file resides on a disk, and is being moved into the system via an I/O transfer. At any rate, when pager receives a "no completion", it ceases activity (blocks itself) and relinquishes the central processor.

When pager receives a "no completion" and gives up the central processor, process A remains blocked. Monitor takes over and looks in the Ready queue for another process to start or a process to return to execution. Monitor executes for an exchange to job mode, and switches the next process in the Ready queue into execution.

Completion After the shared memory transfer completes, pager begins its loop again and moves to unblock process A. There may be more than one page fault in the queue when pager is started by the monitor. There may be I/O completions returned from shared memory manager for page faults processed earlier. If there are, pager processes the completions one at a time, unlocking the newly-written pages in memory and making calls that result in monitor moving blocked processes to the Ready queue.

More on Shared Memory Manager

When pager requests a file transfer from shared memory, it provides shared memory manager with the length of the file and the logical file addresses associated with the requested page. Shared memory manager searches its file object table, a list of the current file addresses residing in shared memory for each file. When a requested file address is found in this table, shared memory manager builds the transfer request blocks (TRB) needed for the transfer, links them into the TRB queue, and starts the memory-to-memory transfer to central processor memory. Once the transfer is started, shared memory manager returns an operation identifier to pager that describes the specific transfer. Pager refers to the transfer's operation identifier when inquiring about the status of any transfer.

When shared memory manager does not find the requested logical file address in the file object table, the file resides on disk and must be moved into shared memory by an I/O transfer.

I/O Transfers to Shared Memory

When a requested file address is not in the file object table, shared memory manager calls the logical file system to translate the logical file address to a physical file address. An appropriate number of blocks in shared memory are allocated by the memory manager to hold the transfer. The logical file system searches its physical file ID map table to obtain the physical file identifiers (PFIDs). The physical file ID map table holds the PFID for each logical file, the length of the file, and the disk device in which the file resides. Shared memory manager sends the physical address to the specific disk physical file system managing the indicated disk, and requests a read of the file into the allocated shared memory blocks.

After setting up the transfer request, shared memory manager returns the operation identifier for the transfer to pager; this return indicates to pager that it should check for completion of the transfer. During its next loop, pager checks the status of any I/O completions. When the disk transfer to shared memory is finished, the completion server is activated by a remote procedure call from the I/O unit.

The Completion Server

The completion server is another critical system server that runs in job mode. Its purpose is to "complete" the I/O transfer from shared memory to central processor memory.

On the indicated disk channel processor, the disk physical file system sets up and manages the disk transfer to shared memory, and then sends a message to the completion server via a remote procedure call. The completion server builds the transfer request blocks required for central processor-shared memory transfers, and begins the transfer to central processor memory. When the transfer is complete, the completion server passes a message to monitor to unblock the pager process, and gives up the central processor. The server will be restarted when a disk requires transfer request blocks to complete another I/O transfer.

Writes Before Reads

Shared memory is deallocated and reallocated upon demand, according to a least-recently-used algorithm. As a result, shared memory is usually kept fully utilized. Flies written out from central processor memory remain in shared memory until their allocated space is needed for another operation. Then the files are written out to disk or to wherever the user has specified.

Even if the least recently used block of memory has been written to or otherwise modified by a write from central processor memory, it is still allocated to the new I/O transfer. When modified areas of memory must be reallocated, shared memory manager immediately write the modified blocks out to disk before before the requested read may take place. Hence the occurrence of writes before reads.

Section 4: The Logical-File System

This part examines the components and implementation of the disk file system. It consists of three sections:

- File directory/catalog manager
- File support module
- Record manager

File Directory/Catalog Manager

The file directory/catalog manager (FDCM) is part of the logical-file system (LFS) that provides for the controlled creation and destruction of logical files and directories. Additionally, FDCM maintains the attributes of files and directories and makes them available on request to other parts of the LFS, other system features, and users.

File System Structure

Directories are used to group other directories and logical files. *Logical Files* are used to store data.

The attributes of a directory are maintained in *directory nodes* and *file nodes*. A directory and its directory node are actually the same thing. Conversely, a file node is an identifier for a logical file.

There is a third type of node called an *alias*. An alias node is an indirect identifier of either a directory or file node. Alias, directory, and file nodes are identified by pathnames.

Each node is a member of the *directory set*. The directory set elements are organized as a true oriented tree structure. The *system root* is the one directory which gives the tree an orientation. It is required to exist at all times. All nodes are linked either directly or indirectly to the system root directory.

Pathnames

Each node has a *nodename* which is regarded as an attribute of the node. Moreover, each node is uniquely identified by an ordered concatenation of nodenames into a *pathname*. The only way in which a node may be identified is by its pathname or by a replacement identifier provided by the file directory/catalog manager after the initial provision of the pathname.

A pathname directly identifies a node. It only indirectly identifies a file system object (directory or logical file). Users and the system both create and destroy these objects through their identifiers rather than directly.

For each process there is a *relative root* and a *working root*. A path name is actually an ordered pair (A,B) where *B* is a list of nodenames and *A* is an identifier of the point of origin of the path name, one of the following:

- System root
- Relative root
- Working root

Alias Pathnames

If the owner of a directory or file has granted the necessary access permission, then a directory or file may be aliased by more than one pathname. That is, it is possible for two different pathnames to identify the same object. All the pathnames of an object are equivalent and have the same attributes.

Global File IDs

A global file ID is a unique identifier of either a directory or logical file.

These two types of objects are included in the scope of global file IDs to support the concept that each of them may be identified as a file object. In addition, there is a performance advantage in the use of a global file ID in that it not only uniquely determines the object, but it also points directly at the object's attributes (i.e. a global file ID is a pointer to the node which describes the object).

Relative Root and Working Root

Pathnames may be specified to originate from various points within the tree's hierarchy. Two of these points are called *relative root* and *working root*. The file system maintains these points on a process basis and provides operations by which users may cause them to be set and listed. Though the names suggest their intended usage, it is entirely the responsibility of the file system user to manage them as appropriate to their own environment.

Directory and File Ownership

Each directory and file has an associated username which is its owner. Other than being an accounting system charging identifier, ownership has an explicit definition. That is, the owner of a directory or file has the implicit right to grant him or herself *access permission* to the object even if he or she has previously been explicitly denied such access permission.

The initial owner of a directory or file is its creator. The owner of an alias pathname is the owner of the object which it aliases. If an object is given to a new user, the recipient username becomes the owner.

Controlling Access to Directories and Files

Each file system object has an associated list of access permissions. All references to a file system object for any purpose is controlled by the associated access permissions.

Permanency of Files

Each pathname is permanent; it can be destroyed only by explicit request. It is the responsibility of the user environment to support other types of path name durations, such as temporary or scratch files.

It is not possible to destroy the last path to a non-empty directory. To delete a directory, all the objects within the directory must first be deleted.

Device Classes

Disk devices are grouped into logical device classes. There are two classes:

- Pre-defined
- Site administrator-defined

Assignment of devices to classes, and maintenance of the device configuration are the responsibility of the disk physical-file system and system configuration control. Each disk file is a member of one *and only one* device class.

The File Catalog

The file catalog is not externalized to users of the file system, but is externalized to the file support module.

The file catalog is a collection of file attribute descriptors. Each attribute descriptor is referred to as a catalog entry. There is a single catalog.

Figure 4–10 illustrates how the global file IDs identify system file objects. It depicts the relationship between the:

- File directory set
- File directories
- File nodes
- File catalog
- File catalog entries
- Logical files

As shown *GFID1* points to directory *DIRa* which has two sub–directories, *DIRb* and *DIRc*. *GFID2* points to *DIRb* which has no file nodes. *DIRc* contains three file nodes, each of which has a corresponding catalog entry. *GFID3* points to file node 1. Global file IDs are not shown for *DIRc*, and file nodes 2 and 3.

Logical Disk Files

The logical file system provides logical files. A logical file is an object by which the ETA10 system and its users store and transmit data. Logical files reside on disk files, which save logically grouped data on non–removable rotating mass storage device(s). A disk file may be a:

- User data file
- Executable file
- Batch input file
- Output file

A disk file is a high–speed, direct or sequential access, mass storage file. A disk file may be shared between currently executing processes.

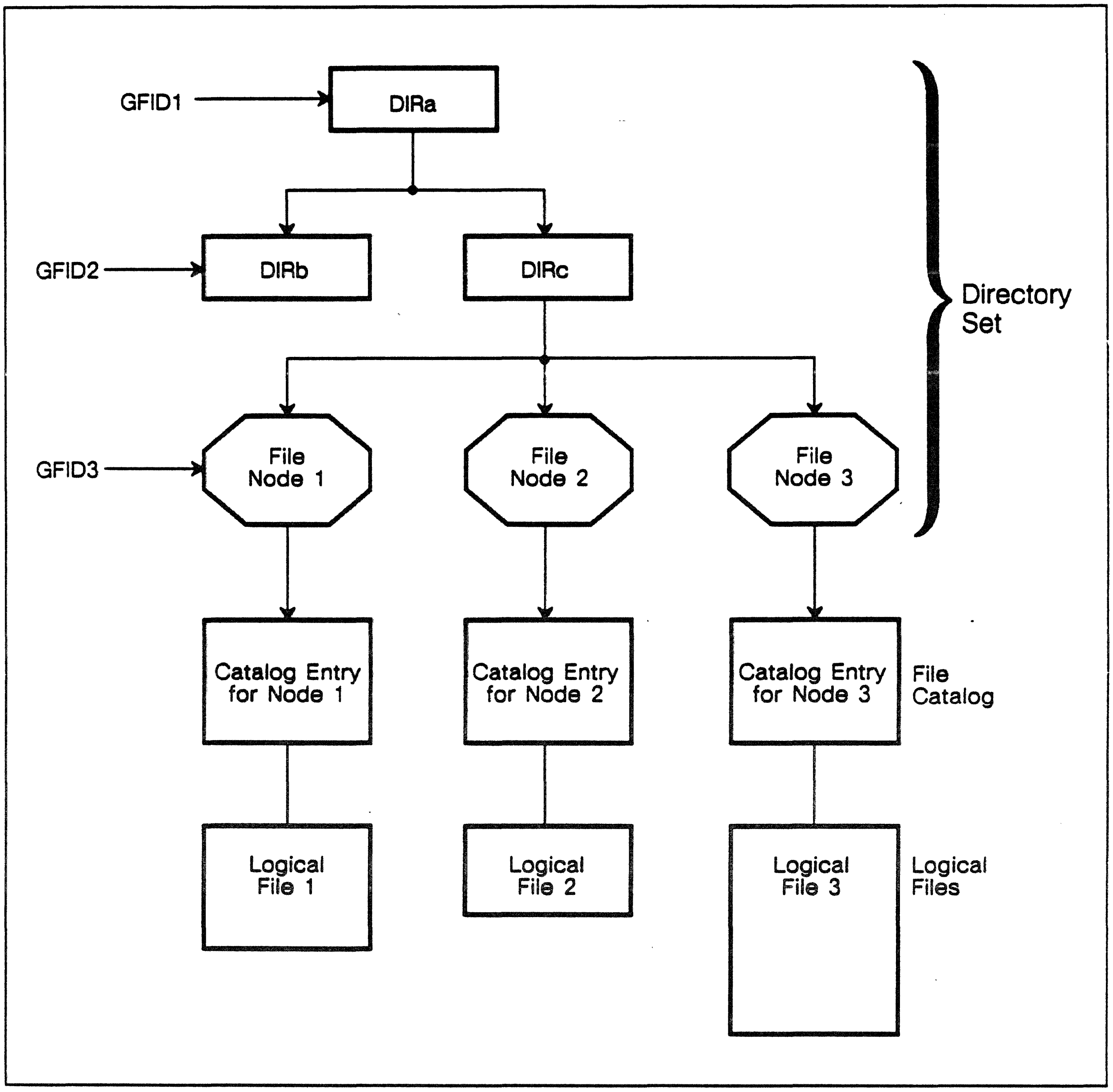


Figure 4–10. Global file IDs and system objects.

The Directory Set

The directory set *directly* describes hierarchical relationships between files and *indirectly* describes them through the file catalog. For each file there is *at least one* file node linked to a directory. Each directory (except for the system root) is linked to one *and only one* other directory.

The complete set of file nodes and directories compose the *directory set*. There are fixed and user defined directories and file nodes in the directory set. The fixed directories and file nodes always exist. The user defined directories and file nodes may be defined or destroyed by users (subject to proper access permission) during normal processing.

Directories

A directory is the "branch" element of the directory set. It is the basic mechanism by which files are logically grouped, and as such may be thought of as a holding place for file nodes. Each directory has the attributes:

- Directory name
- Set of file nodes linked to it
- Parent directory (except for the root directory)
- Set of subdirectories
- Set of access permissions defining directory access
- User name defining the directory owner
- An account ID
- A group ID

File Nodes

A file node is the "leaf" element of the directory set. It acts as an indirect file descriptor within the file hierarchy, and has the attributes:

- File name
- Parent directory
- Set of access permissions defining access to the file node
- Associated catalog entry
- Device type
- Username defining file owner
- Account ID
- Group ID
- Project ID

Directory and File Node Linkage

A directory link is the basic mechanism by which the tree structure of the file system is realized. It is a hierarchical association between directories and file nodes. The linking rules are:

1. The system root directory is not linked to any directory.
2. No directory is linked to itself either directly or indirectly.
3. Each directory and file node (other than the system root directory) is linked to one and only one other directory. A directory or file has only one parent.
4. Any number of directories and file nodes may be linked to a particular directory. A directory may have many subdirectories or files.
5. Nothing may be linked to a file node.
6. The names of all directories and file nodes linked to any particular directory are unique.

The consequences of the linking rules are:

- The linking of directories is unidirectional. If B is linked to A, then A is not linked to B, either directly or indirectly.
- Links establish a hierarchical relationship between all directories. If B is linked to A, then B is an offspring of A and A is the parent of B.
- A path to directory or file X may be defined as an n–tuple of nodes {N(1), ... ,N(n),X} where:
    - N(1) is the system root
    - N(1) through N(n) are directories where N(i+1) is linked to N(i)
    - X is linked to N(n)

Access Permissions

The file system is essentially a closed environment. No access to a file object is permitted without explicitly granted access permission, except that the owner of the object has the implicit right to grant his or herself explicit control access permission.

Each directory and file has an associated set (possibly null) of access permissions which may be altered. An access permission is an ordered 3–tuple of parameters which can be divided into two parts:

- The first two parameters together compose the state ID
  - Username
  - Group ID
- The last parameter is the permission types valid for that state

Each of the parameters of an access permission state ID may take values that contain "wild cards" in order to minimize the number of access permissions required to grant the permission desired.

When a caller attempts to reference a pathname, it does so with an environmentally determined access request state. The access request state is also a 3-tuple:

- The first two parameters together compose the caller's state ID:
  - Caller's username
  - Caller's active group ID
- The last parameter contains the requested access types

In order for the requested access to be granted, the access request state must be obtainable from at least one access permission state ID, and the access requested must be among the access types granted.

Access permissions are order independent. That is, the file directory/catalog manager attempts to derive the access request state from each of the access permissions until it is either successful or the set of access permissions is exhausted.

If the request state is derivable, it becomes the *effective access permission*. The effective access permission has effect beyond the particular access request only in the file open operation.

The access permission types that may be granted to directories include:

- **Alias** permission to create a new pathname (alias) to the directory.
- **Control** permission to change the access permission list and permission to see all existing access permissions.
- **Create** permission to link pathnames to this directory.
- **Delete** permission to destroy the pathname.
- **Give** permission to give the pathname to another user and/or group.
- **Read** permission to query all the attributes of the directory except the access permissions. Only the access permissions granted to the caller with Read permission may be read.

- **Search** permission to search below a directory for a file node.

When a directory is created without the provision of an access permission, a default access permission is automatically given to it. It grants the owner all access permission types under all requested states.

The access permission types that may be granted to files include:

- **Alias** permission to create a new pathname (alias) to the file.
- **Append** permission to append data to the current end-of-file.
- **Control** permission to change the access permission list and permission to see all access permissions.
- **Delete** permission to destroy a file.
- **Execute** permission to execute the file as a process.
- **Give** permission to give the file to another user and/or group.
- **Modify** permission to modify the current contents of the file.
- **Read** permission to read the file.
- **Write** permission to write to the file and permission to alter the file attributes.

A user with any access permission to a file may list all of that file's attributes except the access permission list. A user with any access permission except Control permission may see all access permissions that have been granted on the file.

When a file is created without provision of an access permission, a first access permission is automatically given to the file. It grants the owner all access permissions under all request states.

There are three special characters which may be part of an access permission state ID:

*
?
-

The rules regarding these special characters are:

- The * matches any sequence of characters including the null character.
- The ? matches any single (one and only one) character.
- The * and ? may be used as a substring within a state ID parameter.

- The * and ? may also be used as the entire entry for a state ID parameter, in which case all strings will match the * and any single valid character will match the ?.
- The - is used in the user name field to match all valid usernames except the directory or file owner's name.
- If - is used in the group ID field, it will match any group ID except the directory or file's group ID.
- If - is used in conjunction with any other character in a field, it is not interpreted as a special character. In this case it will match the username or group ID which has a - in the proper place.
- The - is legal only within the username and group ID fields.

Relative Root and Working Root Directories

Each pathname is relative to some point of origin within the directory set. For each process, the file directory/catalog manager maintains two directories which are alterable points of origin for pathnames:

- The relative root
- The working root

These names are mnemonic only. Users may manipulate them at will within the constraints imposed by the access permissions.

Each directory and file node in a system has an associated name which is referred to as its *nodename*. A nodename may contain 1 to 31 characters. The characters of a nodename may resolve to any integer value from 0 to 255 (decimal). The nodename consists of a 31 byte string and a length which specifies the number of significant characters in the nodename.

A pathname is completely specified by a pathname origin and by a list of nodenames. The pathname origin can take on one of the values:

- System root
- Relative root
- Working root

The nodename list is considered to be exhausted when one of the nodenames in the list has a length of zero or the upper bound of the list is reached.

Files have many attributes. Not all files have the same kind of attributes:

- Some files have attributes that are "dormant;" they exist but have no meaning in light of other "active" attributes.
- Some attributes are "intrinsic;" they are necessary in the definition of the file system or are intrinsic to the maintenance of the file system. Intrinsic attributes include:
    - **Physical-File IDs:** An ordered list of physical-file IDs along with the lengths of each physical file. This attribute is applicable only to disk files.
    - **Owner Username:** The user who has the implicit right to grant him or herself Control permission. Otherwise the user who is denied a grant when the username field in an access permission is –.
    - **Device Class Identifier**: All physical files in the file are allocated on logical devices that are a member of the specified device class. This attribute is applicable only to disk files.

Other file attributes are extrinsic in that they exist to support users of the file system. While the file directory/catalog manager is responsible for the maintenance of both intrinsic and extrinsic attributes of inactive files, it should be realized that as regards to extrinsic attributes, the file directory/catalog manager is merely a service feature in that it maintains and provides these attributes on demand, but does not itself use or interpret them.

It is the responsibility of features which use the file directory/catalog manager to specify file attributes which are needed by them. Among the extrinsic attributes are:

- Attributes which enable a file to conform to a specific user environment, where the use of a general file system precludes it. These attributes are contained in the user environment dependency block.
- Record manager attributes that define and control record structures on a file. They are as follows:
    - Maximum record length
    - Record type
    - Padding character
    - File pattern
    - End-of-information
    - Highest byte written

- Accounting/archiving management attributes that include such attributes as:

  - Creation date and time
  - Reference count
  - Account ID
  - Project ID
  - Last open data and time
  - Retention period
  - System file flag

- Resource limitation attributes include such attributes as:

  - Maximum file length
  - Device class
  - Current file length

As previously stated, disk devices are grouped into logical device classes which are predefined, or system defined, as well as site administrator defined. Each username has a list of device classes that it is permitted to use. These are granted to users by the site administrator. User validation provides this facility and makes the appropriate device classes available to the file directory/catalog manager when necessary.

When creating a disk file, the caller may specify a device class on which the file is to reside. The username under which the caller is executing must be permitted access to the specified device class. The system defined device classes are:

- Class which permits primary system files on the device

- Class which permits backup system files on the device

It is the responsibility of System Configuration Control (SCC) to see that there are at all times sufficient logical devices in these device classes to enable the system to run. Moreover, it is the responsibility of system configuration control to see that there is no logical device which is a member of both these device classes.

Shared Information

The physical-file ID map is provided by the file directory/catalog manager and initialized by the file support module during the initialize file system operation. It contains the physical-file IDs of open disk files.

Entries are made in the map by the file directory/catalog manager when a file is opened and deleted when a file is closed. The map is altered by the extension operation. The information is used to resolve logical-file addressing and is made available to the SM manager through the necessary interfaces.

File Directory/Catalog Manager Physical Structure

The physical structure of file directory/catalog manager consists of ten files, two tables, and the routines which control them. The ten files are:

- File system initialization primary and backup files
- Directory set primary and backup files
- Disk catalog primary and backup files
- Access permission primary and backup files
- Physical file ID primary and backup files

The two tables are:

- Physical file ID map
- Flush logical file system table

The file directory/catalog manager files and tables are illustrated in figure 4–11.

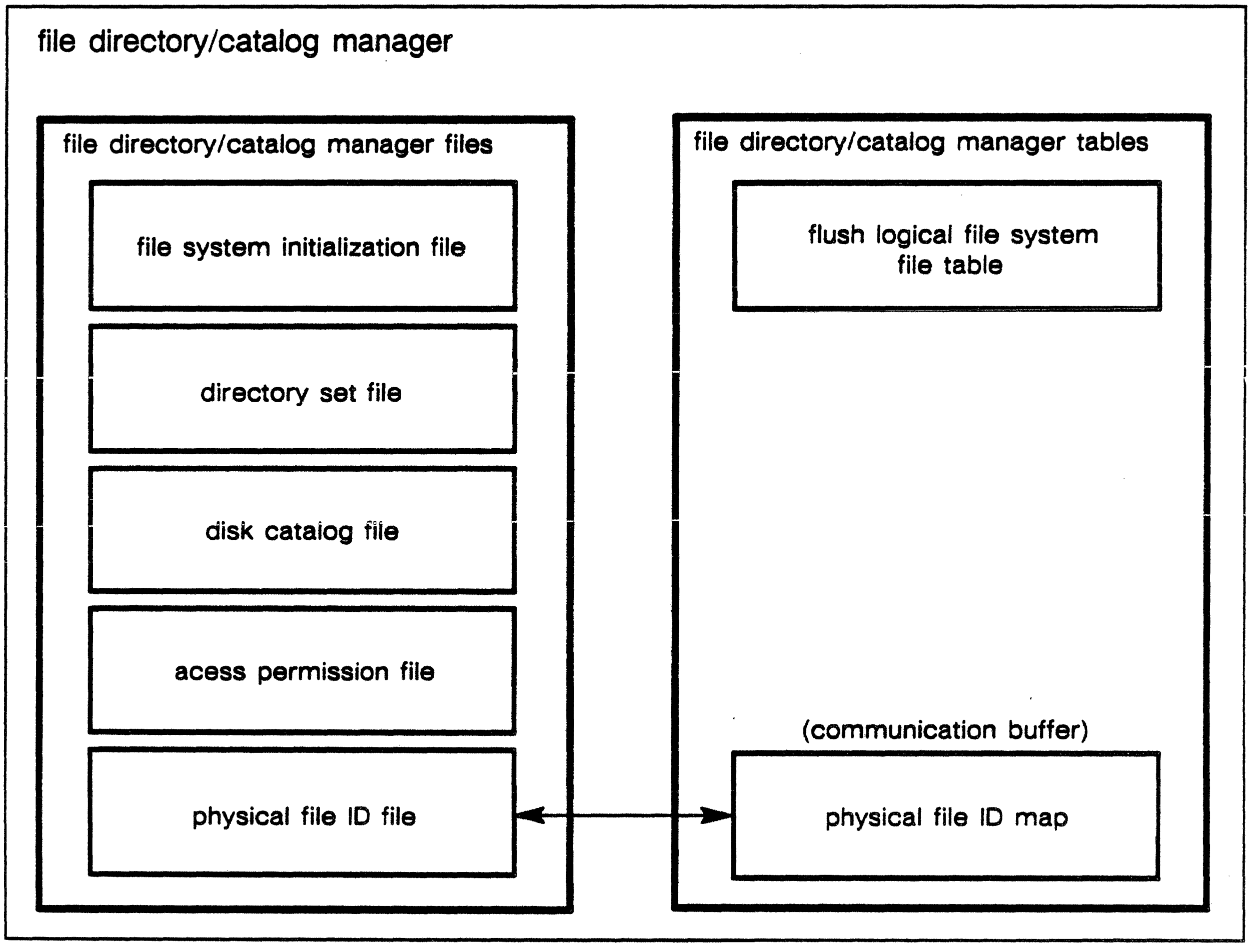

file directory/catalog manager
file directory/catalog manager files
file system initialization file
directory set file
disk catalog file
acess permission file
physical file ID file
file directory/catalog manager tables
flush logical file system file table
(communication buffer)
physical file ID map

Figure 4–11. The file directory/catalog manager files and tables.

File System Initialization File

The file system initialization file is created and initialized by the **install file system** procedure. Installation is the **only** time this file is modified. It contains a file system version ID and physical file IDs for the primary and backup disk file catalogs. Initialization of the file system requires that this file exist.

Directory Set File

The directory set file is created and initialized at file system installation and opened by file system initialization. It contains nodes for all defined files and directories in the system. The rest of the file system is dependent on this file for the representation of the directory tree.

The directory set file is composed of F format records. Each of these records represents describes a node in the directory tree.

Catalog File

The catalog file is created and initialized at file system initialization and opened by file system initialization. It contains the file attributes of all the files defined in the system. The rest of the file system depends on this file for the storage of file attributes for all files defined in the system.

The catalog file is composed of F format records. Each of these records is a catalog image. There exists one catalog image (one record) for each file defined.

Access Permission File

The access permission file is created and initialized at file system installation and opened by file system initialization. It contains all the access permissions that overflow from the directory and file nodes in the system. The rest of the file system depends on this file to verify access to files and directories.

Each directory and file node contains at most three access permissions. Subsequent access permissions are linked from the node into a linked list of access permissions in the access permission file. If one of the access permissions in the node is deleted, the next access permission in its linked list is put onto the node, and its position in the access permission file is returned to the free list.

The access permission file is composed of F format records, each of which is an access entry. Each access entry contains:

- An access permission
- The record number of the next access permission in the access permissions file
- The record number of the current record

Physical File ID File

The physical file ID file is created and initialized at file system installation and opened by file system initialization. It contains entries for all physical files that are associated with logical files in the system.

Each disk file catalog contains at most one physical file ID entry. Subsequent physical file IDs are linked from the catalog into a linked list of entries in the physical file ID file.

New physical file IDs for a file are linked at the end of the file ID list. If a physical file ID is deleted from a files list, its record is placed in the free list in the physical file ID file.

The physical file ID file is composed of F format records, each of which is one physical file ID entry. Each entry contains:

- A physical file ID
- The record number of the current record
- The record number of the catalog image this physical file ID entry is allocated for
- The length of the disk physical file identified by the physical file ID
- The record number of the next physical file ID entry
- The record number of the previous physical file ID entry

Physical File ID Map

The physical file ID map contains lists of physical file IDs for all logical files open in the system. The shared memory manager uses this table to determine the physical files associated with a logical file during I/O.

The physical file ID map is a communication buffer table in which the physical file IDs of all open files are stored. The file support module requests that file directory/catalog manager put the physical file ID for a file into the physical file ID map when a disk file is opened. The file directory/catalog manager sets up a linked list of entries containing the EOS file's physical file IDs and returns the index of the first entry to the file support module.

The shared memory manager requests the file directory/catalog manager to use the information in the physical file ID map to translate a logical file address range into a physical address and range.

The fields included in a physical file ID map entry are:

- The physical file ID for a disk physical file
- A link to the next physical file ID map entry for an open file
- A link to the previous physical file ID map entry for an open file
- The length of the disk physical file identified by the physical file ID in this entry

Flush Logical File System File Table

The flush logical file system file table is a boolean array that indicates whether or not the file system files need to be flushed at the conclusion of external procedures.

The flush logical file system file table is subscripted by the file system local IDs. The boolean corresponding to the proper file is set to TRUE whenever a write to a file system file occurs. At the end of the external procedures a call is made to a procedure which uses this array to flush all logical file system files that have the value set to TRUE.

The Directory Tree

The entire structure of the directory tree is composed of nodes and links between nodes. There are three types of nodes:

- Directory
- File
- Alias

A link is a pointer from one node to another, or from a node to a catalog entry. All links consist of a record number relative to a particular file.

Internal Representation of the Directory Tree

A directory structure segment, as users logically see it, is illustrated in figure 4–12.

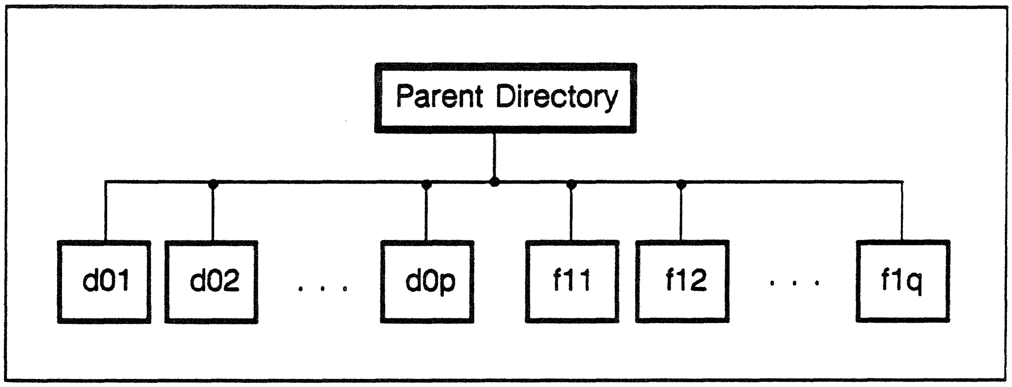


Figure 4–12. A directory structure segment where the d0*i* represent directories and the f1*j* represent files.

To the file directory/catalog manager, the nodes corresponding to f11 . . . f1q and directories d01 . . . d0p are all members of a *sibling set.*

A sibling set is implemented as a binary tree where comparison on the node name is used as a binary sort algorithm.

A sibling set is illustrated in figure 4–13.

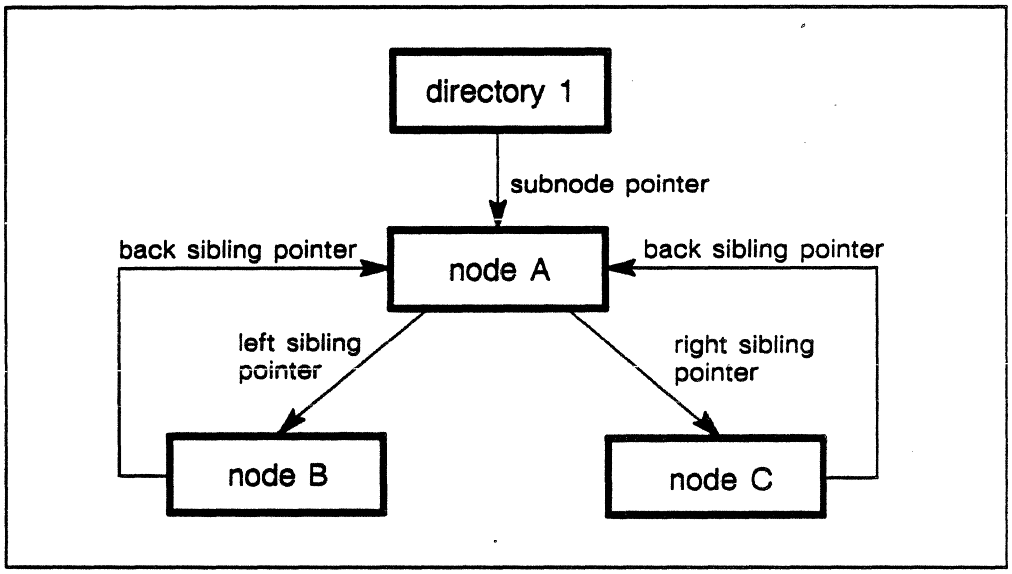


Figure 4–13. Internal structure of a directory set showing the node pointers.

The parent directory (**directory 1**) of a sibling set has a subnode which points to one member of the sibling set, (**node A**). The sibling set consists of nodes **A**, **B**, and **C**, and each member is a subnode of **directory 1**.

All the members of the sibling set have *left sibling* and *right sibling* pointers to the roots of their left and right sibling trees. **Node A** has a left sibling pointer that points to **node B**; and a right sibling pointer that points to **node C**. In turn, nodes **B** and **C** may have left and right sibling pointers that are null.

Note that **node A** has a back sibling pointer whose value is null since it is the root of a sibling set. If any one of the nodes **A**, **B**, or **C** was a directory node, it too would have a subnode pointer to its subnode tree.

Alias Path Name Implementation

An alias path name is implemented as an alias node type in the directory. Each alias node points to another node, an *object node*, which is either a directory or file node.

All attributes associated with an alias path name, besides the node name, are derived from the object node. The way in which alias nodes are linked together is illustrated in figure 4–14.

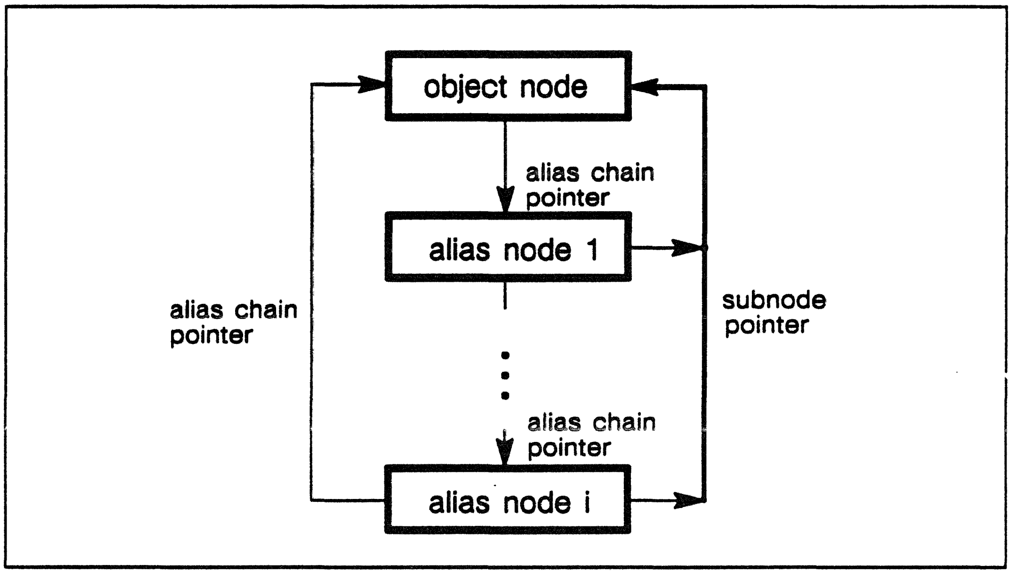


Figure 4–14. An alias chain showing the node pointers.

The subnode field in each alias node is used to point to the object node. Each node in figure 4–14 has an *alias chain pointer* used to form a circular list of nodes known as the *alias chain*. The alias chain field is used to point to the next alias in the chain of nodes, all of which are associated with the file system object.

The alias chain makes it possible for another directory to *adopt* the object node in place of an alias node in the event that the original path name is destroyed.

Global File ID Field Descriptions

The file system constructs global file IDs by concatenating three fields which together uniquely identify a file or directory node:

- The first field is a device type which specifies what type of object the global file ID refers to. It can take the value directory or disk,.
- The second field is the record number. It contains the record number of the node in the directory set file. This record number guarantees the uniqueness of the global file ID within the directory set.
- The last field is the use ID. It gets initialized when a record in the directory set is used for a file or directory node. It gets incremented when the node is reused, after the destruction of the previous object has occurred. It is a pseudo-random number that is incremented by a large prime number after the

destruction of the node. The use ID guarantees the uniqueness of the global file ID over a long period of time.

Configuration of File System Files

For all file system files, the first record is a free list header record. It contains a pointer to the file's free record list.

The free record list initially points to the file's last record. But, as records are deleted from the files (i.e. directories and files are destroyed), entries are inserted onto the free list. The free records are linked such that they are maintained in a sequentially ordered linked list by record number.

The initial configuration of the directory set file is:

Record 01 – Free list header node
Record 02 – System root directory node
Record 03 – Logical file system directory node
Record 04 – Primary directory node for primary logical file system files
Record 05 – Primary initialization file node
Record 06 – Primary directory set node
Record 07 – Primary physical file ID file node
Record 08 – Primary access permission file node
Record 09 – Primary disk file node
Record 10 – Backup directory node for backup logical file system files
Record 11 – Backup initialization file node
Record 12 – Backup directory set node
Record 13 – Backup physical file ID file node
Record 14 – Backup access permission file node
Record 15 – Backup disk catalog file node
Record 16 – Free list tail

The initial configuration of the disk catalog file is:

Record 01 – Free list header
Record 02 – Primary directory set file catalog
Record 03 – Primary access permission file catalog
Record 04 – Primary VSOS catalog file catalog
Record 05 – Primary physical file ID file catalog
Record 06 – Backup directory set file catalog
Record 07 – Backup access permission file catalog
Record 08 – Backup VSOS catalog file catalog
Record 09 – Backup physical file ID file catalog
Record 10 – Primary initialization file catalog
Record 11 – Backup initialization file catalog
Record 12 – Free list tail

Synchronization on Logical File System Files

Synchronization of file system files is done through interprocess communication defined semaphores. An extended semaphore of the reader's or writer's style must be created at file system initialization (one for each pair of files: primary and backup), and activated by the file system process initialization.

Using the style semaphore permits multiple readers and no writers, or only one writer and no readers for each primary and backup file pair. To avoid the deadlock caused by the locking sequence of file directory/catalog manager routines, locking of file system files must be done in the same sequential order by all processes:

1. Directory set
2. Access permission file
3. Disk Catalog file
4. Physical file ID file

Note that this only avoids deadlock situations where the semaphores involved are file directory/catalog manager created. The file directory/catalog manager keeps global (to itself) a table which records the current locked state for each semaphore, for each process.

This table is used to enforce the sequence of locking on file system files.

Synchronization on File Directory/Catalog Manager Tables

The file directory/catalog manager must synchronize the physical file ID map table. Synchronization of this table has two parts:

- First the physical file ID map's free list is synchronized by an interprocess communication defined simple semaphore. Since the free list will only be accessed when it is to be altered, there is no need for a style semaphore.
- Secondly, the entries in the physical file ID map associated with a single file will be synchronized by a bit branch table. This table has an entry for each possible open file table (OFT) entry.

  When the physical file ID map is accessed, it is always through the open file table which contains a pointer to, or an index of, the first physical file ID map entry in the file's physical file ID chain. Because of this, the physical file ID chain for a file can be locked by doing a communication buffer bit branch and swap operation on the entry in the bit branch table that is associated with the file's open file table index.

Splitting the synchronization in this way allows multiple processes to access the physical file ID map simultaneously, as long as they are not accessing the physical file ID chain of the same file.

File Directory/Catalog Manager Routines

The file directory/catalog manager routines are as follows:

- **Add Access Permission to File Object** adds an access permission to either a directory or file's access permissions list.
- **Alter Directory or File Attributes** alters the attributes for a directory or file. The specific attributes which may be altered include all attributes necessary to allow user control over files and directories.
- **Create Alias Pathname** creates a new pathname for an existing pathname.
- **Create Directory** creates a new directory and links it into the directory set. Access permissions to the new directory are initialized. The owner username and group ID of the new directory are set.
- **Create Disk File** creates a pathname to a previously nonexistent disk file:
  - A catalog entry and a file node are created and initialized.
  - The file node is linked to a directory.

  The user may request an initial file length in bytes. Additionally:
  - Access permissions to the new file are initialized.
  - The owner's username and the group ID of the new file are set.
  - The requested file space is allocated.
- **Create Logical File** creates a pathname to a previously non-existent logical file:
  - A catalog entry and a file node are created and initialized.
  - Access permissions to the new file are initialized.
  - The owner's username and the group ID of the new file are set.

- **Delete Access Permission from File Object** deletes a specified access permission from a directory or file's access permission list. The deletion of an access permission for a file is not effective for users that currently have the file open, until they close it.

- **Destroy Pathname** destroys a pathname:

  - If the object identified by the pathname has at least one other path to it, only the specified pathname is destroyed. The object remains intact.

  - If the object is a file which is currently open to some process, it is marked to be destroyed and becomes an unnamed scratch file. It is destroyed when the file is last closed.

- **Extend File** adds a specified number of bytes to a disk file by requesting the disk physical-file system to extend the file's last existing physical file and/or create new physical files which are associated with the logical file.

- **Give File Object to New Owner** changes the owner username and/or owner group ID as requested in a directory or file.

- **List File Object Access Permissions** returns a list of access permissions to a directory or file. A "snapshot" of access permissions to the specified object is taken on the first call for that object. The operation may then be called repetitively to return all access permissions.

- **List File Object Attributes** returns the attributes of a directory or file. A "snapshot" of global file IDs and nodenames of all nodes linked to the specified object is taken on the first call if the object is a directory.

- **List Root Global File ID** lists the global file ID for the system root, relative root, or the working root as specified by an input ordinal.

- **Rename Pathname** changes the name of a directory, file, or alias, and properly relinks it in the directory set.

- **Reduce File** reduces the length of a file to or by a specified amount,

- **Set Root Directory** sets the relative or working root for a process as specified by an input ordinal and an input global file ID.

- **Translate Path Name to Global File ID** returns the global file ID of a pathname.

- **Translate Global File ID to Pathname** returns the pathname for a global file ID.

File Support Module

The file support module (FSM) is a set of operations which together with the record manager and the file directory/catalog manager, comprise the complete Logical File Subsystem (LFS). The file support module performs those functions which make the file system and files available to applications programmers, libraries, and other system features such as the record manager and the file directory/catalog manager.

File System Installation and Initialization

All information needed for the initialization of the file system is contained in a single primary file system initialization file. A duplicate of this file is kept in a backup file system initialization file.

Each of these files has exactly one disk physical file, a *pfile*, associated with it. The physical file IDs of these two files are maintained by system configuration control.

Cold Starting the File System

The cold start system initialization process (CSIP) calls a file support module provided operation called the *LFS Cold Start Operation*. There are no input parameters to this operation, and only a status output. It must be called before any reference or use of the file system, but after the:

- Disk physical-file system
- Interprocess communication
- SM manager
- CB manager

are all initialized and running.

Only the cold start system initialization process may call this operation which will either install or initialize the logical file system. Interfaces are required between this operation and system configuration control (by which the physical file IDs of the initialization files are retrieved from and stored in the system configuration table). This operation also creates all SM, CB and other objects required by the logical file system.

File System Installation

When neither of the physical file IDs of the initialization files are available from system configuration control, the file system, by definition, does not exist and must be installed. This operation installs the file system on the ETA10. It may only be called by the LFS cold start operation. The physical file IDs of the initialization files are generated and returned to the LFS cold start operation.

File System Initialization

When the physical file IDs are available from system configuration control, the file system exists, but must be initialized. This operation initializes the file system on the computer system. It may only be called by the LFS cold start operation. The input parameters are the physical file IDs of the file system initialization files.

CPU Initialization for File System

This operation must be called to warm start a CPU. It must be called after:

- Disk physical-file system
- Interprocess communication
- SM manager
- CB manager

have all been initialized on a CPU, but before any other reference or use of the file system by any other process executing on the CPU. This operation creates and initializes shared CP memory areas required by the logical file system and connects the CPU to the SM, CB, semaphores, and other objects required by the logical file system.

Process Initialization for File System

At the beginning of each process, except for those which must call any one of the aforementioned initialization operations, there are two operations which must be called to initialize the file system for the process.

Start Up File System for Client, is called from Process Create from within a process management process. This operation creates an open file audit table for the process and causes all the files required by the process to be opened to the SM manager.

Start Up Process, must be called within the newly created process, after the:

- Interprocess communication
- SM manager
- CB manager

have all been initialized, but before any other file system operation is called. The operation activates and opens the file system files to the process and connects the process to all SM, CB, shared CP memory, semaphore, and other objects required by the logical file system.

Process Termination

The file support module provides three levels of file system termination to a process. The first two are called from within the process being terminated. The third is a clean-up operation which executes within a process management process.

User File Termination

This operation is the first level of file system termination for a process. It must execute within the unnamed (user) domain. It:

- Flushes all I/O buffers for all files open to the user
- Closes and deactivates all files open to the unprotected system
- Closes and deactivates all files open only to the protected system

The operation may be called under abnormal (though not catastrophic) termination conditions.

Disabling the File System

This operation is the second level of file system termination to a process. It closes all files which have I/O connections (including file system files) and disconnects the process from all SM, CB, shared CP memory, semaphores, and other objects needed by the logical file system.

After this operation completes, the file system is unusable to the process. This operation may be called under abnormal (though not catastrophic) termination conditions.

Terminate File System for Client

This operation is the third level of file system termination to a process. It causes all files whose identifiers remain in the open file audit table to be closed and deactivated. It also destroys the open file audit table.

This operation is only called by Process Terminate from within process management. This operation may be called under abnormal, including catastrophic, termination conditions.

File Activation and Deactivation

When the caller of the file system initially references a file in a process, the reference is done through a file path name or a global file ID, The file system must make a connection between the:

- Pathname
- Global file ID
- Attributes (including physical peripheral storage)

of the file associated with the name. This connection must remain intact until the user requests the file system to break the connection or until the connecting process terminates.

This implies that there may be points in time when a connection exists to a non-existent file. Creating, activating, and destroying a file results in this condition.

Establishing the connection is called *file activation*. Its inverse is *file deactivation*.

When a file becomes active, it inherits a set of saved attributes which become an instantiation of those attributes referred to as *usage attributes*. The user may list and alter the usage attributes of a file without affecting either the file's saved attributes or other instantiations (activations) on the file.

A file activation is externalized to the user by means of a logical file ID. This logical file ID remains intact until the file is deactivated. A file must be activated before its usage attributes may be listed or altered, and before it can be opened. When a file is closed, it remains active. Multiple activations of a file are allowed.

File Sharing

More than one I/O path to a file may exist at one time. There are two senses in which this may happen:

- Shared opens
- Simultaneous opens

Sharing an open file means that many file open attributes such as:

- Current file position
- Active record type

are shared in common among several of the file openers. A shared open is accomplished through the use of a "share mode" or "share token" which is provided through a specific open shared operation.

One user, referred to as the master opener, is to open the file in master share mode. Other users may share this open of the file by retrieving the share token from the master opener and providing it to the open shared operation with a subordinate share mode. Outside the open shared operation, the file system does not distinguish between a master or subordinate opener of a file.

Simultaneous Opens

Simultaneous opens are non-shared opens of the same file. Each open has independent usage attributes and resources associated with it that are not shared with any other opener of the file.

Read/Write Lockout

When opening a file, the caller may wish to lock out further access to it for the duration of the open. This lockout feature is used for both shared and simultaneous opens. Lockout works on both read access and write type (e.g. write, append, or modify) accesses independently and without regard to the access requested by the caller.

Multiple read and write lockouts are permitted.

If the lockout cannot be granted, the open request is denied. A lockout cannot be granted if the file is already open for the access type which is to be locked out.

If a lockout is successful, no further non-shared or shared master open requests for the locked out access types are granted until the open creating the lockout terminates. A process which is to update the contents of a file, dependent on the file's current contents, and requires that no other process be writing the file during the update, should request both read and write access and request that write type accesses be locked out.

Similarly, a process which is to read a file, and requires that no other process modify the contents of the file while it is being read, should request read access only and request lockout for write accesses.

The subordinate shared open operation does not check existing lockouts. This is because all such opens are considered to be one single open of the file which occurred during the master shared open operation. Lockouts requested by the master opener persist until all sharers of the open close the file.

Performance Versus Security Considerations

The file support module and record manager together are responsible for conducting all explicit I/O. Protecting and securing the file system from unauthorized access is an absolute requirement, but I/O performance is also critical. If all the file support module and record manager operations were to execute within the nucleus, each call for I/O would require at least one domain change.

In order for the record manager to be successfully partitioned between unprotected and protected operations, the file support module must also be partitioned between unprotected and protected operations.

This splitting of operations essentially gives definition to two distinct I/O systems for all operations requested between (and including) file activation and deactivation. The relationship of these two systems is that the unprotected system is built upon and uses the protected system. The protected system, however, is unaware of, and has no interface to the unprotected system.

Entry to protected operations is checked by the domain manager to ensure that entry is permitted. If permitted, entry is granted, otherwise entry is denied.

- **Unprotected system** handles the blocking/buffered operations of the record manager. This is the system which provides double buffering in CP memory of file data, and provides interpretation and construction of logical data partitions (records).

  The tables and buffers used by this system are not protected from user tampering. This availability is necessary to prevent domain changes in the normal conduct of buffered I/O.

  This system manages record type and related attributes such as a file's current file position.

- **Protected system** handles the in-place operations of the record manager. This is a more physically-oriented system which provides I/O of byte stream data and overlap to computation facility.

There are no buffers used by this system where data is taken from or put into locations specified by the caller. Tables used by this system are protected from user reference and tampering.

This system is not aware of record type and manages more physically oriented attributes such as the file's end-of-Information (EOI).

Operations provided by both file systems include:

- Activate file
- List file usage attributes
- Alter file usage attributes
- Close file
- Deactivate file

Operations provided only for the unprotected system include:

- Open file for explicit, direct I/O
- Open file for explicit, sequential I/O
- Open file for implicit I/O
- Write bytes
- Write partition
- Write partial partition
- Write partition delimiter
- Flush buffers
- Skip bytes
- Skip partitions
- Read bytes
- Read partition
- Read partial partition
- Read with WAIT
- Write with WAIT
- Flush file with WAIT
- Seek

Operations provided only for the protected system include:

- Open file
- In-place write
- Flush file object
- In-place read

Great care must be taken in changing between these three I/O interfaces:

- User environment interface

- Unprotected system interface
- Protected system interface

If interface usage is inappropriately mixed, uncontrolled and indeterminate results will most likely result, such as data being placed into, or read from, the wrong location, or portions of the file being inadvertently overwritten.

Object Definitions and Usage

Certain objects and attributes of the file support module are defined.

Local File ID

The local file ID is a file identifier used in place of the file path name or global file ID once a file has been activated. It is constructed by the FSM and returned to the caller when a file is activated. It is an identifier to the file system of a connection to an active file.

Each local file ID is unique with respect to the inheriting process and its concurrently active files: No active file has a local file ID which equals that of any other file active to the same process.

There may be two different local file IDs for a file if it is being shared. When a file is deactivated by a process, its local file ID is free for re-use within the process.

Global File ID

The global file ID uniquely identifies a file or directory without regard to its path name(s) or locality of activation. It is algorithmically encoded and decoded by the file system. It is returned by various operations and may be used as an input parameter to several file system operations.

Shared Information

Tables shared by the file support module with other functions in the ETA10 are described.

I/O Connection

The I/O connection (IOC) is provided by the file support module and resides in process private space to which only a named device has access. An entry in the I/O connection table is constructed when a file is activated, and is deleted when the file is deactivated.

The protected operations of the file support module have read/write access to the table, while the protected operations of the record manager and the file directory/catalog manager have only read access to the table.

An entry in the I/O connection table is essentially a set of pointers to other tables, but does hold file state and accessibility information. The other tables referred to are tables needed by other parts of the file system to perform their specific functions.

Global File Usage Attributes Table

The global file usage attributes table (GFAUT) is provided by the file support module, but resides in shared memory. An entry in this table is constructed from attributes in the local file usage attributes table (LFUAT), and the file I/O control block when the file is opened for shared access in master mode. The global file usage attributes table entry is deleted when the last shared opener of the file closes it.

The protected operations of the file support module and record manager have read/write access to this table. Protected file support module operations are provided for accessing the table.

Entries in the table contain file system management information and the attributes of shared open files.

Local File Usage Attributes Table

The local file usage attributes table (LFUAT) is provided by the file support module and resides in process private space to which only the named domain has access. An entry in this table contains attributes necessary for the conduct of in-place I/O. The entry is deleted when the file is deactivated.

The protected operations of the file support module and record manager have read and write access to this table.

File I/O Control Block

The file I/O control block is provided by the unprotected Activate File operations of the file support module. It is modified by the unprotected open and close operations of the file support module, and by the unprotected record manager operations.

It resides in process private space in the user domain. It contains:

- Current file positioning information
- Current EOI

- Padding character
- Record type
- Buffer management data

and other information for non-shared file openings. For shared openings, this information is either not relevant, or it is contained in the global file usage attributes table.

This table is parallel to the I/O connection table. The unprotected operations of the record manager and the unprotected operations of the file support module have read and write access to this table.

An entry is deleted when a file is deactivated via the unprotected deactivate file operation.

Open File Table

The open file table is provided by the file support module and resides in the communication buffer. An entry in this table is constructed whenever a file is first opened. The entry is deleted when all the processes accessing the file have closed it.

The protected operations of the record manager and the SM manager have read access to the table. The protected operations of the file support module and file directory/catalog manager have read and write access. Protected file support module operations are provided for accessing this table.

An entry in this table contains open file attributes which are global without respect to whether the file is opened for shared, simultaneous, or singular access.

Physical-File ID Map

The physical-file ID map is provided by the file directory/catalog manager and initialized by the file support module during the Initialize File System operation. It is used for disk physical files.

Entries are made in the map by the file directory/catalog manager when a file is opened. Entries are deleted when the file is closed.

Physical-file IDs are used by the SM manager to perform I/O.

Open File Audit Table

The open file audit table is provided by the file support manager and resides in SM or CB. This table is created during the Start Up Process for Client operation.

It contains entries for:

- All files required by the SM manager
- The process's object and paging files
- All user files which are open to the process

Each entry contains the:

- Index of the open file table entry
- Mode in which the file was opened
- Indexes into SM manager tables.

This table exists until process termination occurs, when it is destroyed by the Terminate File System for Client operation. The protected operations of the file support module have read and write access to the table.

Functional Interfaces

The file support module contains the following functional interfaces:

- **Cold Start File System** installs or initializes the logical file system depending on whether it already exists or not. It also creates all SM, CB, and other objects required by the logical file system.
- **Install File System** creates all files required by the file system (including the initialization files) and returns the physical file IDs of the initialization files to the Cold Start File System operation for permanent storage in the system configuration table. Additionally, it:

  - Creates
  - Initializes
  - Grants
  - Activates

  all SM, CB, shared CP memory, semaphore, and other objects required by the logical file system.
- **Initialize File System** activates the file system and, if necessary, recovers file system files. Additionally, it:

  - Creates
  - Initializes
  - Grants
  - Activates

all SM, CB, shared CP memory, semaphore, and other objects required by the logical file system.

- **Initialize CPU for File System** creates and initializes shared CP memory areas. Additionally, it:

  - Creates
  - Grants
  - Activates

  all semaphores as necessary to shared CP memory areas, SM and CB objects, and to file system files required by the logical file system.

- **Start Up File System for Client** creates an open file audit table for the process being initialized, and causes all the files required by the process to be opened to the SM manager before the process begins executing.

- **Enable File System to Process** activates and opens the file system files to the process, and connects the process to the process's open file audit table and all SM, CB, shared CP memory, semaphore, and other objects required by the logical file system.

- **Terminate User Files** flushes CP memory block I/O buffers and closes and deactivates all files open to the unprotected system of a process. This operation is the first of three which together terminate the file system to a process. The remaining two are:

  2. **Disable File System to Process**
  3. **Terminate File System for Client**

- **Disable File System to Process** closes and deactivates all files for which there are I/O connections. It does not flush any CP memory block I/O buffers as it is the responsibility of all system features to close their files using normal closing operations if they are using buffered record manager operations.

  This operation must not issue any accounting, dayfile, or user management operations which in turn use the file system. This operation also deactivates all logical file system semaphores, and CB and SM objects to the process.

  This operation is the second of three which together terminate the file system to a process. The remaining two are:

2. **Terminate User Files**
3. **Terminate File System for Client**

- **Terminate File System for Client** causes all files whose identifiers remain in the open file audit table to be closed and deactivated. It also destroys the open file audit table.

  This operation is the third of three operations which together terminate the file system to a process. The remaining two are:

  1. **Terminate User Files**
  2. **Disable File System to Process**

- **Activate File From Pathname** translates a pathname into a global file ID, and calls Activate File From Global File ID. It executes in the domain of the caller as part of a shared library.

- **Activate File From Global File ID** calls Activate File (Protected). If that call is unsuccessful, it then calls List Path Name Attributes from the file directory/catalog manager to initialize the file I/O control block for the caller.

  This operation executes in the domain of the caller as part of a shared library, and enables the use of all unprotected file support module operations.

- **Activate File (Protected)** provides the protected connection between a process and a specified file. A logical file ID is generated and returned to the caller. Entries for the file are initialized in the appropriate tables.

  This operation enables the use of the operations:

  - **Open File (Protected)**
  - **List and Alter File Usage Attributes (Protected)**
  - **Close File (Protected)**
  - **Deactivate File (Protected)**

- **Deactivate File (Unprotected)** deletes the file I/O control block entry and calls the Deactivate File (Protected) operation to break the connection between a process and a file. It makes the logical file ID available for re-use. This function executes in the domain of the caller as part of a shared library.

- **Deactivate File (Protected)** makes a logical file ID available for re-use. It deactivates a connection between a process and a specified file.

- **List File Usage Attributes (Unprotected)** returns the attributes from the file I/O control block for a specified file. This operation executes in the domain of the caller as part of a shared library.

- **List File Usage Attributes (Protected)** returns the attributes from the protected I/O tables for a specified file.
- **Alter File Usage Attributes (Unprotected)** modifies the attributes in the file I/O control block for a specified file. It does not alter any of the file's saved attributes. This operation executes in the domain of the caller as part of a shared library.
- **Alter File Usage Attributes (Protected)** modifies attributes in the protected I/O tables for a specified file. It does not alter any of the file's saved attributes.
- **Open File for Explicit, Sequential I/O (Unprotected)** opens a specified file to allow explicit sequential I/O to be performed. It executes in the domain of the caller. The caller may specify the following attributes:
  - Type of access required while the file is open
  - Position at which to open the file
  - The amount of file data to be buffered in CP memory. Two CP memory buffers, each an integral number of small pages, are established for the file. The record manager double buffers the data in the file, overlapping computation and I/O, using the access strategy for directionality into/from the CP memory buffers. If the amount of data buffered is zero, or the access strategy is Demand, then no or only minimal CP memory buffering is done.
  - A strategy for file access. May be one of the following:
    - **Forward Look-ahead**
    - **Demand**

    The record manager buffers using either Forward Look-ahead or Demand,.
  - Lock out request flags. There is one flag for each type of read and write access lockout.
- **Open File for Explicit, Direct I/O (Unprotected)** opens a file to allow for explicit, direct I/O to be performed. It executes in the domain of the caller. The caller may specify the following attributes:
  - Type of access required while the file is open

- Position at which to open the file
- The amount of file data to be buffered in CP memory. Two CP memory buffers, each an integral number of small pages, are established for the file. The record manager double buffers the data in the file, overlapping computation and I/O, using the access strategy for directionality into/from the CP memory buffers. If the amount of data buffered is zero, or the access strategy is Demand, then no or only minimal CP memory buffering is done.
- A strategy for file access. May be one of the following:
    - **Forward Look-ahead**
    - **Demand**

  The record manager buffers using either Forward Look-ahead or Demand.
- Lock out request flags. There is one flag for each type of read and write access lockout.

- **Open File for Implicit I/O (Unprotected)** opens a file for implicit I/O to be performed on it. This operation executes in the domain of the caller.
- **Open File for Shared Access (Unprotected)** opens a file to allow I/O to be shared between callers. The current file position is shared by all callers that open the file with the same share token. This operation executes in the domain of the caller. The caller may specify the following attributes:
    - A shared mode which may be Master or Subordinate.
    - A share token which identifies the open the caller wishes to share. This attribute is returned to the caller if the share mode is Master.
    - The access mode of the file opening. It may be:
        - Sequential
        - Direct

      This attribute is ignored if the share mode is Subordinate.
    - The type of access required during the open This attribute is ignored if the share mode is Subordinate.
        -

  - The position at which to open the file. This attribute is ignored if the share mode is Subordinate.
  - A strategy for file access. May be one of the following:
    - Forward Look-ahead
    - Demand
  - Lock out request flags. There is one flag for each type of read and write access lockout. This attribute is ignored if the share mode is Subordinate.

- **Open File (Protected)** opens a specified file to permit shared or non-shared explicit or implicit I/O. If called for shared explicit I/O, the current file position is shared by all callers that open the file with the same share token. The caller may specify the following attributes:
  - The I/O mode. It may be Explicit or Implicit. The Implicit mode is used only with disk files..
  - The share mode. It may be:
    - Master
    - Subordinate
    - Non-shared

    The Master and Subordinate attributes are used only with disk files..
  - The share token. This attribute is ignored if the share mode is Non-shared, and is ignored as an input parameter if the share mode is Master.
  - Type of access required. It may be any combination of:
    - Read
    - Write
    - Execute
    - Append
    - Modify

    If the access mode is Explicit, the access mode may not include Execute. If the access mode is Implicit, the access mode may not include Modify or Append.
  - A flag that indicates, when set, that the file is a system file. These files are not closed except by explicit request, or during the Terminate System Files operation. This attribute is ignored if called from the user domain.

  - A strategy for file access.
  - Lock out request flags. There is one flag for each type of read and write access lockout. This attribute is ignored if the share mode is Subordinate.

- **Close File (Unprotected)** closes a specified file. The record manager is called to flush all buffered CP memory data for the file to peripheral storage. This operation executes in the domain of the caller. The caller may specify the position at which to close the file.

  Close File (Unprotected) routine eventually calls the Close File (Protected) routine to complete the close.

- **Close File (Protected)** closes a specified file. SM manager is requested to close the file object in shared memory. The caller may specify the position at which to close the file.

Record Manager

The record manager provides the mechanisms by which users may read and write data from and to files. The record manager interacts with the following to accomplish its functions:

- File directory/catalog manager (FDCM)
- File support module (FSM)
- SM manager (SMM)

Additionally, it interacts indirectly with the CP memory manager (CPMM).

The record manager is accessible by direct calls from all domains. As such, it is the single conduit through which explicit I/O on files is accomplished.

Logical Groupings of File Data

The smallest quantum of data usable by the record manager is the byte. Users may access data from, and write data to files by means of logical groupings of data bytes. All logical groupings of data are called *logical data partitions*, or just *partitions*.

Partitions are either algorithmically constructed logical groupings of file data, or they are constructed by the insertion of control information into the file. If constructed of control information, the control information is useful only to the record manager and in normal usage is never seen by the user.

Records

A record contains logically associated, contiguous data bytes excluding any control information. It should not be regarded as a minimal unit of addressable data. The record is distinct from:

- A sector on disk
- A half-word in shared memory
- A byte or bit

All of these are considered to be minimal units of addressable data in one context or another.

Groups

A group is a set of logically associated, contiguous records, including imbedded record delimiters, but excluding group delimiters.

Files

The definition of a file level partition is dependent on the context. A file level partition is defined to be all the bytes of a file except file level control information. It may also be a collection of records, including any record delimiters, or a collection of groups, including any group delimiters. Multi-file files are not supported or permitted by the record manager.

Partition Hierarchies

All files may contain either no data, or one *and only one* file level partition:

- Some files contain only the file level partition
- Some files contain only record and file level partitions
- Some files contain record, group, and file level partitions

Record level partitions are contained in either group or file level partitions, group level partitions are contained within file level partitions.

Working Storage Area

When all or part of a partition is "given to" or "retrieved from" the record manager it exists as data contained in the working storage area (WSA). The working storage area is a set of virtually contiguous bytes in virtual CP memory aligned on a byte boundary. It is provided by the caller to the record manager as appropriate and is accessible to the caller.

Associated with each working storage area is a length in bytes referred to as the working storage length (WSL). While a partition exists in the working storage area it has no type.

Accessing Modes

When a file is opened for explicit I/O, it is opened in either sequential or direct access mode. The caller must specify which access mode is desired by choosing the correct open operation. Access mode is a file usage attribute and has meaning only when the file is open for explicit I/O. To change from mode to the other, the caller must close and reopen the file.

All files may be opened in sequential access mode. Only type *F* or *U* record format disk files may be opened in direct access mode.

Sequential Access

Sequential access has a restricted and explicit definition. Sequential access is defined as a mode of access in which:

- Every write operation to the file defines a new end of file position which is the last data byte written.
- Modify access is not permitted.

Direct Access

Direct access also has a restricted and explicit definition. Direct access is defined as a mode of access in which:

- A write operation to the file defines a new end of file position if *and only if* the byte address of the last byte written is greater than the byte address of the EOF that existed before the write operation.
- Append access is not permitted.

Shared Files

The LFS supports simultaneous and shared access to a file. The record manager is involved in supporting this type of access. The record manager does not synchronize I/O operations on shared files, this is the responsibility of the file users.

While files may be shared, CP memory buffer space on files may not be shared. The single aspect of shared files for which the record manager is responsible is that of current file position.

Simultaneous Access

When a file is open for simultaneous access, calling the record manager to read, write, or skip on the file causes a change in the current file position for the caller's process and logical file ID pair. It does not in any way alter the current file position on the file for any other process which is simultaneously accessing the file.

It should be noted that while file positions are not altered, they may be invalidated as a file's end of information may be set to a value lower than a process's current file position on that file.

Shared Access

When a file is open for shared access, calling the record manager to read, write, or skip on the file causes a change in the current file position on that file for all process and logical file ID pairs sharing the access.

CP Memory Block I/O Buffers

The record manager provides, on request, block data buffering of file data in CP memory through the provision of CP memory block I/O buffers. These buffers are maintained by the record manager and are intermediate data buffers between the working storage area and shared memory. There are two strategies employed by the record manager, and selectable by the caller, that support data buffering:

- The first is *"look-ahead"* buffering. The caller provides the length of the desired buffer. The record manager allocates two CP memory buffers, each with half that length, although neither being less than one CP memory small page per buffer unless the requested length is zero.

  In conducting I/O on the file, the record manager will "double" buffer using the computation overlap facility. Management of the buffers are optimized by the record manager to minimize I/O within the constraints of the buffer length supplied by the user.

- The second strategy is "demand" strategy. In this strategy the record manager determines if an intermediate buffer is needed on each I/O call. If one is necessary, the record manager:

  - Allocates it
  - Transfers the data into it
  - Properly disposes of the data
  - Deallocates the buffer
  - Returns to the caller

  If an intermediate buffer is not needed for an I/O request, then none is allocated and the data is moved directly.

Type *R* and *V* files always require intermediate buffers in order to insert or strip record delimiters.

Concurrent, In-place I/O

The record manager provides facilities to conduct in-place I/O. By this it is meant that through these facilities, data is transferred directly to or from the caller's working storage area buffer from or to shared memory without being routed through intermediate CP memory buffers.

The in-place I/O operations have the ability to begin the transfer and return to the user without waiting for transfer completion. This provides for an I/O-to-computation overlap facility. The wait or no-wait option is user selectable. A check I/O call with wait option is also provided so that callers may be assured that the I/O necessary for computation is complete before entering computational logic which depends on the data.

The following restrictions are placed on usage of these facilities:

- The transferred data is treated strictly as byte stream data regardless of the file's record type.
- Intermixing partition level I/O operations with in-place I/O operations has undefined (and for the user most likely disastrous) results.
- The amount of data moved through the in-place operations must be a multiple of half-words to or from a file address that is on a half-word boundary, from or to a working storage area address that is on a half-word boundary. Otherwise an exception case status is returned and no data is moved.

There is a request serial number (RSN) returned from each in-place I/O operation which identifies the operation, and the data and memory associated with it. A request serial number is a positive integer which can be interpreted to be the number of in-place I/O operations requested for a particular logical file ID, since the file associated with the logical file ID has been opened.

Opening a file causes a request serial number associated with the opening to be set to the integer 0. Requesting in-place I/O on the file causes the request serial number to be incremented by one, and that value is then returned to the caller.

It should be noted that request serial numbers associated with I/O operations for one open file are not guaranteed to be unique among request serial numbers associated with I/O operations for another file, even in the case of two different opens for the same file.

The request serial number is an input parameter to check the I/O function which should be called when the status or the completion of the I/O is desired.

Partitioning of Record Manager Operations

There are essentially two types of record manager operations. One of these has been referred to as in-place concurrent I/O. An in-place operation functions without regard to the file's logical partition structure, and will not transfer data to or from intermediate CP memory buffers.

The operations of the other type do buffer data in CP memory buffers and also provide and support logical structures. These operations are collectively referred to as buffered/blocking operations.

The caller is permitted access to both types of operations without restriction. Because the buffered/blocking operations themselves use the in-place operations, you are strongly cautioned against intermixing the types of operations requested of the record manager. If intermixing does occur, the results are not defined.

File Data Versus Control Information

Throughout the discussion of the record manager there are references to control information in a file which is used to organize file data. The relationship between control information and data should be described.

Note that control information that delimits a particular level of partition is always treated as data within higher level partitions. Thus, group and partial group, and record and partial record delimiters are treated as data within a file level partition.

Record Constructs or Types

There are four mechanisms by which records (and thus groups) may be constructed and recognized. Additionally, there is a fifth (null) record construct needed by the record manager to signify that files with this record construct have no internal record structure.

For simplicity, a record construct is referred to as a *record type*. Record types cause records to behave differently with regard to I/O performance, cause some of them to be unsuitable for certain types of data storage, and result in various partition hierarchies.

All of the data in a file must be of one *and only one* record type.

Record Mark Delimited (R) Records

A type *R* record is defined to be a variable amount of data between bytes with the integral value $0A_{16}$ or $1F_{16}$. The record manager writes $0A_{16}$ as the record delimiter. The delimiter is appended to the end of each record to signify record end.

Type *R* files provide high efficiency in the use of data storage while providing a rich hierarchy in data organization. On the other hand, the performance in accessing type *R* files is poor relative to other record types. Type *R* records are not recommended for storing non-ASCII data since the data might be interpreted as a delimiter.

Type *R* records are provided in order to support ASCII text files. The record manager recognizes hierarchical partitions within type *R* files as follows:

- $1F_{16}$ Record delimiter
- $0A_{16}$ Record delimiter
- $1D_{16}$ Group delimiter
- $1C_{16}$ File delimiter

All other characters are interpreted as data.

Control Word Delimited (V) Records

The *V* type record has control words placed at the beginning of each partition to signify the location of the end of the partition as well as the location of the next control word. Each control word is one word (64 bits) of data storage aligned on a word boundary.

Type *V* records are slightly less efficient in the use of data storage than type *R* records, but provide significantly better performance. Type *V* records are safe for the storage of binary structured data.

Type *V* records have been provided to support FORTRAN requirements.

The record manager recognizes a hierarchy of partitions in type *V* files. The hierarchy, in ascending order, is:

1. Records
2. Groups
3. Files

Fixed Length (F) Records

The type *F* record is defined to be a fixed but specifiable number of bytes. The record length is a file attribute.

The efficiency of data storage using type *F* records varies, as does the performance in accessing data. Used correctly, it is both highly efficient and provides much greater access performance than either type *V* or *R* formats

Type *F* formats have been provided in order to support FORTRAN requirements.

Type *F* formats are safe for the storage of binary structured data. They contain only record and file level partitions.

Unstructured (U) Records

The type *U* file does not contain records. There is no logical grouping of data in the file recognized by the record manager. The file is considered to be one continuous byte string of a specified length.

The caller specifies the number of bytes to be read or written during each I/O operation to a type *U* file. The type *U* file may be both highly efficient in data storage and access performance, but it provides no facility for the logical organization of data.

File Positioning

Several concepts and attributes associated with open files are described.

Partition Numbers and Byte Numbers

The partitions in a structured file are sequentially ordered. Thus, partitions can be given a *partition number* which may be used for subsequent addressing.

The partition number of any partition is the number of partitions of the same level which precede it, plus one. A record numbered 100 is always the one hundredth record in a file regardless of any group structure the file may contain.

Each group and record in a file is uniquely and consecutively numbered. Because multi-file files are not permitted, each file is always numbered 1.

There is an analogous concept to unstructured files. The bytes of an unstructured file are consecutively numbered. The *byte number* of any byte in the file is the number of bytes which precede it plus one. The first byte in the file is byte 1.

File Beginning and End Positions

Each non–null partition within a file has a well–defined beginning and end position (except during partial partition write operations). These positions are known as *beginning-of-partition* (BOP) and *end-of-partition* (EOP). A level partition may be null (empty), in which case its beginning-of-partition and end-of-partition are nonexistent.

The *beginning-of-file* (BOF) is the first file level partition data byte of a file. The *end-of-file* (EOF) is the last file level partition data byte of a file. An unstructured file has only a beginning-of-file and end-of-file for beginning and end positions. A structured file may have, in addition:

- Beginning-of-group (BOG)
- End-of-group (EOG)
- Beginning-of-record (BOR)
- End-of-record (EOR)

Current Partition and Current Byte

When a structured file is open for explicit access, it always has, depending on its record type, a defined *current group* and *current record*. At any point in time, these two quantities can assume any number of values. The record manager is responsible for managing these conditions.

There is an analogous concept for unstructured files. When an unstructured file is opened for explicit access, there is always a *current byte* defined.

Current Partition Address and Current Partition Offset

Within the current partition there is always a *current partition address* and *current partition offset*. The current partition address is the file address of the beginning-of-partition. The current partition offset is the number of bytes read from or written to the current partition. It is also the point at which a read or write partial partition operation is to begin.

Any time a complete partition has been transferred to or from the working storage area, the partition offset is set to zero. If a partial read or write operation is performed, and only part of some partition is transferred, the partition offset is set to the partition offset plus the number of bytes transferred.

Note that where the respective partition levels exist, the following values are always equivalent:

- Current file address plus the current file offset
- Current group address plus the current group offset
- Current record address plus the current record offset.

Current File Position

The current file position is defined as being the byte address of the first byte of the current partition plus the:

- Current partition offset if the file is structured
- Byte address of the current byte if the file is unstructured

When a file is opened at beginning-of-file, the current partition is set to the first partition, even if the partition is nonexistent. When a file is opened at end-of-file, the current position is set to the last partition byte plus one.

If a structured file is opened, the current record offset is set to zero (i.e. structured files are always opened at some beginning-of-record, because a file cannot be closed without a record having been completely transferred). When a file is opened without repositioning, the current partition is not changed from what is was after the last close.

For structured files, each write operation to the current record completes that record, by either:

- Write partition at the record level
- Write partial partition at the record level with the terminate partition flag set
- Write partition delimiter at the record level

The current record is redefined as the last full record written plus one. The current record offset is set to 0. Each read operation to the current record which completes it causes the:

- Record count to be set to the last full record read plus one
- Current record offset to be set to 0.

For unstructured files, each write operation causes the current byte to be redefined as the previous current byte plus the number of bytes written. Each read operation causes the current byte to be reset to the previous current byte plus the number of bytes read.

End-of-Information

For all files there is a point that marks the end of all data and record manager control delimiters. This point is the *end-of-information* (EOI), which for disk files is the highest byte number written to the file.

Management of Record Manager Related Attributes

There are several file attributes needed by the record manager in order for it to accomplish both saved file attributes and file usage attributes. The file attributes include:

- Record type
- Padding character
- Maximum record length or fixed record length
- Buffer length

Record Manager Shared Information

The record manager contains a number of tables which are shared with other processes in the system. These tables are described.

I/O Connection Table

This table contains:

- Connection information to other file system tables and files
- File usage state information
- Effective access permissions

An entry is created in this table by the file support module when a file is activated It is deleted when the file is deactivated. This table resides in process private memory and is not accessible to the blocking/buffered operations.

Global File Usage Attributes Table

This table contains file system management information and the attributes of files as they are being used and are open for shared access. An entry in this table is created by the file support module when the file is first opened for shared access. It is deleted when the file is no longer opened for shared access.

The usage attributes are inherited from the local file usage attributes table of the master opener. This table resides in the communication buffer and shared memory and is not accessible to the blocking/buffered operations.

Local File Usage Attributes Table

This table contains file system management information and the attributes of files that are needed by the in-place operations. An entry in this table is created by the file support module when a file is opened. It is deleted when the file is closed. The table resides in process private memory and is not accessible to the blocking/buffered operations.

Open File Table

This table contains open file attributes which are global without respect to how the file was opened. An entry in the table is created by the file support module when a file is first opened. It is deleted when the file is no longer open to any process. This table resides in the communication buffer and shared memory and is not accessible by the blocking/buffered operations.

File I/O Control Block

This table contains the set of file CP memory buffering attributes that are needed and managed by the record manager while the file is open. An entry in this table is created when the file is opened. This table resides in process private memory and is not accessible by the in-place operations.

Record Manager Routines

The record manager internal functions are as follows:

- **Write Bytes** transfers a specified number of bytes of data from the working storage area into an I/O buffer associated with the specified file. The bytes are moved starting at the current file position or at a specified byte address.

- **Write Partition** transfers a full partition (delineated by the WSL parameter) whose level is transferred from the working storage area to an I/O buffer associated with the file.

  If the operation immediately preceding was a Write Partial

Partition without the terminal partition flag being set, this operation calls the write partition delimiter operation. Then, if both:

  - The current partition offset of the implied or specified partition level is not 0
  - The partition number to be written is not specified

the operation skips forward one partition of the same type and positions at the beginning of the partition skipped to.

If the partition number to be written is specified, the file is positioned at the beginning of that partition before writing begins.

The data transferred is a complete partition starting from the beginning of the current or specified partition. Partition delimiters are added to the partition as appropriate (type *R* or *V* files).

- **Write Partial Partition** transfers a partial partition (delineated by the working storage length parameter) from the working storage area into an I/O buffer associated with the specified file. The partition level to be written is specified by the caller:

  - A partition delimiter is written to the current partition
  - The current record is padded if the preceding operation was a write partial partition and the terminal partition flag was not set

  If the implied or specified partition to be written is the same as the current partition, the data is written starting at the current file position. No partition delimiter is appended to the data unless a terminal partition flag is set.

  If the file is type *F* record format and the terminal partition flag is set, the current record is padded out, if needed, after working storage length bytes are written to the file.

- **Write Partition Delimiter** writes a partition delimiter to the current position for types *R* and *V* record format files, or pads out the remainder of a type *F* format record in the specified file. The partition level is specified by the caller.

- **In-place Write** writes a specified amount of data from a specified CP memory virtual address to the specified file's SM buffered space without going through the record manager's buffering/blocking operations with optional I/O to computation overlap.

The record type of the file is ignored and the data is written without interpretation. The data is written to the specified or implied file address.

This function returns a record sequence number to be used by the caller to determine the status of the I/O at a later time. Positive identification of the file's I/O status is returned to the caller when the function completes.

- **Flush I/O Buffers** writes the contents of CP memory block I/O buffers associated with a specified file to shared memory and optionally requests the SM manager to write the data to peripheral memory.

  If the user is doing minimal or no CP memory buffering, this operation does nothing unless the destination is peripheral storage. This operation does not invalidate the contents of any buffer.

- **Read Bytes** transfers a specified number of bytes of data into the working storage area from an I/O buffer associated with a specified file. The bytes are moved starting from the current file position or the specified byte address. The number of bytes actually transferred is returned to the caller.

- **Read Partition** transfers a full partition with the specified number and level from an I/O buffer associated with the specified file into the working storage area. The length of the working storage area is specified by the caller.

  If the immediately preceding operation was a Write Partial Partition without the write_partial_partition flag set, this operation calls the Write Partition Delimiter operation. This results in an attempt to write past the EOF if the partition number is defaulted.

  If the partition number is defaulted and the current partition offset of the implied or specified partition level is not 0 (i.e. the previous operation was a read partial partition), this operation skips forward one partition of the same type and positions at the beginning of the partition skipped to.

  If the partition number to be read is specified, the file is positioned at the beginning of that partition before reading. The data transferred is a complete partition starting from the beginning of the implied or specified partition.

  Partition delimiters are stripped from the partition as appropriate for type *R* and *V* format records. The number of bytes actually transferred to the working storage area is returned to the caller. If a partition level equal to or higher than the specified level is found, that level is also returned to the caller.

- **Read Partial Partition** transfers a full or partial partition with the specified number and level from an I/O buffer associated with the specified file into the working storage area.

  If the immediately preceding operation was a Write Partial Partition without the terminal partition flag set, this operation calls the Write Partition Delimiter operation. This results in an attempt to read beyond the end of information if the file is opened in sequential access mode and the partition number to be read is defaulted.

  Partition delimiters are stripped from the partition as appropriate for type *R* and *V* record format files. The number of bytes actually transferred is returned to the caller as is an indicator of whether the end of the partition was found.

- **In-place Read** transfers a specified amount of data directly from a specified file's SM buffered space to a specified CP memory virtual address without going through the record manager's buffering/blocking operations with optional I/O to computation overlap.

  The record type of the file is ignored and the data is read without interpretation. The data is read from a specified or implied file address.

  This function returns a record sequence number to be used by the caller to determine subsequent I/O status. Positive identification of the file's I/O status is returned to the caller when the function completes.

- **Skip Bytes** positions a specified file to:

  - A specified offset from the current file position
  - The beginning or end of a file
  - A special offset from the beginning or end of a file

- **Skip Partitions** repositions a file a specified number of partitions from its current position. If the preceding operation was a Write Partial Partition without the write_partition_flag set, this operation calls the write partition delimiter operation.

  The file is positioned at the beginning or end of the specified partition depending on a position within partition parameter. The caller is informed of the partition level encountered in the skip.

- **Seek** positions a specified file to a specified logical file offset. This routine is to be used with the in-place I/O system.
- **File Object Flush** causes a file's data to be written from shared memory to an I/O unit with optional I/O computational overlap. This operation does not invalidate the contents of any buffered data. This function returns a record sequence number to be used by the caller to determine the status of the I/O at a later time. Positive identification of the file's I/O status is returned to the caller when the process completes.
- **Check I/O Request Status** returns the status of a specified I/O operation on a file. A wait for I/O completion option is provided which waits for I/O completion and checks the requested I/O operation's status. Positive identification of the file's I/O status is returned to the caller when the function completes.

Section 5: The Memory Managers

This part examines the three computer system memory managers and their properties. It consists of three sections:

- CP memory manager
- SM manager
- CB memory manager

Central Processor Memory Manager

The CP memory manager allocates and deallocates CP memory for CPU processes. There are four types of CP memory manager (CPMM) functions:

- **Page Fault Processing** is a single function that:
  - Reads logical disk file data into CP memory
  - Detects and reports faults for undefined addresses
  - Detects and reports faults for inaccessible addresses
- **Implicit File Access** is a series of functions that provide the ability to read and write a logical disk file by referencing or updating CP memory. These functions also allow a process to obtain accessible virtual address space.
- **Physical CP Memory** is a series of functions that support operating system use of CP memory. Included are the CP memory object functions which permit sharing and use of physical CP memory without references to a logical disk file, and the Transfer Address/Buffer functions which permit the SM manager to obtain the physical address of the CP memory pages to be transferred.
- **Process Management** is a series of functions provided to support the process management functions that create and destroy processes.
- **Resource Management** is a series of functions provided to manage CP memory resource limits, and to determine a process' current CP memory usage.

An independent CP memory manager resides on each CPU, managing the processor's CP memory. The CP memory manager does not coordinate the use of CP memory in multiple CPUs.

Central Processor Memory Objects

CP memory manager is the feature that allows processes to use CP memory. Site and user resource limits are provided to support the use of CP memory.

Physical CP memory, in the form of CP memory objects, is used and shared between processes by the operating system. A memory transfer buffer is provided for the SM manager (SMM) to both:

- Obtain a CP memory physical address
- Make a CP memory page unreplaceable or locked down

Memory addressable files are provided to allow implicit read and write access to a logical file. They provide the mechanism for defining a virtual address space a process may reference. Processes may share CP memory by sharing a memory addressable file.

The paging file is a default memory addressable file provided by the CP memory manager to support scratch space. Scratch space is virtual memory that is accessible without having a logical disk file explicitly created and opened for implicit access.

A system code map template initializes system code as a series of memory addressable files in a new CPU process.

Central Processor Memory Manager Feature Interaction

The following paragraphs describe the usage of CP memory manager functions. The function names are illustrated in *italics*.

Implicit File Access

CP memory manager works with the logical file system (LFS) to provide implicit access to a logical disk file. The process is:

1. The file support module (FSM) *Open Implicit* function is performed.
2. The Q5MAPIN System Interface Library subroutine associates the file to a virtual address range, creating a memory addressable file.
3. Within the virtual address range the file is accessed through page faults processed by the Page Fault Processing function.

4. When access to the file is no longer desired in the particular address range, the Mapout File or Mapout Address functions are used to remove access to the file and to the virtual address range.

The file support module Close File function may also be called to remove access to a memory addressable file. The file support module Close File function uses the Mapout File function for files opened for implicit access.

The Mapin Scratch Space function associates a specified virtual address range with the paging file, which is created and opened at process initialization. This permits a process to reference virtual address space without creating and opening a logical disk file.

Address Space Definition

A process' addressable virtual memory is potentially an address space of from 0 to 2^{48}–1 bits. The domains of a process define its virtual address space. The keys of a domain define a domain's accessible virtual address space.

The domains of a process share CP memory data by sharing a key to the virtual address space. The address space of the domain of a given process is unique to that domain, except for the address ranges of data shared between domains.

The virtual address space of a process is made accessible by its association to a memory addressable file. A majority of a process' data is associated to memory addressable files by the pre–linker at the time linking occurs. CP memory manager uses the linker's static definitions of the accessible address space at process initialization.

A process may dynamically expand its address space by either mapping a virtual range to a file or defining a virtual range as scratch space mapped to the paging file. Alternately, to avoid a collision with other memory addressable files, the Allocate Virtual Memory function can be used to obtain an unused virtual address range from the system heap.

Addresses mapped to the paging file are a special case. If the paging file data has not been referenced, a file or CP memory object may be allocated in the virtual address space.

Failure to define a virtual address before referencing it is detected by the Page Fault Processing function.

Page Fault Processing

Mapin File obtains an SM manager open file ID from the file support module (FSM) and associates it to the specified virtual address range. Mapin Scratch Space associates the virtual address range to the SM manager open file ID of the paging file. When a process page faults for a virtual address, the page fault processing function looks up the SM manager open file ID and uses it to read and write the file as requested.

The Mapin File and Mapin Scratch Space functions obtain the key they need from domain management. This specifies the lock Page Fault Processing uses in the associative word in the page table.

Upon detection of page faults that cannot be processed, Page Fault Processing sends the process either an Undefined Address signal, or an Access Violation signal via the process management software signal functions.

Process Initialization

Before a new process begins execution, the CP memory manager calls process management to:

- Invoke Create Paging File
- Invoke Initialize Process
- Invoke Initialize Process Address Map

Initialize Process allocates and activates a list of CP memory objects for the new process and the process management server.

Process Termination

Within a terminating process, the logical file system closes and maps out any remaining files opened for implicit access. The same process that performed Initialize Process must invoke Terminate Process, which deallocates all remaining pages and either destroys the paging file or writes out its modified pages.

CPU Initialization

CPU Initialization is used by the system initialization boot program to initialize the page tables of the processes read into CP memory from the service unit (SU). Once this function has completed, other CP memory functions may be performed. Special exceptions exist for the CP memory objects of several features to permit bootstrapping of the system from the system initialization process.

Central Processor Memory Transfers

Functions are provided to obtain the physical address of CP memory that will be involved in a shared memory transfer. Functions are also provided to the SM manager to translate the virtual pages of a CP memory transfer to physical memory and to lock the pages into memory until the transfer has completed.

To make the translation of an address used in multiple transfers more efficient, CP memory transfer buffers are provided.

SM manager is provided with functions to lock and obtain the physical address of a transfer buffer. Resource management (RM) is provided functions to define a virtual address range as a transfer buffer. SM manager provides resource manager with functions to read and write a file to a buffer address.

Central Processor Memory Resource Management

CPU Initialization and Initialize Process use global scheduler, system configuration control (SCC), and site administration interfaces to obtain the resource usage attributes required to manage CP memory.

Page Fault Processing and Working Set Evaluation will use site and session category attributes as well as current CP memory usage to allocate and deallocate a process' CP memory.

Current resource usage attributes are recorded with accounting by Terminate Process, and may be retrieved through the accounting system by any user.

Central Processor Memory

One small page size is supported on a CPU. The small page size is 2K words.

The different page sizes possible on a CPU require a common measurement unit be adopted for specifying CP memory amounts. The CP memory block (1K words) is the unit of measurement supported by CP memory manager. All CP memory and memory addressable files are specified in terms of CP memory blocks. The smallest unit of CP memory that can be allocated, however, is a small page (2K words).

Central Processor Memory Objects

The CP memory object procedures permit low level operating system functions to use shared, locked CP memory not associated to a logical disk file. CP memory objects are provided for the operating system features that must share CP memory between processes, but which can not perform the map in and page fault operations for data on logical disk files.

For virtual address space definition, CP memory objects are considered similar to memory addressable files. They may not overlap other objects or memory addressable files except for unreferenced paging file space.

Shared CP memory objects do not have to reside at the same virtual address within the different processes. Only the activated object offset must be the same.

Features that use CP memory objects for process information must provide a Process Terminate function. This function must:

- Back up the object for error recovery purposes
- Destroy and/or deactivate the object at process termination

Central Processor Memory Object Supporting Functions

CP memory objects are:

- Created by Create Object
- Destroyed by Destroy Object
- May be referenced after an Activate Object
- May no longer be referenced following a Deactivate Object.
- Physical memory allocated by Allocate Object
- Physical memory deallocated by Deallocate Object

Create Object defines the maximum size of the object, but allocates no memory to it. Since no provision is made to extend a CP memory object, they should be created with the maximum required length.

CP memory is allocated to the object with the Allocate Object function. This function permits the user of the CP memory object to extend the object as required, up to the maximum length specified in the Create Object function. Deallocate Object reduces a CP memory object as requested.

A special function is provided to Process Management at system initialization. It allows Process Management to create a CP memory object from a preallocated physical address by providing as input the:

- Virtual address

- Object offset
- Object length

Memory Addressable Files

A memory addressable file is the mechanism for defining to CP memory manager the virtual memory of a process, and for providing implicit access to a logical disk file. For a process to access virtual memory, it must allocate a memory addressable file, associating a logical disk file to the virtual address space of the process.

Memory Addressable File Segments

Either one or more segments of an addressable file, or the entire logical disk file, may be associated to virtual address space. Separately allocated segments of a logical disk file may reside in the same memory addressable file. A memory addressable file may be segmented into discontinuous address ranges.

The virtual address of a process is assigned to only one memory addressable file. The logical file address in a modifiable memory addressable file may be assigned to only one virtual address. The logical file address of a memory addressable file with read only, or read and execute access, may reside at more than one virtual address of a process.

Memory Addressable File Access Permissions

Access permissions may be specified when allocating a memory addressable file. The access permissions may be any combination of read, write, or execute permissions, but must be a subset of both the access permissions allowed to the open file and the access permissions allowed to the virtual memory.

Allocation or deallocation of a memory addressable file at a virtual address requires access to the virtual address. A write temporary memory addressable file permits modification of the virtual memory, but not the logical disk file. For write temporary access, only read permission to the virtual memory is required. Write temporary is mutually exclusive with write access.

Memory Addressable File Allocation Units

Memory addressable files cannot be allocated in units less than a full CP memory page.

All the starting virtual addresses of a memory addressable file segment must begin on a CPU page boundary.

If the file being mapped is smaller than the page size of the data object, the file will be extended to the page size if write access is requested. If write access is not requested, the memory past the end of the file is patterned and the file length is not altered.

Memory Addressable File Supporting Functions

Mapin File is used to allocate a memory addressable file. Once allocated, a memory addressable file may be accessed by memory reference until the file is deallocated.

To ensure modified virtual memory pages are written out of CP memory, the program must perform either the map out or close file functions. A memory addressable file is deallocated from a process by the Mapout Address or Mapout File procedures.

The linker allocates all uninitialized data areas as scratch space. If the scratch space has not been previously referenced (causing a paging file modification), Mapin File will deallocate the scratch space at the virtual address prior to allocating the memory addressable file.

Memory Addressable Paging File

The paging file is a default memory addressable file provided by CP memory manager. It contains the scratch space pages. It also contains modified pages of write temporary memory addressable files.

The pager allocates space in the paging file on demand when a page fault occurs for scratch space. Paging file space is also allocated when a logical file is mapped in for write temporary access.

Memory Addressable File Supporting Functions

The paging file is created by Create Paging File and opened by Initialize Process. The paging file is optionally destroyed or saved for later analysis by Terminate Process.

Memory Addressable File Scratch Space

Scratch space allows a process to reference virtual memory without requiring that a logical disk file be used as a memory addressable file. Scratch space is assigned to a default memory addressable file, the paging file.

A small scratch space allocation request will cause explicit paging file allocation prior to a page fault. An installation parameter is provided to determine how large of a scratch file space will be explicitly allocated in the paging file.

Scratch space is allocated with Get Virtual Memory or Mapin Paging File, and deallocated with Mapout Address.

Central Processor Memory Resource Objects

There are several concepts CP memory manager uses in allocating CP memory to a process:

- Resident set
- Working set

These concepts are each examined.

Resident Set

The *resident set* is the number of CP memory blocks currently allocated to a process. The resident set size can never be smaller than the working set size.

Working Set

The *working set* is the number of pages reserved for a process (the maximum resident size). A *working set evaluation period* is the amount of CPU time a process uses before its working set is evaluated.

The page fault rate is used to evaluate a working set. For the purpose of a working set, a page fault is considered to be any reference to a page outside the working set (including pages still in the resident set). If the number of page faults during an evaluation period is less than a threshold, the working set is reduced. If it is greater than the threshold, it is increased.

An initial working set is estimated from a percentage of the initial paging file size:

- The working set is considered to be the minimum number of pages a process requires
- The resident set is not reduced beyond the working set measured during the previous evaluation period
- The working set never exceeds the working set maximum
- Reaching the working set maximum causes more frequent page faults to occur when memory is fully committed.

The working set is only used for accounting and scheduling purposes. A process is only charged for the CP memory it uses in its working set.

Central Processor Memory Management Policies

There are several policies CP memory manager follows in deciding how to allocate CP memory to a process:

- Page replacement policy
- CPM commitment level
- Process activation
- Process deactivation

These concepts are each examined.

Page Replacement Policy

Page replacement policy determines which physical page of CP memory is replaced when a process requests access to a nonresident CP memory page. In general, the page replacement policy chooses the least recently used page that resides outside the process' working set.

An easily accessible list of free pages is kept to reduce the overhead in finding an available page to replace. The pages of suspended and terminated processes are added to the free page list.

There is a priority in determining page replacement candidates. The order of priority is:

1. A free page.
2. A page outside the faulting process' working set.
3. A page belonging to a process that is blocked due to memory over commitment.
4. A page outside the working set of any process.
5. A page within the faulting process's working set.

Central Processor Memory Commitment Level

A CPU's commitment level is the sum of:

- The working sets of all the processes in the ready and blocked queues of the CPU

- The shared working set.

Note that shared system code and shared pages are not included in a process' working set.

Process Activation

When CP memory is not fully committed, processes outside the CPU are examined for possible assignment. These processes possess an initial working set.

Central Processor Memory Manager Shared Information

There are a number of tables, blocks, and queues which CP memory manager shares with other parts of the operating system. These sources of shared information are described.

Page Table

CP memory manager maintains a page table for each process. The page tables are shared between CP memory manager and the operating system. The operating system swaps page tables into and out of the space table during a process switch. At process initiation, process management informs the operating system of the location of the process' page table.

A single page table is maintained for system shared code pages. This page table is loaded into the space table with a process' page table. The space table is sorted into a shared code page table and the process' page table during a process switch.

Working Set Evaluation Queue

The operating system maintains a queue of processes that need to have their working sets evaluated. Pager references this queue to find those processes that require working set evaluation.

Page Fault Queue

The operating system maintains a queue of processes that have faulted for a page. Pager references this queue to locate the processes that require page fault processing.

Pager Message Table

The operating system monitors this table and uses its information to determine when to schedule the pager to process a pager message.

Process Package

Page fault processing requires read access to the following invisible package information in the process package of every process:

- Keys
- Access interrupt address
- Access interrupt cause bits

The CP memory manager Process Initialization function grants the pager access to the process package.

System Configuration Table Coldstart Attribute Block

At system initialization, CP memory manager uses the coldstart attribute block of the system configuration table to build page tables for processes loaded into CP memory by the service unit. This attribute block must specify the:

- Names of the domains that are in each server process
- Physical address of domain and server executable files
- Physical address and length of the system configuration table

System Configuration Table Process Image Attribute Block

The names of domains that are in the process image are used at system initialization and reconfiguration time to build a system domain address map template.

System Configuration Table Attribute Block

The global file ID of the domain's executable file is used at system initialization and reconfiguration time to build a system domain address map template.

Executable File Header

The following information is read by process initialization:

- Target page size
- Address map
- Initial paging file length

Central Processor Memory Manager Functions

The CP memory manager functions perform the tasks requested by CP memory manager. The specific CP memory manager functions are:

- **Page Fault Processing** satisfies a process' request for CP memory. Page fault processing is performed by a separate server process known as the *pager*. The processing of page faults consists of the following:

  - Validation of the virtual address to permit only page faults within accessible memory addressable files.

    Pager generates an access violation signal when a process accesses a virtual address which has no assigned key.

    Pager generates an access violation signal when a process accesses a virtual address which is assigned to a memory addressable file the faulting domain has no access to.

    An undefined address signal is generated when a process faults for a virtual address that has an assigned key but is not associated to a memory addressable file.

  - Allocation of paging file space for the faulted page.

    A page fault for an address within a large scratch space region will cause file space to be allocated in the paging file, and causes the virtual address to be associated to the paging file address.

  - Location of a replacement page, cleaning modified pages.

  - Generation of I/O requests to bring data into CP memory.

    Note that the I/O requests cause the page faulted process to wait for I/O completion, but not the pager.

    Because file extensions may be done during a write operation, and because pager can not be blocked for the duration of the extend file operation, file extensions are performed by another process. The extend server extends in serial. Thus, if two write operations cause extends, the second must wait for the first to complete before starting.

Pager writes modified pages of write temporary memory addressable files to the paging file instead of the logical disk file. If the page has not previously been written to the paging file, pager associates the page's virtual address to the paging file. Any subsequent faults for that page cause it to be read from the paging file.

- Generation of page table entries for the page requested by the page fault.
- Rescheduling of the process to the appropriate queue.

- **Evaluate Working Set** periodically measures a process working sets and the system working set and produces working set seconds for accounting.
- **Lock Transfer Address** permits the SM manager to obtain the physical addresses of CP memory pages that will be transferred to shared memory. If necessary, the pages are made resident in CP memory.
- **Unlock Transfer Address** notifies pager that the specified physical address may now be removed from CP memory.
- **Create Object** creates a shared CP memory object. This function does not allocate CP memory to the object, but it does define the total length of the object.
- **Destroy Object** destroys a shared CP memory object.
- **Allocate CP Memory to Object** allocates physical CP memory for a CP memory object. A request for multiple pages will optionally be allocated as contiguous physical pages. The smallest unit that can be allocated is one small page.

  This function also performs an Activate CP Memory Object.
- **Deallocate CP Memory From Object** deallocates pages from a CP memory object. If the page is activated in this process, a deactivate is performed.
- **Activate Object Page** associates the virtual memory of a process to a CP memory object logical address.
- **Deactivate Object Page** removes the ability to access the shared CP memory object associated to the specified virtual address range.
- **Assign Page to Predefined CPM Object** is used at system initialization to permit the locked-down server processes to record SU allocated CP memory as a CP memory object. There are predefined CP memory objects to which the CP

memory pages may be assigned. The smallest unit which may be assigned is one page.

- **CPU Initialization** initializes CP memory manager tables so that CP memory manager functions can be performed. This function is distributed between the system initialization process and the pager.

- **Mapin Paging File** allocates virtual memory to a process as scratch space. The paging file is used as the memory addressable file for scratch space.

- **Mapin File** allocates a memory addressable file, associating a logical disk file region to a virtual address range. An option is available to map in the file, thus permitting multiple processes to share the disk file in the same CP memory.

- **Mapout Address** disassociates a virtual address range from a disk file. Subsequent access to the address range causes an access violation.

  Mapout waits for any outstanding I/O to the specified address range to complete before processing the request. Mapout writes any CP memory resident modified pages out of CP memory and deallocates the physical pages.

  An option permits process management to perform the map out function for a virtual address without access permission to the address.

- **Mapout File** performs the map out function for all virtual address ranges of a specified logical disk file.

- **Allocate Virtual Memory** allocates virtual address space in the system heap. The space is guaranteed to not overlap other memory addressable files.

  An option is available to request address space that is not mapped to the paging file. Otherwise, the request maps the address range to the paging file.

- **Update CP memory** writes the modified CP memory pages within the specified virtual address range to the associated memory addressable files.

- **Initialize Process** initializes the CP memory manager tables necessary prior to execution of a new process.

  An option is provided to reinitialize a suspended process from a paging file.

This function accepts a list of requests to:

- Allocate CP memory objects to the calling process
- Allocate a process package page to a CP memory object in the calling process
- Activate CPM objects in the new process

CP memory manager gives the pager process access to the process package pages.

- **Initialize Process Address Map** initializes a process' address map from its executable file and process image system address map template.
- **Terminate Process** deallocates paging file and CP memory manager tables for a terminating process.

  All CP memory object pages are deactivated. This function performs the map out operation for any remaining system code and data for the terminating process. The paging file is closed.

  An option is available to specify whether or not to destroy the paging file.
- **Create Paging File** creates a paging file. The initial paging file length is obtained from the executable file header.

Shared Memory Manager

The shared memory (SM) provides for storage of the problem variables for applications. Additionally, shared memory provides temporary storage for code and data that is being transferred between the disk subsystem and CP memory.

Shared memory consists of two equal halves, each half ranging in size from 32 million words to 128 million words (256 million words total). There are eight high speed ports for CPU connections to SM (four for each half), and 20 low speed ports (10 each half) for I/O Unit (IOU) and Service Unit (SU) connections.

All access to the shared memory from CPUs, the service unit, and the I/O units is through the SM manager. The SM manager supports operations on *shared memory objects* and *file objects.*

Shared memory objects define regions in shared memory that can be used for data storage. The caller may:

- Create a shared memory object
- Read and write information in the shared memory object
- Destroy the shared memory object

Data in a shared memory object resides in shared memory for the life of the object.

File objects permit the caller to store or retrieve data from a device. File objects are built and maintained by the file directory/catalog manager. The file support module is called to open a file object, and calls SM manager.

Once the file object is open, the caller utilizes SM manager's read/write operations to access data in the file. During a read/write operation, SM manager calls the disk physical-file system to move the data from the physical location on the disk.

Only the file support module, pager, and the record manager may call SM manager file object operations. SM manager provides no security for file objects.

Externalized File Objects

File objects are built and maintained by the file directory/catalog manager. SM manager supports operations on file objects that permit the caller to access data from a device. All data moves through SM, there is no direct path from a device to CP memory.

Data from a file object can reside in either shared memory or on the specific device. The caller specifies a *logical file address*, it is the responsibility of SM manager to find where the data resides and move the information into CP memory.

Shared Memory Manager does not provide mutual exclusion across processors or across file opens.

SM manager allocates shared memory to files in fixed length blocks. To receive the greatest benefit of shared memory, the necessary data must be in shared memory when the caller needs it.

Shared Memory Objects

SM objects define regions in shared memory that can be used for data storage. Information in SM objects always resides in shared memory.

The intent of SM objects is to provide data storage that permanently resides in shared memory for system domains and special processes. The application programmer is not permitted to use SM objects.

Although these objects reside in shared memory, they are not necessarily shared between processes. It is possible for a system domain to:

- Create an SM object for each process
- Use of the object for data storage while the process is executing
- Destroy the object when the process terminates

SM manager does not provide *mutual exclusion* for SM objects. Mutual exclusion is provided by the interprocess communication feature.

Because data in SM objects permanently resides in shared memory, there are restrictions on the number of SM objects that may be created. In many cases, file objects may be used instead of SM objects.

Memory Management Requirements

SM manager will migrate data into and out of shared memory based on expected future usefulness and application advisory actions. SM manager allocates shared memory in fixed length blocks. Shared memory is not directly accessible by the caller.

SM manager uses the following rules when allocating space in shared memory:

1. All operations in shared memory compete globally for blocks in shared memory.
2. SM manager will attempt to replace the least recently allocated block in shared memory.

SM manager allocates memory in fixed length blocks. Because memory is allocated in blocks, most data movement between shared memory and the I/O unit is performed in multiples of the block length.

Shared Memory Object Procedural Interfaces

There are six SM object procedural interfaces:

- **Create Shared Memory Object** creates an SM object. An SM object is a region of space in shared memory that can be used for data storage. An SM object resides in shared memory for the life of the object. The caller creates an SM object by specifying an object name and length.

  SM manager returns a capability to the SM object. The capability is used when accessing or destroying the object. The creator of the object must pass the capability to any other process or domain that is going to share access to the object.

- **Read Shared Memory Object** provides the interface that permits the caller to read data in an SM object. SM manager returns the data to a CP memory buffer specified by the caller. SM manager does not provide mutual exclusions for operations on SM objects. Any number of processes may be reading and writing an object at the same time.

- **SU/IOU Read Shared Memory Object** permits the service unit or I/O unit to read an SM object. SM manager returns the data to a buffer specified by the caller.

- **Write Shared Memory Object** provides the interface that permits a caller to write data into an SM object. SM manager writes the data from the CP memory buffer specified by the caller. SM manager does not provide mutual exclusion for operations on SM objects. Any number of processes may be reading and writing an object at the same time.

- **SU/IOU Write Shared Memory Object** permits the service unit or I/O unit to write an SM object. SM manager writes the data from the buffer specified by the caller.

- **Destroy Shared Memory Object** destroys an SM object. All data residing in the object is destroyed.

File Object Procedural Interfaces

There are eight file object procedural interfaces:

- **Open File Object** opens a file object for access. The file directory/catalog manager information is passed to SM manager during the open. Only the file support module is permitted to make this call. All callers attempting to open a file object must call the file support module. The file support module will call SM manager. Multiple opens are permitted on each file object, and each open is permitted to have different attributes.

- **Read File Operation** moves data into CP memory. The caller specifies a logical file address and a virtual CP memory range. It is SM manager's responsibility to find where the data resides (SM or disk) and deliver the data to CP memory.

  An operation identifier is returned with every disk read operation. The caller uses this identifier to check for completion of the operation. A read file operation is considered complete when the data is moved into a CP memory buffer.

- **Write File Operation** moves data from CP memory to disk. The call, however, does not guarantee that the data is *immediately* written to disk. To force the data to disk, the caller must either use the Flush File operation or close the file object.

  SM manager returns an operation identifier to be used to check for completion of the write file operation. The operation is considered complete when the data is moved out of the CP memory buffer into shared memory.

- **Pager Transfer File Operation** is intended for use only by pager. It permits pager to transfer data to and from a physical range in CP memory.

  An operation identifier is returned. Pager uses this operation identifier to check for completion of the operation. A pager transfer of file operation is considered complete when the data is moved into or from the CPU.

- **Flush File** writes a copy of all modified data in shared memory for an open file out to the disk file. An operation identifier is returned which may be used to check for completion of the operation. A flush file operation is considered complete when a copy of all modified data is written to the device. The data also still resides in shared memory after the operation is completed.

- **Check for Operation Completion** returns the status of a file operation. If the wait option is used, the routine waits for the completion of the operation.
- **Pager Check for Operation Completion** permits pager to check for file operation completion or completions. A list of completed identifiers is returned.
- **Close File Object** informs SM manager that the file will no longer be accessed through the specified open identifier. All modified data in shared memory is written to disk.

Miscellaneous Procedural Interfaces

There are four miscellaneous procedural interfaces:

- **System Initialization** is called by system initialization to permit SM manager to initialize shared memory.
- **Completion Server** is used to complete object operations. The routine checks for completed disk and memory transfer requests. Any process waiting for a completed operation will be restarted.
- **Process Initialization** is called by each process to allow SM manager to initialize its process local tables.
- **Process Termination** is called at process termination to allow SM manager to clean up any remaining operations initialized by the terminating process.

Shared Memory Objects versus File Objects

There are some trade-offs to be considered when using shared memory and file objects.

Shared Memory Objects

SM objects are used only by the system, never by users. The properties of SM objects are:

1. SM objects permanently reside in shared memory, no disk space is allocated to SM objects.

2. The I/O unit or service unit may directly read or write SM objects. A file object cannot be accessed directly from either the I/O unit or service unit (It is possible to read the file object in a CPU and send the data to an I/O unit or service unit using interprocess communications).

3. If the information is needed by SM manager to access the file object, an SM object must be used. This restriction applies mostly to the interprocess communication and logical file system features.

4. SM manager does not automatically extend SM objects. It is up to the caller to decide how much and when the object is to be extended.

File Objects

The properties of file objects are:

1. Disk space is allocated to a file object. The amount of disk space allotted depends on the amount of data written, and the device class where the file resides.

2. Information in a file object does not have to reside in shared memory. SM manager will page the information into and out of shared memory based on how the information is being used.

3. More information may be kept in a file object than in an SM object. Since data in an SM object resides in shared memory, there is a limit to the amount of memory that may be allocated to SM objects.

Communication Buffer Management

CB Manager (CBM) manages the use of the communication buffer. The communication buffer is a high-speed memory used to communicate information between processes that may be running in any of the processors of the computer system.

The Communication Buffer is divided into two discrete halves, side 0 and side 1. The size of the memory (both halves) is one million words, which may be expanded to four million words.

Each side of the Communication Buffer has its own Communication Buffer Interface (CBI) that connects the communication buffer with the other components of the computer system. Each side has 10 ports, one for each of the eight possible CPUs, and two I/O interfaces (IOIs). The CPUs, I/O units, and the service unit access the communication buffer through these ports. The logical connections between the sides of the communication buffer and the computer system processors is illustrated in figure 4-15.

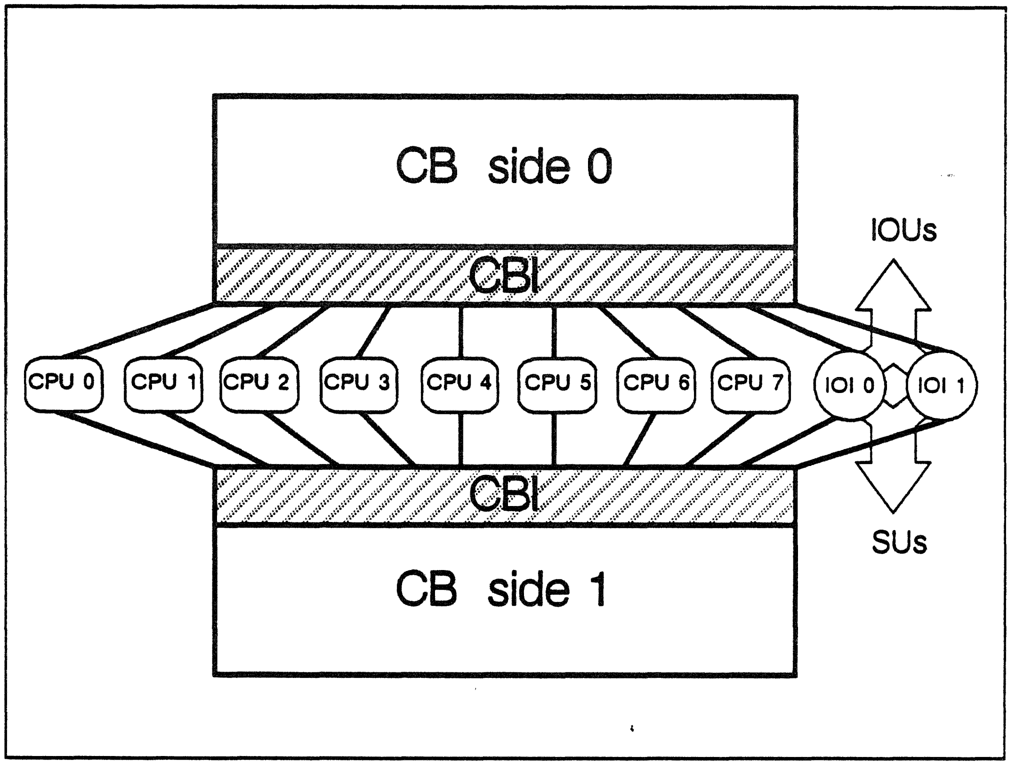


Figure 4-15. Logical relationship between the Communications Buffer sides and the computer system processors.

The operations performed by the communication buffer hardware are:

- Transfers between the communication buffer and I/O units, service unit memory, or CPU registers
- Semaphore post and wait operations
- Conditional swaps between the communication buffer and an I/O unit or service unit memory word, or a CPU register (these operations are commonly referred to as *bit test and swap*)
- Conditional stores of a CPU register, or I/O unit or service unit memory word, into the communication buffer and loads of the following communication buffer word (these operations are commonly referred to as *bit test and load/store*

The CPUs have special hardware to protect areas of the communication buffer that belong to a *domain*. Each domain has its own specific range of central processor memory addresses which it can access, and each has its own set of operating registers known as a *domain package*.

The domain package contains registers that denote the lowest communication buffer address the domain can access, the *base address*, and the highest communication buffer address the domain can access, the *limit address*. The domain package also contains the access rights that determine the operations the domain is allowed to perform on that range of communication buffer memory. This information fits into two registers, referred to as a Base/Limit/Access Pair (BLAP). Figure 4–16 illustrates how the base/limit/access pairs in a domain package specify address ranges.

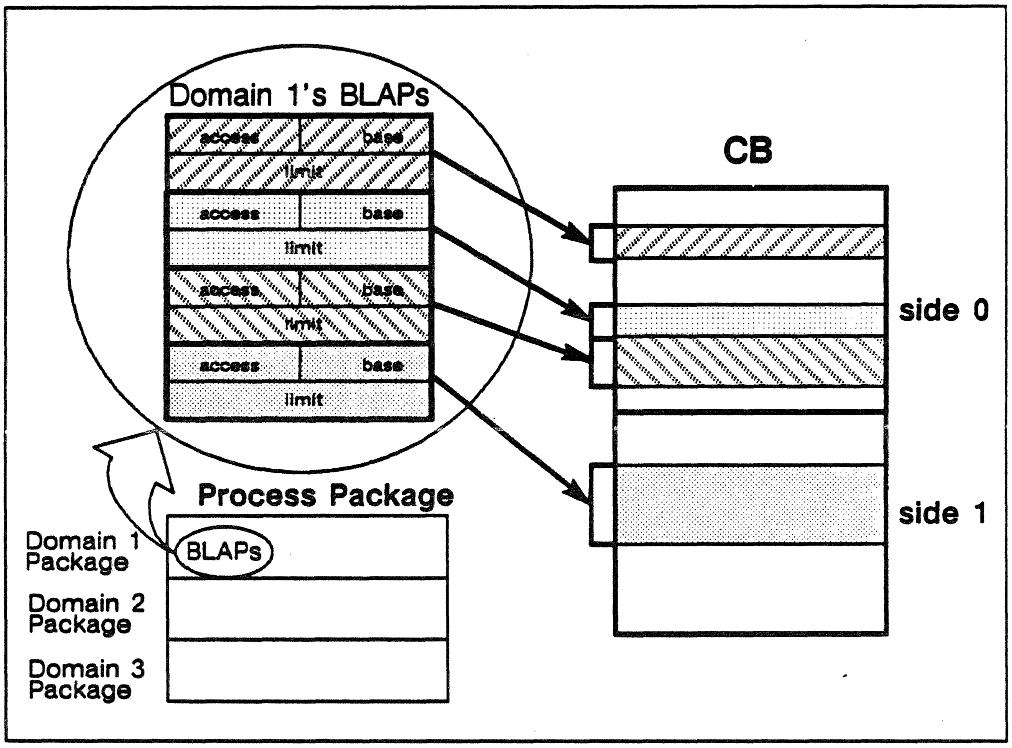


Figure 4–16. Base/limit/access pairs in the domain package specify address ranges in memory that the process has access to.

For efficiency, each domain has four base/limit/access pairs, permitting a domain to use up to four different ranges of communication buffer memory at once.

The CPU hardware protects the communication buffer in a manner analogous to CP memory protection. If a domain attempts an operation in an area of the communication buffer that is not specified in the domain's base/limit/access pairs, an access violation results and the operation is not performed.

The use of communication buffer hardware is controlled by the CB manager. This software enables an operating system kernel feature to reserve a portion of the communication buffer for its own use, with the assurance that its portion is protected from unauthorized or inadvertent manipulation.

The CB manager features differ depending on the processor type:

- On a CPU, CB manager performs management functions
- On the I/O units and the service units, CB manager uses management information generated by the CPU functions to operate on the communication buffer.

Communication Buffer Objects

The portions of the communication buffer that an operating system kernel feature reserves are called *CB objects*. Each object has a unique name assigned to it, and access rights that determine the type of operations that may be performed on it, and who may perform those operations.

CB manager controls the physical placement of CB objects. The operating system kernel accesses CB objects by name via the CB interfaces. CB Manager relates each name to its physical location by manipulating a base/limit/access pair. Thus, a CB object is *logically* accessed by its name, and *physically* accessed by its base/limit/access pair.

There are three types of CB objects:

- **Permanent** objects remain in the system until the system is initialized
- **Transient** objects lasts as long as there is any process still using them
- **Queue** objects have a permanent lifetime

A queue object may have communication buffer semaphore instructions performed on it; these instructions can not be performed on the other object types.

Queue objects are also available to a restricted set of operating system kernel features. Other kernel features may use permanent and transient objects; application features are limited to transient objects.

CB manager defines the logical structure of the communication buffer. The CB manager views the communications buffer as a series of groupings of similar type objects. These groupings are called *regions.*

Regions are analogous to a subdirectory. Regions minimize fragmentation. The initial size of each region is determined by a system configuration parameter, and cannot be changed until the next system initialization. An entire side of the communication buffer is known as a *primary region*.

All other regions are contained within a primary region; no region straddles a communication buffer side.

Communication buffer sides may have different region configurations. Each region has a header that contains information about the region. The logical structure of a typical communication buffer region is illustrated in figure 4–17.

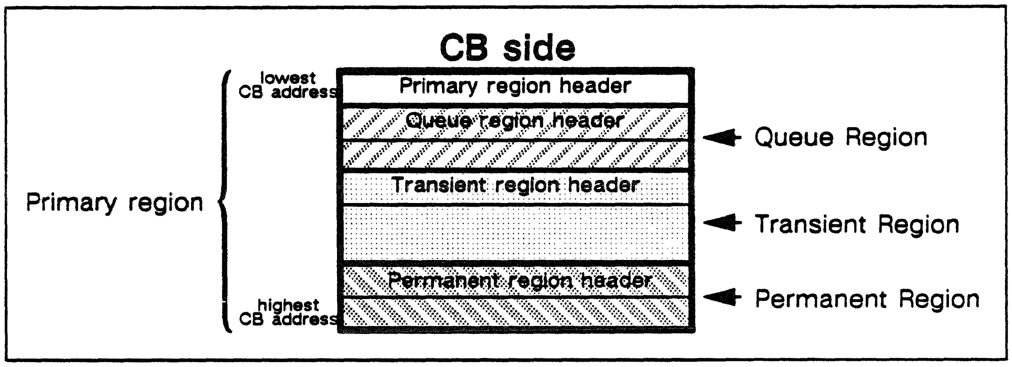


Figure 4–17. A Communication buffer side is typically divided into regions for the three object types.

Communication Buffer Management in the CPU

CB manager in the CPU manages access rights and the physical placement of CB objects within the communication buffer. The actual communication buffer instructions (e.g. load, store), however, are issued directly by the object's user.

CB manager in the CPU provides these functions to manage CB objects:

- Create a CB object
- Destroy a transient CB object
- Grant access to a CB object
- Get information about a CB object
- Activate (set a base/limit/access pair for) a CB object in a domain
- Deactivate (clear a base/limit/access pair for) a CB object in a domain
- Preactivate (set a pseudo base/limit/access pair for) an I/O unit or service unit CB object

An Example of CB Management in a CPU

The capabilities feature communicates information between several CPU processes. Within each process, the domain management feature must also read the information. The communication is accomplished via CB manager on the CPU:

1. The capabilities feature creates a CB object for its information, which reserves a portion of the communication buffer.

2. To retrieve the communication buffer attributes of the object, the capabilities feature gets information about the object.

3. The capabilities feature grants read access to the object, so that either feature can activate the object.

   . Activation sets a base/limit/access pair within the domain package to the object's communication buffer address range. The activating domain may now legally issue communication buffer instructions for the object:

   - If domain manager activates the object, the access rights in the base/limit/access pair limit domain manager's domain to read (i.e. communication buffer load instruction).

   - If the capabilities feature activates the object, the access rights are read and write (i.e. communication buffer load and store instructions).

4. When either domain manager or the capabilities feature is done accessing the object, it is deactivated, which clears the base/limit/access pair associated with the object.

5. If the object is transient, the capabilities feature can destroy it to prevent subsequent processes from activating the object, and to have the object removed from the communication buffer when the last process using it terminates.

The CB Manager in the CPU also provides functions that perform general services for the operating system:

- Initialize CB manager for the CPUs

- Get system information

- Initialize CB management for a process

- Terminate CB management for a process

- Terminate CB management for a process cluster

Communication Buffer Management in the I/O and Service Units

CB management for the I/O units and service units uses management information generated by CB management on the CPU to operate on preallocated portions of the communication buffer. They do not manage storage objects themselves.

A CB object to be used by an I/O or service unit feature must be allocated during system initialization via the communication buffer on the CPU. During initialization, CB manager in the CPU records pertinent information about the object as a *pseudo base/limit/access pair*. A pseudo base/limit/access pair contains the same information as a normal base/limit access pair, but is enforced by software instead of hardware.

Later, when an I/O or service unit feature accesses a CB object, CB manager in the CPU refers to the object's pseudo base/limit/access pair for the physical location of the object, and its access rights.

It is important to note that unlike CB management in the CPU, the communication buffer instructions are issued via CB management in the I/O units or service units; the I/O and service unit features do not directly issue communication buffer instructions. Thus, for the I/O unit to execute a load from the communication buffer, the I/O unit feature calls a procedure in the CPU rather than issue the load instruction directly.

CB management functions in the I/O unit are asynchronous; the caller does not need to wait until its CB management operation is complete before continuing with the next operation. CB management operations in the service unit, however, are synchronous; the caller must wait for each CB management operation to complete before continuing.

CB management in the I/O units provides functions that perform communication buffer operations on preallocated objects (via CB management in the CPU):

- Asynchronous semaphore post
- Asynchronous semaphore wait
- Asynchronous, bit test and swap
- Asynchronous, bit test and load/store
- Asynchronous load from communication buffer
- Asynchronous store into communication buffer

CB management for the I/O units also provides general services for the operating system:

- Initialize CB management for an I/O unit

CB management in the service unit provides functions that perform communication buffer operations on preallocated objects (via CB management in the CPU):

- Synchronous semaphore post
- Synchronous semaphore wait

- Synchronous, bit test and swap
- Synchronous, bit test and load/store
- Synchronous load from communication buffer
- Synchronous store into communication buffer

CB management in the service unit also provides general services for the operating system:

- Initialize CB management for a service unit

CB Management Between the CPU and I/O Unit

Information often needs to be communicated between processes running on the CPUs and the I/O units. This communication is accomplished via CB management in the CPU and I/O Unit:

1. During system initialization on the CPU, a permanent or queue CB object is created via CB management in the CPU.
2. The object is preactivated as an I/O unit object via CB management in the CPU.
3. After initialization, authorized CPU features may operate on the object. On the I/O unit, features simply invoke the appropriate CB management function in the I/O unit, specifying the name of the object to be operated upon.
4. CB management in the I/O unit retrieves the pseudo base/limit/access pair information for the object, and issues the communication buffer instruction. Thus, to write to a preallocated CB object, an I/O unit feature calls CB management in the I/O unit for the Asynchronous Store into Communication Buffer function.

Communication Buffer Restrictions and Limitations

There are several restrictions and limitations associated with the communication buffer that should be noted:

- The communication buffer is not a paged memory. A CB object, therefore, must be a contiguous piece of the communication buffer.

  Each object's range is denoted by a base/limit/access pair, so that a domain can use a maximum of four CB objects at once.

- A transient CB object is deleted only after the last process accessing the object terminates. Thus, a transient CB object will not be immediately destroyed if at least one process that activated the object is still executing. Permanent and queue objects cannot be destroyed.

- No relocation or swapping of CB objects may be performed. Once an object is allocated in the communication buffer, it does not move.

- All transient objects are restricted to a fixed size. The size is set at system initialization, and cannot be changed during the life of the system. This eliminates communication buffer over committment and fragmentation problems in the transient object region.

- Queue objects may be used by a restricted set of operating system kernel features. Communication buffer semaphore instructions manipulate physical communication buffer address information within a communication buffer process word, and, if improperly used, can have detrimental effects on CB management, and undermine the hardware protection of the base/limit/access pairs. The operating system,features that have demonstrated a critical need for queue objects are the only features that CB management will permit to execute these instructions. Those features that are permitted are:

  - Semaphores
  - Remote process communication
  - Network management features
  - Global scheduler

- The sizes and locations of all regions are fixed at system initialization. There is no dynamic communication buffer reconfiguration.

- There is no recovery from a lost communication buffer side. Controlled deconfiguration of a communication buffer side is not available. Recovery from a catastrophic communication buffer hardware failure is not possible.

- There are no maintenance objects. The requirements for reserving portions of the communication buffer for maintenance and diagnostic purposes have not been defined.

- There is no flaw support for the communication buffer. The requirements for marking portions of the communication buffer as flawed have not been defined.

Communication Buffer Management Externalized Objects

CB management provides *CB objects* for operating system kernel features. CB objects are contiguous, non–overlapping areas of CB memory. They are the basic logical units of the communication buffer. They are managed and manipulated via the CB management functions.

CB management protects CB objects according to the specified criteria, the *access rights*.

A CB object is a contiguous block of the communication buffer created via CB Management in the CPU. The size of an object is fixed at creation; it cannot be changed. Its maximum size is the size of the communication buffer region in which it resides. An object may reside in either communication buffer side, as long as a region appropriate to the object exists there. A specific side may be requested.

There are the three CB object types: *permanent, transient,* and *queue*. The major differences between the types are:

- The communication buffer semaphore instruction access rights are allowed for a *queue* object, but they are forbidden for a *permanent* or *transient* object.
- *Permanent* and *queue* objects remain in the communication buffer until the next system initialization, while a *transient* object remains in the communication buffer only until the termination of the last process within the owning process cluster, and the termination of every other process that activated it.

Communication Buffer Object Access Rights

It is the responsibility of CB management to protect each object from inadvertent or malicious corruption. CB management in the CPU accomplishes this by the manipulation of a domain's base/limit/access pairs.

CB management in the CPU ensures that each domain's base/limit/access pairs correspond exactly to the domain's CB objects; this enables hardware access validation. An operating system kernel feature executing in one CPU can directly issue communication buffer instructions; the CPU hardware guarantees the domain cannot access areas of the communication buffer outside its base/limit/access pairs.

It is the responsibility of CB management in the I/O units and service units to provide the same protection, even though there is no hardware support for the base/limit/access pairs on these processors. The task is accomplished by allowing only communication buffer on either of these processors to issue communication buffer instructions, so that validation may be performed by software.

CB management in the CPU also provides operations so that each feature can protect its CB objects from other features in the same domain. For this intra-domain protection, a feature:

1. Activates the object immediately before its use.
2. Does not call other features (except CB management) while the object is activated.
3. Deactivates the object immediately after its use.

CB management protects a CB object according to its access rights. The access rights are the operations a caller is allowed to perform upon a CB object:

- The right to execute a communication buffer load instruction
- The right to execute a communication buffer store instruction
- The right to execute a communication buffer bit test and swap, or a bit test and load/store instruction
- The right to execute a communication buffer semaphore instruction
- The right to grant access rights to the object to other callers
- The right to destroy a transient object

Note that the communication buffer instruction access rights are the same as the access bits in the domain base/limit/access pair; these are enforced by the hardware. The other rights are enforced only by CB management in the CPU. For CB management in the I/O units and service units, all rights are enforced by software.

The creator of an object specifies the maximum set of access rights during allocation. The creator may grant all or a subset of these access rights to others, called *grantees*, and may allow the grantees, in turn, to grant access rights to still others.

A grantee may be specified in two ways, depending on the intended scope of the access rights:

1. **Domain Feature ID**. In any process, any domain that contains the specified feature is granted rights. This is illustrated in figure 4–18.

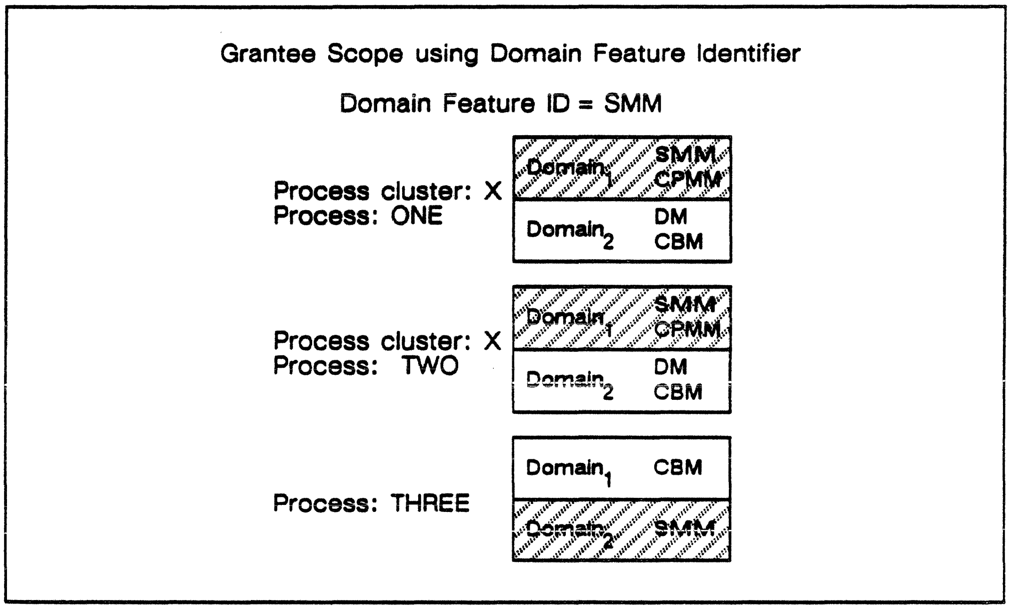


Figure 4–18. Granting of domain privileges using domain feature identifier.

2. **Process Cluster ID and Domain Feature ID**. In processes within the process cluster, any domain that contains the specified feature is granted rights. This is illustrated in figure 4–19.

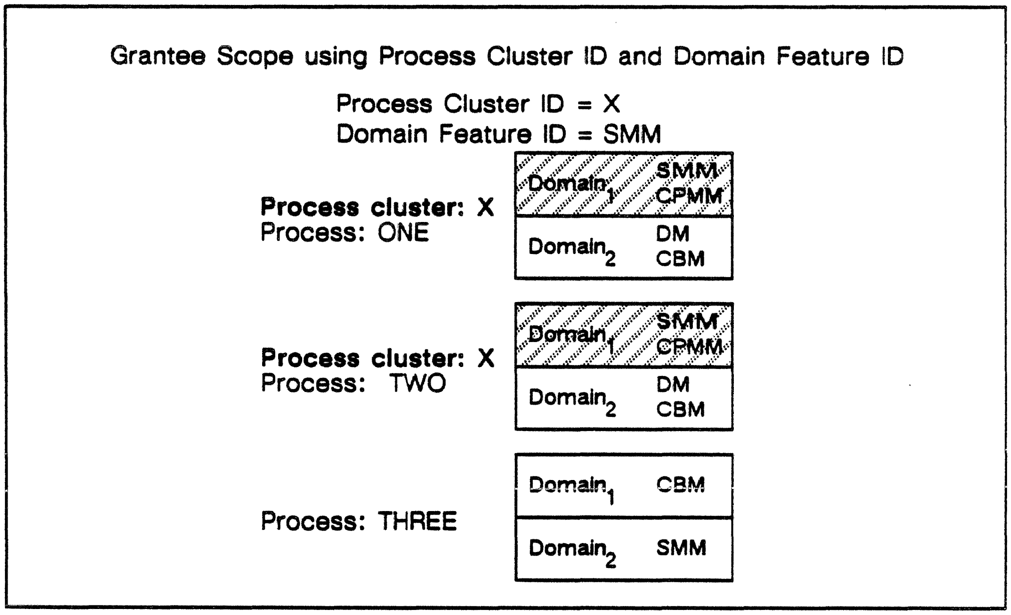


Figure 4–19. Granting of domain privileges using cluster ID and domain feature ID.

Permanent Communication Buffer Objects

A permanent object is to be used by system features that need to communicate information between all processes throughout the life of the system.

The minimum size of a permanent CB object is one communication buffer word. The maximum size is the size of the region in the Communication Buffer that holds permanent objects. Individual features, however, may be limited to a smaller size to prevent resource exhaustion.

CB management assigns a name to each permanent object. The name is subsequently used as an input parameter to CB management object operations. The name must be unique. If a unique name cannot be generated, the object will not be allocated. A permanent object name consists of:

- **Domain Feature ID**. Identification of the feature that created the object.
- **Symbolic ID**. A character string specified by the creator of the object. It is the responsibility of each operating system feature to ensure that the symbolic ID is unique within the feature.

The access rights that can be granted for a permanent object are:

- Read (load)
- Write (store)
- Bit branch (conditional swap or load/store)
- Grant rights

The grantee's scope can be either domain feature ID, or process cluster ID and domain feature ID.

The operations of a permanent communication buffer object may be summarized as follows:

1. To *create* a permanent object, the creator specifies its domain feature ID and a symbolic ID. This is verified by CB management via domain management interfaces. A permanent object with a unique name is created. The object remains in the communication buffer until the next system initialization.

2. To *grant access, activate,* or *get information*, the caller specifies the object's name, and its own domain feature ID (and process cluster ID if access was granted in that manner). The caller's identification is verified, and checked with the object's list of authorized users. If authorized, the requested operation is performed.

3. To *deactivate* a permanent object, the caller specifies the object's name. If the object was activated (i.e. a base/limit/access pair is set for the object), the appropriate base/limit/access pair is cleared.

Transient Communication Buffer Objects

A transient CB object is to be used by any caller, application or system, that needs to communicate information between a specific set of processes (i.e. a process cluster). The object is automatically deleted when the last process in the set terminates.

All transient objects are the same size. This size is fixed at system initialization time by a system configuration parameter.

CB management assigns a name to each transient object. This name is subsequently used as an input parameter to CB management object operations.

The name must be unique. If a unique name cannot be generated, the object will not be allocated. A transient object name consists of:

- **Process Cluster ID**. The identification of one of the creator's process clusters. The termination of this cluster causes automatic destruction of the object. The creator must be a member of this process cluster, at least while the object is being created.

- **Symbolic ID**. A character string specified by the creator of the object. It is the responsibility of each operating system kernel feature to ensure that the symbolic ID is unique within the feature.

The access rights that can be granted for a transient object are:

- Read (load)
- Write (store)
- Bit branch (conditional swap or load/store)
- Grant rights
- Destroy rights

The grantee's scope can be either domain feature ID, or process cluster ID and domain feature ID.

The operations of a transient CB object may be summarized as follows:

1. To *create* a transient object, the creator specifies its domain feature ID, the appropriate cluster ID, and a symbolic ID. The creator's domain feature ID is verified by CB management via domain management interfaces, and its process cluster ID is

verified by CB management via process management interfaces. A transient object with a unique name is created. The object remains in the communication buffer until either it is destroyed or the last process in the process cluster terminates. Access is granted to the creator on a process cluster ID and domain feature ID scope when the object is created.

2. To *grant access, activate, get information,* or *destroy* a transient object, the caller specifies the object's name, and its own domain feature ID (and process cluster ID if access was granted in that manner). The caller's identification is verified, and checked with the object's list of authorized users. If authorized, the requested operation is performed.

   For a destroy operation, the object is not immediately destroyed if it was activated by one or more processes that are still executing. When these processes terminate, the object is destroyed.

3. To *deactivate* a permanent object, the caller specifies the object's name. If the object was activated (i.e. a base/limit/access pair is set for the object), the appropriate base/limit/access pair is cleared.

Communication Buffer Queue Objects

A queue object is a specialized type of a permanent object. It is intended for use by a restricted set of operating system kernel features. Queue objects allow the operating system kernel features to make more effective use of communication buffer hardware via communication buffer semaphore instructions, without impacting CB management's operation.

A queue object can only be allocated by certain operating system kernel features. Communication buffer semaphore instructions can be directly performed on a queue object by these features.

Since a semaphore operation requires communication buffer physical addresses, CB management puts the beginning physical address of a queue object in the first word of the object when it is allocated. This ensures that a queue object's users can calculate the physical address of any location within the object.

The minimum size of a communication buffer queue object is two communication buffer words: one for the starting physical address of the queue object, and one for data. The maximum size of a queue object is the size of the region in the communication buffer that holds queue objects.

CB manager assigns a name to each permanent object. The name is subsequently used as an input parameter to CB management object operations. The name must be unique. If a unique name cannot be generated, the object will not be allocated. A permanent object name consists of:

- **Domain Feature ID**. Identification of the feature that created the object.

- **Symbolic ID**. A character string specified by the creator of the object. It is the responsibility of each operating system feature to ensure that the symbolic ID is unique within the feature.

The access rights that can be granted for a permanent object are:

- Read (load)
- Write (store)
- Bit branch (conditional swap or load/store)
- Semaphore (post, wait)
- Grant rights

The grantee's scope can be either domain feature ID, or process cluster ID and domain feature ID. The semaphore rights can only be granted to interprocess communication features.

The operations of a permanent CB object may be summarized as follows:

1. To *create* a permanent object, the creator (one of a limited set of operating system kernel features) specifies its domain feature ID and a symbolic ID. This is verified by CB management via domain management interfaces. A permanent object with a unique name is created. The object remains in the communication buffer until the next system initialization.

2. To *grant access, activate*, or *get information*, the caller specifies the object's name, and its own domain feature ID (and process cluster ID if access was granted in that manner). The caller's identification is verified, and checked with the object's list of authorized users. If authorized, the requested operation is performed.

3. To *deactivate* a permanent object, the caller specifies the object's name. If the object was activated (i.e. a base/limit/access pair is set for the object), the appropriate base/limit/access pair is cleared.

Communication Buffer Management Functions

There are a number of CB management functions that perform the tasks of the communication buffer:

- **Activate BLAP for Object** is a CPU function that sets up a domain's base/limit/access pair (BLAP) registers so that the named object can be accessed by an authorized feature.

- **Create CB Object** is a CPU function that allocates a unique portion in CB memory for the specified object. A unique name is also assigned to the object.

- **Deactivate BLAP for Object** is a CPU function that clears a domain's base/limit/access pair (BLAP) registers associated with a specific object. The object, therefore, cannot be accessed by the domain until it is reactivated.

- **Destroy Transient CB Object** is a CPU function that deletes a transient CB object when all processes that used the object have terminated.

  It should be noted that a transient object is automatically destroyed when the last process in its process cluster terminates. This interface, therefore, should not be called unless there is a special need to delete the transient object before the process cluster terminates.

- **Get CB Object Information** is a CPU function that retrieves CB management directory information about a particular object to authorized callers.

- **Get System Information** is a CPU function that retrieves system level information about communication buffer usage.

- **Grant Access to CB Object** is a CPU function that adds access rights to a specific CB object for the specified grantee.

- **Initialize CB Management for the CPUs** is a CPU function that initializes the CB management feature during system initialization of CPUs. This includes building CB management's directory structures to reflect the system's physical and logical communication buffer configuration.

- **Initialize CB Management for a Process** is a CPU function that initializes CB management within a new process. This function also sets up the CB management domain data structures.

- **Preactivate a CB object for IOU or SU** is a CPU function that preactivates a CB object that will be used by an I/O unit or service unit feature. A pseudo base/limit/access pair is created for later use by an I/O unit or service unit feature so that base and limit addresses and access rights can be associated to an existing CB object.

- **Terminate CB Management for a Process** is a CPU function that terminates CB management within an executing process.
- **Terminate CB Management for a Process Cluster** is a CPU function that terminates CB management within a process for the specified process cluster.
- **CB Bit Test and Load/Store** is an I/O unit and service unit function that issues a bit test and load/store communication buffer instruction for a preactivated CB object.
- **CB Bit Test and Swap** is an I/O unit and service unit function that issues a bit test and swap CB instruction for a preactivated CB object.
- **CB Load** is an I/O unit and service unit function that issues a load communication buffer instruction for a preactivated CB object.
- **CB Semaphore Post** is an I/O unit and service unit function that issues a semaphore post CB instruction for a preactivated CB object.
- **CB Semaphore Wait** is an I/O unit and service unit function that issues a semaphore wait communication buffer instruction for a preactivated CB object.
- **CB Store** is an I/O unit and service unit function that issues a store communication buffer instruction for a preactivated CB object.
- **Initialize CB Management for an IOU** is an I/O unit function that initializes the CB management feature during initialization of the computer system I/O units. Interprocess communication mailboxes are established between CB management in the I/O unit and the data pipe controller.

  A CB management server process is started on the I/O unit so that asynchronous communication buffer operations can be performed. The CB management CPU data for preactivated CB objects is also retrieved.

  Note that CB management in the I/O unit determines which communication buffer instructions must be issued; the data pipe controller issues the communication buffer instructions and returns completion status to CB management in the I/O unit.

- **Initialize CB Management for a service unit** is a service unit function that initializes the CB management feature during initialization of the computer system service units. Interprocess communication mailboxes are established between CB management in the service unit and the maintenance interface driver. The CB management CPU data for preactivated CB objects is also retrieved.

  Note that CB management in the service unit determines which communication buffer instructions must be issued; the maintenance interface driver issues the instructions and returns completion status to CB management in the service unit.

Section 6: The Process Managers

This part examines the three features that schedule, manage, and communicate between processes in the computer system. It consists of three sections:

- Global scheduler
- Process management
- Remote procedure calls

The Global Scheduler

The global scheduler (GS) is the part of the operating system responsible for scheduling sessions and processes within the multiple CPUs of the computer system. Additionally, global scheduler provides functions for the management and control of system input and output queues.

Global scheduler is a system supervisor responsible for coordinating activities at a global level to optimize the computer system's activity. It is heavily dependent upon the accuracy of the advice and information supplied by other system resource managers.

Global scheduler is distributed across all of the computer system's CPUs. It interfaces with many operating system features, providing functional interfaces at more than one level.

There are three functional levels to the global scheduler:

- Queue management subsystem
- Session scheduler
- Process scheduler

The Queue Management Subsystem

The queue management subsystem (QMS) provides a generalized queueing facility for use by global scheduler.

An input queue is an ordered list of session input *queue items* waiting to be processed by the system. An output queue is an ordered list of queue items waiting to be transferred out of the system. A queue item contains queueing information about a particular file.

The queue management subsystem maintains a higher level object known as a *queue set*. The queue set is used to represent a collection of queues used by one or more system features.

In order for a system feature to use a queue managed by the queue management subsystem, it must have access permission to the queue set. Any system feature can call the queue management subsystem to have a queue set created for it. There is no limit to the number of queue sets a system feature can use.

The queue management subsystem supports several features:

- Ability to add queue items to the tail of any queue.
- Direct access to any queue item within a queue set.
- The ability to place queue items in a hold state.
- Support for the listing of items in a queue.
- Access control for queue sets.
- Support functions for the recovery of queues.

The queue management subsystem database is kept in one file, which contains only queue-item information. A tree of directories and files is used to represent the structure of a queue set, queue, and queue items. Using a directory tree for storage of queues allows the logical file system to manage access control, and it facilitates queue recovery.

The hierarchical relationship of objects managed by the queue management subsystem is illustrated in figure 4-20.

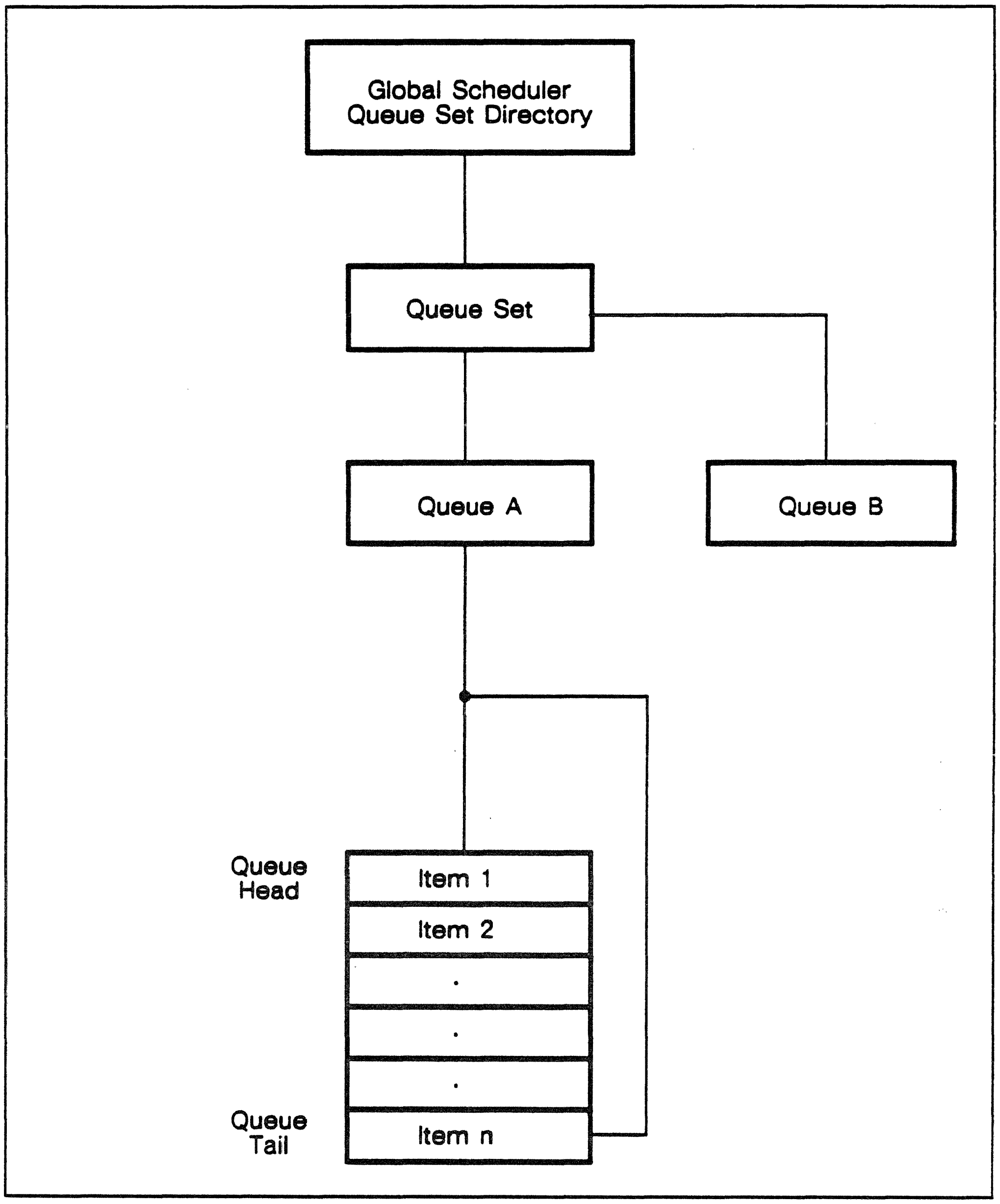


Figure 4-20. Hierarchical relationships of objects managed by the queue management subsystem.

The relationship of the queue management subsystem procedure to the user environments is illustrated in figure 4–21.

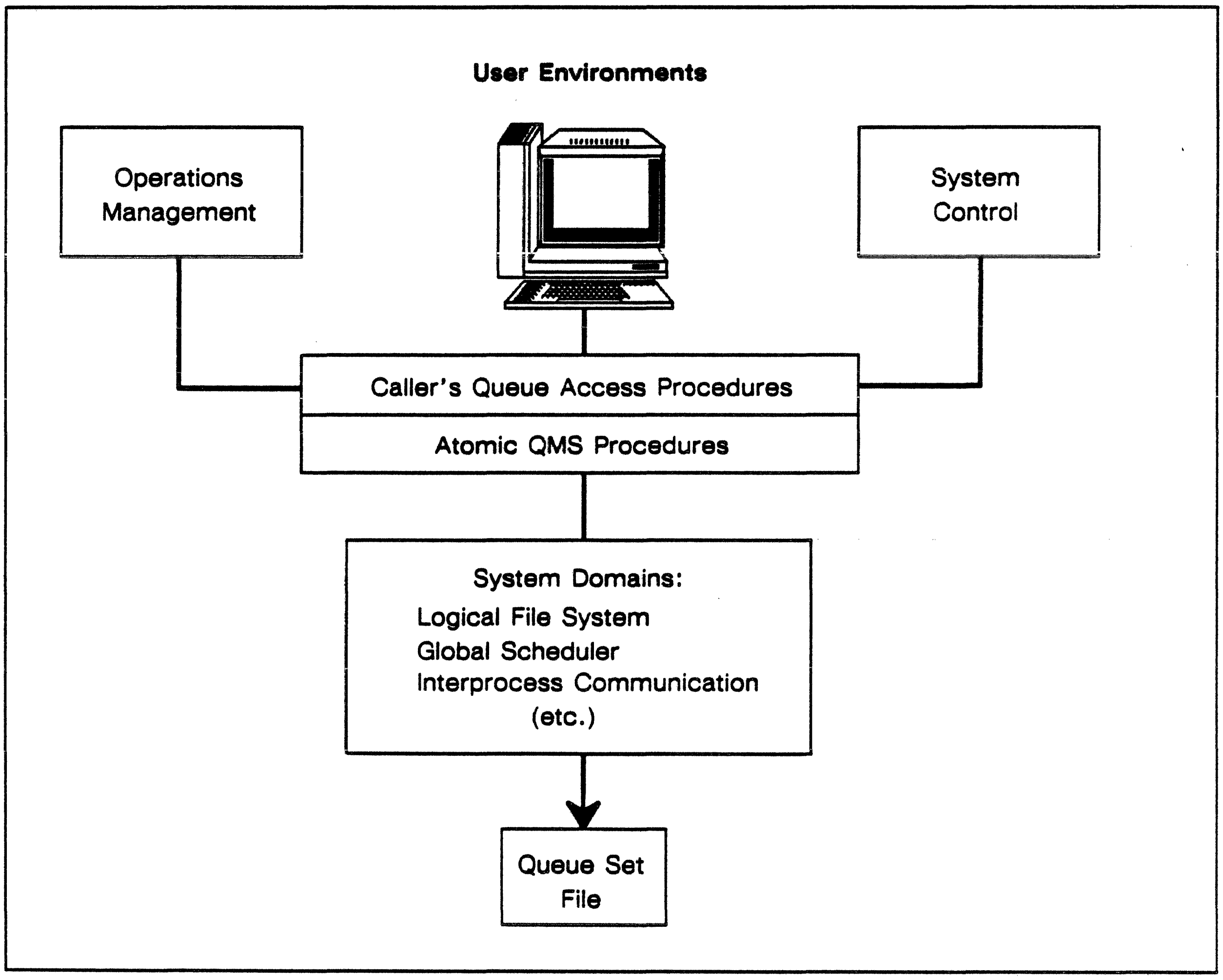


Figure 4–21. Relationship of calling features to Queue Management procedures.

Queue Management Subsystem Procedural Interfaces

The queue management subsystem procedural interfaces manipulate system queue sets, queues, and queue items. Access to queue sets is restricted by access rights provided by the logical file system. Any feature with read/write access permission to the queue set is permitted to use any of the queue management subsystem procedural interfaces.

The queue management subsystem procedural interfaces are atomic: they preserve the current setting of the queue set read and write locks. If the appropriate lock is not already set, the procedure will implicitly perform the proper locking and unlocking of the queue set.

The queue set procedural interfaces of the queue management subsystem are:

- **Create Queue Set** creates and initializes a queue set.
- **List Queue Set Attributes** returns the current attributes of a queue set.

The queue procedural interfaces of the queue management subsystem are:

- **Create Queue** creates a new queue for the user of the queue set.
- **List Queue Attributes** returns queue attributes for all the queues within a queue set.

The queue item procedural interfaces of the queue management subsystem are:

- **Enqueue Queue Item** creates and enqueues a new item into the specified queue. The enqueued item is added to the tail of the queue.
- **Requeue Queue Item** moves a queue item from one queue to another within the same queue set.
- **List Queue Item Attributes** returns the attributes of all queue items in the specified queue set.
- **Alter Queue Item Attribute** changes the following queue item attributes:
    - Reserve flag
    - Search key
    - Item information block
    - Ready flag
- **Get Next Queue Item** retrieves the next ready queue item with a specific search key. The queue item returned is the one nearest the head with a search key equal to that specified. Reserved queue items are not eligible.
- **Dequeue Queue Item** deletes an item from a queue. This procedure is used after the queue item has been processed by the queue user.

Session Scheduler

The session scheduler (SS) software is responsible for:

- Managing input queue(s)

- Scheduling and dispatching batch sessions
- Logon of interactive sessions
- Keeping track of all sessions
- Providing a set of operations for controlling sessions

The first release of the session scheduler will operate using a first-in-first-out algorithm. That is, the first task received by the session scheduler is the first one acted upon. The session scheduler is limited to one input and one execution queue.

Session Scheduler Procedural Interfaces

Session scheduler procedural interfaces describe operations which can be performed on sessions. These interfaces are provided primarily for use by:

- User management
- User environments
- Operations management

The session scheduler procedural interfaces are:

- **Logon Session** creates and initiates an interactive session on the computer system. A queue item is created in the input queue set and initialized with the attributes of the session. System accounting is called to initialize an accounting table for the session. Process management is called to create the command shell process of the session.
- **Queue Session** queues a batch session for processing. The attributes of the session are stored in a queue item which is enqueued in the input queue of the session category.
- **Run Session** causes the specified batch session to be placed into execution immediately, if possible (resources must be available to do so)..
- **List Session Attributes** returns the attributes of a session.
- **Alter Session Attribute** modifies an attribute of a session.
- **List Sessions** returns the attributes of all sessions currently recorded in the input queue set.
- **Drop Session** initiates removal of an executing session from the system. The user environment is signaled to drop the session. The VSOS environment interprets this to mean

terminate the current task and go to the next EXIT or CONTINUE command.

- **Kill Session** results in one of the following actions:
  - If the session is in an input queue, the session status attribute is set to kill and the session is initiated. The session is permitted to execute just long enough to generate a dayfile.
  - If the session is already executing, the session status is set to kill and a kill signal is sent to the command shell process. The user environment is responsible for ensuring proper termination of all processes stemming from the shell process and the deletion of the batch input file when necessary. A dayfile is generated.

When all processes associated with the session have terminated, the session entry is removed from the input queue set.

Process Scheduler Procedural Interfaces

The process scheduler procedural interfaces manipulate processes and are provided for use by process management.

The procedural interfaces are:

- **Return Process Inheritance Information** gets global scheduler specific inheritance data which is passed to a child process.
- **Schedule Process** initiates the scheduling of a process for the first time.
- **Terminate Process** is called by process management to perform global scheduler process termination.
- **Initialize Process** is called by process management to perform global scheduler process initialization.

Queue Sets

The queue set is an object used to maintain a set of queues for a system feature. Operations are provided to create and list a queue set. The name of a queue set is specified by the system feature which requested creation of a queue set. The queue set name must be unique among all queue sets. By convention, the queue set name is prefixed with a feature identifier (a character acronym).

Before any system feature can request operations on a queue set, the queue set must exist and the requesting feature must have access permission to it.

A queue set is created with a Create Queue Set procedure. Once a queue set exists for a given feature, the feature can request operations on objects within the queue set.

Queues

A queue object is used to organize items in a first-come, first serve order. The queuename is specified by the calling feature. The queuename is unique within the queue set.

Queue Item

A queue item stores the information needed by the user of a queue set. A list of queue item attributes includes:

- **Queue Item ID** - A unique system wide ID generated by the queue management subsystem when the queue item is created. The queue item ID is used by the caller to reference a specific queue item.

- **Item Information Block** - Attribute used to store queue item information specified by the calling feature. The size of the block is fixed by a global scheduler configuration constant. The content and format of this block are left to the queue user.

- **Search Key** - An attribute set and used by the calling feature to get the next queue item of a specified type. For example, the input queue manager (IQM) or session scheduler can use this attribute to store the current state of a session as Waiting for Memory. When memory is available, input queue manager can request Global Scheduler Get Next Queue Item with a search key of Waiting for Memory.

- **Reserve Flag** - A Boolean attribute that indicates the item is reserved for exclusive access by a single process. When the flag is set, other processes are prohibited from making any changes to the queue item. The process which reserved the item is permitted to perform any operation on the item, and is responsible for releasing reservation of the item when it is done using it. Normally, a queue item is placed in this state while being processed by the user of a queue.

- **Last Reserved Process ID** - The process ID of the last process which reserved the queue item

- **Ready Flag** - A Boolean attribute that, when set, indicates the item is ready for processing and can be selected by the procedure to Global Scheduler Get Next Queue Item.

Input Queue Set

The input queue set is used by session scheduler to maintain a structure of input and execution queues. Every session known to the system is represented by a queue item in the input queue set.

Queue items are used for storage of session attributes.

Session Attributes

Session attributes describe the characteristics of a session. Some attribute values are generated internally by the session scheduler, while the bulk are derived from parameters passed through a user management request to log on or queue a session.

Only attributes unique to the session are maintained by the session scheduler. These attributes should not be confused with attributes defined by other system features unique to a user which are stored in the user registry.

The session attributes are:

- **Session ID** – A unique, system wide ID generated when a session is created or queued. The session ID is a queue item ID used by the caller to reference a specific session.
- **Session Queued Time** – The time and date when the session was queued for processing. This time is available to user environments for use in generating session dayfiles.
- **Session Initiate Time** – Date and time when the session was initiated. This time is available to user environments for use in generating session dayfiles.
- **Session State** – State of a session waiting in an input or execution queue.
- **Session Control Code** – Communicates to the user environment shell process what action to take at the start of a session:
  - It can be started in normal fashion
  - It can be terminated because the session was either killed or dropped while in the input queue.
- **Session Inheritance Block** – A block of information provided by the originator of the queued session to pass information to be inherited by the new session. The size of the information block is limited by a global scheduler constant. The content and format of this block are left up to the session originator.

- **Terminal Connection ID** - The interactive terminal ID for the session. The user environment reads session commands from this terminal connection. This attribute is only valid for interactive sessions.

- **Queued Input File ID** - The queued input file ID is the global file ID of the input file. The user environment reads the session commands from this file. This attribute is valid only for queued sessions.

- **Image Name ID** - the software unit name of the command shell as configured in the system configuration table.

- **Session Process Cluster ID** - Identifier of the process cluster created for the session. Process clusters are discussed in the section titled *Process Management.*

- **User ID** - Identifier of the username the session belongs to. This attribute is needed by other system features so they may retrieve user attributes stored in the user registry which they define. This attribute is further discussed in the section titled *User Management.*

- **Project ID** - Unique project ID needed by the system accounting feature. This attribute is further discussed in the section titled *User Management.*

- **Account ID** - Unique account number needed by the system accounting feature. This attribute is further discussed in the section titled *User Management.*

- **Environment ID** - Identifies the user environment the session is working under. This attribute is needed by certain utility programs to determine processing based on environment.

Process Management

Process management includes the activities of:

- Process creation
- Process initialization
- Process switching
- Process termination
- Process recovery

Process management (PM) is responsible for the recording and modifying of process related variables. Process management is also involved in CPU initialization, building the process queues for the monitor, and setting up the process packages and register tables for all locked-down servers.

From the viewpoint of process management, a process is simply a logical association between a process ID, an executable file, and the PM objects initialized to allow the process to execute. The process ID is unique across CPUs.

After a process has been created, but before it has been initialized on a CPU, it consists only of an executable file and process ID. After it has been initialized, it also has a:

- Process package
- Alternate process package
- Process descriptor block
- Register block
- Process attributes list

Every process is also associated with an inheritance information block. The inheritance information is gathered in creating the process and is available, at any time, to the new process.

Process management code exists in the system domain of every process, and a process management server runs on each CPU. Each server has a mailbox for communications between CPUs, and between I/O units and CPUs. The process management server is responsible for:

- Initialization
- Recovery

- Termination

of all processes on its CPU. If a process wants to create a process on another CPU, the Create Process procedure (via global scheduler) notifies the process management server on the appropriate CPU.

In all these cases, residence of the target process is hidden from the caller. The caller must invoke a local procedure, which then determines on which CPU the target process resides. To all features other than the global scheduler, process management appears to function as a single unit.

Due to the amount of locked CP memory required for each active process, the number of processes allowed on a CPU at one time is limited to 128. Each process on a CPU has a CPU process index that ranges from 0 to 127. The index is used by the following to reference their local tables:

- Process management
- CP memory manager

Process Objects

In order for a process to run on a CPU, certain process objects must have been created and initialized. They are:

Executable File

An executable file is created when an object file is successfully linked. It is opened by a process management server at process initialization. An executable file contains the:

- Process image name and version
- Register and map information
- Transfer address for the new process

Process Descriptor Block

The process descriptor block holds process specific data, including:

- Process inheritance information
- Process attributes
- Monitor scheduling parameters
- Monitor process attributes

- Buffer for messages from the process management server
- Buffer for passing messages passing between process management and monitor

The process descriptor block is created by process management and maintained by both process management and the monitor.

Register Block

The register block contains the process's registers at the time of a process give-up. It is initialized by process management from the executable file and maintained by the hardware.

Process Package

The process package contains the process's:

- Invisible package
- Domain packages
- Domain stack

Memory for the process package is allocated by process management. It is initialized by domain management and maintained by both domain management and the hardware.

Alternate Process Package

The alternate process package is used for error recovery. It is a copy of the original process package, but has a different transfer address.

Process States

When a process decides to create another process, it places a call to the Process Create function of process management. Process management creates an entry for the new process in the process catalog. Inheritance information is gathered and temporarily stored in the new process' uninitialized paging file. The global scheduler (GS) is notified that a new process exists and needs to be scheduled.

Some time later, global scheduler calls a process management server on the appropriate CPU to initialize the new process. The new process's

- Process package
- Process descriptor block

- Register block
- Paging file
- Map file

are set up. The process's inheritance information is copied into the process descriptor block. When this external initialization of the process is complete, the monitor is asked to place the new process in the ready queue.

The new process begins running an internal process initialization procedure in the process management domain. The initialization procedure calls those features which must be initialized before transferring control to the application domain.

As different procedures are initialized, they may call process management to retrieve their inheritance information. When internal process initialization is complete, domain management is called to set the true transfer address into the registers, and the Forward Domain Change instruction is executed to give control to the application.

When a process terminates normally, a call is made to Process Terminate. Process Terminate:

- Sets the process status in the process catalog to terminating
- Calls each feature's process termination routines
- Stores the reason for termination in the process descriptor block
- Asks the monitor to move the process to the termination queue

If the process terminates abnormally because of an access violation or an illegal instruction, it is moved to the recovery queue. The process is restarted so it can perform termination cleanup functions such as unlocking files that were locked when the process terminated. Information about the process' termination is sent to the dayfile in batch sessions, or sent to the screen in interactive sessions.

As soon as the monitor moves a process to the termination queue, it notifies the process management server. The server removes the process from any clusters it belongs to. The server also deletes the process entry in the process catalog and completes external termination of the process.

The states a process can go through are shown in figure 4–22. While process management requests many of the changes, the actual process switching is done by the monitor in monitor mode.

Process Create

Process Create generates a new process, which is placed in the initiating state. The new process type and its inheritance information are determined at this time.

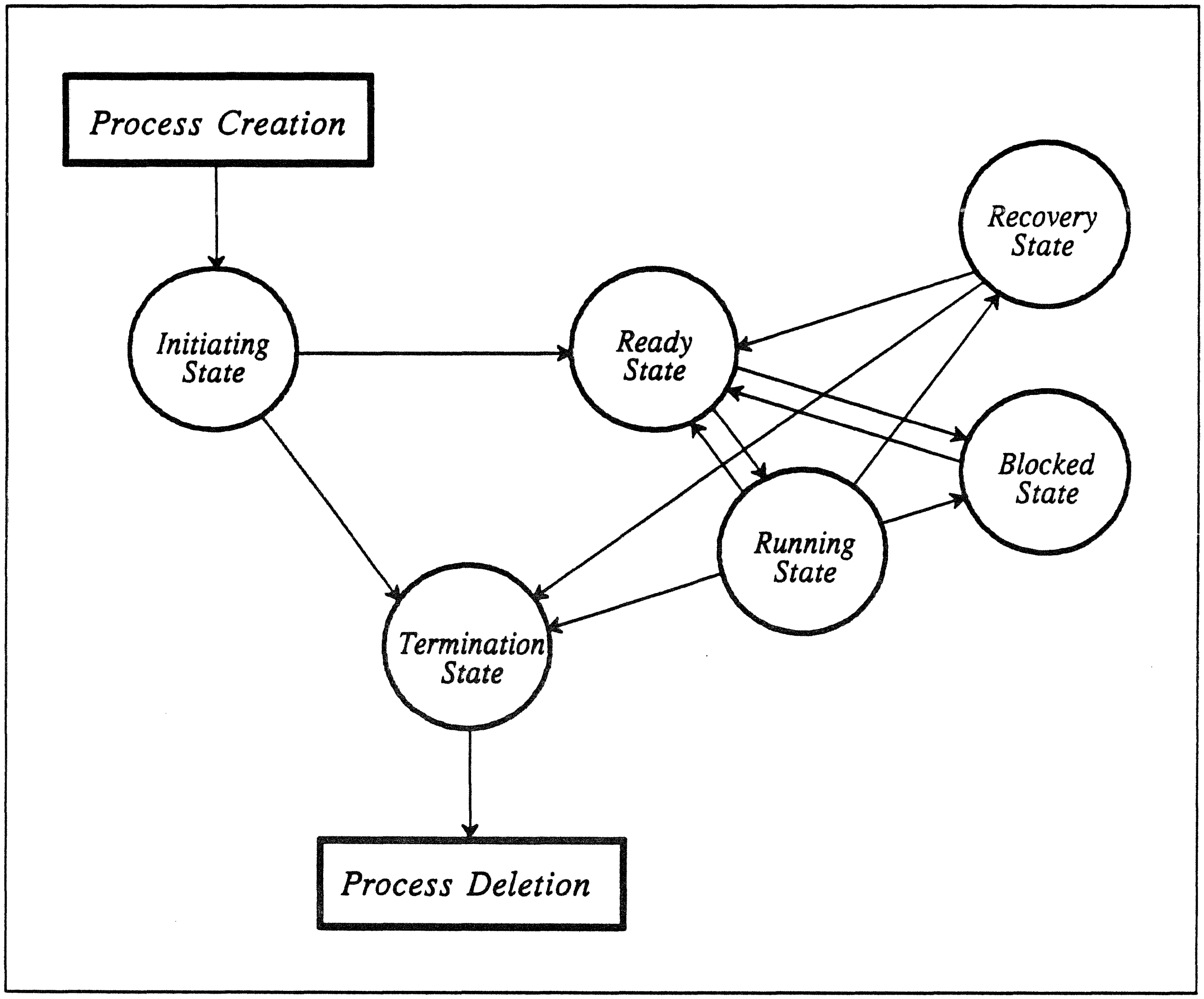


Figure 4–22. A mapping of the possible states and transitions within process management.

Initiating State

Processes in the Initiating state are not yet eligible for execution upon a particular CPU. Processes in this state do not yet have any system resources assigned to them; they have only enough information to begin initialization.

Initiating State to Ready State. Initialize requests are required to move from the Initiating state to the Ready state on a CPU. The initialize request may only be made by the global scheduler, which provides the scheduling parameters for the new process. In moving from the Initiating state to the Ready state the process is committed to a CPU, and the following resources are assigned to it:

- Paging file
- Process package
- Register file

Initiating State to Process Deletion. If an error occurs prior to the assignment of resources, no termination is required. The process is simply deleted from the system.

Initiating State to Termination State. If an error occurs after the assignment of resources, the process is moved to the Termination state to receive the processing necessary to release the resources.

Ready State

A process in the Ready state can execute as soon as the monitor performs an exit force to the process. All processes in the Ready queue are in the Ready state.

Ready State to Running State. The monitor chooses the next process from the top of the Ready queue to process switch into the Running state. If the pager is modifying this process's page tables or process management is switching its process package, flags are set telling the monitor that the process should not be scheduled. If this is so, the next qualified process is scheduled.

Ready State to Blocked State. Normal completion of the process management Block procedure will move the process from the Ready to the Blocked state.

Running State

A process in the running state is executing in the hardware.

Running State to Ready State. For a time-slice expiration or a hardware interrupt, monitor may move the process from the Running state to the Ready state.

Running State to Termination State. When the Terminate Process procedure is called, it will call other feature's termination procedures. It then asks the monitor to move the process to the Termination state.

Running State to Blocked State. A process may move from the Running state to the Blocked state for several reasons, including:

- Page faults

- Process management wait calls
- Wait for I/O completion

Running State to Recovery State. If a run time error occurs that can not be handled within the process, the process may be moved to the Recovery state. For example, if an access violation occurs, the process must be moved to the Recovery state.

Termination State

A process in this state has completed its internal termination processing or has been through recovery. Once in Termination state, processes remain there until they are deleted from the system.

Termination State to Process Deletion. Process management deletes a process in the Termination state by:

- Returning the process's paging file and other files explicitly mapped in the system
- Removing the process from the list of active processes
- Deleting the process objects (register block and process descriptor block)

Recovery State

Processes in the Recovery state await disposition after an abnormal condition. Process management scans processes in the Recovery state and decides whether they should be:

- Put back into the Ready state with a new process package to allow for error termination
- Moved to the Termination state after recovery processing

Recovery State to Ready State. After the process management server has switched the process's process package, the process is moved from the Recovery state to the Ready state where its termination code will execute.

Blocked State

Processes in the Blocked state are temporarily ineligible for execution. They retain all resources and need only be moved back into the Ready state in order to execute.

When process management asks the monitor to move a process to the Blocked queue, a time limit is given the monitor. Processes wait in the Blocked state for unblocks from other processes, or until the time limit is reached.

Blocked State to Ready State. When the monitor is requested to wake a process, or when a timeout occurs, execution of the monitor's scheduling code moves the process back to the Ready state.

Process Clusters

A process cluster is an arbitrary collection of processes with the relationship between processes in a cluster specified by and known to the part of the system which invokes this software. Information is stored with each cluster which can be retrieved by the caller at a later time. This information is stored in the *cluster packet.*

The cluster packet contains some process management information and a buffer which can hold user defined information. Cluster information is intended to be global in the sense that it can be easily retrieved or modified from any CPU.

The process that creates a cluster is the cluster owner. The cluster owner always has rights to:

- Add or delete processes from the cluster
- Destroy the cluster
- Change the cluster packet and attributes

The rights of the other cluster members depend on how the owner sets up the cluster.

The owner is returned a capability for the cluster. The owner can specify whether other members of the cluster can update the cluster packet and whether all descendant processes will be members of the cluster.

Members of a cluster can always get a list of other cluster members, find out who owns the cluster, and get a list of the clusters to which the caller belongs.

Process Management Objects

The process management objects contain specific process management information about process management initialization, clusters, and processes in the system.

CB Data Block

The CB data block contains:

- Process management initialization information
- Mailbox handles
- Handles for SM objects

The data block is created at system initialization and is shared by all processes.

Cluster Catalog

The cluster catalog contains information about all clusters in the system. It is organized by cluster ID and contains cluster names, cluster packets, and cluster members. This catalog is created at system initialization and is shared by all processes.

CPU Process Catalog

There is a CPU process catalog on each CPU. It is a table in CP memory that contains information about the processes that are on the particular CPU.

A fixed number of processes can be resident on a CPU at one time. This catalog contains an entry for each possible process on the CPU. The index for an entry into this table is called the *cpu_process_index*. The CPU process catalog includes:

- The process ID.
- The state of each process.
- Physical addresses of the process packages and process descriptor blocks.
- A CPU process index indicating which CPU slots processes inhabit.

The CPU process index is shared with CP memory management and accounting. The CPU process catalog is initialized at the time of CPU initialization and is shared by all processes on the CPU.

SM Process Catalog

The SM process catalog maintains a list of all processes on the system. It contains the following specific information about each process:

- Current state
- CPU used

- The process's global file ID
- A list of clusters of which the process is a member.

The SM process catalog is created at process initialization and is shared by all processes.

Shared Information

A number of lists, blocks, and queues exist within process management for sharing of information.

Blocked State List

A blocked state list contains all of the processes on a CPU in the Blocked state. This list is initialized by process management at CPU initialization, but is owned by the monitor.

Process Descriptor Block

A process descriptor block is a locked-down table associated with each process on a CPU. It is shared with the monitor and the PM server on the CPU. It contains:

- The kernel's process attributes
- monitor process attributes (including scheduling information)
- Feature-specified inheritance information

The process descriptor block also contains buffers for passing information between the process and monitor, and between the process and the process management server.

Recovery State List

The recovery state list contains all of the processes on a CPU that are awaiting disposition after an abnormal condition such as an access violation or illegal instruction. This list is created by process management at CPU initialization. It is managed by the monitor and read by process management.

Termination State List

The termination state list contains all of the processes on a CPU awaiting final termination. This list is created by process management at CPU initialization. It is managed by the monitor and read by process management.

Process Management Functions

The functions within process management are briefly described. They consist of Initialization functions, and process and cluster operations.

Process Management System Initialization:

Entered during system initialization to create the process management tables.

Process Management CPU Initialization:

Used during system initialization to bring process management into existence on a CPU.

Process Create:

Places a preliminary process object in the Initializing state. The process is not able to execute on a CPU until initializing actions are performed as a result of the global scheduler calling process management to initialize the process.

Process Initialize:

Makes a process eligible for execution by moving it from the Initiating state to the Ready state for a CPU.

Wait:

Provides the CPU give-up mechanism. This routine can be called to make a simple time delay, or to wait for one of a set of defined wait reasons with a specified timeout.

Process Block:

Permits one process to request that another process be blocked.

Process Unblock:

Requests that a process be moved from the Blocked state to the Ready state.

Process Terminate:

Begins the termination of the process. This function calls each feature's termination routine and moves the process to the Termination state.

Get CPU Process Index:

Returns the index into the CPU process catalog for the specified process.

Get Process Status:

Returns the status of the specified process.

Set Inheritance Block:

Allows a feature that is not automatically called at process creation to specify inheritance information.

Retrieve Inheritance Information:

Returns a feature's inheritance information.

Process Attribute Inquiry:

Returns the specified process attribute to the caller.

Change Process Attribute:

Allows callers to change some of the attributes associated with a process.

Create Cluster:

Provides a logical grouping of processes. The operation does not create any new processes, it merely groups a list of existing ones.

Add Process(es) to Cluster:

Adds a list of one or more process IDs to a cluster. While the list of clusters to which the processes belong is updated, the processes are not notified of the change.

Delete Process(es) From Cluster:

Deletes one or more process IDs from the process cluster with the specified cluster ID. If all process IDs are deleted as a result of the operation, and the 'can-be-empty' characteristic is not set (the cluster can only exist with members), the cluster ID is deleted from the cluster list. While the list of clusters to which the processes belong is being updated, the processes are *not* notified of the change.

Set Cluster Packet:

Provides the mechanism to record cluster specific information by updating the user defined part of the cluster information packet.

Return Information From Cluster Packet:

Provides the mechanism to return the cluster packet to the caller.

Return Processes in Cluster:

Gives a list of processes which are members of a specific cluster.

Return Clusters of Which Process is a Member:

Gives a list of the clusters in which a process is a member.

Change Cluster Characteristics:

Provides the mechanism to change the characteristics of a cluster from those specified when the cluster was created.

Destroy Cluster:

Deletes a cluster. While the list to which the processes in this cluster belong is being updated, the processes are *not* notified of the change.

Remote Procedure Calls

The remote procedure call (RPC) feature is one of the system features that coordinate interprocess activities. The remote procedure call feature provides the ability to transfer messages between processes running on the three types of processors - central processing unit (CPU), I/O unit (IOU), and service unit (SU).

How Messages are passed

The remote procedure call feature permits processes to send and receive messages using *mailboxes*.

The following conventions are observed. Any process expecting to receive messages can connect to, or *export*, a mailbox. The process is referred to as a *server*. Any process can connect to, or *import*, a mailbox to which it can send messages. The sender is called a *client*.

Only processes that have exported or imported mailboxes can use them to pass messages. A client always sends messages to a mailbox, and a server always receives messages from a mailbox. A server can also send a reply to a client after receiving a message.

A server waiting for a message from a mailbox that is currently empty, or a client waiting for a reply to a message, can suspend execution (*block*) until the expected event occurs. Or, they can continue with other tasks and periodically check whether the message or reply has been sent.

The simplest scenario for mailbox use is one server receiving messages from one client. There are other options available. As shown in figure 4–23, one server (S) might receive messages from multiple clients (C), multiple servers receive messages from one client, or multiple servers receive from multiple clients.

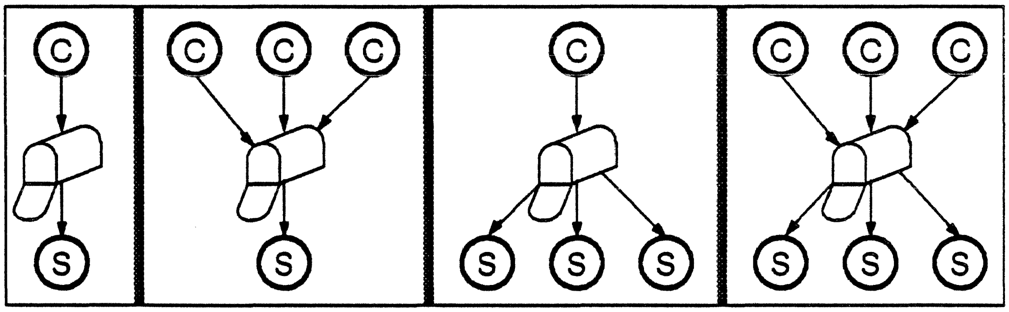


Figure 4–23. Typical client and server relationships.

The same process can act as a server taking messages from some mailboxes, and as a client sending messages to other mailboxes (figure 4-24).

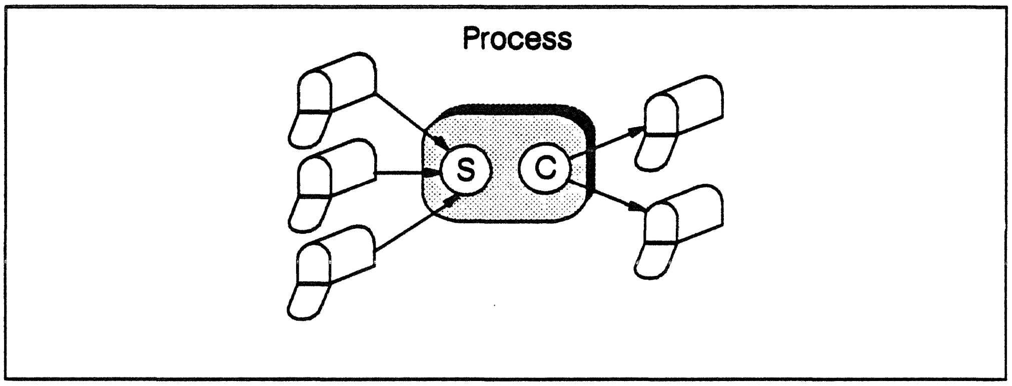


Figure 4-24. A process which is both a client and server to different mailboxes.

A process can even act as a server and a client to the same mailbox, though this is an inefficient means of communicating, and procedure calls should be made instead.

Complex Receive

A single server may have several active mailboxes from which it takes messages, and a client may send multiple messages to various mailboxes, and expect replies to those messages. By issuing a *Complex Receive,* servers and clients can avoid constantly polling for data they are expecting. The server can request to read the next message from a specified list of mailboxes, and the client can request to read the next reply to a specified list of messages.

Figure 4-25 shows a server with two mailboxes, A and B. During a Complex Receive, the next message posted to either mailbox is received by the server. In this case, mailbox B contains the next message.

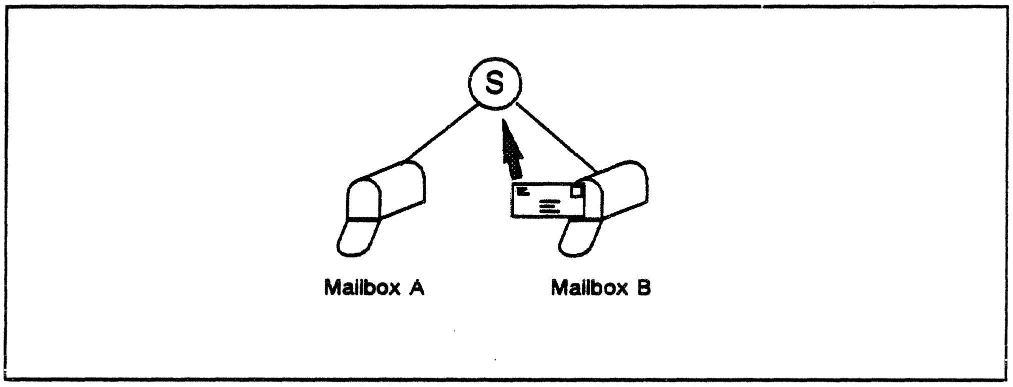


Figure 4–25. A server receiving the next message posted to one of its mailboxes.

In figure 4–26 below, a client has sent two messages to which replies are expected. During a Complex Receive, the next reply to be sent is received by the client. In this case, message *A*'s reply was next.

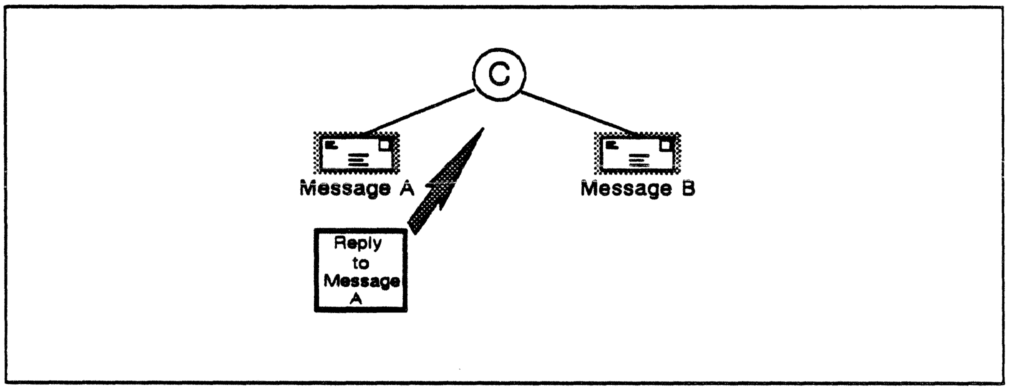


Figure 4–26. A client receiving the next reply sent to it.

A process can act as a server and a client. During a Complex Receive, the process can wait for either the next message or the next reply. In the case illustrated in figure 4–27, a message was posted next. The process is notified of the message, so that the process can act as a server of the mailbox.

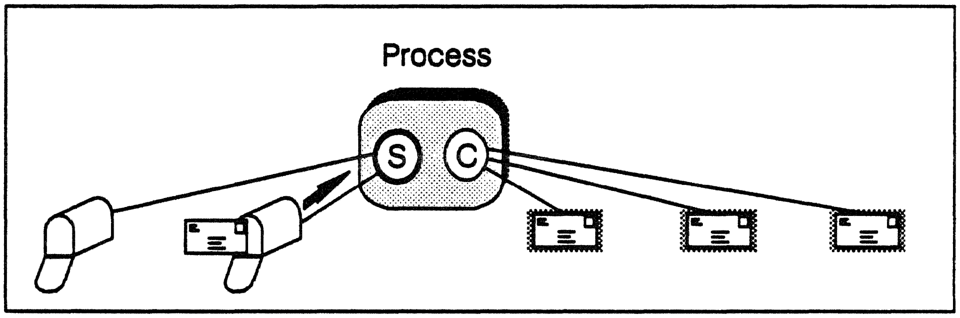


Figure 4–27. A process receiving the next message or reply destined for it.

Waiting for Messages and Replies

Both clients and servers can block while waiting for messages and replies. If a mailbox contains the maximum number of messages that can be enqueued at one time, it is full, and no additional messages can be sent to it until a server has received at least one message from that mailbox. A client sending a message to a full mailbox can block and wait until the mailbox is no longer full, at which point the message can be sent to the mailbox. If several clients are blocked on the same mailbox, for each message received from the mailbox by a server, one client is unblocked and allowed to send its message.

The same scheme applies to servers waiting for a message to be sent to an empty mailbox. Servers can block until messages arrive in the mailbox. For each message placed in the mailbox by a client, one server is unblocked and can retrieve the message.

RPC Objects

There are three types of RPC objects: mailboxes, messages, and replies. The software supports various operations performed by server and client processes on these objects.

RPC Mailbox Objects

The RPC *mailbox* object holds messages sent by clients and received by servers. It has a system-wide unique name and access rights determining which processes are authorized to use it.

The mailbox is actually a queue of message control blocks (MCBs) which reside in the communication buffer. There is one message control block for each message. The message control block is enqueued when the message is sent, and dequeued when the message is received. Synchronization of queuing is internally handled by communication buffer semaphore instructions.

The message control block holds information about the message status, its size, its location in global memory, the sender's identification, whether a reply is needed, and where the reply is to be stored.

Three types of mailboxes can be defined - permanent, transient, and system.

Permanent mailboxes can be created at any time and are explicitly activated by a process. Once created, they remain in the system until the next system initialization. System features that need to transfer messages between all processes use permanent mailboxes. Any feature executing in the system domain or on an I/O unit or service unit can access a permanent mailbox. Processes in the application domain cannot access a permanent mailbox.

Each permanent mailbox is defined with a unique name that is used by processes performing import and export operations. The name consists of the domain feature ID of the feature that created the mailbox, and a symbolic ID, which is a character string specified by the creator, unique to that feature.

The first export request referencing a permanent mailbox creates the mailbox. The creator specifies the mailbox name. After the mailbox is created, a *handle* is returned to the caller, to be used in message transfer requests for this mailbox.

Transient mailboxes can be created and destroyed at any time, and are explicitly activated by processes. They are destroyed when the last process in the creating process' cluster terminates.

The transient mailbox is created on the first export request referencing it. It is assigned a unique name consisting of the ID of a cluster to which the creating process belongs, the ID of the feature creating it, and a symbolic ID specified by the creator.

Any caller belonging to a transient mailbox's creating process' cluster, or any process running on an I/O unit or service unit has access to system mailboxes. An authorized user may export, import, and destroy a transient mailbox.

A transient mailbox can be destroyed when there are no processes using it.

System mailboxes are created only at system initialization. They are used by a restricted set of features that need to transfer messages between all processes and must not block during remote procedure call operations. System mailboxes are preactivated in every process throughout the system's lifetime. Naming conventions for system mailboxes are the same as for permanent mailboxes, except that the handle returned from import and export operations is always the same value, no matter what process invokes it. Any feature executing in the system domain or on an I/O unit or service unit can access system mailboxes. An authorized user can perform export and import system mailbox functions.

To preactivate a system mailbox the caller specifies its domain feature and symbolic ID during system initialization. Preactivation operations allocate the mailbox and ensure that all subsequently executing processes can perform mailbox operations.

Internally, the remote procedure call feature uses three types of semaphores to control mailbox operations - Full, Empty, and Message semaphores. Post and wait functions manipulate the semaphores.

Full semaphore is used to determine when a mailbox is full. When a mailbox is created, its Full semaphore is set to the maximum number of messages that can be enqueued to the mailbox. The semaphore value decreases by one for each message sent to the mailbox, and increases by one for each message taken from it. A zero semaphore indicates that the mailbox is full. Clients cannot send more messages to the mailbox until some messages have been retrieved. If the client blocks and waits until it can send a message, the semaphore is decreased by one. A negative semaphore indicates the number of waiting clients.

Empty semaphore is used to determine when a mailbox is empty. When the mailbox is created, the semaphore is set to zero. Each message sent to the mailbox increases its value by one, and each message received decreases it by one. When the semaphore value is less than or equal to zero, the mailbox is empty. The server can block until a message is sent. The semaphore value decreases by one for every blocked server. A negative semaphore value indicates the number of blocked servers.

Message semaphore synchronizes the addition (sending) and deletion (receiving) of a mailbox's messages. The Message semaphore value is set to zero when a mailbox is created. Sending a message enqueues an message control block to the mailbox, and decreases the Message semaphore by one. Receiving a message dequeues a message control block from the mailbox and increases the semaphore by one. The absolute value of the semaphore value indicates the number of message control blocks queued to the mailbox.

RPC Message Objects

A message is information that is transferred from a client to a server via a mailbox. The message can range in length from zero to the maximum message length configured in the system.

The remote procedure call feature views each message object as a *message buffer* and an associated message control block. If there is data associated with the message, the remote procedure call feature copies it into a message buffer in system global memory whose length equals the maximum system message length. If there is no buffer space available when a message is to be sent, the remote procedure call feature blocks the requesting client until space frees up.

After the message is copied to the global buffer, a message control block is allocated. Message control blocks contain information about a message buffer, including: the message status, (the request was sent or received, a reply was sent, or a request canceled), a flag indicating whether a reply is expected, the sender's identification, and the message buffer size and location.

A remote procedure call enqueues the message control block to the tail of the mailbox queue. To receive a message, a server issues a request to the remote procedure call feature. The remote procedure call feature dequeues the message control block at the head of the queue and copies the corresponding message buffer, if any, into the server's local memory.

RPC Reply Objects

A server sends a reply to a message if the sending client so requested. The server forms the reply and issues a remote procedure call to deliver it. The remote procedure call feature copies any reply data from the server's local memory into the global buffer and updates the message control block to indicate a reply has been sent.

When the client issues a request to the remote procedure call feature to receive the reply, the remote procedure call feature copies the message buffer into the client's local memory, and releases the global message buffer and its message control block for use for subsequent messages.

The server that received the message must provide the reply, except on I/O units, where a process different from the receiving server can reply. The client that sent the message is the only process that can receive the reply.

RPC Functions

Callers manipulate mailbox objects by issuing remote procedure call feature requests. The types of requests can be divided into *server, client, server and client,* and *system.* Most functions can be invoked on any of the three processor types, but some are restricted to CPUs only. The following function descriptions will indicate when a function is restricted.

Server Functions

Export Mailbox connects a process to a mailbox. The caller becomes a server which can receive messages from that mailbox. The mailbox name, the maximum number of servers allowed to access the mailbox, the maximum number of messages that can be queued to the mailbox at one time, and the creator's access rights are specified on the call, but are ignored if the mailbox already exists. If the mailbox does not exist, it is automatically created and the creator given the specified access rights. A mailbox handle is returned, to be used in subsequent calls referencing the mailbox. Once the mailbox is exported, clients can connect to it (import) and send messages.

Receive a Message requests the remote procedure call feature to take a message from the mailbox and pass it to the caller. If the mailbox is not empty, the remote procedure call feature copies the message at the head of the mailbox queue into the server's local memory, along with the sender's ID, a flag indicating whether the client expects a reply, and the request ID. If the mailbox is empty, the calling server can choose to block until a message is placed in the mailbox.

Send a Reply sends a server's response to a message, if the sending client so requested. After validating that the server is allowed to issue the specified reply, the remote procedure call feature copies it into global memory from the server's local memory.

Cancel Export Mailbox disconnects a server from a mailbox. The server cannot receive any more messages from the specified mailbox.

Destroy a Transient Mailbox prohibits any new processes from using the mailbox, and destroys it when all its permitted user processes have terminated. A transient mailbox is automatically destroyed when the last process belonging to the process cluster of its creator terminates.

This function is restricted to CPU processors only.

Client Functions

Import Mailbox connects the caller to an existing mailbox The caller becomes a client able to send messages to that mailbox. The mailbox's handle is returned.

Send a Message to a Mailbox sends a message to any mailbox to which the caller has access rights and which it has imported. The client may request a reply from the server that received the message. If the mailbox is full, the client may also request that the remote procedure call feature temporarily block it until there is room for another message. The remote procedure call feature queues the message to the mailbox.

Cancel a Message requests that a message sent to a mailbox be deleted. The message can only be deleted if a server has not yet received it. Only messages that require a reply can be canceled.

Receive a Reply retrieves a server's response to a message. If the server has not yet replied, the client can block until the reply is sent, or until a time limit expires. The remote procedure call feature returns the reply to the caller's local memory.

Cancel Import disconnects a client from a specified mailbox. The client cannot send any more messages to that mailbox.

Server and Client Functions

Complex Receive allows a client or server waiting for several specified messages and/or replies to be notified when one is sent. The caller is blocked until one of the expected messages or replies is issued. The message or reply is returned to the caller. If no message or reply is sent before a specified time limit, an error is returned.

System Functions

Initialize Remote Procedure Call Feature for System initializes the remote procedure call feature for the CPUs and one I/O unit. This function is called when the first CPU and every I/O unit is initialized. If running on the CPU, this function must be called from the CPU cold start system initialization process. If running on an I/O unit, it must be called after the remote procedure call feature CPU initialization completes and all system mailboxes have been created. The service unit software does not call this function. Upon completion, a processor can communicate with other processors using remote procedure call facilities.

Create System Mailbox is a CPU-only function, called from the CPU cold start initialization process. It must be called after Initialize Remote Procedure Calls for System, and before any processes which use system mailboxes execute. The caller specifies the maximum number of servers allowed for the mailbox, the maximum number of messages that can be queued, the creator's access rights, and whether the mailbox can access privileged resources.

Initialize Remote Procedure Calls for Process sets up remote procedure call data structures within each new process. It is automatically invoked by the first remote procedure call in a new process if remote procedure calls are not yet initialized for the process. It can also be explicitly coded. After initialization, the process can perform the remote procedure call client and server operations.

Terminate Remote Procedure Calls for Process discontinues use of the remote procedure call functions within an executing process. Process management calls this function when it terminates a process. All the process' messages that were not yet received are canceled. Any replies not yet received are discarded. All the process' connections to mailboxes are canceled.

Record Destruction of a Process Cluster is called by process management when a process cluster is terminated. Any transient mailboxes created by a process belonging to the destroyed cluster are immediately destroyed if not in use. Transient mailboxes that are still in use will be destroyed when the last using process terminates.

Section 7: Domain Management

This part examines the feature that creates and manages domains within the system. It consists of one section:

- Domain management

Domain Management

Domain features support protected routines and data structures from unauthorized access or modification. Domain features are part of the operating system.

Domain Hardware

The computer system hardware provides interprocess virtual address space protection via the *process switch* mechanism. The operating system depends on the process switch to ensure that one process does not violate the address space of another.

While the process switch offers some protection, it also has its deficiencies:

- It is relatively slow because of the required intermediary switching routine
- The smallest unit of protection is a process

The process switch mechanism is illustrated in figure 4–28.

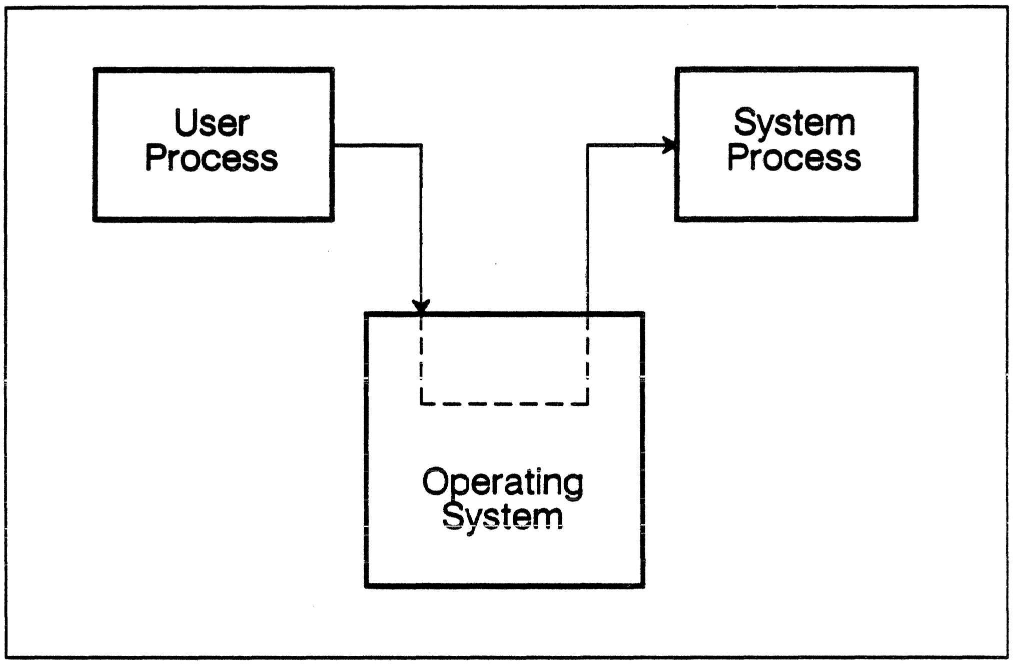


Figure 4–28. Process switch protection mechanism.

New features of the computer system hardware make intraprocess protection possible within the CP memory (CPM). A process's address space may be partitioned into distinct areas known as *domains*.

Each domain has a set of *keys* that unlock the pages of CP memory the domain has access to. All operations in the domain are limited to this CP memory space.

Hardware mechanisms, known as the forward and backward domain change instructions, enable movement between domains while preserving the integrity of each domain.

The strict enforcement of intraprocess address space protection eliminates the need for a costly process switch to service most of the operating system requests. With domains, a call imbedded in a process is made to an appropriate operating system routine that is isolated in another domain of the same process. After hardware validation of the call, the domain containing the operating system routine is entered.

Domain protection is illustrated in figure 4–29.

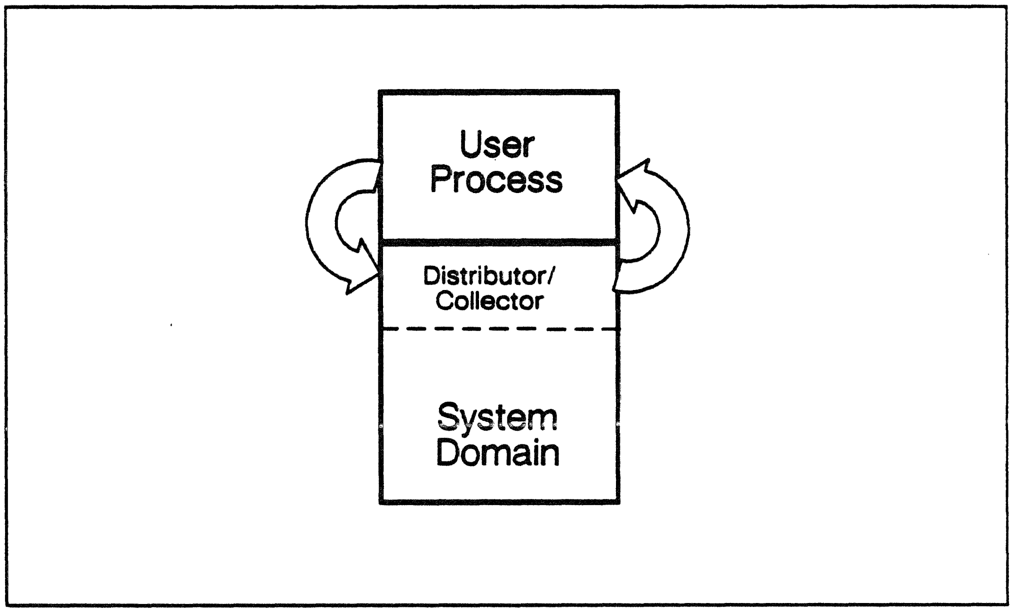


Figure 4–29. Domain protection using a domain distributor/collector.

Entering a domain means that access is permitted to the addresses that are unlocked by the domain's specific set of keys. Until the domain is entered, these addresses are inaccessible to the process, and upon exit from the domain, they are again inaccessible. Thus, the domain is part of a process, though its address space is hidden from the rest of the process. Similarly, the rest of the process cannot be seen by the domain.

Domain Software

The operating system software, especially the domain management feature, provides a means for the operating system to use the computer system domain hardware. The primary intent is to protect the operating system software from unauthorized access.

Partitioning software into domains is protective, because the address space of a domain cannot be accessed by another domain unless it is explicitly shared. Each domain is structured so that there is a single point of entry and a single point of exit. At these two points, processing specific to the domain can be performed.

Domains also define the packaging of the operating system's software. The smallest replaceable unit of software is a domain. A *process image* is a linked set of one or more domains. A process image consists of the domains that are accessible to a process; a domain cannot be used by a process unless it is part of the process image. Thus, a process image is the smallest package of a software release.

There are three types of domain management facilities:

- **Generation facilities** create and install domains and process images so tat they may be accessed later during linkage and execution.
- **Linkage support facilities** retrieve domain information that is required to link a program to a process image.
- **Execution facilities** initialize the domain information for each process, and dynamically manipulate domain information within an executing process.

There are two domains:

- **System domain** is a library of operating system procedures. A system domain has no entry point.
- **Application domain** contains the program entry point. It calls the system domain when necessary.

Generally, the application domain is a user program. An operating system server program, however, such as the CP memory pager, is also an application domain.

Domain Generation

Domain generation facilities provide a means to reconfigure the system's CP memory software, in particular the domain and process images. Reconfiguration is static: All configuration changes are made to a copy of the current configuration. The changes are recorded during domain generation so that they may be applied to the system's software configuration during system initialization.

The two major elements of domain generation are:

- **Domain editor** which collects information about the domains and process images to be manipulated. This utility is intended for ETA Systems use only.
- **Software reconfiguration** which manipulates domain or process image information in the system configuration table. These procedures may be invoked by the domain editor or system configuration management utilities.

To build object modules into a domain, an analyst specifies directives to the domain editor which describe how the code and data within the object modules are to be shared among the two domains.

Domain editor translates these directives into a set of domain keys. As illustrated in figure 4-30, these keys and other domain attributes, and the file containing the domain's object modules are used by the software reconfiguration feature to add a domain to the configuration.

Domain editor and software reconfiguration can also delete inactive versions of a domain, and display information about a domain.

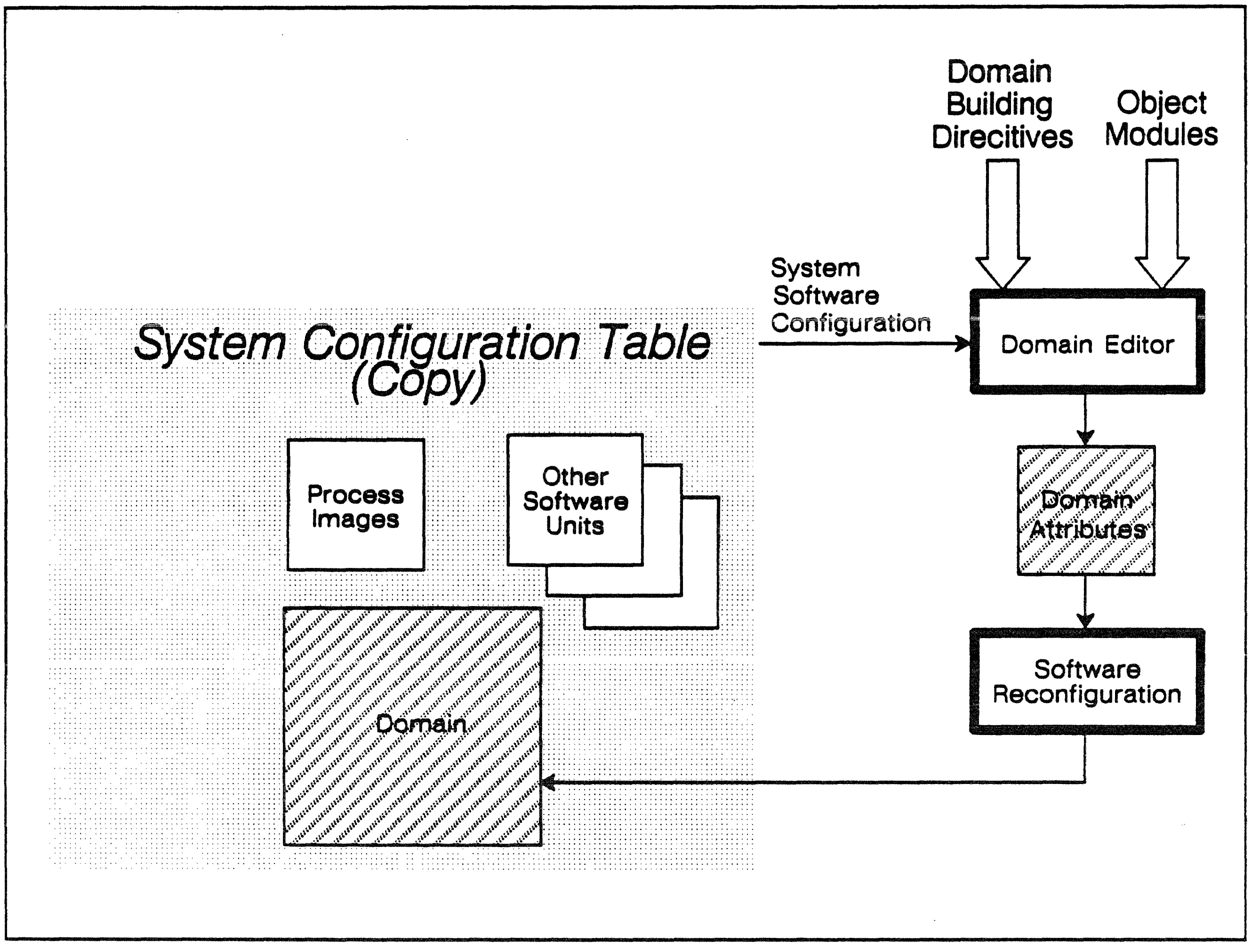


Figure 4-30. Building a domain.

Domain editor creates a process image by linking the two domains together. To build these domains into a process image, the analyst specifies directives to domain editor which describes the process image. Using the specified configuration, domain editor determines the attributes of the new process image. As illustrated in figure 4-31, these attributes are used by software reconfiguration to add a process image to the configuration.

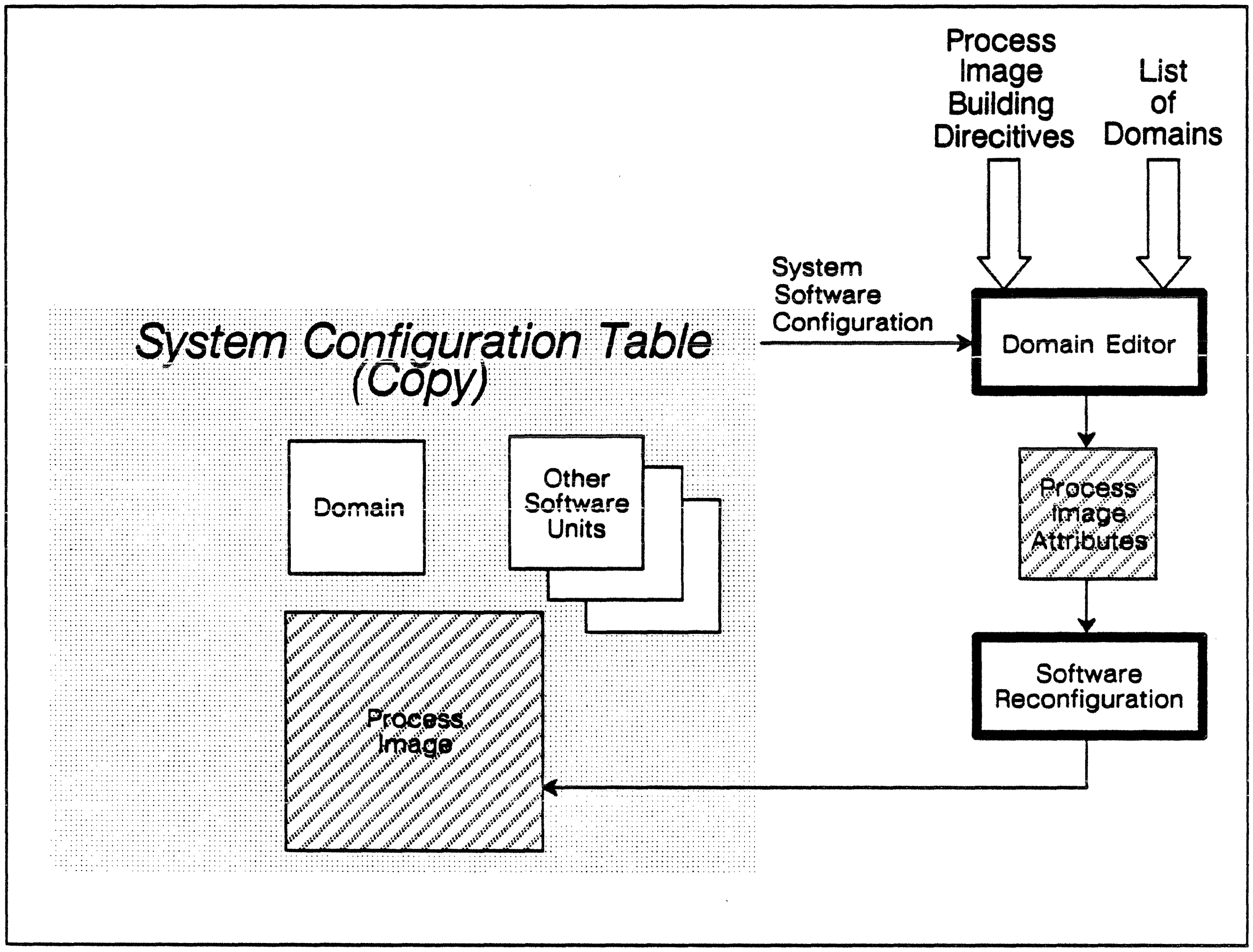


Figure 4–31. Building a process image.

Later, the process image is used by the operating system kernel features to create and execute processes.

Domain editor and software reconfiguration can also set a default image of the process image, delete inactive versions of a process image, and display information about a process image.

Domain generation facilities ensure that each domain and process image have a globally unique name, and that each of the two domains in a process has a unique virtual address space to execute within.

Once domains and process images have been manipulated in the local copy of the system configuration table, the analyst calls to the domain editor to dump a record of the modifications in a format that can be transferred to the actual system configuration table during system initialization.

Domain editor copies the system configuration table modifications to a file, and also provides a list of computer system files that correspond to the system configuration table modifications (e.g. domain executable files and process package templates). These files are transferred to the device that will perform actual system configuration table modification during system initialization.

Domain Linkage Support

A procedure may call any procedure that is in the domain. Outside of a domain, however, a procedure may call only those procedures the other domain has externalized. These externalized procedures are known as *domain entry points*.

During generation of a domain, a list of domain entry points is specified as an attribute of the domain. The list is stored as a domain information attribute in the system configuration table.

During generation of a process image, all of the member domains' entry points are gathered into a single list and stored in the system configuration table as a component of a process image. Domain management uses this list to restrict linkages between the domains to only those entry points.

When linking a program to a process image, if a called procedure is not in the program's search path (i.e. within the program or its associated libraries), the LOAD utility invokes domain management's linkage support facilities to see if the procedure is a domain entry point. If it is a domain entry point, domain management returns information so that LOAD can complete the linkage. If the entry point is not available, the reference to the procedure remains unsatisfied.

Domain Execution

Each process executing in the computer system must have a *process package* containing the information that is required and manipulated by the hardware during process execution. During process creation, Process Package Initialization creates a process package from the appropriate process image information in the system configuration table. This procedure is illustrated in Figure 4–32.

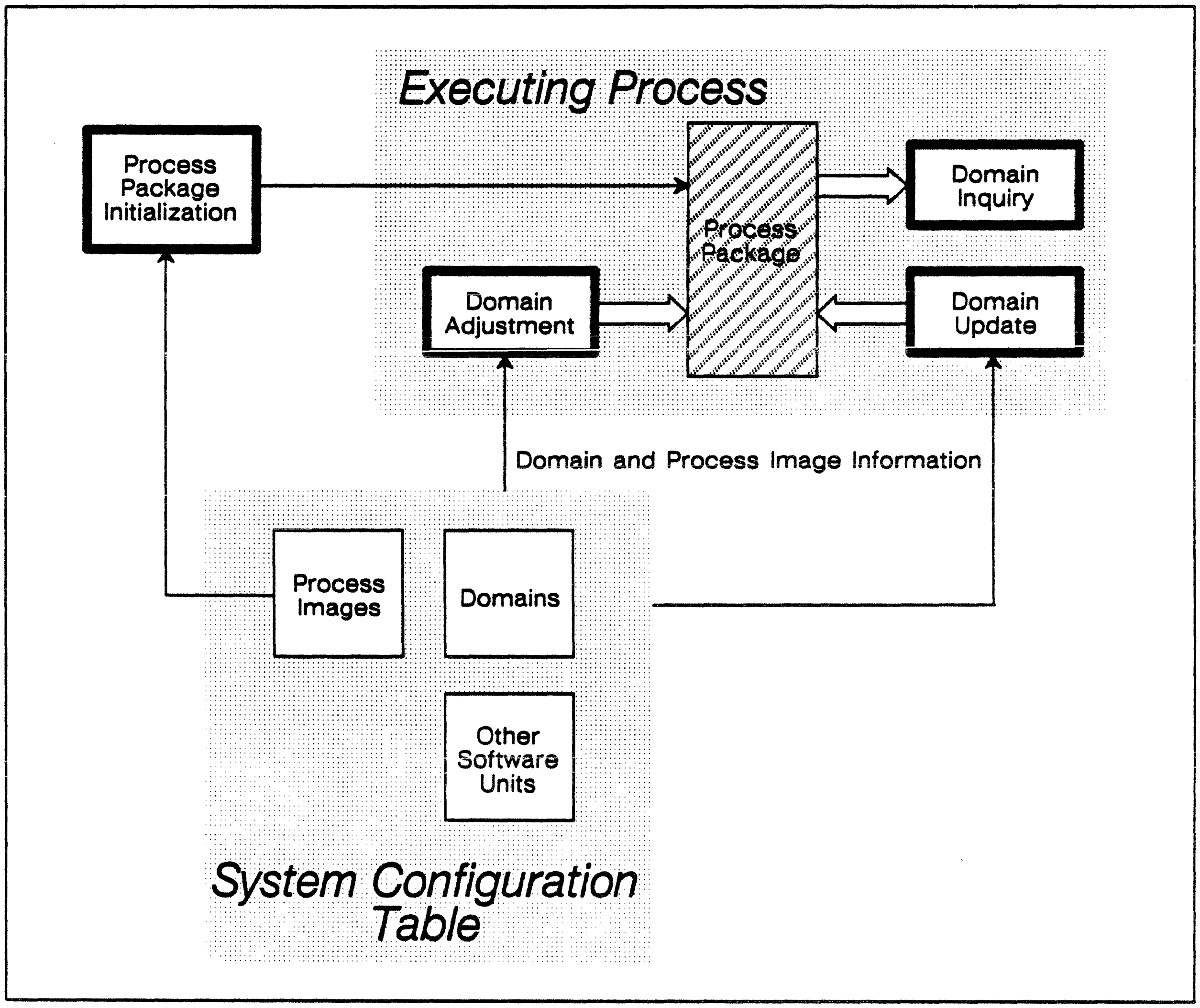


Figure 4–32. The process of process initialization and execution.

A *process shadow package* is also created. The shadow becomes the process package when special error or interrupt conditions are processed.

During execution, various operating system kernel features require information about their domain or the domain that is calling them. Domain inquiry procedures provide information about the current domain and the calling domain. This information is retrieved from the process package and/or the system configuration table.

When information about a domain must be modified during execution, it is handles by domain management's domain update procedures. The domain information being modified is verified based on the current configuration.

During execution of a process, the Domain Adjustment procedure returns information about the keys of a domain.

Domains

A domain is the basic unit of virtual CPM address space protection offered by the computer system hardware and software. This protection includes:

- **Coupling each virtual address with a lock**. The lock can be thought of as a virtual address qualifier that is needed to determine the corresponding physical address.
- **A set of keys for each domain.** These keys are compared with the lock of every virtual address referenced by the domain. If the domain does not have a key that matches a lock, if it is impossible for the hardware to determine the corresponding physical address, thereby forbidding access.
- **Specific access rights for each key in a domain.** These access rights define the operations that the domain can perform on the address space associated with the key. A domain may have the proper key, yet be denied particular operations because the domain does not have the proper access rights.
- **Restricting the list of domains any particular domain may call.** A hardware mask limits the domains that a particular domain may call.

A domain, in software terms, is code and data that has been packaged into discrete functional units. The code provided with the computer system is divided into functions.

Using domain management features, functions are assigned:

- A set of keys
- Access rights
- A list of domains it may call

Thus defined, the hardware is then able to isolate functions from each other, as well as from user code and data.

Domain management defines two domains:

- **System domain** is a domain built from object modules and registered in the system's software configuration before to being used within a process.
- **Application domain** is a domain containing the code and data from a user program (as well as the compilers, applications,

and libraries provided by the system) during the execution of a process.

Hardware does not distinguish between these two domains. Software, however, recognizes several differences:

- There is, at most, one application domain per process. The other is the system domain.
- During the building of the system domain, directives may be specified to control the accessibility of the address space. The application domain does not have this flexibility.
- The system domain must be registered in the system configuration before it can be accessed by a process. An application domain is not registered in the system configuration.

Domain Identification

Domain identification uniquely distinguishes a system domain in the software configuration. The identification has four components:

- Domain name
- Version name
- A list of the domain's functions
- System unique identification

A system domain's list of functions indicates the operations that may be performed in the domain. This list is used by a system domain that is being called by another to verify the that the caller has a legal intent.

Caller Domains

Each system domain has the option of restricting which domains may call its routines. A domain that calls another is called a caller domain.

A list of caller domains is associated with each system domain via the domain editor. The default list of caller domains is all domains.

During execution, the hardware uses a forward domain change mask. This mask is a list of valid caller domains. The hardware permits a system domain to call only those domains that are listed in its mask.

Domain Distributor and Collector

When a routine within a system domain is called from another system domain, the routine is not directly branched to. Instead, the forward domain change instruction forces the called system domain to be entered at one particular address, no matter which routine in the domain is being called. Domain management ensures that this address is the start of the domain distributor.

The domain distributor is added to a system domain via domain editor. This distributor performs special processing for the system domain before branching to the routine that the caller explicitly requested. Since the distributor is a system domain's single point of entry, it can screen incoming calls for all routines in the domain.

A minimal distributor contains:

- Domain entry points
- Dynamic stack adjustment
- Data flag branch manager adjustment

When the called routine completes, the return to caller is not direct. Instead, the exit is through a domain collector whose last instruction is a backward domain change. The collector is also added to a system domain via domain editor. The domain collector comprises the:

- Dynamic stack adjustment
- Data flag branch manager adjustment

Each system domain has a distributor and a collector.

Domain Entry Points

A system domain exposes only the routines that other domains are permitted to call. These callable routines are known as *domain entry points*. The true entry point is the domain distributor, and the distributor branches to the correct routine when it finishes its processing.

A domain distributor with multiple entry points and two hidden internal routines is illustrated in figure 4–33.

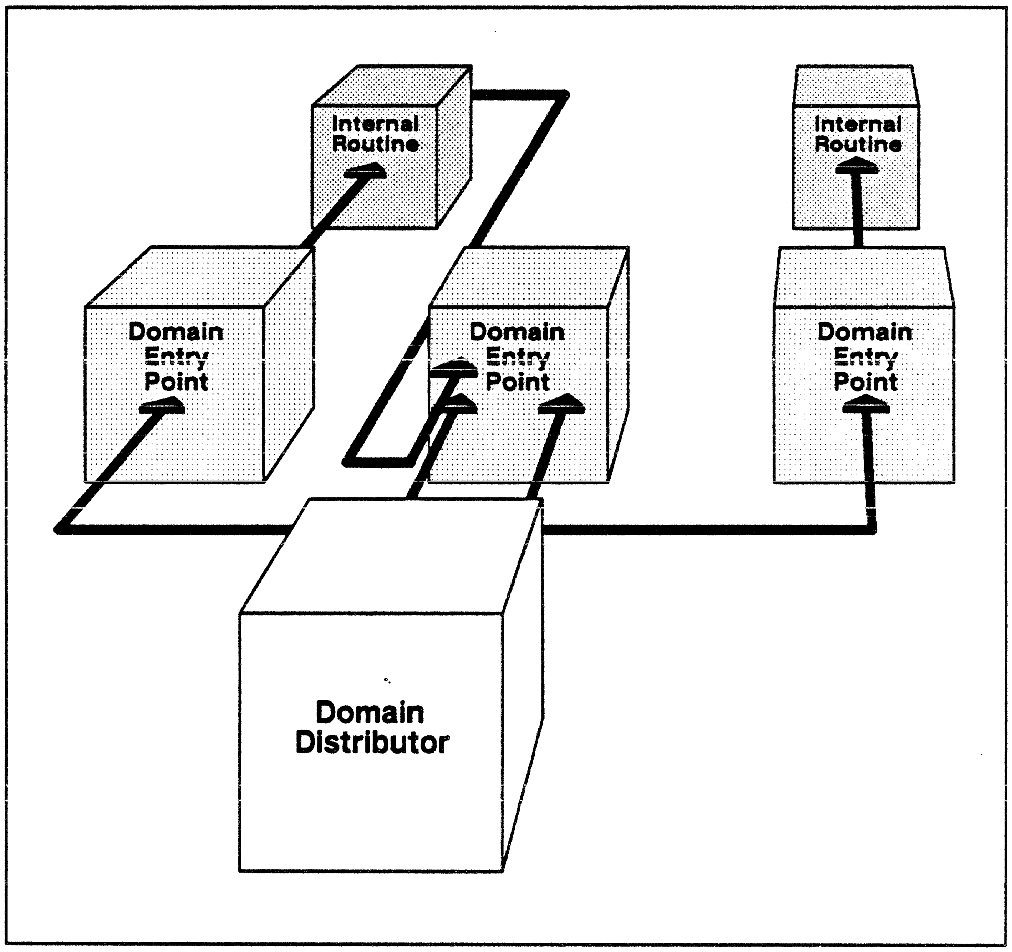


Figure 4-33. A domain distributor with multiple entry points and two hidden internal routines.

Before branching to the called domain entry point, the domain distributor sets the database register to the called routine's database.

Dynamic Stack Adjustment

Each routine in a process has a dynamic stack which serves as a storage area for the caller's environment registers and dynamic working storage area. The domain distributor and collector manages the dynamic stack for the domain's entry point routines.

Entry points within a system domain can be invoked by other domains several times in the chain of execution. The domain distributor must therefore keep each instance of a domain (i.e. each domain entrance) in a separate frame in the domain's dynamic stack.

When an instance of a domain ends through a backward domain change, the collector removes that frame from the dynamic stack.

Thus, the distributor and collector ensure that each instance of a domain has access to its own dynamic stack information.

An instance of a domain should not access another's dynamic stack frame, though domain management does not forbid it.

Certain languages store parameter lists on a dynamic stack. These languages must share the dynamic stack between the caller and callee domains.

Data Flag Branch manager Adjustment

The data flag branch manager is available to the application domain. No data flag branch interrupts are lost by a forward domain change because this instruction waits for all interrupts to be posted before its execution. Some interrupts might not be processed, however, before the domain change. They will be processed upon return to the application domain.

Prologue and Epilogue

The operating system kernel calling sequences require that a called routine store critical portions of the caller's register file, so if the called routine uses these registers, the caller's values can be restored when the called routine completes its processing.

For a typical routine, compilers generate a *prologue* for register storage, and an *epilogue* for register restoration. The prologue also establishes the registers needed by the called routine.

Keys

The domain management feature is responsible for managing the keys of each domain during its generation and execution. A domain's keys completely define the virtual addresses that it can access during its execution in a process. This is enforced by the computer system hardware.

A key is assigned to each virtual address when a virtual address range is added to a domain within a process. CP memory manager records this information in the page table. The page table equates a virtual address in software to a physical address in hardware. Each process has its own page table.

When stored in the page table, this key value acts as a lock; for every virtual address reference, the hardware scans the page table for a matching address with a lock that matches one of the executing domain's keys. If none matches, the hardware cannot determine the physical address, and access is thereby forbidden.

A key in a domain describes a set of address ranges that are accessible to the domain. The address ranges do not need to be contiguous. A virtual address range cannot be assigned to more than one key in the process because there is only one lock per virtual address.

When a domain is built, the domain editor interprets directives to specify the intended structure of the domain. This determines how the keys are assigned to the virtual addresses in the domain.

Domain management implements four types of keys:

- **Parameter keys** are used for the application domain. The application domain's code and data is placed entirely under two parameter keys.
- **Private keys** are used for system data. All system data is placed under private keys.
- **Shared code keys** are used for domain code and for read only data.
- **Unused keys** are keys with no virtual addresses assigned to them. Unused keys are place holders for keys that are not used.

The computer system hardware supports only three key types:

- Incoming parameter keys
- Outgoing parameter keys
- Non-parameter keys

Internally, domain management relates its key types to one of the hardware key types. The hardware supports exactly 12 keys per domain during its execution. Domain management ensures that each domain has 12 keys.

Four of the 12 keys are parameter keys (two incoming and two outgoing). The other eight keys are some combination of private, shared code, or unused keys. Each of these keys has memory access rights.

Parameter Keys

Parameter keys permit the calling and called domains to share data. A domain cannot access its caller's virtual address space unless the keys to the space are in the invisible package during its execution. The parameter key hardware provides a mechanism allowing a domain to leave keys in the invisible package for the next domain called.

In an executing process's process package, the invisible package contains the executing domain's keys. The process package also contains a domain package for each domain.

The domain package is a unique storage area for each domain's keys. Upon a forward domain change, most of the caller's domain keys are replaced in the invisible package by the callee's domain keys from the callee's domain package. The keys that are not replaced are the parameter keys.

The computer system hardware provides four parameter keys in the invisible package and two in each domain package. The two in the domain package are *outgoing* parameter keys.

While a domain is executing, its outgoing parameter keys are in the invisible package. During a forward domain change, the outgoing parameter keys are copied to the *incoming* parameter key locations in the invisible package just before loading the callee's domain keys. This activity is illustrated in figure 4–34.

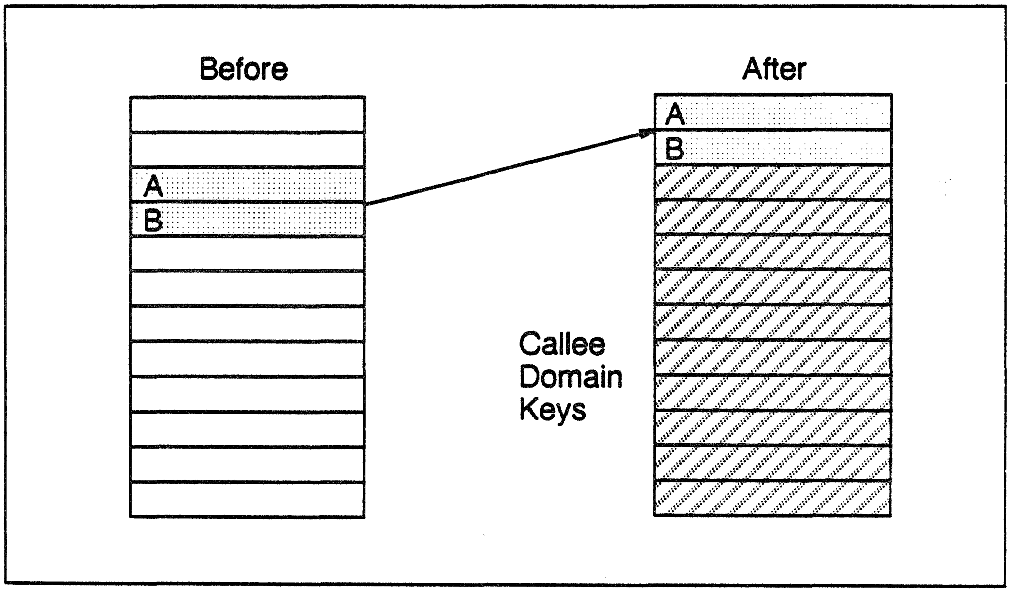


Figure 4–34. Shifting incoming parameter keys to outgoing parameter keys in a forward domain change.

The caller domain controls access to its address space by giving the callee these two keys and their corresponding access rights. The incoming parameter keys are not saved in the domain package, so that the callee domain has access to the space only when called from that particular domain.

The hardware stipulates that no two key values in a domain can be equal. Thus, a domain cannot execute a forward domain change to itself because the incoming parameter keys would then equal the outgoing parameter keys.

The LOAD utility enforces this by first searching the external references within the domain. If the entry point is found within the domain, the link is made with a branch instruction instead of a forward domain change.

The hardware also stipulates that a virtual address range cannot have two keys assigned to it. Thus, a domain cannot directly pass an incoming parameter as one of its outgoing parameters. It must instead copy the parameter to an address range that is assigned to either an outgoing parameter key or a shared key. Domain management does not do this copy; the parameter must be manually copied by the domain itself.

Private Keys

A private key guarantees that the virtual address space assigned to the key is inaccessible to the other domain. In other words, code and data assigned to a private key are private to the domain. Domain management assigns a key value that is unique throughout the process to each private key.

Shared Code Keys

A shared code key is assigned to virtual addresses that must be accessed by more than one domain. Sharing may be done with parameter keys, but this requires a caller/callee relationship between the sharing domains. Shared code keys require no such relationship.

A shared code key is used for any system domain code (as denoted in the object text) that needs to be shared between domains. Domain management ensures that each process has the appropriate shared code key values in the appropriate domain packages so that the code can be read and executed.

The application domain does not have shared code keys.

Unused Keys

Unused keys are placeholders in the domain package. No virtual addresses are assigned to an unused key.

The keys which are not reserved by domain editor during a domain's generation are unused keys. Like any other key in the domain package, an unused key cannot equal any other key in the domain, nor can it equal any key that the domain could receive as an incoming parameter.

Memory Access Rights

Domain management associates one or more of the memory access rights with each key in a domain:

- **Read** permits virtual addresses to be read with the key
- **Write** permits virtual addresses to be written with the key
- **Execute** permits virtual addresses to be executed with the key
- **Null** does not allow virtual addresses to be accessed with this key.

Null, or any combination or read, write, and execute constitute a legal memory access right specification.

The memory access rights define the operations that a domain may perform on its accessible virtual addresses. All keys have memory access rights.

Memory access rights are domain-specific. For example, two domains may share a key value, but the first domain may have read and write memory access rights, while the second has only read access rights. If the second domain tries to write to a virtual address covered by that key, it results in an access violation.

The memory access rights associated with each type of key are as follows:

- **Parameter keys** may have either read/execute or read/write/execute access rights
- **Private keys** may have read/write/execute access rights
- **Shared code keys** have read/execute access rights.
- **Unused keys** are not assigned access rights.

It should be noted that the file system permits domain-specific access rights to be associated with a file. The memory access rights, however, cannot be greater than the rights permitted the key, though they may be less.

Process Package

The process package is an object that contains information about all the domains in a process. The process package is required by the hardware, and managed by domain management. It contains:

- The hardware information about the executing domain (the invisible package)

- The storage area for the hardware information on all the domains in the process (the domain packages)
- A stack for domain packages that were executing, and which will be returned to later (the stacked domain packages)

A typical process package is illustrated in figure 4–35. Its structure is stipulated by the hardware. It must occupy contiguous addresses and, for an active process, it must be locked in CP memory.

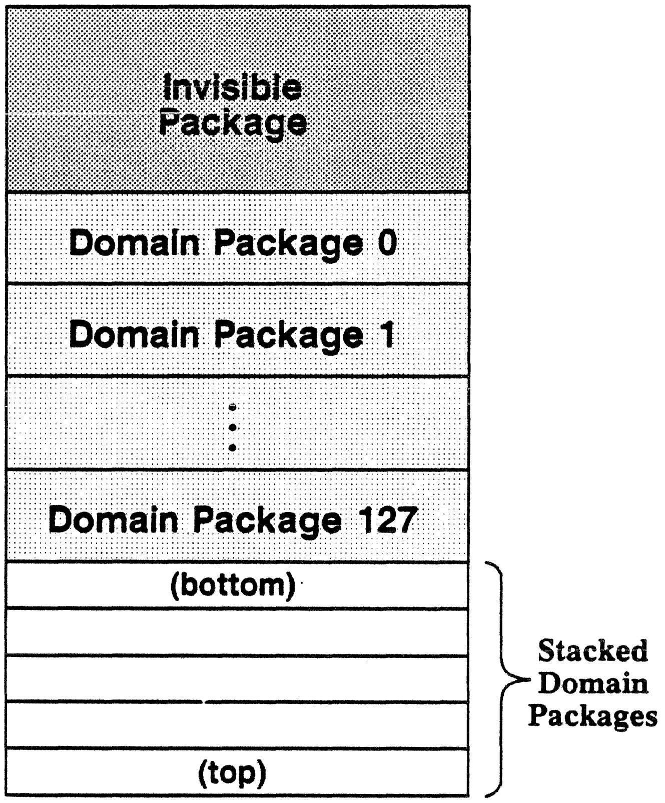


Figure 4–35. Process package structure.

Each process has a process package. The process package initialization function of domain management sets up the process package of each process from a template. The template varies depending on the environment in which the process will execute.

There is a unique process template for each process image. The template varies depending one the domains that are included in the process image. The template is a component of the process image; it is created by the software reconfiguration feature when a process image is built, and is recorded in the system configuration table.

Invisible Package

The invisible package contains hardware information that is required for process execution. Domain management is responsible for initializing the invisible package. During execution, however, it can access the invisible package only indirectly, through the domain packages.

During a forward domain change instruction, the following items are loaded from the called domain's domain package:

- Current program address
- Forward domain change mask
- Illegal instruction mask
- Data flag branch manager register
- Keys and access rights
- Instrumentation counters and select codes
- Communication Buffer base/limit address pairs

Prior to the load, some information about the calling domain is saved on the domain stack, such as:

- Program address
- Current and previous domain package numbers
- Current incoming and outgoing parameter keys

The calling domain's instrumentation counters are copied back into its domain package. Hardware then adjusts the domain stack pointer, current and domain package numbers, and the parameter keys, and loads the called domain's information. After this, the process is executing in the called domain.

Domain Package

Each domain in a process has its own domain package. It contains domain-specific information that must be in the invisible package when a domain is executing. The domain package can be built statically prior to execution in a process, or dynamically during execution of a process.

The domain package is domain management's primary data structure:

- **Domain update** modifies other domain package information.

- **Domain inquiry** returns information from the domain package.

Domain management provides interfaces to certain components of a domain package:

- Keys and memory access rights
- Forward domain change mask
- Illegal instruction mask
- Current program address
- Communication buffer base/limit pairs
- Instrumentation select codes

There are a maximum of 128 concurrent domain packages in a process. All domain packages are initialized from a process package template. The domains that are in the process image have their actual domain package information initialized in the template. All other domain packages are initially empty, and are filled in when the domain is first invoked during the process's execution.

Stacked Domain Packages

When a domain invokes another domain, the caller's state is automatically preserved by the hardware. The hardware moves a portion of the caller's domain information from the invisible package into a frame of the domain stack. Information such as the following are put in the stack:

- Current program address
- Current and previous domain numbers
- Data flag branch manager status
- Parameter keys

When a backward domain change is executed, the hardware restores this information to the invisible package. The remainder of the invisible package is copied from the caller's domain package.

Process Shadow Package

Each process has a second set of process package information called a *process shadow package*. It permits a domain to have different process package information during special situations, such as processing of illegal instruction conditions.

The address of the process shadow package is known to process management and the operating system. When a special condition is encountered, domain management is called to build a shadow package. The shadow is used to execute the process instead of the original process package. Thus, the shadow package can be used to recover from conditions that leave the original process package in an unexecutable state.

When recovery is complete, process management invokes domain management to reinstate the original process package. Process management then causes execution to resume in the original process package

Process Images

In the absence of dynamic domain linkage, any call to the system domain that might be made during a process's execution must be included in the process when it was created. Thus the image of the domain, called a *process image*, must exist in a process at the start of execution.

Domain editor provides the human interface to build process images from the system domain name as listed in the system configuration table. Software reconfiguration manipulates the process images in the system configuration table.

The domain packages for the domains in a process image are initialized in a process package template. The process and shadow package templates are a component of the process image. Any domain that is a member of the process image is also recorded in the system configuration table. These system configuration table components are used by various operating system kernel features to create and execute processes.

All domain entry points for the process image are also recorded in the system configuration table. This list is used to link programs to the process image.

A process image may also include the program (i.e. the application domain). This is known as a *server process image*. Server process images are complete images of the process; they are not used to link other programs. A *non-server process image* contains only the system domain, and must have a program linked to it before it can be executed as a process.

A process image has an ASCII name and version, and a name unique within the system that is similar to a domain's identification. There is, however, no domain function list for a process image. One version of the process image may be designated as the default version, and this version will be used when a version name is not explicitly specified.

Application Domain

The application domain contains the transfer address of the program that is to be executed in a process.

For non-server process images, the application domain is the only domain in the process image that is not in the system configuration table. The application domain of a non-server process image is not processed by domain editor or software reconfiguration. It is simply a collection of object modules, including a program, that is linked to a specified process image prior to execution.

A user's program is a typical application domain.

For server process images, the application domain is an operating system program. The program's object text file is processed by domain editor and software reconfiguration. The executable version of the program is recorded in the system configuration table.

The application domain has keys and access rights that are different than the system domain. The application domain is not permitted to use the backward domain change or shared memory instructions.

Domain Management Shared Information

Some information is implicitly shared between domain management and other operating system kernel features.

Process Package and Shadow Process Package

The memory copy of the process package may be read by either the system or application domain.

Dynamic Stack

The domain distributor manages a dynamic stack frame pointer for each instance of a domain. Domain management Inquiry also uses the dynamic stack to determine if the caller of a domain entry point is inside or outside of the domain.

The LOAD utility and compilers also manipulate the dynamic stack.

Domain Management Functions

The Domain management functions are divided into the following categories:

- Domain inquiry
- Domain update
- Process package adjustment
- Domain linkage support
- Software reconfiguration

- Domain distributor/collector
- Domain editor

Domain Inquiry Functions

The domain inquiry functions provide information about the domains in a process. These interfaces reside in both domains, no domain change is required to invoke them. The domain inquiry functions are:

- **Get Caller Domain ID** identifies the domain that called the inquiring domain. The information returned is the domain name, version, and domain function list.

  It should be noted that this function always identifies the caller (previous) domain, even if the last procedure call came from within the current domain.

- **Get CB Base Limit Access Pair** provides the contents of the communication buffer base/limit/access pair from the domain package of the specified domain.

- **Get Domain Access Rights for Address** returns the access rights that the domain has to the address.

- **Get Domain ID** identifies the inquiring domain. The information returned is the domain name, version, and domain function list.

- **Get Domain Instrumentation Counters** provides the values of all instrumentation counters for the executing domain.

- **Get Instrumentation Counters** provides the values of all instrumentation counters for the specified domain.

- **Get Origin of Caller** indicates whether the domain entry point procedure was called from inside the current domain, or from the caller (previous) domain. This function can be used to determine the type of validation that must be done for the caller of the domain entry point.

Domain Update Functions

The domain update functions change information other than keys in the domain package. The function Set Domain Instrumentation Counters is available to the application domain; the others are not.

All these interfaces, except Get Domain Software Unit ID require a domain change because domain information in the invisible package can only be updated indirectly via domain package updates. It is the domain change instructions that move the updated information into the invisible package.

The domain update functions are:

- **Get Domain Software Unit ID** retrieves the system configuration table software unit identification of any domain in a specified process image.
- **Set Application Domain Program Address** sets the transfer address for the application domain.
- **Set CB Base/Limit/Access Pair** sets the contents of the communication buffer base/limit/access pairs in the domain package of the specified domain.
- **Set Domain Instrumentation Counters** sets the select code of the specified instrumentation counters for the requesting domain. The values of these counters are reset to zero. Counters 6 and 7 cannot be set.
- **Set Instrumentation Counters** sets the select code of the specified instrumentation counters for the specified domains. Counters 6 and 7 cannot be set.

Process Package Adjustment Functions

The process package adjustment functions prepare process packages and process shadow packages for execution, and manipulates keys in the domain packages.

The process package adjustment functions are:

- **Initialize Process Package** creates the initial process package for each new process prior to its execution. The resulting process package is used by Process Management during the initiation of a new process.
- **Initialize Process Shadow Package** creates the initial process shadow package for each process. All information in the process shadow package is initialized from the process shadow template, except for the domains' instrumentation counters. This information is copied from the specified process package. The resulting shadow package is used by process management for interrupt processing.
- **Reset Process Package After Interrupt** copies a specified process package to a target process package, without overwriting the instrumentation counter information. This function is used by process management to reinstate the original process package after interrupt processing.

- **Get Actual Key Value** returns an actual key value given the type of key and its access rights, as well as the domain and process image identification.

Domain Linkage Support Function

The domain linkage support function provides domain entry point information for a process image. This interface may be called from any domain, but is intended for use by the LOAD utility. This procedure does not require a domain change, but they must be domain entry points if their callers are in a different domain.

The domain linkage support function is:

- **Get Domain Entry Point Information** locates the specified entry point within a domain, and returns the domain information needed to create a linkage to that entry point.

Software Reconfiguration Functions

The software reconfiguration functions support manipulation of domains and process images in the system configuration table so that they may be used by an executing process. These functions are to be called by domain editor. The domain editor executes in the application domain, and these functions may also reside in the application domain, unless restrictions are imposed by the operating system kernel features called by these functions.

The software reconfiguration functions are:

- **Add Domain to Configuration** installs a version of a domain in the specified copy of the system configuration table.
- **Add Process Image to Configuration** installs a version of a process image in the specified copy of the system configuration table.
- **Delete Domain from Configuration** deletes a version of a domain from the specified copy of the system configuration table. This function does not delete the files (object modules and the executable file) associated with the domain.
- **Delete Process Image from Configuration** deletes a specified version of a process image from the specified copy of the system configuration table. This function does not delete the files (program object modules , program executable file, process package template, process shadow package template, and domain entry point file) associated with the process image.

- **Dump Configuration Information** dumps the information from a specified copy of the system configuration table. This information is used to manipulate a domain or process image in the system's system configuration table at the next system initialization.

  It should be noted that this information must be transferred to the service unit that executes the system initialization.

- **Get Domain Information** retrieves information about a domain from a specified copy off the system configuration table. In general, all information in the domain software unit entry and domain information attribute block is retrieved.

- **Get Process Image Information** retrieves information about a process image from the specified copy of the system configuration table. In general, all information in the process image software unit entry and process image domain list attribute block is retrieved.

- **Set Default Process Image in Configuration** marks a configured version of a process image as the default version in the specified copy of the system configuration table. A default version is used by the operating system kernel features that do not specify a version name when accessing a software unit.

Domain Distributor/Collector Function

The domain distributor/collector function defines the true entry and exit points of a domain. Each domain contains a domain distributor/ collector module. The module contains a function which is both distributor and collector.

This function is not explicitly called by any other operating system kernel feature. It is, however, implicitly called by each forward domain change instruction.

The domain distributor/collector function is:

- **Domain Distributor/Collector** provides a single point of entry (distributor) and exit (collector) for a domain. The distributor prepares the domain for execution by sorting pertinent registers of the caller for later restoration, and setting its own registers, including a new frame of the dynamic stack.

  The distributor portion of the procedure is terminated when it branches to the domain entry point that was actually called (i.e the procedure call which resulted in a forward domain change to this domain).

The collector portion of this procedure begins when the called domain entry point is exited. The collector restores the callee's registers and issues a backward domain change.

Domain Editor

The domain editor is the human interface to domain management's software reconfiguration functions. Domain editor directives correspond directly to the functions described for software reconfiguration. The domain editor directives are:

- Build domain
- Build process image
- Delete domain
- Delete process image
- Display domain information
- Display process image information
- Display system configuration table modifications
- Dump configuration information
- Set default process image

In general, domain editor directives are a tool to provide input to, and get output from software reconfiguration interfaces. An analyst or system installer uses the domain editor utility to construct a domain that can later be installed in the software configuration.

Input to domain editor includes:

- A set of directives that can be entered interactively or from a file

- Object text files

Output from domain editor includes:

- Status messages that can be viewed interactively or from a file

- Executable files

- Various information files

Section 8: The System Managers

This part examines the features that monitor the system for status, record and report system status, and keep track of system configuration. It consists of four sections:

- System configuration control
- System monitor
- System logging and analysis

System Configuration Control

System configuration control (SCC) is responsible for maintaining an accurate record of all elements of the system configuration. The definition and status of these elements, as well as their interrelationships, is contained in the system configuration table (SCT) which is managed solely by system configuration control.

The relationships between system configuration control and other system features is illustrated in figure 4–36.

System configuration control executes in the service unit and the CPUs. Multiple copies of the system configuration table are maintained in the service unit to ensure reliability. For performance and availability, the system configuration table is maintained in each CPU. Access to shared portions of the system configuration table is provided through the communication buffer.

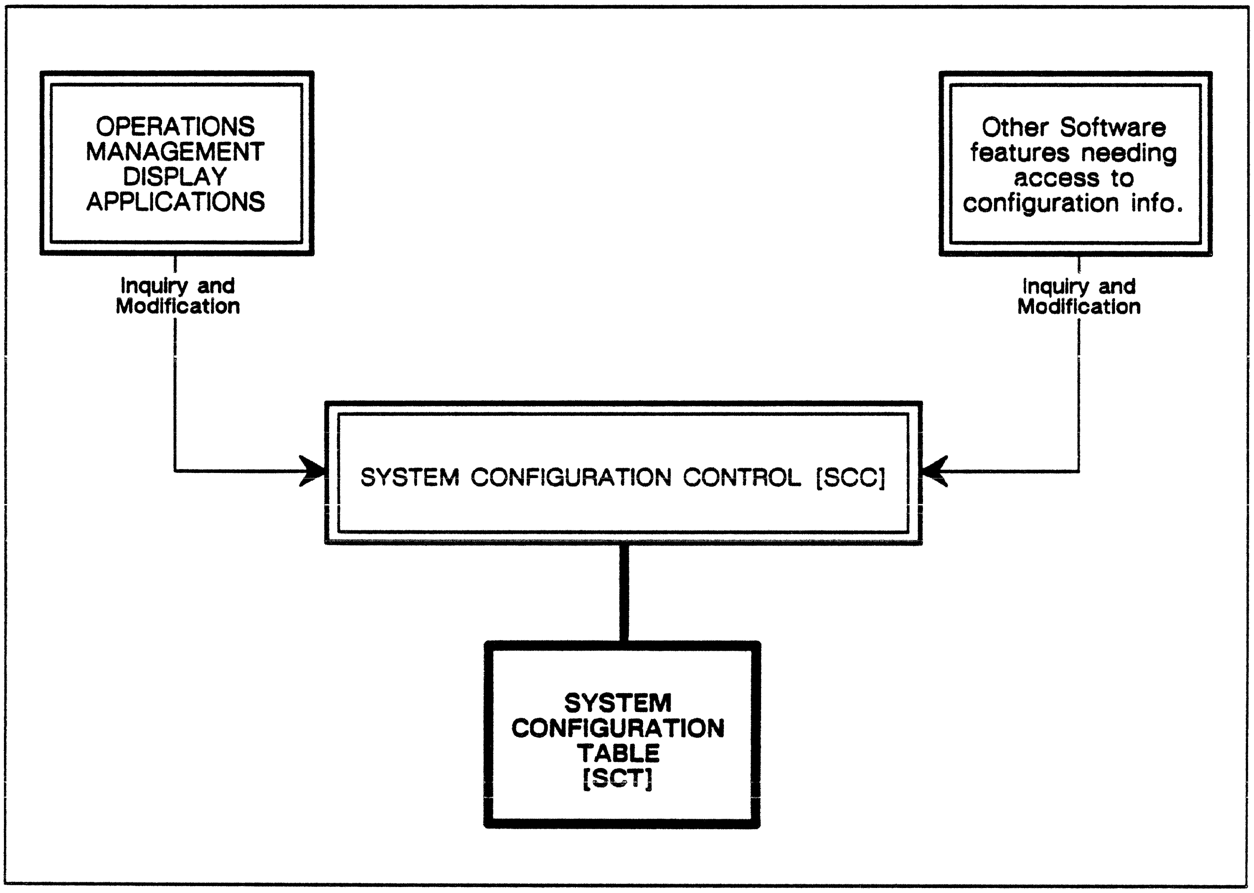

OPERATIONS MANAGEMENT DISPLAY APPLICATIONS
Other Software features needing access to configuration info.
Inquiry and Modification
Inquiry and Modification
SYSTEM CONFIGURATION CONTROL [SCC]
SYSTEM CONFIGURATION TABLE [SCT]

Figure 4–36. Relationship between system configuration control and other system features.

System Configuration Table

The system configuration table contains a wide variety of configuration information, including:

- Hardware unit definition and interrelationships
- Hardware unit status and attributes
- Software server/driver status and communication path identification (i.e. mailboxes)
- Software unit versions and attributes
- Workloads, status, and CPU partitions with their parameters.
- Logical disk configuration and allocation information
- Network configuration parameters and statistics
- Site-modifiable system parameters (defaults, limits, thresholds, tuning parameters)

- System clock maintenance parameters

The system configuration table can be extended. New attributes may be defined for the system or for any configurable unit. These attributes are maintained in extension blocks which system configuration control treats as a block of information associated with a name. The format and content of the extension block is defined and interpreted by the the software using the capability.

Additionally, new hardware and software configurable units can be dynamically defined and added to the system configuration table in a similar fashion. This is necessary in order to support hardware and software enhancements such as new peripherals and software packages.

Hardware Configurable Units

In order to dynamically manage the configuration, the following hardware configurable unit information must be maintained in the system configuration table:

- Status
- Usage
- Interrelationships

Changes in the status of these units are a result of system initialization activity or the detection of a system failure. These changes are coordinated by system configuration management with all relevant operating system components. The resulting status changes are recorded in the system configuration table.

A configurable unit is considered *demoted* when it is made less active due to status changes. The only valid demotion activity is system monitor reporting system failures.

A unit is considered *promoted* when it is made more active as a result of status changes. The only valid promotion activity is the initialization of the system.

When a unit is made inactive, it may be moved from one management environment to another, such as moving a unit from the operating environment to the maintenance environment (a unit in the maintenance environment is not available for subsequent autoloads). System configuration control is responsible for immediately recording these status changes in the system configuration table.

Configurable units are distributed in such a manner that diagnostic operations and maintenance actions may be performed on these units without the need to demote more than half the system at one time (in the fully redundant configuration). Maintenance activities which are destructive to data are prohibited on units which are part of normal system operation. Non–destructive maintenance activities such as reading CP memory or the communication buffer are allowed for certain applications which require them. System configuration control helps coordinate this activity by maintaining a record of the current management environment and the status of each configurable unit.

For configuration management purposes, software servers and drivers such as the following are treated as hardware configurable units:

- The disk physical-file system
- Remote Host Facility applications methods
- Maintenance interface server and driver

Each unit of this type performs a specific function and is responsive to changes in status and management environment. All hardware configurable units in the system are uniquely identifiable by a configurable unit name.

Software Configurable Units

For a given software configurable unit, multiple copies of different versions may exist. Each version is identified by a *version name*. For each unit, there exists a *default version* (usually the current version), which is used by the system if the process domain does not specifically request a named version.

System configuration control provides for changing the specified default version for each configurable unit. Associated with each software version is a *capability* which incorporates the unit and version names.

In addition to managing multiple versions for a configurable unit, identical copies of the same version are maintained for backup purposes. These copies are identified by a *backup copy number*. System configuration control provides for adding or removing different version and/or backup copies of a software unit. The location and identification of these software configurable unit copies is maintained in the system configuration table.

Logical Disk Devices

As a service function for the disk physical-file system and the logical-file system, system configuration control provides the interlocking necessary for logical disk device allocation. This requires maintaining allocation information associated with logical disk devices and being able to lock a logical device while space is being allocated or deallocated.

The definition and management of *device classes* is also provided by system configuration control. With the exception of a number of predefined classes, these device classes are defined by the site, and are associated with groups of logical disk devices.

System Configuration Elements

Both the hardware and software of the computer system is divided into configuration elements. These elements can be demoted and promoted by system configuration management.

Hardware Classifications

For maintenance purposes, there are three classifications for hardware elements in the computer system:

- **Field replaceable unit (FRU)** is that part of the system that can be quickly and easily removed and replaced.
- **Field degradable unit (FDU)** is that part of a system that must be removed from the active resource list to protect the system from further interruptions due to a known failed field replaceable unit.
- **Field serviceable unit (FSU)** is that part of the system which the user cannot access during a maintenance action on a failed field replaceable unit.

Software Classifications

The hardware concepts of field replaceable, degradable, and serviceable unit do not directly relate to system software due to the different aspects of hardware and software. It is appropriate, however, to make the following software considerations:

- A field replaceable unit is any collection of one or more software features. Typically software field replaceable units are associated with corrective code.
- A field serviceable unit is a collection of field replaceable units which define a product or a portion of a product. Typically software field serviceable units are associated with a new system release.

Since hardware field degradable units have no clear analogy with software, a software field degradable unit is considered to be the same as a software field serviceable unit.

For CPU software, a field replaceable unit may be a domain or, depending on the type of software, a set of one or more object modules. These field replaceable units are packaged into a domain cluster, an object file, or a library to form a field serviceable unit.

For I/O unit software, field replacealbe units consist of the I/O unit nucleus and the software for each variety of I/O processor. An I/O unit field serviceable unit is a collection of one or more I/O processor field replaceable units.

For service unit software, a field replaceable unit is a bound process or application library and a field serviceable unit is a collection of bound processes or application libraries.

Configurable Units

Generally, configurable units are the set of field degradable units. These are the system elements whose usage must be controlled and coordinated among the processing elements of the machine.

When dealing with large system components like system halves, system configuration changes are propagated to all subcomponents.

Unit Ownership

In determining how a configurable unit may be used in a given situation, it is useful to consider the unit as belonging to a particular management environment. This *ownership* can be used to reflect the status of a configurable unit in regard to what operating system and maintenance activities are permitted for it.

There are two classes of ownership for a configurable unit:

- Operating system
- Maintenance

When a unit is being used and managed by an operating system, it belongs to that operating system. If a unit has been removed from operating system use for diagnostic or maintenance activities, it is assigned to a maintenance owner.

A maintenance owner may be a diagnostic, a systems engineer, or simply unassigned. Maintenance ownership is determined by the activity required for a unit. When a unit belongs to a maintenance owner, it is isolated, on the operating system side, from that class of units which belong to the operating system and which may remain connected to other units belonging to the same maintenance owner. Maintenance software provides for isolation between maintenance owned units.

Unit Sponsorship

Each configurable unit that has a maintenance interface, such as the high-tech boards, is considered to be a *sponsorable unit*. Each sponsorable unit has one and only one sponsor which happens to be one of the service unit server nodes. Only the unit's sponsor may access it through the maintenance interface, thus ensuring that no simultaneous access can occur through maintenance interface.

The power and cooling supervisor (PCS) is also considered a sponsorable unit since only one power and cooling supervisor processor (PCSP) can actively probe the data acquisition processors (DAPs) at any given time. There are two power and cooling processors, one of which is active, the other is passive. Each is connected to a different service unit server node.

Sponsorable units have a sponsor regardless of their ownership status. This is required, as it is sometimes necessary to access units through their MI even while they are owned by the operating system.

Physical Status

Associated with each configurable unit may be one or more *physical status* attributes. These status indicators reflect physical traits associated with hardware units. The physical status type currently defined is:

- **Prepped status** indicating whether the unit has been initialized for use by the operating system. This status is only applicable for non-volatile memory units (such as disk devices) that need to be prepared or formatted before they can be used.

Logical State

Associated with each configurable unit is a *logical state* attribute which indicates the state of the unit. This state reflects how the unit is being used by its owner, including on-line and off-line.

The logical state also indicates what transitions the configurable unit may be undergoing, such as initializing. It is the logical state of a configurable unit that system configuration management uses and manipulates in performing its functions.

Logical Devices

When under the ownership of the operating system, RMS devices are grouped into logical devices. One logical device contains a disk device.

All logical status changes under operating system ownership are done at the logical device level. All logical status changes under maintenance ownership are done at the disk device level. Changes of ownership between the operating system and maintenance are done using disk devices.

Device Classes

Logical disk devices are grouped into logical device classes. These device classes are used to classify logical disk devices according to particular attributes. Some device classes are predefined by the system, the remainder being defined by the site.

A particular logical disk device may belong to many logical-device classes at one time. System configuration control is responsible for maintaining a list of valid device classes and for coordinating the assignment of logical devices to these classes.

The predefined device classes consist of:

- **System primary device class** which are the logical disk devices where the primary system files are
- **System backup device class** which are the logical disk devices where the backup system files are

At least one logical disk device must belong to each of these device classes. Furthermore, a particular logical device cannot belong to both of these device classes.

Device classes may be used by a site to partition logical devices a number of different ways. For example, device classes may be used to define:

- Family groups
- Performance categories
- Disjoint I/O paths (each disk physical-file system managing devices in different classes)

Service Unit

The service unit configuration consists of an service unit chassis and a minimum of two workstations. Up to six additional workstations may be added.

The service unit chassis contains two server nodes, each with a relatively large disk drive. Each server node connects to other system elements:

- Power and cooling subsystem
- Mainframe maintenance interfaces
- I/O unit maintenance interfaces
- Workstations

Additionally, a graphics printer is connected to one of the server nodes to provide a limited print capability and workstation screen snapshots.

Each workstation consists of a desk-like cabinet which contains:

- A color display node
- A relatively small disk drive
- A tape cartridge assembly

Two of these stations contain a modem for remote site support.

The server nodes are used for the following functions:

- Logging
- Monitoring
- System interface
- Control

The display nodes, however, are used for service unit application programs which require a user interface. Processes in the server nodes are typically invoked under program control. Processes in the display node, however, are typically invoked by a user or operator.

Each server node has a number of servers and drivers which are configurable entities themselves. They include:

- Servers and drivers for the maintenance interfaces
- Power and cooling subsystem
- Diagnostics

- B.E.S.T.
- Logic analyzer
- System control functions such as
  - System monitor
  - System logging and analysis
  - System configuration control

Additionally, one of the server nodes also supports server processes for system configuration management and interprocess communication.

Multi-Host Network

The Multi-Host Network physical configuration is maintained in the system configuration table. This includes entries for:

- Hosts (local and remote)
- Local and remote Network Access Devices (NADs) and their respective trunk control units (TCUs)
- Trunks between remote hosts and the local host.

The ETA10 Remote Host Facility software uses this physical configuration to selectively configure its own logical configuration.

System Clock

As a service function, system configuration control provides other system features with the current value of the system clock. There are two formats for the system clock:

- The time and date represented in binary coded decimal (BCD)
- An integer value which represents the amount of elapsed time since January 1, 1970, 12:00 a.m. Greenwich Mean Time.

System configuration control provides the functionality necessary to set and subsequently synchronize the system clock with the real time and date.

Configurable Unit Attributes

Many of the configurable units in the computer system have attributes which can change from one configuration to another. Some of these attributes are managed by system configuration control. The attributes include:

- Chip sizes for memory units
- Unit address (e.g. service unit node address, NAD address)
- Part number
- Serial number
- Revision level
- Software name
- Version

As more peripheral hardware becomes available for the computer system, each new type of configurable unit may have hardware attributes that can change from one configuration to another.

Provision is also made for other features to manage *extensible attribute blocks* which are global to the system or are specific for configurable units (either hardware or software) that the feature is responsible for managing.

Extensible attribute blocks may be specified as shared memory resident. These blocks are shared among CPUs and are not maintained in the service unit copy of the system configuration table. Extensible attribute updates are performed through system configuration control as a fixed block of information: no attention is paid to the content.

System Configuration Table

All hardware and software configurable units are defined in a system configuration table. This table resides in each CPU and in shared memory as a shared memory object. Access to shared portions of the system configuration table is controlled by the communication buffer.

The system configuration table resides on service unit nodes as a mapped file. Each service unit node that has a system configuration control server has a copy of the system configuration table.

All updates and inquiries to the system configuration table are coordinated through system configuration control. When status and attribute information is changed, system configuration control makes the appropriate changes to *all* copies of the table. Note that information which is memory resident is changed only in the CPU shared copy but is not changed in the service unit disk copy.

System Configuration Table Structure

The system configuration table contains both general information about the total system and information about each configurable unit. Each of the following is checksummed:

- System level entries
- Unit entries
- Extensible blocks

Hardware units are identified by unique hardware names. The name is derived from a hardware type name which has zero or more digits appended identifying its position with respect to identical devices in the system configuration table.

Note that for the purpose of providing hardware interrelationships, there are some additional hardware units in the system configuration table which are not configurable, like power and cooling subsystem.

Each unit entry may be considered as a node in a structure similar to the hardware itself. The location of this node within the structure is derived from the hierarchical relationship the node has with predecessor nodes. The information contained in the table entry is dependent on the configurable unit type.

Software configurable units are identified by software and version names. For backup redundancy on different storage devices, each software version may point to multiple file copies. Each software name has a default version which is used by other system features including the linker and system initialization.

Because a software unit may not necessarily be contained in a single file, one entry in the system configuration table may not be sufficient to describe the software unit. Multiple entries, therefore, may be grouped together to form software families.

System Configuration Table System Information

The general system information contained in the system configuration table includes the following:

- System configuration table attributes and statuses
- Table sizes and limits
- Hardware and software unit type definitions
- Extensible attribute type definitions

Some examples of ETA-defined system level extensible attribute blocks include:

- Site information:
  - Site identification
- Operating system features:
  - System table locations
  - Configurable parameters
- Logical file system:
  - Primary and backup master directory locations
- Cold start initialization:
  - Physical memory layout

System Configuration Table Hardware Unit Information

Hardware configurable unit information, depending on the specific device type, may include the following:

- Configurable unit type
- Unit number
- Hardware unit name
- Software server mailbox name
- Part and serial number, revision level
- Owner
- Sponsor
- Physical status
- Logical state
- Unit address (e.g. node ID, NAD number)
- Relevant software unit name(s) (for processors)
- Maximum amount of allocatable space (for memory devices)
- Links to predecessor nodes
- Links to sibling nodes
- Links to successor nodes

- Information blocks for extensible attributes

System Configuration Table Software Unit Information

The software configurable unit information for the system configuration table may include:

- Configurable unit type
- Software unit name
- Version name
- Backup copy number
- Software status indicators (e.g. ETA controlled, service unit file, family member, family parent, inaccessible, active)
- Global file ID or service unit file name
- Software unit capability with imbedded software name and version
- Software function list
- Date and time unit published
- Date and time unit defaulted
- Link to predecessor node in a multiple file unit
- Link to successor node in a multiple file unit
- Information blocks for extensible attributes

Some examples of ETA defined software extensible attribute blocks are:

- Domains:
  - Domain information
  - Domain entry points
  - Shared libraries used by each domain
  - Shared sections used by each domain
- Process package:
  - Process package information
  - Domain list

System Table Operations

System table operations provide procedures to add, list, retrieve and remove:

- Attribute types
- Hardware unit types
- Software unit types

There are also procedures to modify and retrieve extensible attribute information and to retrieve information about the system configuration table itself. The system table operations are:

- **Add attribute type** defines a new extensible attribute for the system or for the existing configurable unit type. This attribute may be defined for a site feature or for another operating system feature. An option is provided to indicate whether this attribute is to be maintained only in memory on the CPU, or on the service unit disk as well.

- **Add hardware unit type** defines a new hardware configurable unit type. The new unit type may be defined as a site extension or as another operating system feature. The relationship this unit type has with other unit types in the configuration is specified by providing the names of the parent unit types.

  Some characteristics of the unit type are specified when it is added, for example:

  - I/O processor channel driver
  - Dual access
  - Sponsorable
  - Disk device

  Add attribute type procedure defines extensible attribute blocks for the unit type. A limit is imposed on the number of unit types that may exist in the system configuration table.

- **Add software unit type** defines a new software configurable unit type. The new unit my be defined as a site extension or as another operating system feature.

  Add attribute type defines extensible attribute blocks for the configurable unit type. A limit is imposed on the number of unit types that may exist in the system configuration table. Site defined unit types have no relationship with ETA defined unit types.

- **Change attribute block** replaces the contents of the attribute block(s) associated with a configurable unit. Its use is intended for operating system features or site extensible features which have defined their own attributes.
- **Get attribute block** retrieves the contents of attribute block(s) associated with a configurable unit. Its use is intended for operating system features or site extensible features which have defined their own attributes.
- **Get attribute type information** retrieves all information associated with an attribute type definition.
- **Get hardware type information** retrieves all information associated with a hardware unit type definition.
- **Get system information** retrieves information associated with the system configuration table itself.
- **List attribute types** lists all attribute types associated with a particular unit type, or with all unit types.
- **List hardware unit types** lists all hardware unit types.
- **List software unit types** lists all software unit types.
- **Remove attribute type** removes an attribute type from the system configuration table.
- **Remove unit type** removes either a hardware or software unit type from the system configuration table.
- **Set system information** changes information associated with the system or the system configuration table itself.
- **Set system version** changes the version name associated with the system. This is the version used when system tables are registered in the system configuration table. It is also the default version of the system family unit.

Hardware Unit Operations

Hardware unit operations provide procedures to add, list, and remove hardware units in the system configuration table, and to retrieve and change hardware unit information and status. The hardware unit operations are:

- **Add hardware unit** adds a new configurable hardware unit to the system configuration table. The relationship this unit has with other units in the configuration is specified by providing the names of the parent units.

- **Change hardware unit information** changes the predefined attributes associated with a hardware configurable unit. These *modifiable* attributes include:

  - Unit address
  - Part number
  - Revision level
  - Serial number
  - Relevant software name and version
  - Maximum amount of allocatable space

  It may also contain a small number of fixed length attribute fields which are used by operating system features or site extensible code.

- **Change hardware unit status** changes one of the statuses associates with a hardware configurable unit. These statuses include:

  - Unit sponsor
  - Unit owner
  - Active access
  - Physical statuses
  - Logical state

- **Get hardware unit information** retrieves information associated with a hardware configurable unit. This information includes hardware type characteristics such as:

  - Processor
  - Sponsorable
  - Removable

  This information also includes hardware unit characteristics such as:

  - Dual access
  - High density

  The information also includes modifiable attributes such as:

  - Unit address
  - Part number

  The information also includes the name of the parental units.

- **List hardware unit information** retrieves attribute and status information associated with a list of hardware configurable units. It is essentially the combination of three operations:
  - **List unit names**
  - **Get hardware unit status**
  - **Get hardware unit information**
- **Get hardware unit status** retrieves the statuses associated with a hardware configurable unit. These statuses include:
  - Unit sponsor
  - Unit owner
  - Active access
  - Physical statuses
  - Logical status
- **List unit names** lists the names of hardware units. Two options are provided:
  - All units which are immediate children of a parent
  - All units of a specified type which are descendants (at any level) of a parent

  The parent may be the ETA computer system itself.
- **Move hardware unit** moves a hardware unit from one set of parents to another. This operation supports the physical reconfiguration of hardware without having to remove a unit and add it back later.
- **Remove hardware unit** removes a configurable hardware unit from the system configuration table. Only certain units are removable, such as peripheral hardware and software servers.

Software Unit Operations

Software unit operations provide procedures to add, list and remove software units, and to retrieve and change software unit information and status. The software unit operations are:

- **Add software file copy** adds a software file copy to an existing software unit version.
- **Add software unit version** adds a new configurable software unit or a new version of an existing unit to the system configuration table.
- **Add software unit** adds to the system configuration table either:

  - A new configurable software unit
  - A new version of an existing unit
  - A new copy of an existing version

  If the unit is new, it may be optionally linked with other software units to form a multifile unit.

- **Change attribute block using capability** replaces the contents of attribute block(s) associated with a software configurable unit. It is intended for use by operating system features that reference software units by their capability rather than their software name and version.

- **Change software status** changes one of the statuses for a software unit. Software statuses include:

  - Default version designation
  - Unit activation
  - Copy accessibility

- **Get attribute block using capability** retrieves the contents of attribute block(s) associated with a software configurable unit. It is intended for use by operating system features which reference software units by their capabilities rather than their software name and version.

  This procedure "displays" the software name and version from the capability and then calls the get attribute block procedure.

- **Get software unit information** retrieves information associated with a particular software configurable unit.

- **List family units** lists all the software units of a specific family, or lists all families a specific software unit belongs to.

- **List software copies** lists all software unit copies of a software unit name and version.

- **List software units** lists all software unit names of a specific type, or of all types.

- **List software versions** lists all the software versions of a particular software unit name.

- **Remove software file copy** removes a software file copy from an existing software unit version.

- **Remove software version** removes a configurable software unit or a software version from the system configuration table.

It should be noted that certain software units such as service unit servers and drivers, CPU kernel and nucleus, I/O unit nucleus and channel drivers, must exist in order for the system to function. They cannot be entirely removed.

- **Remove software unit** removes from the system configuration table either:

  - A configurable software unit
  - A software version
  - A software copy

  Certain software features, such as the service unit servers and drivers, and the operating system and its features, must exist in order for the system to function. These features, therefore, cannot be entirely removed.

Logical Device Operations

Logical device operations provide procedures to manage logical devices and their device classes. This includes procedures to:

- Add, list, and remove logical disk devices
- Add, list and remove device classes
- Manage the assignment of disk devices to device classes
- Coordinate logical disk device allocation

The logical device operations are:

- **Add device class** adds a new device class to the system configuration table. It may also be used to alter the device allocation selection method.
- **Add device to device class** assigns a logical disk device to a device class.
- **Add logical device** adds a new logical disk device to the system configuration table.
- **Get device class** retrieves all information associated with a device class. This includes:

  - Device class allocation selection method
  - Device class name
  - Device class handle
  - Amount of space available for allocation
  - Amount of space allocated
  - Last logical device selected for allocation

  - Least full logical device
  - Number of logical disk devices

- **Get logical device** retrieves all information associated with a logical disk device. This includes:

  - Logical disk device name
  - Logical disk device handle
  - Amount of space available for allocation
  - Amount of space allocated
  - Number of device classes the device belongs to
  - The number of disk devices in the logical device
  - List of the disk devices in the logical device

- **Get logical device mailbox name** returns the remote procedure call mailbox name associated with a logical disk device.

- **List device classes** lists all device classes to which a specific logical disk device belongs, or it lists all device classes currently defined in the system.

- **List disk physical-file system mailbox names** lists the remote procedure call mailbox names for all disk physical-file system's which are on-line.

- **List logical devices** lists all the logical devices belonging to a specific device class, or it lists all the logical disk devices contained by the system.

- **Locate logical device** is used by the disk physical-file system in the CPU to locate a logical disk device upon which the first segment of a file is to be allocated. If the device is located, it is locked such that no further allocations or deallocations may be made to that device until the update device allocation procedure has been performed for the device. The device allocation selection method is defined by the device class.

- **Remove device class** removes a device class from the system configuration table.

- **Remove device from device class** removes a logical disk device from a device class.

- **Remove logical device** removes a logical disk device from the system configuration table.

- **Select logical device** is used by the disk physical-file system in the CPU to select a logical disk device for allocation or deallocation. If the device exists, it is locked such that no further allocations or deallocations may be made to that device until the update device allocation procedure has been performed for the device. If the device is already locked, the procedure waits until it is unblocked.

- **Set allocation amount** sets the amount of allocatable space for a particular logical device. It is called after a logical device has been configured for use.
- **Update device allocation** is used by the disk physical-file system when space on a device is allocated or deallocated, and by system configuration management when a device is activated, to update the allocation information associated with a logical disk device. Following the update, the device is also unlocked so that further allocations or deallocations may be performed on the device.

  It should be noted that the device may have also been locked as a result of the locate logical disk or select logical disk operations.

System Time Operations

System time operations are procedures that set and retrieve the system clock. The system time operations are:

- **Convert real-time clock to time stamp** converts the free running system clock (RTC) to a time stamp integer clock value.
- **Decode time stamp** decodes a time stamp integer clock value to a binary coded decimal clock value.
- **Get system clock** retrieves the current value of the system clock. Two values are returned: the binary coded decimal (BCD) clock value and the integer clock value.
- **Get time stamp** retrieves the integer value of the system clock.
- **Set system clock** permits the system initialization feature to establish a relationship between current time and the real-time system clock.

Error Handling and System Configuration Table Access Operations

Error handling operations provide table validation and recovery procedures. System configuration table access operations provide procedures to initialize and manipulate the various copies of the system configuration table. The error handling and system configuration table access operation is:

- **Capture system configuration table** is provided on the service unit to capture the contents of the system configuration table and write them to a specified file. It is intended for use by System Initialization when it coldstarts the system. It may also be used for state capture.

- **Check system configuration table** permits system monitor and system configuration control itself to perform table checks on the system configuration table.

  Only one copy of the system configuration table is checked at a time. If this operation is called on the CPU, only the CPU copy is checked, if it is called on the service unit, only the service unit copy is checked. If an abnormality is found during this operation, a message is sent to system monitor indicating the error.

- **Initialize CPU** permits the system initialization feature to initialize system configuration control during a CPU warm start.

- **Initialize allocation tables in the communication buffer** permits the system initialization feature to initialize the logical device allocation tables used by system configuration control in communication buffer memory.

- **Initialize events** is provided by system configuration control client stubs on the service unit to open a mailbox to the system configuration control server and to initialize the event counter for it.

- **Read events** is provided by system configuration control client stubs on the service unit to get a message from the system configuration control server containing system configuration control events that occurred.

- **Replace system configuration table** is provided on the service unit to replace the contents of the system configuration table from a specified file. The new system configuration table is validated using the check system configuration table operation prior to replacement. This operation is intended for use by the system initialization feature after it cold starts the system, or when the system configuration table needs to be reinstalled.

- **Set events** is provided by system configuration control client stubs on the service unit to establish the event(s) that the application wishes the system configuration control server to monitor.

- **Set model number** is provided by system configuration control on the service unit to set the model number for the ETA10 computer system. It is only significant when creating an initial system configuration table file.

- **Set system configuration table file pathname** is provided by system configuration control on the service unit to set the file pathname of the system configuration table to a local file. This call is intended for service unit applications which need access to their own local copy of the system configuration table.

System Table Registry Operations

System table registry operations provide procedures to record and retrieve information about system tables. This type of information is used by the system dump analyzer, service unit debugger, and system termination. The system table registry operations are:

- **Add system table** identifies and records the location of a system table in the system configuration table.
- **Add table field** successively defines the fields of a particular system table.
- **Get system table information** retrieves information for a specific version of a system table.
- **Get table field information** successively retrieves information for each field of a system table.
- **List system tables** successively lists the system table names which are registered in the system configuration table.
- **Remove system table** removes one or more versions of a system table. If version is not specified, then all versions of the system table are to be removed.

System Monitor

The system monitor receives fault notification from throughout the computer system. It provides periodic verification of hardware and software operation. All reconfiguration decisions made by system monitor are based on *single* fault notification. Correlations between multiple fault notifications are made by the system logging and analysis feature.

System monitor watches available hardware status as well as process fault indications reported by other software. The interfaces to system monitor are illustrated in figure 4–37.

System monitor has pieces executing in each processing element of the computer system. System monitor consists of three separate processes and their interfaces:

- System monitor service unit server
- System monitor I/O unit server
- System monitor service unit probe

The interface for reporting errors to system monitor is the same in all processing elements, but system monitor may be required to handle errors differently depending on the processing element.

System monitor provides the focal point for reporting system errors. Fault notifications are received from:

- Local processes
- Interprocess communication

All fault notifications are validated and passed to the system logging and analysis feature to be logged.

System monitor is responsible for running hardware and software verifications. It periodically polls for hardware error information, including power and cooling subsystem status.

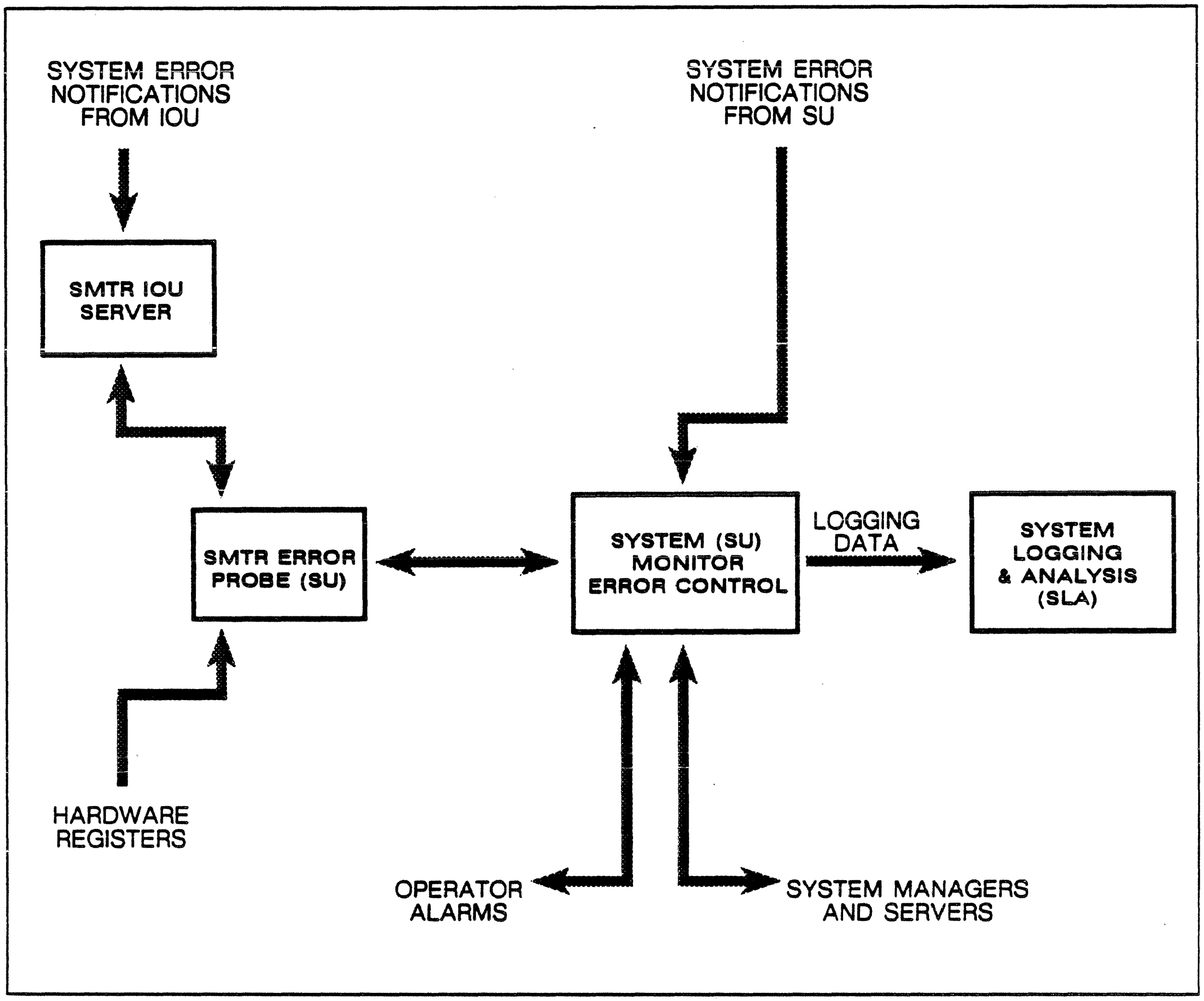

SYSTEM ERROR NOTIFICATIONS FROM IOU
SYSTEM ERROR NOTIFICATIONS FROM SU
SMTR IOU SERVER
SMTR ERROR PROBE (SU)
SYSTEM (SU) MONITOR ERROR CONTROL
LOGGING DATA
SYSTEM LOGGING & ANALYSIS (SLA)
HARDWARE REGISTERS
OPERATOR ALARMS
SYSTEM MANAGERS AND SERVERS

Figure 4-37. Components and interfaces of the system monitor.

Hardware Fault Detection

Hardware faults are detected by system software either by:

- Examining status after an I/O operation or memory transfer
- Acknowledging and responding to hardware initiated exception processing.

These fault indications are provided to system monitor by the software that detects them.

The computer system hardware provides *go* or *no go* fault indication with each data transfer operation. This indication is available to the software controlling the transfer. If there is a hardware detected fault involved, detailed information is available to the service unit.

Some hardware faults are detected internally by system monitor through polls of the high-tech board hardware registers or power and cooling subsystem status.

Failures from which the hardware automatically recovers, (e.g. single bit memory errors), result in detailed information being recorded and saved in the registers until the service unit reads them. System monitor polls the registers and logs any error information found.

Hardware Fault Processing

The processing a feature implements in response to hardware errors that might occur is *diagnostic* in nature. The code must determine what the failing element is, and whether or not the error is recoverable. The calling feature may set the severity level of the error report to:

1. Event
2. Error
3. Failure

The feature may also set a flag requesting a response from system monitor.

System monitor is notified of hardware faults in two ways:

1. An unexpected event occurs and data concerning it is to be logged. The calling feature can and will handle any recovery or alternate activities. The feature sends a report to system monitor with an Event severity level.
2. An error occurs that indicates a device has failed and is no longer usable. The feature sends a message to system monitor with a Failure severity level. System monitor logs the event.

When an error condition is identified, information is recorded about the error condition. The data is passed to system monitor for logging and/or action.

System Monitor Functional Interfaces

Hardware and software configurable units are identified to system monitor with a unique identifier. This identifier is assigned and managed by system configuration control.

System monitor communicates with these configurable units through its functional interface. This functional interface includes an error data pointer as an input parameter. It points to a variable length data area containing information specific to the error condition being reported.

When either an entry condition is not met, or an invalid calling parameter is detected by system monitor, a system parameter is used to determine whether an appropriate exception status is to be returned, or if a software failure is to be assumed. All calls to system monitor result in a call to the system logging and analysis feature to log the event, *even if the calling parameters are invalid.*

Report System Error

This interface is used to report all observed error conditions. A data error pointer that may not be interpreted by system monitor is included in the calling parameters so that the caller may pass pertinent information for logging and/or interpretation by the system logging and analysis feature.

A *wait* or *no wait* flag is also included in the parameter list so that critical features such as SM manager are not blocked while waiting for a response form system monitor.

Any process may use this interface to report a software error detected anywhere in the system. A severity parameter should be set by the calling feature to either:

- Event
- Error
- Failure

Regardless of the severity, system monitor calls the system logging and analysis feature to record the report.

The severity level should be set to Event to log an informative report of a condition from which the calling feature can recover. This interface may also be used with an Event severity to report changes in system performance such as CP memory thrashing or other software events that need to be logged.

The severity level should be set to Error when a software error which might be recoverable is detected. System monitor will collect any error specific data and pass the report to the system logging and analysis feature for logging. The calling feature should follow up with an Event report if recovery succeeded, or a Failure report if it did not.

The severity level should be set to Failure when an unexpected or inconsistent status is detected in a system configurable software unit. The caller should pass the error upward.

Any process may use this interface to report a hardware error detected anywhere in the system. A severity parameter should be set by the calling feature to either:

- Event
- Error
- Failure

Regardless of the report severity, system monitor calls the system logging and analysis feature to record the event.

The severity level should be set to Event to log an informative report of a condition from which the calling feature can recover. This interface may also be used with an Event severity to report changes in system configuration such as promoting a CPU or other hardware events that need to be logged.

The severity level should be set to Error when a hardware error which might be recoverable is detected. System monitor will collect any error specific data and pass the report to the system logging and analysis feature for logging. The calling feature should follow up with an Event report if recovery succeeded, or a Failure report if it did not.

The severity level should be set to Failure to advise system monitor that a hardware unit is no longer usable. The error is not recoverable and the caller should pass the error upward.

Human Interfaces

System monitor interfaces with an operator at a service unit display node to either:

- Toggle probing with probe manager, a utility which runs in the service unit shell

- Advise of a hardware or software failure

These operations are performed through the operator alarm interface to the alarm server. The alarm server is provided by operations management.

Fault Detection

A *fault* is an observation made during normal system operation that either hardware or software is not performing as intended. These situations do not include the erroneous operation of user code.

A *failure* is an unexpected or unintended condition in either hardware or software that results in a fault. The fault is the symptom, the failure is the problem that must be treated and cured.

Shared Memory Transfer Errors Indicated by Transfer Request Block Status

The following maintenance interface transfer errors are detected by the transfer request block status and returned to system monitor.

Shared Memory Transfer Hardware Error

This class of faults includes:

- Double SECDED errors in either CP memory or shared memory
- Data or address parity errors between the CPU and SM Interface

This fault indicates either:

1. CP memory hardware has failed
2. SM interface hardware has failed
3. Shared memory hardware has failed
4. Reference to a shared memory location containing a double SECDED fault written by the hardware in response to a prior data parity error.

A SECDED error could occur outside the area of shared memory that was read because the SECDED check is enabled on 128-bit boundaries.

A data parity error detected by the SM interface during a store to shared memory, writes a double bit SECDED error so that the next read of that area will get a double SECDED fault.

Shared Memory Transfer Bounds Error

This fault occurs when the SM interface detects an attempt to transfer data across the boundary between shared memory halves or to a missing shared memory rank. This fault indicates either:

1. A hardware failure occurred in reading the transfer request block
2. A software failure occurred in constructing the transfer request block

3. A CPU hardware failure

4. Corrupt system tables due to either a hardware or software failure.

This fault may also indicate a "runaway" transfer due to a failure in either the CPU or SM interface. The SM interface works with the current memory address. When the CPU concludes a transfer it sends a terminate signal. If the terminate signal is missed, or is not sent, shared memory is destroyed out to the end of contiguous shared memory, which is usually the end of the shared memory half.

Note that shared memory may not be contiguous within a shared memory half. This can occur if a rank is deconfigured. The service unit has the ability to do this by setting controls in the SM interface.

Faults Detected by System Monitor

The following faults are detected by system monitor through polls of the maintenance registers.

CPU Instruction Stack Parity Error

This fault indicates CPU hardware has failed. The CPU stops. The fault is detectable only by a service unit poll of the maintenance registers.

CPU Multiple Match Error

This fault occurs when the hardware detects duplicate associative register data. The CPU stops. This condition is detectable only by a service unit poll of the maintenance registers. This fault indicates that:

1. Hardware incorrectly read the virtual translation data (a CPU failure)

2. A software failure has occurred in system domain code

3. System tables are corrupt due to either hardware or software failure

CPU Illegal Monitor Mode Instruction

This fault occurs when hardware detects an attempt to execute an illegal instruction in monitor mode. The CPU stops. This condition is detectable only by a service unit poll of the maintenance registers. This fault indicates:

1. The hardware incorrectly read or decoded an instruction (a CPU failure)
2. A software failure has occurred in monitor code
3. Monitor code is corrupt (a hardware or software failure has occurred elsewhere)

CP Memory Double SECDED Error

This fault, including both double and odd multiple faults, CPU hardware has failed. The CPU stops. This condition is detectable only by a service unit poll of the maintenance registers.

This fault indicates either:

1. I/O interface hardware failure
2. I/O unit hardware failure
3. Light pipe failure

I/O Interface Communications Error

Communications between an I/O unit and its I/O interface are protected by cyclic redundancy checks. Hardware automatically retries message transmissions up to three times before indicating a transmission failure to the I/O unit.

When message transmission from the I/O interface to the I/O unit fails, a failure status is placed in the maintenance registers. The service unit must poll for this indication.

This fault indicates either:

1. I/O interface hardware failure
2. I/O unit hardware failure
3. Light pipe failure

Power and Cooling Subsystem Faults

The power and cooling subsystem hardware is charged with protecting the computer system hardware from catastrophic hardware conditions. The service unit receives notification that an environmental parameter has moved outside an early warning limit. System monitor can idle the affected parts of the system before the power and cooling subsystem shuts them down.

Single Bit SECDED Error

These faults can be detected only by a service unit poll of the maintenance registers associated with CP memory, shared memory or CB memory. A correctable memory failure is indicated.

Communication Buffer or Shared Memory Faults

Faults detected by the service unit when accessing the communication buffer or shared memory through the service unit interface to the I/O interface are essentially equivalent to those detectable by an I/O unit.

CP Memory Faults

Faults that occur while the service unit is reading CP memory are reported via the CPU maintenance registers. If the CPU is also accessing CP memory, any fault indication in the maintenance registers could be a result of either CPU or service unit access.

System Logging and Analysis

System logging and analysis (SLA) is a service unit resident operating system feature responsible for logging events. The system logging and analysis feature records the minimum number of events necessary to maximize support.

The classes of events captured by the system logging and analysis feature are:

- **Error events** including all detected hardware and software errors
- **Environmental events** including related environmental conditions monitored by the power and cooling supervisor.

System logging and analysis provides a display interface to permit monitoring of events as they occur. The display monitor also permits scrolling and searching through events that have been logged.

System Logging and Analysis Architecture

The architecture of the system logging and analysis feature is illustrated in figure 4–38. Each feature is discussed.

Event Report Interface

An externalized interface is used by system monitor to report system logging and analysis events. Through this interface event reports are posted to the event report first-in-first-out (FIFO) queue.

Event Report First-In-First-Out Queue

The event report first-in-first-out queue is processed sequentially by the logging processor.

Logging Processor

The logging processor is a process charged with entering event reports into the system logging and analysis database. The logging processor may also relay event reports to the query processor, if it is active.

The logging processor limits the size of the database by aging the entire database when the configured limit on the number of records in the database has been reached. The aging of the database and starting of a new database file is coordinated with the query processor based on current query activity.

System Logging and Analysis Database

This file is the actual system logging and analysis database file. The file contains fixed length binary records formatted like the raw event reports. The length of a record is determined by the maximum size raw event report. This record type permits scrolling and searching, and requires a minimum of disk space.

Aged Database

The aged database is an aged image of the system logging and analysis database that resides in file */su_os/trace/aged_sla_database*.

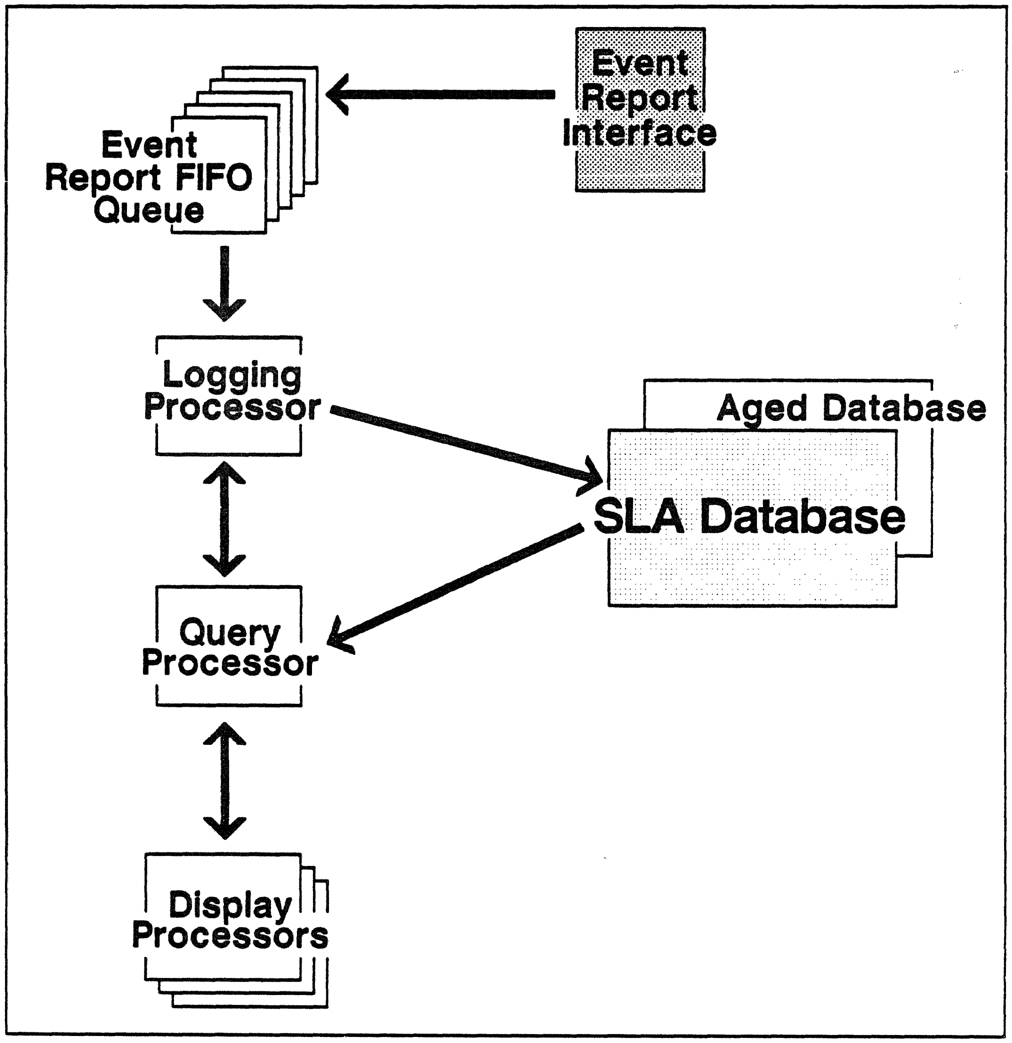


Figure 4-38. SLA Architecture.

Query Processor

The query processor is the process charged with processing display processor queries and for returning query results. Event reports are displayed as they occur. The ability is also provided to scroll and search the database.

The query processor coordinates the aging of the database with the logging processor. Aging does not occur while the display processor is making queries to the database.

The query processor enables the display processor to view archived database files as well as the current database file.

Display Processor

The display processor permits you to select via a menu item one of two modes.

The first mode is that of monitoring event reports as they happen.

The second mode is the database query mode that permits you to scroll through the event database via menu items. The second mode also permits the user to select events for scrolling. The event field to search is controlled by selecting a menu item and then responding to a prompt for the content that should be looked for in the file.

You may select an alternate database file via menu item and prompt.

A toggled menu item selects summary or detailed display format. Summary format displays:

- Date and time of the event
- Type
- Process
- Processor
- Failing unit
- Exception information

Detailed format displays summary information plus a status message and event specific data in ASCII and hexadecimal.

Report Writer

Report writer is a display application which permits you to select search criteria (including trailing wildcards) for the generation of a text file report that may be viewed from the service unit environment. The report may also be printed.

Input to the report writer is the ID of the node containing the system logging and analysis database. The database file name may also be used. If the report writer is unable to communicate with the system logging and analysis server on the selected node, it will attempt to access the default database directly.

Several report types are available:

- Raw summary, a line by line report similar to the display summary mode
- Raw summary with ASCII, containing an ASCII interpretation of the event specific data
- Raw summary with hexadecimal, containing a hexadecimal interpretation of the event specific data
- Time histogram, a histogram generated in half hour intervals
- Day histogram, a histogram generated in daily intervals

The possible date and time increments are:

- Today
- Since yesterday
- Since two days ago
- A specific range of dates or times
- All

The selection of process, processor, and failing system unit is case insensitive and may contain the trailing wildcard character %. To search for all events having a failing central processing unit, you would enter the wildcard character in *CP%*.

The report writer output file is a text file with FORTRAN page control. The output width is 130 columns.

System Logging and Analysis Design Philosophy

The design philosophy of the system logging and analysis feature is to provide logging and display capability. Event reports are used in raw form by each part of the system logging and analysis feature due to the different modes of access to the data in each component:

- The logging processor views the data as raw data which needs to be recorded and distributed
- The query processor views the data from a scroll or search standpoint.
- The display processor views the data from a cosmetic or formatting standpoint.

This philosophy permits each component to manipulate the data without adversely affecting the other components.

Event Report

An event report contains information about an event that an operating system feature wants to log in the system logging and analysis database. An event may be:

- An error
- An informative message
- A system action
- A configuration or status change
- An operator action
- Any system occurrence that impacts system operation

An event report consists of:

- **Date/time stamp**, the date and time of the event's occurrence
- **Type**, event, error, or failure
- **Process name**
- **Processor**
- **Failing unit**, the name of the suspected failing unit
- **Exception information**, the standard exception record
- **Event specific data**, data which is specific to the event

Section 9: System Entry

This part examines the feature that validates user entry into the system. It consists of one section:

- User management

User Management

User management (UM) provides several functions. One of these is the validation of sessions and connections when they first enter the computer system.

A *user* is anyone who has the privilege to use the computer system. Users are divided into several classes, including:

- A system administrator which controls the tuning of the system and who has access to everything inside the system
- An account administrator who assigns accounts and projects

A user may also be someone who has no privileges other than to log on and run one specific program. The restrictions for accesses to system resources are determined and assigned for each user.

To support accounting functions, a user is validated to use one or more account and project pairs for their system charges.

On a typical system, the system administrator has the privilege to change things for all users of the system. The account administrator has the privilege to change things for all users in a specified account. A typical user may change things only under their own user identifiers.

Site administration provides the human interface to the system for users and their resource usage and limits. It includes the system human interface called *UREGUTIL* that is needed to specify and maintain user validation parameters and limits in the user registry.

The Common Login Processor

The common login processor (CLP) is composed of two parts:

- A process that runs continuously and senses when a user is trying to log into the system interactively. It handles the interactive prompts for user identification and password.
- A process that validates the user.

User management provides a function so that the user environments may provide an interface for a batch submission utility (*ENQUEUE* is the VSOS command for this).

The common login processor issues requests to the communications control subsystem (CCS) and receives requests from the batch submission utility. When a user enters something at a terminal, the request is satisfied and a connection is made.

The common login processor goes through the login sequence. If the login is successful, the connection and assorted user information is handed off to global scheduler for creation of the session.

Login validation occurs before selection of a user environment.

When logging in to the system, the common login processor prompts the user with:

```
Please log in :
```

Once a user identification has been entered, a prompt for password is added:

```
Please log in : username
Password      :
```

The password is echoed to the terminal when it is entered.

In response to the username prompt, users are required to enter their username. Additionally, they may enter:

- Account
- Project
- Jobname

When these items are not entered, the default entries from the user registry are used. The completed entry appears as:

```
Please log in :username,account=acct,
  project=proj,jobname=name
```

To use defaults for everything except the project, a user would answer the prompt with:

```
Please log in :username,project=new_project
```

For local batch sessions, the ENQUEUE command has similar arguments

For *interactive* local batch, the format is:

> ENQUEUE(*filename*,userid=*name*,account=*acct*,jobname=*jname*, output=*outfile*,project=*projid*)

An interactive local bactch session prompts for the password.

For *batch* local batch, the format is:

> ENQUEUE(*filename*,userid=*name*,password=*passwd*,account=*acct*, jobname=*jname*,output=*outfile*,project=*projid*)

A batch local batch session includes the password in the ENQUEUE command format.

The minimum form of the ENQUEUE command is:

> ENQUEUE(*filename*,output=*outfile*)

The common validation may be divided into two areas:

- Interactive users
- Local batch users

Interactive Validation

Interactive validation involves logging in to the system as an interactive session, and at some future time, logging out. The process for each is examined.

Logging In as an Interactive Session

The following sequence of events occurs when you log in as an interactive session:

1. You initiate a connection from your device terminal server indicating to the communications control subsystem (CCS) that it should complete a connection to the common login processor.

2. The common login processor prompts you for your username. If you need something other than defaults for your account, project, or jobname, you may enter them on the same line as your username. After the user identification line has been entered, you are prompted for your password.

3. The common login processor validates your username and password by calling a validate login procedure. If the

information supplied is valid, the common login processor calls global scheduler to create the session. If either your username, password, or other information is incorrect, an invalid login indication is returned.

Logging Out of an Interactive Environment

The following sequence of events occurs when you log out of an interactive session:

1. If it exists, your epilogue command file for the selected user environment is executed by the environment.
2. The terminal session is terminated.

Batch Validation

Batch validation involves logging in to the system as a batch session, and at some future time, logging out. The process for each is examined.

Logging In as a Local Batch Session

The following sequence of events occurs when you execute a batch session:

1. The job is submitted locally to the input queue. If it is to run under a different username, optional parameters on the ENQUEUE command must include the filename, output filename, and optionally the account, project, username, password, and jobname.
2. ENQUEUE passes its arguments to the common login processor, enabling it to proceed with checking the login.
3. The common login processor validates the username and password by calling a validate login procedure. If the information supplied is valid, the common login processor calls global scheduler to create the session. If either the username, password, or other information is incorrect, an invalid login indication is returned.

Completing a Batch Session

The following sequence of events occurs when a batch job completes:

1. If it exists, the user's epilogue command file for the selected environment is executed by the environment.

2. The batch job is terminated.

User Registry

The user registry contains the privileges, limits and other information relevant to each user of the system. The information contained for each user is composed of several parts, some for each resource manager.

Each entry that is not accessible to the user has a *privilege to modify* flag associated with it. The first privilege in each resource manager's area is the privilege to change the privilege to modify attribute. This privilege is its own protection, if a resource manager does not have privilege to modify the attribute, it may not give itself privilege.

The privilege to modify an attribute have access associated with either a user, project, or account, or with any number of users, projects, or accounts.

Account and Project File

This file contains all the accounts available on the system and the user identifiers of those validated to be the account administrators. For each account, there is an account limit and the names of the projects in the account. For each project, there is:

- A list of usernames that are validated for that account and project combination
- The limits associated with the project

System Administrator Functions

User management provides a function for the system administrator to change anyone's password in the system.

User management provides functions for the system administrator to add or remove users, and to examine and change their privileges.

Project Administrator Functions

User management provides a function for a project administrator to add or remove a user from an account or project. It also provides a function that allows users to change their passwords.

User Access Functions

User management provides a function permitting users to change their passwords.

User management provides a function permitting users to get and change their user registry attributes.

Site Administration Utilities for User Management

There are two site administration utilities for performing user management:

- **ACCTUTIL** has a command line entry field update capability to add or remove new accounts and projects to the system.
- **UREGUTIL** has a command line entry capability. Fields may be modified to update the user registry.

CHAPTER 5

Chapter 5

The Mainframe Components

Chapter Contents

List of Figures

Chapter 5

The Mainframe Components

In This Chapter...

This chapter provides a functional description of the mainframe hardware components found in a super-cooled ETA10 configuration. The central processors and system memories are described individually and as they interact and interface with each other. The categories of ETA10 machine instructions are discussed, and a brief listing of the instruction set is included.

The hardware descriptions in this chapter are intended to provide a general technical background in the ETA10 mainframe hardware. The *System Reference Manual* is intended to be used as a basic training reference for many of the courses offered by ETA Systems training organization.

- Section 1: Overview of the ETA10 Mainframe
- Section 2: Central Processor Architecture
- Section 3: The System Memories
- Section 4: The ETA10 Instruction Set

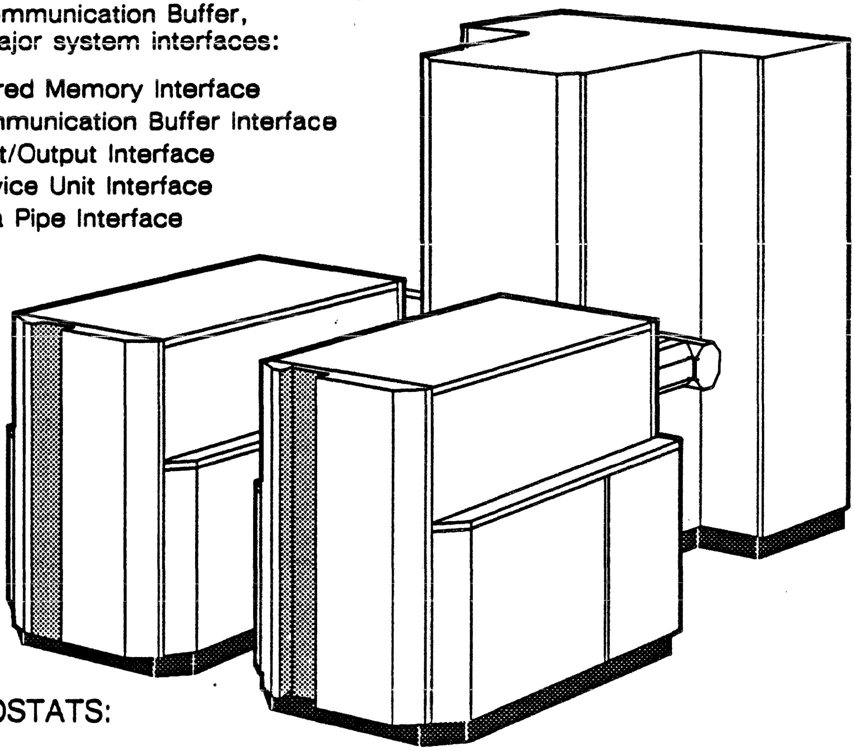
Section 1: Overview of the ETA10 Mainframe

The ETA10 mainframe pictured here shows two cryostats and the SCMI cabinet. A cryostat houses one or two central processors and central processor memories. The tall SCMI cabinet at the right houses shared memory, the communication buffer, and the major system interface and interconnection boards.

SCMI CABINET:

This cabinet houses Shared Memory the Communication Buffer, and major system interfaces:

- ○ Shared Memory Interface
- ○ Communication Buffer Interface
- ○ Input/Output Interface
- ○ Service Unit Interface
- ○ Data Pipe Interface



CRYOSTATS:

These are special containers that immerse the Central Processors in a liquid nitrogen bath.

Figure 5-1. Cabinets housing the ETA10 mainframe components.

The Cryostat

A cryostat is a sealed container that vertically suspends one or two central processor boards in a bath of liquid nitrogen. Nitrogen liquefies at near minus 320 degrees F (77 degrees K). For insulation, the stainless steel cryostat is vacuum-insulated, as are all the lines going between it and the cryogenerator that liquifies the nitrogen. The cryostat also houses the central processor memories. These memories are installed at the top of the cryostat to connect to each central processor board, and are sealed off from the liquid nitrogen bath.

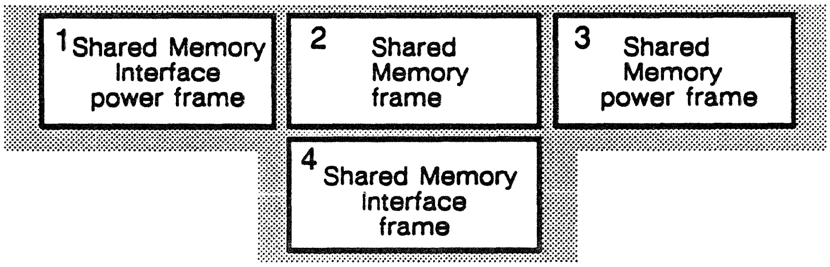
The cryostats cool the processor boards to enhance processing performance. The low temperature of the liquid nitrogen bath nearly doubles the speed of logic within the central processor, and so doubles the board's processing speed. The CMOS technology in the central processor boards is also capable of running at room temperature (at half speed).

Refer to chapter 10, "Power and Cooling", for more information on the cryostats and the ETA10 cooling system.

The SCMI Cabinet

Note that SCMI is an acronym for Shared and Communication Buffer Memories and Interface.

This T-shaped cabinet is actually four frames assembled as one unit:


1 Shared Memory Interface power frame
2 Shared Memory frame
3 Shared Memory power frame
4 Shared Memory Interface frame

The shared memory frame holds the shared memory maxi boards on which the memory stacks are mounted. The shared memory power frame holds the shared memory power supply and cooling assemblies. The shared memory interface frame holds the shared memory interface boards, the communication buffer interface and memory boards, the I/O interface boards, the service unit interface boards, and the data pipe interface boards. The shared memory interface power frame holds the power supplies and cooling assemblies for the boards in the shared memory interface frame.

The SCMI cabinet connects to the cryostats via coaxial cabling. The cabinet is large enough for all increments of communication buffer memory, shared memory and any associated power and cooling units.

The I/O Unit

The I/O units provide the means to attach peripherals and networks to the ETA10 system. Functioning as a set of independent, multi-functional channels, the high bandwidth I/O units directly access both shared memory and communication buffer. Each I/O unit can contain a variety of channels that interface to disk drives, local area and remote host networks, and other peripherals. Independent I/O processors manage the I/O transfers from I/O channels through the data pipe. A fiber optic cable, the data pipe, connects each I/O unit to its I/O port in shared memory. I/O unit cabinets house power supplies and the backplanes for the I/O unit boards, including the I/O processors, data buffers, memory, clock, data pipe interfaces, and the channel interface and protocol processor boards.

The Service Unit

Operator and maintenance communication with the ETA10 system is through display workstations on the service unit network. All system monitoring and maintenance operations are performed using the service unit. The service unit includes one or two server nodes, two graphics workstation nodes, and an amount of mass storage shared by all nodes. These components are connected by a 12 million bit/second network. Up to six additional workstations may be added to the service unit network. The service unit cabinet houses the server nodes, power supplies, and mass storage for the network.

Mainframe Components

The ETA10 mainframe components include:

• Central Processors	- independent computational engines
• Shared Memory	- system memory accessed by all processors
• Communication Buffer	- enables system processors to communicate
• I/O Interface	- connects mainframe to I/O units
• Interrupt Network	- handles processor and I/O interrupts
• Maintenance Interface	- special system access via the service unit

The ETA10 is a multiprocessor system with all processors able to communicate with each other through the communication buffer and able to access the shared memory and communication buffer. As shown in the mainframe diagram below, the independent central processing units directly connect to shared memory and the communication buffer, and interface to the service unit for maintenance.

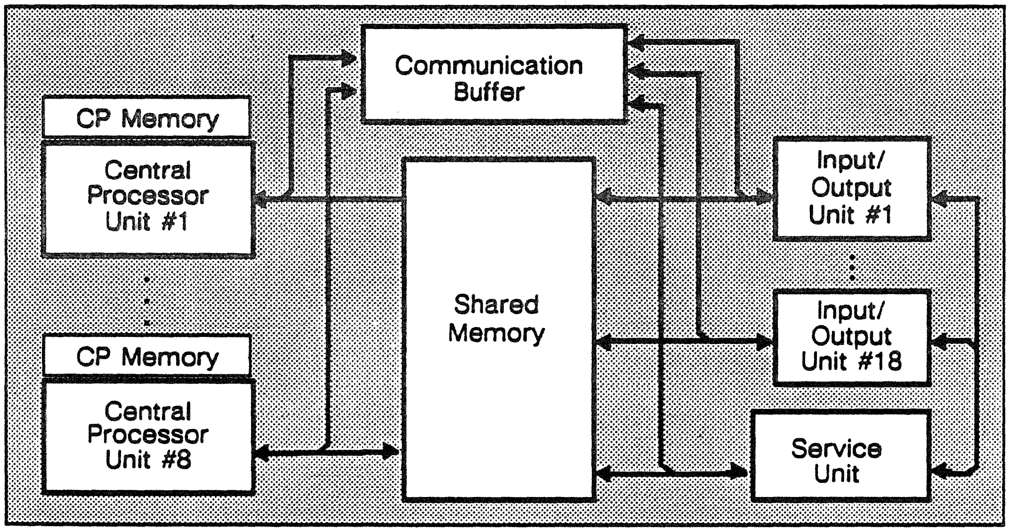


Figure 5-2. Communication paths between the ETA10 mainframe components, the I/O subsystem, and the service unit.

The I/O subsystem provides channel interfaces that connect to system peripherals and networks. I/O processors control and manage transfers to/from shared memory and the communication buffer via the fiber optic data connection, also called the data pipe. The service unit is interfaced to the mainframe components and the input/output subsystem to provide system monitoring, maintenance, and communication capabilities.

Central Processing Unit

A central processing unit consists of the central processor and its local internal memory. Each of the central processing units of the ETA10 contains:

- double-pipelined vector processor
- scalar processor
- general purpose, high speed 256 register file
- 4-million word central processor memory interfaced with an 8 64-bit word/cycle (512 bit) bandwidth

The central processor directly connects to the shared memory and the communication buffer for data transfers, and to the communication buffer for communication with other central processors.

The central processor board is a 44-layer printed circuit board populated with 240 CMOS ALSI chips. The processor board functions at room temperature but can be super-cooled to double its processing speed. The basic components of the central processing unit are shown as follows:

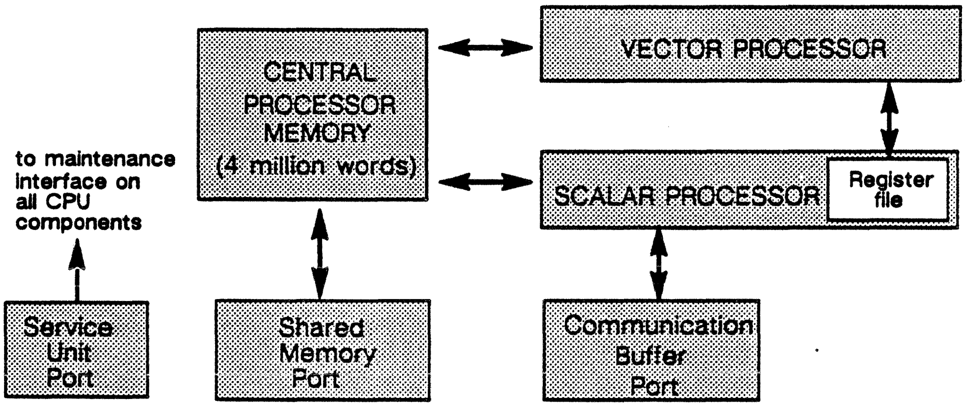

CENTRAL PROCESSOR MEMORY (4 million words)
VECTOR PROCESSOR
SCALAR PROCESSOR
Register file
to maintenance interface on all CPU components
Service Unit Port
Shared Memory Port
Communication Buffer Port

Figure 5-3. Components in the ETA10 central processing unit.

The shared memory port and communication buffer port provide the hardware interfaces from the central processor to each of those memories. The maintenance interface logic on each central processor component allows the service unit to perform diagnostic and maintenance functions.

Vector Processor

The vector processor has two arithmetic/logical pipelines that compute memory-to-memory results at the rate of two results per machine cycle in full-precision or 64-bit mode, and four results per machine cycle in half-precision or 32-bit mode. Vector performance doubles in the 32-bit, half-precision mode.

Scalar Processor

Scalar arithmetic results are produced one per machine cycle. The scalar processor contains the instruction pipe and controls the execution of instructions.

The Memory Hierarchy

The memory hierarchy includes five components: the central processor register file, the central processor memories, the shared memory, communication buffer memory, and the disk subsystem. The register file gives very fast access to data close to the processors. Each central processor has exclusive access to its own 4 million word memory. As the primary system storage for data, the large shared memory is accessible to all central processors and ranges from 32 to 256 million words. The communication buffer is used for system communications and some storage of system data. The high-capacity magnetic disk subsystem acts as a staging memory for the system.

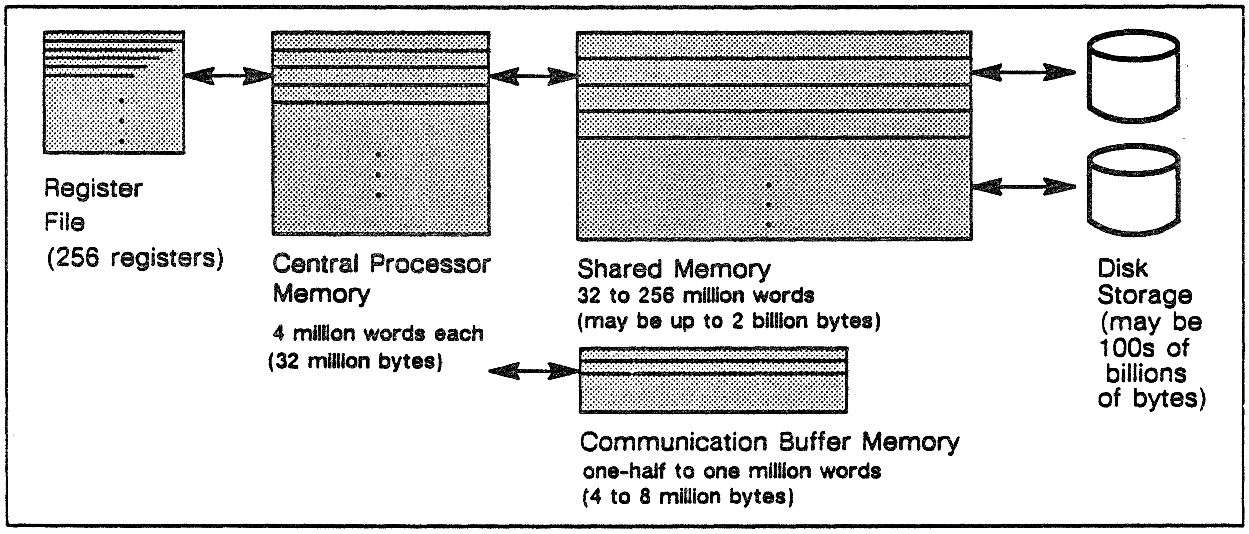

Register File (256 registers)
Central Processor Memory
4 million words each (32 million bytes)
Shared Memory
32 to 256 million words
(may be up to 2 billion bytes)
Disk Storage (may be 100s of billions of bytes)
Communication Buffer Memory
one-half to one million words
(4 to 8 million bytes)

Figure 5–4. Components of the memory hierarchy.

Shared Memory

Shared memory is the large, high speed staging memory for the system. It ranges in size from eight to 256 million 64-bit words, and supports system virtual address space of up to two trillion words. The shared memory interface provides nine I/O ports and one service unit port to the memory.

Shared memory provides major resources for a large virtual memory system and can simultaneously provide a 64-bit word per cycle to central processors and the I/O interface.

Central Processor Memory

Central processor memory is a fixed size of four million virtually addressed words. Its bandwidth to the processor is 512 bits per cycle. Single error correction/double error detection (SECDED) protection is provided on each 32-bit half word. Central processor memory is accessed by the central processor with which it is associated, a maintenance port to the service unit, and a shared memory port.

Communication Buffer

The communication buffer is a unique memory structure that enables communications between all system processors and enables central processors to share data during multiprocessing operations.

The communication buffer is accessed by all system processors for interprocessor communication. It transmits high speed synchronization messages and signals between the central processors, as well as between the mainframe and the input/output and service subsystems. Communication buffer is one-half million words that can be expanded if necessary.

The Interrupt Network

Any system processor - central processor, service unit processor, or I/O processor - can communicate with any other processor(s) by causing an interprocessor interrupt. The interrupt network controls the multiplexing and sequencing of system processor interrupts. Interrupts between the central processors and the I/O ports enable the I/O subsystem to make transfers to and from shared memory and the communication buffer.

The Maintenance Interface

The maintenance interface communicates with all mainframe components enabling the service unit to provide system-wide monitoring, maintenance, and control. Through the maintenance interface, the service unit is able to make transfers to and from central processor memory, shared memory, and the communication buffer in addition to other maintenance functions.

Overview of the ETA10 Hardware System

The diagram below shows a simplified schematic of the hardware elements in a super-cooled ETA10 system. It is included in this mainframe chapter to show the relationships of the mainframe components with the other hardware components in the system.

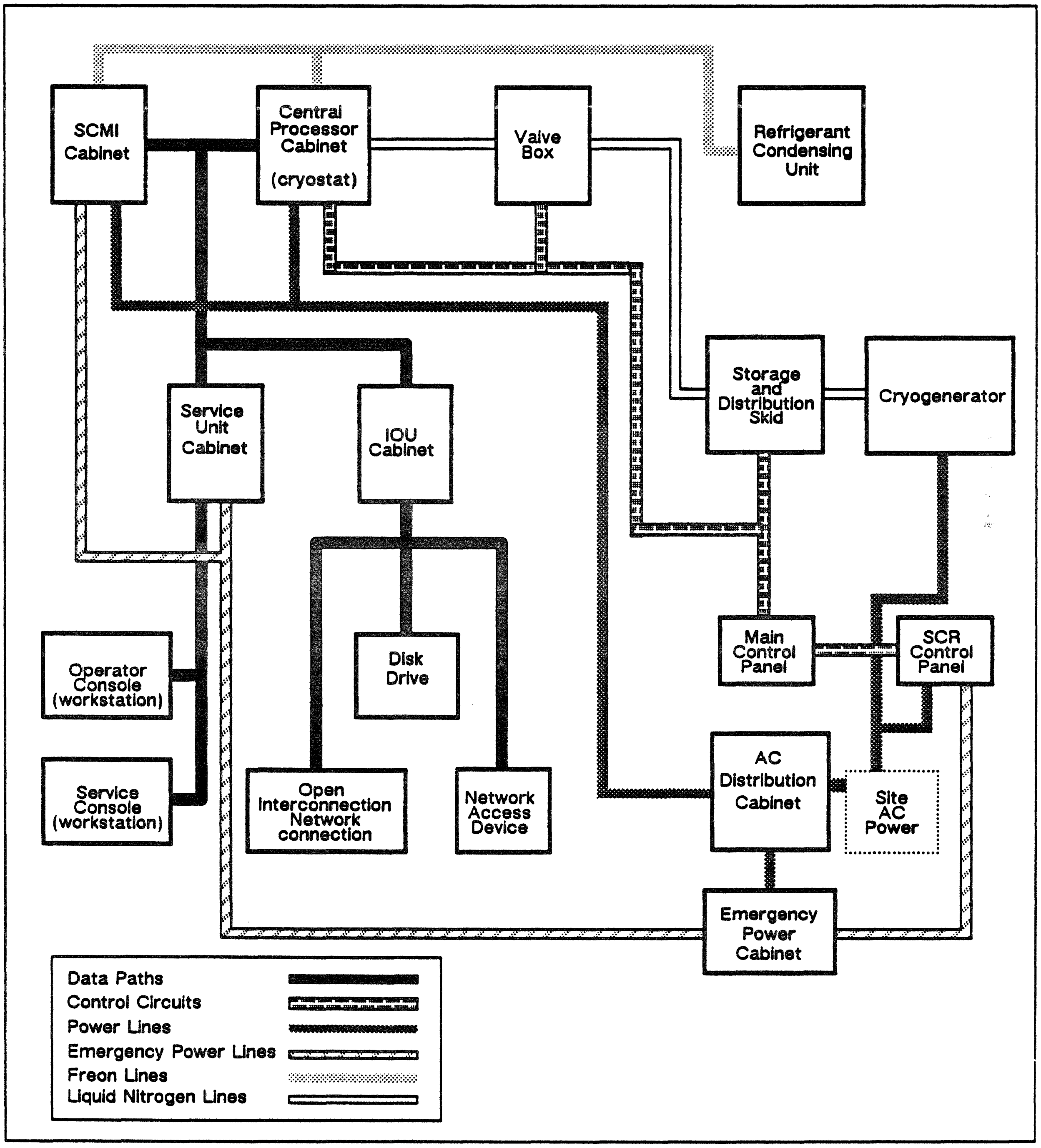


Figure 5-5. Diagram of major ETA10 hardware components and interconnections.

These components are grouped into several equipment types:

Mainframe Components

- central processing unit cabinets (cryostats)
- shared and communication buffer memories and interface cabinet (SCMI)

Service Unit

- operator console (workstation)
- service console (workstation)
- service unit cabinet

Input/Output Subsystem

- input/output unit cabinet

Peripheral Equipment

- disk drive
- network access device
- open interconnection network

Power and Cooling System

- motor starter panel
- main control panel
- silicon controlled rectifier control panel
- AC power distribution cabinet
- emergency power cabinet

Cryogenic System

- valve box
- cryogenerator
- storage and distribution skid
- refrigerant condensing unit

Section 2: Central Processor Architecture

The ETA10 central processor is contained on one 16.5" x 22.5" board manufactured by ETA. The 1/4" thick board has 44 layers that hold more than 80,000 interconnect nodes and nearly 1.25 miles of internal wiring.

Central Processor Components

The primary components of the central processor are:

- scalar processor
- double pipelined vector processors
- interface to central processor memory
- communication buffer ports
- shared memory ports
- maintenance interface

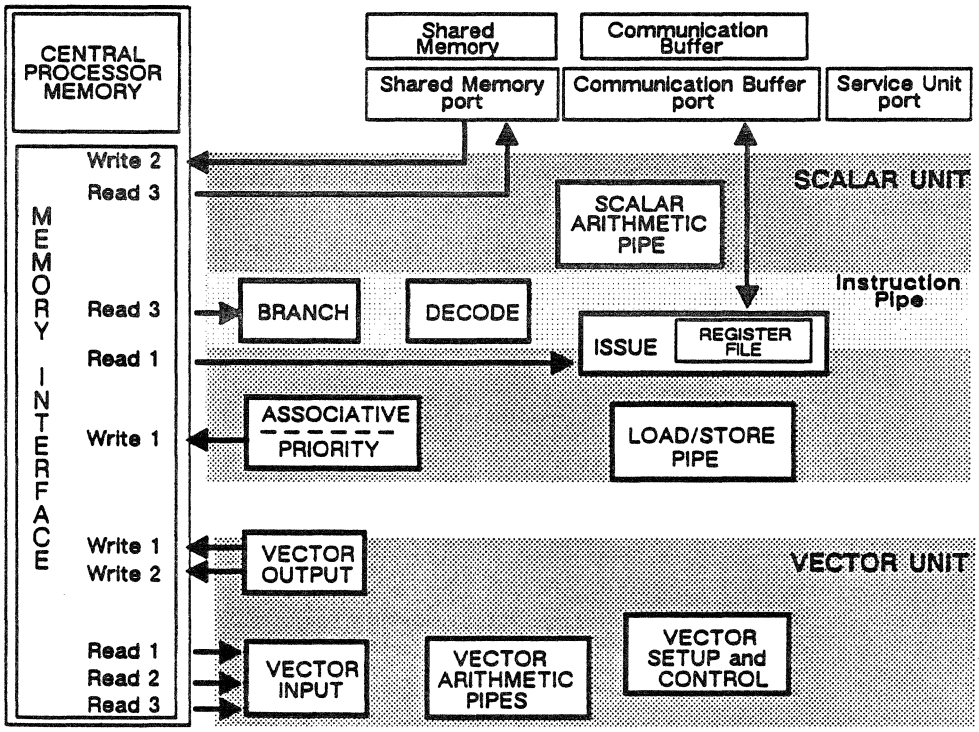


Figure 5-6. Components in the central processor.

A central processing unit is comprised of the central processor and its four-million word local memory. The central processor has independent scalar and vector processors that can operate simultaneously. Scalar and vector instructions can overlap because the processors have separate paths to the central processor memory. Communication buffer and shared memory ports provide rapid access to those memories.

Scalar Characteristics

There is a register file of 256 64-bit words which holds scalar data. The register file supports two operand-read operations and one operand-write operation every clock, providing the capability to issue a new instruction every clock cycle. The register file supports 32-bit operands in a packed manner. This means that 256 32-bit operands appear in the lower 128 64-bit locations in the register file (32-bit register 01 is not available).

The various functions within the scalar arithmetic pipeline are independent and can be overlapped with other operations. The principle exception to this is the divide/square root unit; it can process only one operation at a time. Instructions for both the scalar and vector functions are issued sequentially as a single instruction stream. When a request for an instruction is made to memory, eight 64-bit words are returned. These eight 64-bit words are called a block. An instruction cache holds eight blocks. The cache is fully associative so that sizable loops, even discontinuous loops, can easily be held in the cache, relieving memory traffic and the resulting delays.

Vector Characteristics

The vector processor has twin pipelines; once the vectors have started through the pipelines, two 64-bit results are produced every clock cycle. When 32-bit data is used, four results are produced per clock. The vector string unit allows vectors to be processed as an individual string of bits rather than as a 32-bit or 64-bit word. The vector processor is also equipped with a shortstop function that, instead of sending the result to memory, feeds it back into the pipeline. This improves the startup time for the vector unit, and enables short vectors to be rapidly processed. Operands are streamed into the pipelines, results are streamed out; the vector pipes act as assembly and production lines for vector operands.

Central Processor Memory Characteristics

The central processor memory is virtually addressed, and has a standard size of four-million words of 64K CMOS static RAM memory. The 512 bits/cycle bandwidth is high enough to sustain both the scalar and vector processors operating at maximum performance. single error correction-double error detection (SECDED) is provided on each 32-bit word.

Central processor memory and its virtual-physical addressing mechanism are discussed further in section 3, "System Memories".

Shared Memory Port

There is one shared memory port on each central processor. Through the port, shared memory transfers data at one 64-bit word per cycle.

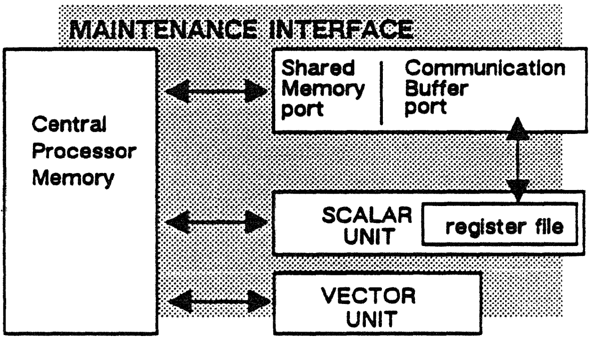
Communication Buffer Port

The communication buffer port connects the communication buffer to the register file. Small amounts of data and messages that are to be shared among central processors during multiprocessing activity are accessed from the communication buffer.

Maintenance Interface

There is also a maintenance interface that has access to each component.

Processor and memory diagnostics are run through the interface. Monitoring of processor performance is also done through the maintenance interface.



Central Processor Operations

Instruction Flow

Instructions are moved from central processor memory through the memory interface into the instruction stack of the branch unit. The branch unit tracks the corresponding central processor memory address for each instruction word from that memory. The instruction words and their corresponding instruction addresses move into the input registers of the decode unit. If the decode unit is busy with a previous operation or is performing a higher priority activity, then the instruction/address pair remains in the register until the other operations are completed.

When the decode unit is free, it performs function code translations on the new instruction and moves it into its output register. The output registers send control signals to the issue unit. When the issue unit receives a scalar instruction, it reads the data from the register file and sends it to the correct scalar arithmetic unit where the instruction is finally executed. If it is a vector instruction, the memory address of the vector operands and other instruction data is sent to the vector setup and control unit. The vector setup and control unit forwards the addresses and makes memory requests to the priority unit. The priority unit sends data about the starting address to the vector input unit, and data about the output to the vector output unit; at this point, the instruction is ready to be executed.

Monitor Mode and Job Mode

The central processor executes in two alternate modes: job mode and monitor mode. Different processor operations are allowed in each mode. A change between modes is called an **exchange**. An interrupt or an EXIT FORCE (09) instruction causes the central processor to change from job to monitor mode through the exchange operation. While in monitor mode, interrupts are disabled.

The exchange operation provides the central processor the capability to manage processes more efficiently. As an example, when a process incurs a page fault because its data is not found in central processor memory, it is blocked, and the processor moves to another process. The page fault causes an interrupt which in turn causes the central processor to exchange to monitor mode. While in monitor mode, the processor makes an I/O request for the page needed by the blocked process and prepares to execute the next waiting process.

When an interrupt stops a process, all the process's relevant information is stored so that the process is able to resume execution as if there had been no interruption.

Monitor Mode

During monitor mode, the operating system:

- performs system work
  (tracking task queues, for example)
- communicates with other system resources
  (with I/O processors and other central processors, for example)
- responds to user requests
- performs general housekeeping tasks
  (manages virtual memory, for example)

Privileged instructions that are not allowed in normal job mode execute in monitor mode to enable the operating system to perform these functions.

While the processor is in monitor mode, the virtual memory mechanism is disabled and memory is accessed by its real (physical) addresses. During normal monitor mode operation, interrupts are disabled. However, if an IDLE (00) instruction is executed, the processor halts and interrupts are then enabled.

Job Mode

While in job, or user, mode, the central processor fetches, executes, and returns results from instructions. The virtual memory addressing mechanism is enabled and central processor memory is virtually addressed. Virtual memory faults, such as a missing page or an attempt to execute from a read-only page, cause interrupts.

In job mode, external interrupts to the processor are enabled. The set of privileged instructions is not allowed; an attempt to execute one causes an interrupt. Job mode is entered only when an EXIT FORCE (09) instruction is executed in monitor mode.

Exchange Types

One type of exchange puts the processor into execution:

- *half exchange* (performed after a master clear, it loads the register file package from processor memory into the register file, loads the program counter from the address in register 03, and starts executing in monitor mode)

A second type with two variations puts the processor into job mode:

- *return from interrupt exchange* (an exchange to job mode performed to restart interrupted process code)
- *initial exchange* (an exchange to job mode performed to initialize the process's register file package, invisible package, page table, and so on...)

A third type, with two variations, puts the processor into monitor mode:

- *interrupt exchange* (an exchange to monitor mode performed for any bits set in the interrupt register)
- *exit job exchange* (an exchange to monitor mode performed to bring an orderly end to the execution of process code)

Exchange to Job Mode Operation

(return from interrupt exchange)
(initial exchange)

During an exchange to job mode operation, these steps are performed:

1. Store the monitor register file to central processor memory.
2. Load the executing process's copy of the register file to the hardware register file.
3. Load the process's invisible package.
4. Toggle the central processor to job mode.
5. Toggle the interrupt enable to receive interrupts.
6. Branch to the instruction whose address is found in word zero of the invisible package.

Exchange to Monitor Mode Operation

(interrupt exchange)
(exit job exchange)

During an exchange to monitor mode operation, these steps are performed:

1. Store the process's current register file to the process package in central processor memory.
2. Store the process state in the process's invisible package.
3. Load the copy of the monitor register file to the hardware register file.
4. Toggle the central processor to monitor mode.
5. Toggle the interrupt enable to disallow interrupts.
6. Branch to the instruction whose address is in register file word 03.

Monitor Interval Counter

This is a 32-bit timer that keeps track of the time the central processor spends in monitor mode. It is activated by setting it to a value greater than zero; it decrements automatically from this set value until it reaches zero. At that time, it causes an interrupt exchange to occur. Master clear deactivates this timer. The TRANSMIT (R) TO MONITOR INTERVAL TIMER instruction sets interrupted monitor mode intervals. The monitor interval counter is discussed later in this chapter.

Job Interval Timer

The job interval timer is used by application programs to time intervals. This timer is loaded only in job mode using the TRANSMIT R TO JOB INTERVAL TIMER instruction. Once loaded, the timer decrements until an exchange to monitor mode or a domain change occurs, or until the timer decrements to zero or is loaded with a value of zero. When an exchange or domain change occurs, the decrementing stops and the contents of the timer are stored in the appropriate invisible or stacked domain package. When execution is resumed, the job interval timer is loaded from the package and begins decrementing. The job interval counter is discussed later in this chapter.

Invisible Package

The invisible package consists of registers that contain address and control information necessary to begin execution of a new process or to continue an interrupted process. Each process has an invisible package built as it is readied for execution; the invisible package for each process active in a central processor is stored in processor memory.

During an exchange to monitor mode, the content of the executing process's invisible package is stored again into memory as part of the process structure. Monitor stores the process's invisible package during an exchange to monitor mode and loads it during an exchange to job mode; monitor itself has no invisible package. At process completion, some accounting information is saved and the invisible package is discarded. At an abnormal termination, the invisible package is saved along with the other process data to be printed or dumped.

Figure 5-7 illustrates the general contents of the invisible package. It is also described in section 3, "Central Processor Memory Protection".

Invisible Package:

0	current domain package number / program address
1	current instruction address
2	previous domain package number / job interval timer
3	breakpoint address
4	data flag register
5	stack limit / illegal instruction mask / stack index
6	Key 0 Key 1 Key 2 Key 3
7	▨
8	Key 4 Key 5 Key 6 Key 7
9	communication buffer base address 0
10	Key 8 Key 9 Key 10 Key 11
11	communication buffer limit address 0
12	forward domain mask 0
13	communication buffer base address 1
14	forward domain mask 1
15	communication buffer limit address 1
16	instrumentation counter 0
17	communication buffer base address 2
18	instrumentation counter 1
19	communication buffer limit address 2
20	instrumentation counter 2
21	communication buffer base address 3
22	instrumentation counter 3
23	communication buffer limit address 3
24	instrumentation counter 4
25	▨
26	instrumentation counter 5
27	▨
28	instrumentation counter 6
29	▨
30	instrumentation counter 7

Figure 5-7. Contents of the invisible package.

Scalar Processor and its Operations

The scalar processor contains the following units:

- scalar arithmetic unit
- branch unit
- decode unit
- issue unit with register file
- associative unit
- priority unit
- load/store unit

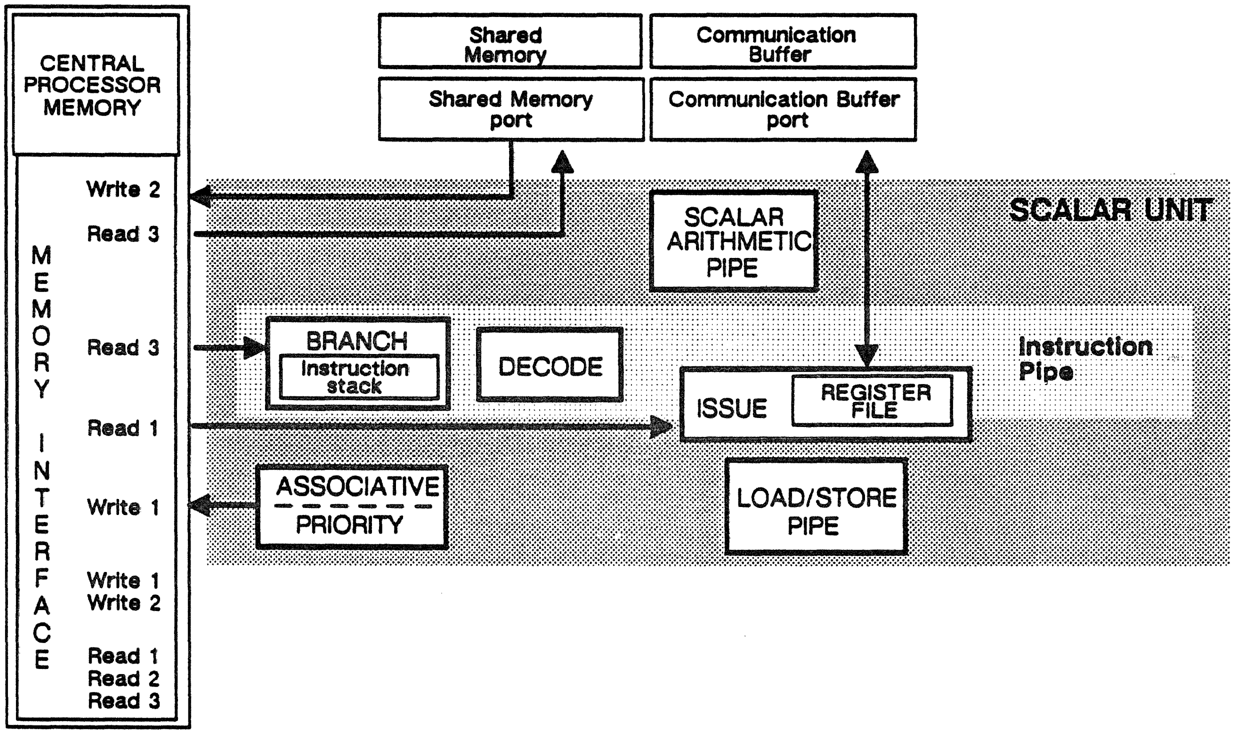


Figure 5–8. The internal components of the scalar processor.

The branch, decode, and issue units are sometimes referred to as the instruction pipe.

Scalar Arithmetic Unit

Independent units within the scalar arithmetic unit perform the arithmetic and logical operations for most instructions which return a result to the register file. (There are also vector instructions that return scalar results to the register file.) Some units handle arithmetic testing for most conditional branch instructions, and perform operations pertaining to the data flag register. The scalar arithmetic unit also contains various counters and timers, and the interrupt register.

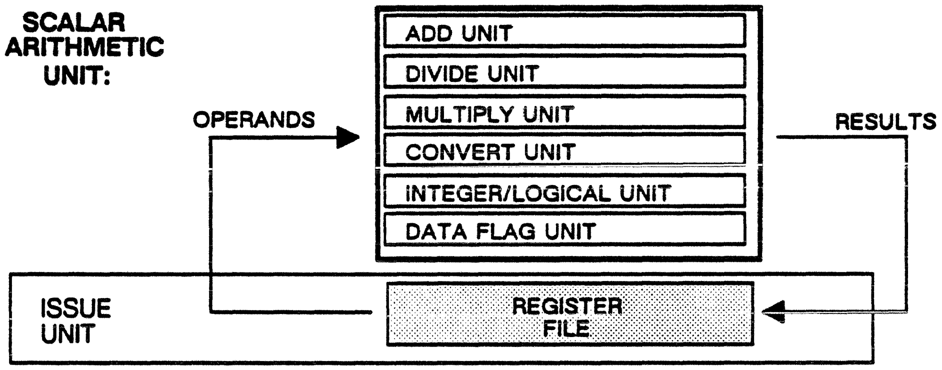


Figure 5–9. The scalar arithmetic unit.

Add Unit The add unit performs floating-point addition and subtraction, and branch compare instructions.

Divide Unit The divide unit performs floating-point divide and square root, and adjust instructions.

The divide and square root instructions are performed using standard non-restoring iterative algorithms.

Multiply Unit The multiply unit performs floating-point multiply operations.

Convert Unit The convert unit processes the CONVERT BCD TO BINARY and CONVERT BINARY TO BCD instructions. A binary coded decimal number of up to 16 digits is converted to a signed, twos complement binary number, and put into the destination register of the register file. The second instruction converts a twos complement binary number to a 15 digit number. Binary coded decimal numbers are represented as a sequence of 4-bit digits followed by a 4-bit group representing the sign of the number.

Integer/Logical Unit This unit processes the set of logical instructions that include scalar versions of logical And, inclusive Or, exclusive Or, and so on. It also processes address and length arithmetic, data movement, and shift instructions.

Data Flag Unit This unit processes several types of instructions including the data flag register instructions that read and set the data flag register. It also performs the real-time clock instructions, and the job interval and monitor interval timer instructions. The central processor interrupt register is maintained by this unit. The data flag register is discussed later in this section.

Branch Unit

The branch unit is the first component of the instruction pipe, and controls the processor's access to instructions. The primary function of the branch unit is to read blocks of instructions from central processor memory and to transmit these instructions to the decode unit. The instructions are put into a normalized form, and placed into the instruction stack. Instructions are sent to the decode unit one at a time.

The branch unit also controls the branch operations and processes the branch instructions. The issue unit sends the branch base address and index to the branch unit. The branch unit evaluates the data, and performs the necessary calculations to determine the correct branch address. For some conditional branches, it must validate that the branch can be performed. It also checks the address to which the branch is to be made and determines whether it is in the instruction stack or is out of the stack, in memory.

Instruction Stack

The instruction stack stores up to eight blocks of instructions. Each block contains eight 64-bit instructions, sixteen 32-bit instructions, or some mixture of both 32- and 64-bit instructions. 64-bit instructions can exist across blocks. There is a central processor memory address associated with each block. A record of what is currently in the stack is kept by saving the memory address of each block in the register. Two blocks are used for instruction lookahead. This instruction lookahead has the effect of limiting the largest program loop that can fit into the stack to six blocks or 96 instructions maximum.

Decode Unit

The decode and issue units are tied together functionally, and are parts of the same process.

The decode unit receives an instruction from the instruction stack, determines which instruction it is and which instruction format is being used. From this decoding, the unit determines which registers in the register file are to be used. Decode divides complex instructions into instruction segments which are sent one at a time to the issue unit.

Issue Unit

The register file is part of the issue unit. When the issue unit receives an instruction segment, it reads the data from the register file and sends it to the correct arithmetic unit if it is a scalar instruction. If it is a vector instruction, the data is read to the vector setup and control unit. When the arithmetic unit returns a result, the issue unit writes the result to the specified destination register in the register file. Vector results are handled in the vector processor except for those vectors that return scalar results.

The issue unit also manages conflicts. A conflict is a condition in which sending an instruction segment to its intended arithmetic unit would cause a problem or an error. When an instruction requires the result of a prior instruction, the instruction cannot be issued until the result has been written to the register file. A signal that stops the instruction flow is sent to the decode unit. The issue unit has a buffer register to hold data from the decode unit that cannot be processed until the conflict is satisfied.

Some additional explanation about this last statement is necessary. Results written to the register file can also be returned directly to the scalar arithmetic unit. This saves time because the issue unit can send a register file operand to scalar processing without both operands coming from the register file. The process of moving results from scalar outputs to scalar inputs is called shortstopping, and saves time spent waiting for results.

Instruction Segments

The primary function of the issue unit is to process instruction segments; when there is no conflict, one instruction segment is processed per minor clock cycle. In most cases, one scalar instruction is one segment.

Instruction segments are comprised of fields that describe the segment's intended function. Some of these fields are: the R and S address fields that define the registers holding the two source operands, the T address field that defines the register in which to store the result, and the function field that specifies the arithmetic unit that is to perform the operation.

Source Operands

Each instruction segment generally specifies two source operands. (Some exceptions are floor, ceiling, and adjust significance instructions that specify one.) Both operands are usually register file operands. The issue unit has special registers that can supply other operands, one of which is the stack index register. The issue unit can extract a variety of bitfields from the original instruction and supply them as operands. (24-bit and 48-bit immediate operands may be supplied.)

Load/Store Unit

The load/store unit receives the central processor memory base address and the index from the issue unit. The index is left-shifted three, five, or six places as appropriate, and added to the base address to form the virtual/absolute memory address for the load/store operation. (Virtual addresses in user mode; absolute addresses in monitor mode.) The adder output plus function code is placed in one of the eight holding registers. The holding registers are read in a round-robin order and the contents are sent to the associative and priority units. The priority unit returns an accept for the load/store operation requested if the requested memory bank is not busy.

Associative Unit

The associative unit contains 16 associative registers used to map a virtual address to an absolute address. When the central processor is in job mode, the branch, load/store, and priority units can make virtual address requests for memory.

The associative unit compares the virtual addresses with the virtual pages identified within the 16 associative registers. If a match is found, and the associative word lock matches one of 12 keys supplied by the process, the associative unit converts the virtual memory address. If no match is found, the branch and load/store units are halted, and the priority unit is informed of an impending space table search. The priority unit reads additional associative words from a page listing of central processor memory called the space table. If a match is found in the space table, the matching word is placed in the associative registers and normal operation resumes. If no match is found, the associative unit sends the scalar arithmetic unit an access interrupt and terminates the space table search.

When the search is completed, any matching word from either the associative registers or from the space table is placed at the top of the associative table. If the matching word is found in the space table, then the last word of the associative can be pushed to the top of the space table. In this way, virtual pages that have recently been accessed are kept in the associative registers. If a space table search is completed with no match, a null word is left in the associative registers at the completion of the search.

The combination of a lock and a virtual page address in an associative word is a protective mechanism. Any given combination of a lock and a virtual page address should occur in only one associative word in the space table. If a multiple match is detected in the associative registers, a match fault occurs, and the central processor is stopped. Locks and keys are explained later in this chapter, in the "Central Processor Memory Protection" section.

Priority Unit

The priority unit schedules and validates all requests to central processor memory, and tracks memory references. These requests come from the shared memory interface, and from the load/store and branch units. Scheduling is done using the following criteria:

- shared memory interface requests have higher priority than central processor requests.
- only one request may be sent to memory per minor cycle: any request busies the address bus for one cycle.
- a block request busies one bank of central processor memory for seven cycles and a data bus for four cycles. (An eight word block is moved two words at a time; most major paths within the central processor are 2 words wide.)
- a word request busies 1/64th of central processor memory for seven cycles and a data bus for one cycle.
- a half word request busies 1/128th of central processor memory for seven cycles and a data bus for one cycle.

Each memory request has to successfully pass four checks: slot conflict, bus conflict, bank conflict, and associative match. When the request passes the checks, the priority unit sends one signal to the memory interface to start the transfer and an accept signal to the requesting unit. If any check is not passed, the request is delayed.

The memory activity for space table searches and the STORE and LOAD ASSOCIATIVE REGISTERS instructions are handled in the priority unit, although these operations are initiated by the associative unit.

The priority unit operates in both scalar mode and vector mode. In scalar mode, the load/store unit can freely access memory. In vector mode, the priority unit freely accesses memory to transfer vector instructions. The branch unit and the shared memory interface can request data in either mode of operation.

Scalar Mode

All scalar data written to memory goes through the priority unit. As a result, slot conflicts can occur in scalar mode. There is a slot conflict when two or more requests attempt to access memory on the same minor cycle. Three types of requests can create these conflicts: shared memory interface, fetch instruction block, and load/store requests. They have priority in the order listed here.

Vector Mode

The priority unit keeps the flow of data moving as quickly as possible for the processing of vector instructions. This flow of data goes from the central processor memory to the vector input, to the vector pipe, to the vector output, and back to memory.

The priority unit receives memory addressing information from the vector setup and control unit, and generates the memory requests for the vector instructions.

Scalar Shortstop

When two instructions are chained together, the shortstop option enables the instructions to execute more quickly. For example, an ADD instruction chained to a MULTIPLY instruction means that the result from the ADD is sent to the register file and then read as a source operand for the MULTIPLY operation; processing the two instructions takes 12 clock cycles. Shortstop takes the ADD result, holds it in a temporary buffer for one cycle and the sends it into the MULTIPLY unit. The shortstop function saves 4 clock cycles.

The Scalar Pipe

The scalar arithmetic unit functions as a pipe, and operates the same way as the vector pipe. For example, when an add instruction is moved into the add unit, the first part of the addition is done in the first segment of the pipe and the partial result is moved to the second segment. At the next clock, a new add instruction can be introduced and a new operand moved into the first segment. Over the next few clocks, the partial results of the original instruction are obtained until the sum is completed and the computed result is moved out to the register file. In this way, new operands and increasingly-complete results consecutively move through the scalar pipe, and this produces a result each clock cycle.

Divide/square root and binary coded decimal operations are not pipelined.

Vector Unit Operations

The vector processor contains the following units:

- vector setup and control unit
- vector input unit
- vector output unit
- vector arithmetic unit

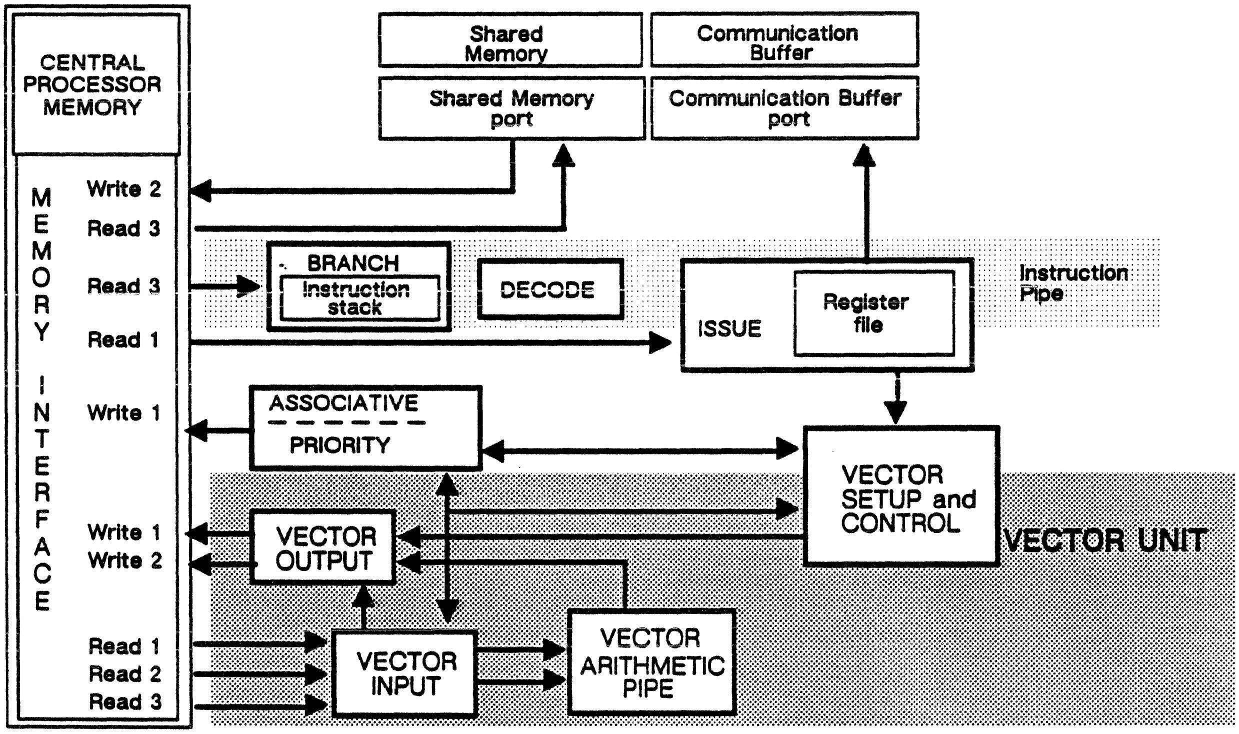


Figure 5–10. Data paths between memory, vector processor, and associated scalar components.

Vector Pipelines

In the vector unit, there are two vector pipelines. Both pipelines perform identical operations, but only one type of operation at one time. Both pipelines perform only adds, or only divides, and so on, at one time. The pipelines run in lockstep. For 64-bit operations, one pipe performs operations on the odd elements in the vector, the other performs on the even elements. For 32-bit operations, consecutive pairs of operands are operated on in each pipe. The consecutive operand pair 0 and 1 are operated on in the first pipe, pair 2 and 3 in the second pipe.

Two 64-bit results can be produced every clock cycle, one from each pipe. Four 32-bit results can be produced every clock cycle, two per pipe. If results need to be combined or used in a final result, they are put into temporary 64-bit registers between clock cycles.

Overlapping

Operations are overlapped in the pipelines; while one operation is being completed, another independent operation can be started. As an example, while vector operand A is being added to operand B for operation A + B, operands Y and Z could be fetched for the operation Y * Z.

Vector Shortstop

Vector shortstop provides the same sort of performance enhancement on the vector processor as scalar shortstop does on the scalar processor. In the shortstop operation, either pipeline result stream can be shortstopped. In addition to the usual two (A and B) input buffers in each vector pipe, there are two additional shortstop buffers. The vector input unit sends shortstop control bits that direct the pipes to use the shortstop buffers instead of the input stream for data.

Chaining

Multifunctional vector pipelines enable operations to be chained. In chaining, more than one type of operation is performed on operands as they pass through the pipeline. Chaining requires a shortstop mechanism that inserts a result into a pipeline rather than sending a result to a register. In the comparison in figure 5-11, input operands A and B are multiplied together and their result is added to input operand C. The multiply is done first, its result is shortstopped to the addition pipeline where operand C has also been sent. As the figure shows, chained operations obtain a final result more efficiently than typical operations.

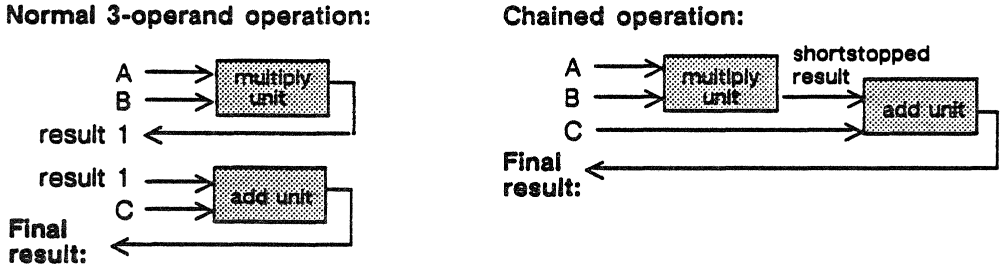


Figure 5-11. Comparison of chained and normal 3-operand operations.

Vector Setup and Control Unit

The vector control unit provides execution control and monitoring for the vector and string instructions executed in the vector processor. Shared memory and communication buffer instructions are not monitored by vector control. The decode and issue units in the scalar processor transmit the translated function code and the required register file contents of the vector and string instructions to vector control. Vector control includes the control to initiate execution of these instructions by sending function code and setup information to the priority, vector input, and vector output units, and to the pipes. This information is also sent to the interrupt counters contained in the vector control unit. These counters permit the restart of an interrupted vector instruction at the point where the interrupt occurred.

During execution of vector instructions, data is processed using two data input streams, one control vector input stream, and one data output stream. Vector instructions specify these streams in terms of field lengths, base addresses, and offsets.

Vector setup contains the registers and adders to perform the address and field-length calculations for setting up vector and string instructions. Vector setup uses registers to buffer the register file data received from the issue unit, and the stream setup information sent to the priority unit and to vector input.

Vector Input Unit

The vector input unit takes data from the memory interface, aligns it properly, and sends it to the vector pipelines for processing. At the same time, the input unit sends control information to other vector subfunctions to ready them for the data coming from the pipes. The vector input unit maintains a streaming rate (every minor cycle) until the end of the instruction unless there is a bubble (gap) in memory.

When the input unit finishes one instruction, termination information is sent to the pipes, to the vector output unit, and to vector control so they can prepare to move to the next instruction.

The input unit has two input buffers, 48 words by 128 data bits, plus six control bits. The buffers receive data from the error correction logic in the Read 1 and Read 2 ports (refer to figure 5–10). Because the read operations flow separately, both buffers can be written at the same time. Two other input buffers receive control vector data from the Read 3 memory stream. These buffers are also 48 words by 128 data bits, plus four control bits. Each buffer stores the data as it is received, and if the input length is short, data for more than one instruction is stored at a time. A five-rank ripple register allows data for up to five instructions to be stored. The priority unit keeps track of how much data it has sent to the input buffers so it stops sending when they are full. Data is moved from the unit's input buffers to output buffers as it flows through the pipes.

As interrupts arrive from the priority unit, the vector input unit processes them. If the interrupt occurs during a vector instruction before the input unit has completed the instruction, the input unit stops data flow and passes the interrupt to the vector output unit and to the pipes. If the interrupt is honored, the data is flushed from the input unit and the unit prepares for the job-to-monitor exchange.

Vector Output Unit

The main functions of the vector output stream unit are to receive data from the vector pipes and send it to the Write 1 memory interface port (refer to figure 5-10), and to receive string logic data and send it to the Write 2 port. Vector output also sends scalar results from appropriate vector operations to the register file.

The vector output unit manages data from up to ten vector instructions. Vector output also holds and modifies the index counters for compare type instructions, and sends the results to the memory interface or to the register file when the instruction terminates. Vector output forms the order vectors and sends them via Write 2 to the interface. Free data flag bits are sent to the data flag register for vector operations that can affect data flag bits.

From 32 and 128 bits of data, in 32-bit increments, may be written into the Write 1 buffer at one time. The priority unit initiates reads of the buffer into the memory interface.

The Write 2 buffer is four blocks of 16 by 132 bits. Up to three instruction results can be stored into the Write 2 buffer at one time. Data is written into the buffer one half word every other cycle at the maximum rate. As the result from one instruction terminates, the buffer is prepared for the next instruction data. Results from the string pipe are sent to Write 2.

Vector Arithmetic Unit

The vector pipelines perform all arithmetic and logical operations for the vector arithmetic unit. The vector arithmetic unit consists of two pipelines, pipe 1 and pipe 2, which are identical in most respects. Pipe 1 contains some additional logic not included in pipe 2. Figure 5-12 illustrates the components of both vector pipelines.

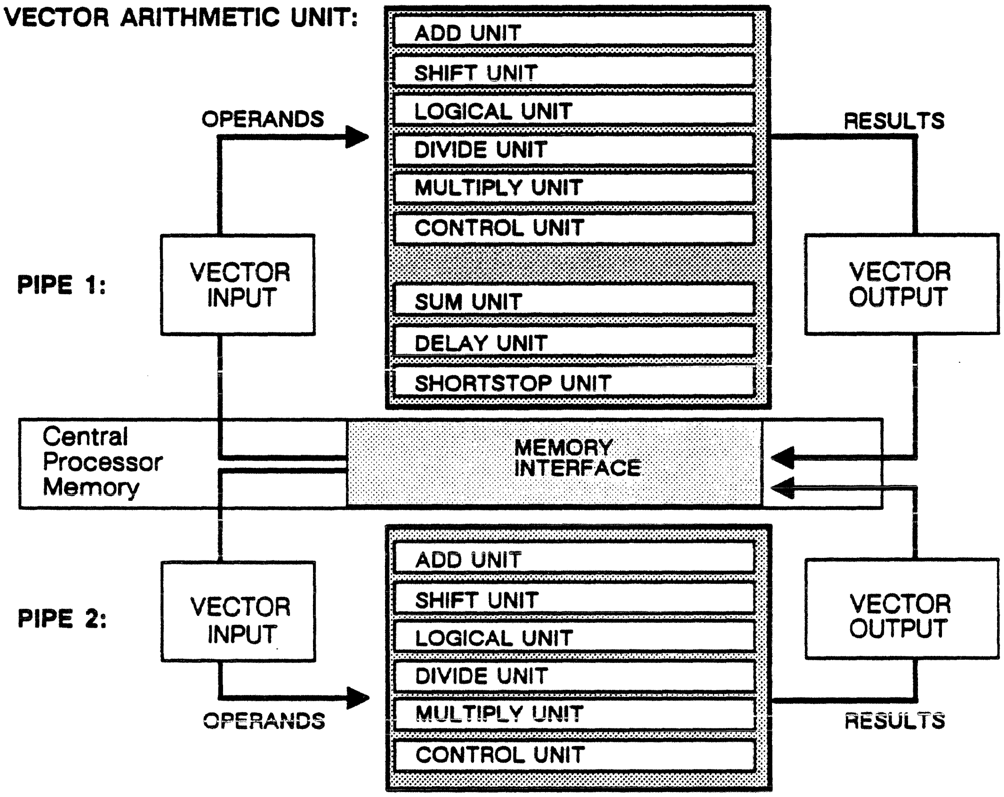


Figure 5-12. Internal components of the vector pipelines.

Add Unit The add unit performs floating-point addition and subtraction, and compare instructions. It contains a front-end section and two independent add sections, one for 32-bit operand sizes, one for 64-bit/32-bit operand sizes. The front end receives the A and B input streams from memory and formats them for 32- or 64-bit arithmetic. The resultant streams and any input end-cases are sent to the appropriate add section to be processed. Results are sent to the vector output unit.

Shift Unit The shift unit performs the shift instruction. The shift section consists of a combination right sign-extend/left end-around shifter, and a shift-count network. The shift-count network determines the direction and amount of shift. The results are sent to the vector output unit.

Logical Unit The logical unit performs Boolean and pack/unpack operations, implementing any Boolean function of two variables. The pack/unpack section assembles and disassembles floating-point operands or vector descriptors. The result from the logical section is sent to the vector output unit.

Divide Unit The divide unit performs floating-point and square root instructions. It contains a front-end section, eight iterative sections, and two back-end sections. There are eight iterative sections so they may be overlapped to obtain a higher streaming rate.

The divide and square root functions are done using standard non-restoring iterative algorithms. The divisor and dividend/radicand must be positive, and the divisor must be normalized for the algorithms to function properly. The divide unit's front-end performs these operations. Partial adders are used in the eight iterative sections to speed up the iterative process, but it is not possible to resolve the quotient in one cycle. The result is that the decision to add or subtract on the next iteration cannot be made until the next cycle. To eliminate this problem, both a partial add and a partial subtract are done every iteration, and the proper operation is selected when the quotient bit is resolved.

Multiply Unit The multiply unit performs floating-point multiply and adjust instructions. It contains a front-end section and two independent multiply sections, one for 32-bit and one for 64-bit/32-bit arithmetic operations.

The front-end receives the data streams and sends them through an input-formatting network. The resultant streams, significance and normalize counts, and input end cases are sent to the appropriate multiply section. This section recodes the operands, allowing the multiplication to be done two bits at a time, or as the sum of 24 intermediate products. The final result is formatted for 32- or 64-bit arithmetic and sent to the vector output unit.

Control Unit The control unit decodes the instruction function codes and manages the control information for the vector arithmetic unit.

Sum Unit This unit is included only in pipe 1. It performs double-precision adds for the sum and dot product instructions, and also performs the interval instruction.

Delay Unit The delay unit is used to provide delay cycles for certain operations that sequence the inputting of data. For example, at the start of a linked triad operation, there are three inputs. The third input is sent through the delay unit for as many cycles as are required for the first two inputs to be operated upon. When the result from inputs one and two is obtained, it can then be put into operation with the suitably delayed third input.

The delay unit receives the input streams from pipe 1 and pipe 2 units, and inserts them at variable points in the chain of 64-bit delay registers as determined by control lines from the control unit. The delay outputs can be interchanged and sent to either pipe 1 or pipe 2 as the next operation requires.

Shortstop Unit The shortstop unit buffers the result stream coming from either pipe for use as an input stream for a subsequent instruction executed in that pipe.

The Register File

The register file functions as the highest level of the memory hierarchy, providing the fastest data access for the central processor.

The register file is a set of 256 directly addressed 64-bit general purpose registers. The lower 128 64-bit registers can also be addressed as 256 32-bit registers (with the reservation that any address to 32-bit register 01 is illegal). Instructions have 8-bit fields to directly address the register file. The instruction's function code or, in some cases, a word size bit in its subfunction field, determines whether references are to 32-bit or 64-bit operands.

Regular register (scalar) instructions contain R and S fields for the source operands, and a T field where the result operand is to be sent. Source operand fields contain 8-bit designators that reference specific registers holding the source operands. The destination field holds an 8-bit designator that references the register that is to contain the result operand. In contrast, fields in vector instructions reference registers containing the locations in central processor memory for source and result operands.

Register File Structure

This diagram of the register file shows both 64-bit and 32-bit registers. At the right of the registers are the bit addresses; at the left are the values of 8-bit designators used by the instructions to address 64-bit registers. Within each register half are the values of 8-bit designators used by the instructions to address 32-bit registers.

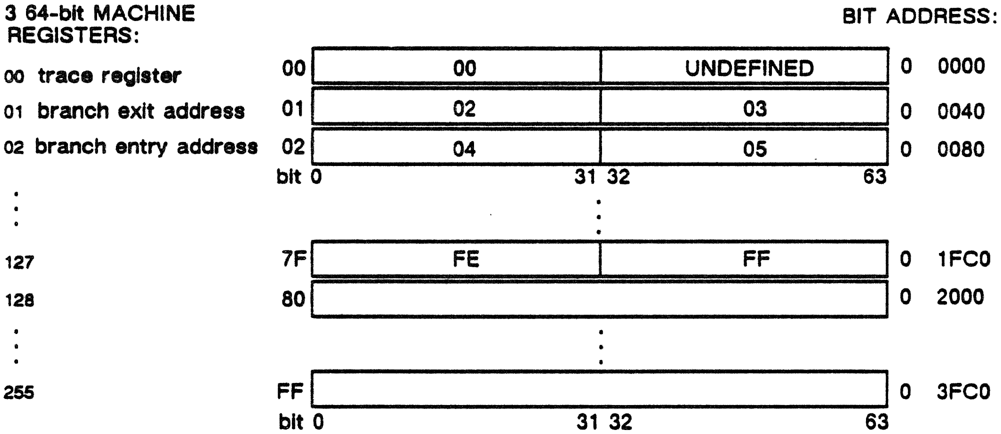


Figure 5–13. Structure of the register file.

Register 00 is a special register that provides machine zero or all zero bits depending upon the type of instruction referencing it. When register 00 is specified as the destination register, no data is stored. A write of register 00 is done to dump an unwanted result in an instruction that must specify a destination register. The 32-bit register 01 is the odd half word of 64-bit register 00; any read or write to 32-bit register 01 produces undefined results.

Part of the exchange to monitor mode operation is to load and store the entire register file to central processor memory. During an exchange operation, the contents of the trace register and the appropriate central processor memory location for register 00 are exchanged (swapped). When the processor is in monitor mode, register 03 holds the central processor memory address of the first instruction to execute after the exchange to job mode occurs. The trace register keeps a record of where the last branch came from, so that a programmer can back track to the last instruction issued before the branch. The current instruction address register, or program counter, is stored in the trace register.

Branch Registers

The register file has specific registers reserved for use when the data flag branch is being used. For 64-bit branch conditions, register 01 and 02 must be reserved exclusively for the branch. Register 01 is the data flag branch exit address, register 02 is the data flag branch entry address. For 32-bit branch conditions, registers 02 through 05 must be reserved. The data flag branch register is described in the next section.

Register File Conflicts

There are two basic types of register file conflicts: operand conflicts and unit busy conflicts. Operand conflicts occur when an instruction has a read or a write reference to a register that is waiting for a result of a prior instruction.

Operand Conflicts

A register file operand conflict can occur when instructions are chained together. Chained instructions are instructions with operand dependencies; an instruction cannot execute until it receives the result of a preceding instruction. The result operand from one instruction is to be a source operand in a later instruction; the later instruction cannot execute until the result is ready. Instructions in conflict do not have to be successive.

When chained instructions are successive, the second instruction is ready for the result of the first instruction on the second clock (because new operands are fed into the arithmetic units every cycle). At this point, the register file is searched for the operand; the register file is in conflict because several more cycles are required to return a result to the register file. Until the conflict is resolved, no new instructions are issued.

Unit Busy Conflicts

Unit busy conflicts occur when scalar and vector processing units near overloading or become overloaded with operands/instructions. The issue unit monitors the ongoing status of the various load/store and scalar and vector units, and determines if sending another operand/instruction to a unit would exceed the processing capacity of that unit. For example, the load/store unit can hold eight pairs of unprocessed operands; no more operands are sent until at least one pair is processed.

Vector Operands

Vector operations are performed memory-to-memory; operands are obtained from central processor memory and results are placed there. The register file contains vector descriptors that point to memory locations where vector operands are stored. Vector descriptors are 64 bits in length and contain a 48-bit memory address and a 16-bit length descriptor. The issue unit reads the vector descriptors from the register file and sends them to the vector setup and control unit.

Scalar Operands

Except for memory load and store operations, scalar operations are register-to-register. The register file is the source of all scalar data and the destination of all scalar results. Source operands from the register file are sent to a particular scalar arithmetic unit; some number of clocks later the result is obtained and is returned to the register file.

Write Operations to the Register File

A register file write operation occurs when the instruction specifies a register file result. Only one write to the register field can occur during a minor clock cycle. (During exchanges, two words per clock are moved in or out.) When a load from memory instruction is issued, data is moved from memory and written to the register file.

The issue unit schedules results from the scalar arithmetic units to be written into the register file and manages competition for register file slots. Iterative instruction results are held in a scalar output rank until the issue unit collects it and sends it to the register file. Iterative instructions are divide and square root, for example. Results from non-iterative instructions are immediately written to the register file.

Communication buffer and shared memory read instructions have results destined for the register file. The issue unit has an 8-word first-in-first-out buffer in which results from communication buffer and shared memory instructions are stored. The oldest entry in the buffer is written as the issue unit finds an unscheduled register file write slot. Some vector instructions have register file results that are stored in a second 8-word first-in-first-out issue unit buffer.

Write Conflicts

When there are entries from both issue unit first-in-first-out buffers waiting to be written, priority is given to the results from communication buffer and shared memory instructions.

Data Flag Branch Register

The data flag register enables a program to branch to special routines upon the occurrence of certain operands, results, or conditions. The branch can be automatic, and saves the time required for the program to explicitly check for enabled conditions. A program may contain a condition that is specified to cause an automatic branch. When such a condition occurs during an instruction, the address of the next instruction to be executed is stored in register 01 and a branch is made to the address contained in register 02.

If parallel vector and scalar processing are occurring, the central processor will execute a variable number of instructions before the automatic branch actually takes place. Automatic branches can occur between instruction issues in the scalar processor.

Four 16-bit fields define the 64-bit data flag register. This register is stored into word 4 of an invisible package, a domain package, or a stacked domain package. Bits 0 through 2, 16 through 18, 32 through 34, and 48 through 50 are undefined. The result of an instruction attempting to sample, set, or clear these bits is undefined.

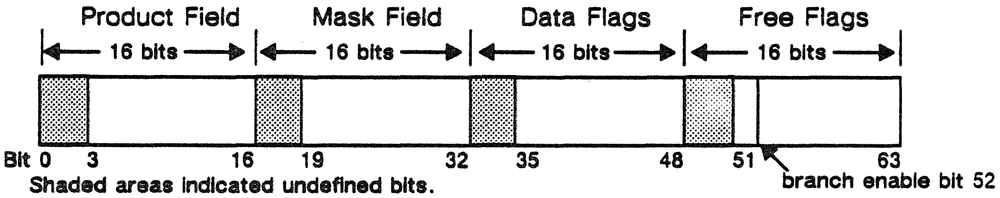


Figure 5-14. Structure of the data flag branch register.

Data Flags

Data flag bits 35-47 indicate special conditions that have occurred. Succeeding instructions will not clear any set bits. Bits 35-47 are cleared only by the DATA FLAG REGISTER BIT BRANCH AND ALTER and the DATA FLAG REGISTER LOAD/STORE instructions.

Mask Field Bits

Mask bits are used to select the conditions which are to cause a data flag branch and a data flag branch manager interrupt. For example, bit 25 enables a data flag branch on a floating-point divide fault. Although a mask bit is associated with each of the data flags, the associated mask bit does not have to be set in order to set a data flag bit. The mask function enables a particular data flag to cause a bit to be set in the product field. Setting the mask bit and the data flag bit may be done in any order without affecting the result, the result being that their associated product bit is set.

Product Field Bits

Each product bit is the dynamic logical product of a data flag bit and its associated mask bit. Branches are performed when there is at least one "1" in the product register and the data flag branch enable bit is set.

Free Data Flags

The first of the free data flag bits is the dynamic inclusive OR of the product field. The second bit, bit 52, is the data flag branch enable bit.

There are no product or mask bits associated with bits 53-55. These bits are initialized during the initial phases of the instructions that do set them, and setting these bits does not cause a data flag branch.

There are no product or mask bits associated with bits 56-63. Their purpose is to assist software in determining what operation caused specific data flag bits to be set. For example, if a vector operation generates a divide fault, data flag bit 41 and free flag bit 59 will both be set and so indicate a divide fault. If a scalar operation causes the fault, only data flag bit 41 is set.

Data Flag Branch Enable Bit

The data flag branch enable bit (bit 52) must be set for a data flag branch to occur. If this bit is a one and free bit 51 becomes a one, or vice versa, a data flag branch occurs at the end of the current instruction. The enable bit is automatically cleared by the hardware when a data flag branch takes place and must be reset to re-enable branching.

Branches

When a data flag branch occurs, the address of the next instruction to be executed is stored in register file address 01 and a branch is made to the address contained in register 02. The address of the data flag branch manager software entry point is placed in register 02 during execution-time initialization. Subsequent processing is determined by the settings in register 01 and specifications made in special program calls.

The address in register 01 does not necessarily point to the instruction immediately following the instruction that caused the branch. The hardware initiates a data flag branch only after all currently executing instructions have completed. Because instructions may be executing in parallel when the condition causing the data flag branch occurs, the branch can occur several instructions after the instruction causing it.

Data Flag Branch Manager

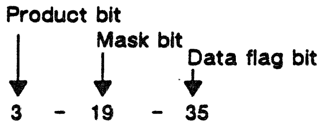
The data flag branch manager is a library routine that processes data flag branches as they occur during execution of a program. The data flag branch feature is used to eliminate the time penalty that would be required for the programmer to perform explicit checks for special conditions. Any of the following conditions can cause a data flag branch to occur:

- a square root operation attempted with a negative operand
- a division operation attempted with a zero divisor
- an exponent overflow in computation of a number too large to be represented internally
- an exponent underflow in computation of a number too small to be represented internally
- an operation attempted using an indefinite operand
- reduction of the job interval timer to zero
- execution of a hardware breakpoint instruction under certain usage conditions

The programmer selects the conditions to cause a data flag branch. When the branch occurs, control passes to the data flag branch manager to perform interrupt processing for the condition. The data flag branch manager interrupts the executing program, issues an error diagnostic, dumps the contents of the data flag branch manager, and aborts the program. Default interrupt processing that does not cause the program to abort can be selected. The programmer can also specify the processing that is to be performed as a result of the interrupt.

Data Flag Branch Register Bit Assignments

Assignments are made with combinations of associated bits that are set together to indicate a condition or occurrence:



As an example, the 3-19-35 assignment shown above indicates a soft interrupt. In a soft interrupt, monitor software sets bit 35 of the job's data flag branch register while the register is stored in the invisible package. When the exchange to job mode is made, a normal data flag branch occurs if bit 35 and its corresponding mask bit are set.

Assignments indicate a variety of conditions or occurrences, including:

- job interval timer decremented to zero
- select condition not met, no match on MASKED BINARY COMPARE instruction
- binary result exceeds range of $(+2^{47}-1)$ to (-2^{47})
- floating-point divide fault
- an exponent overflow
- result machine zero
- negative source operand encountered in a square root operation
- indefinite result placed into central processor memory or register file, or both operands of a floating-point compare were indefinite
- breakpoint occurred

Central Processor Interrupt Register

The central processor interrupt register is 64 bits long. It receives interrupt information from sources within the central processor as well as from the system-wide interrupt network. Within the central processor, the internal interrupt input register receives interrupts and passes them to the interrupt register. The external interrupt input register receives interrupts from other system processors via the interrupt network and transfers them to the interrupt register.

An interrupt signal is sent to the central processor decode unit when any bit is set in the interrupt register. Every 64 clock cycles, the contents of the interrupt register are sent to the transmit interrupt register. This register converts the data and sends it to the service unit. The central processor interrupt register is read by the READ INTERRUPT REGISTER TO T instruction.

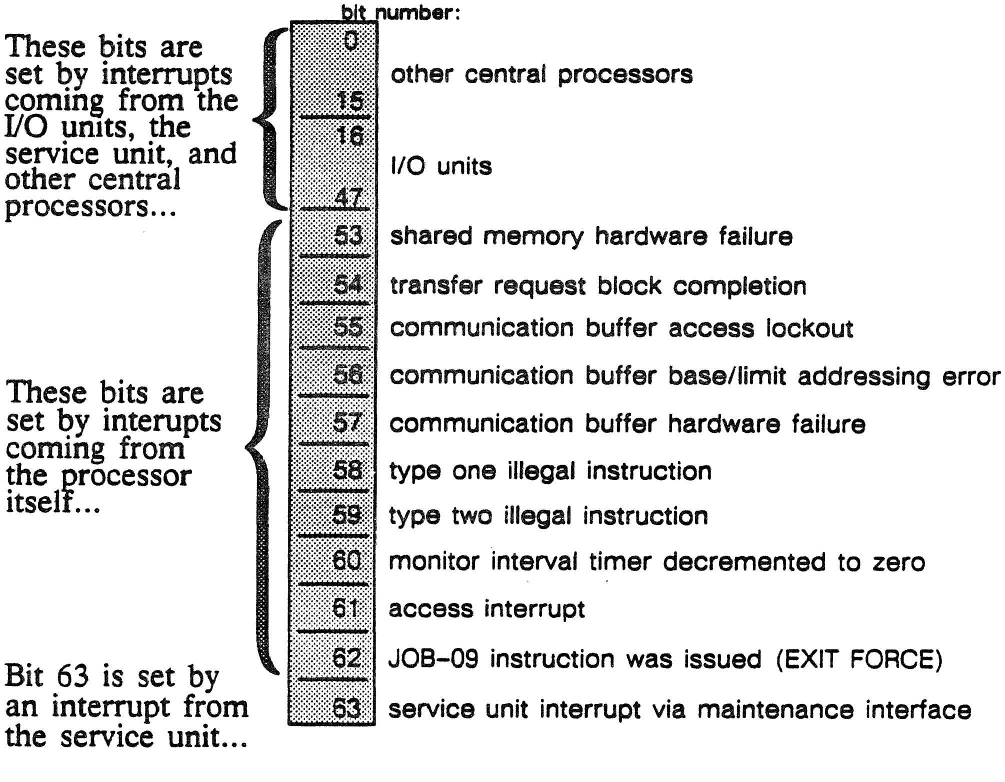

bit number:
These bits are set by interrupts coming from the I/O units, the service unit, and other central processors...
0
15
other central processors
16
47
I/O units
These bits are set by interupts coming from the processor itself...
53
shared memory hardware failure
54
transfer request block completion
55
communication buffer access lockout
56
communication buffer base/limit addressing error
57
communication buffer hardware failure
58
type one illegal instruction
59
type two illegal instruction
60
monitor interval timer decremented to zero
61
access interrupt
62
JOB-09 instruction was issued (EXIT FORCE)
Bit 63 is set by an interrupt from the service unit...
63
service unit interrupt via maintenance interface

Figure 5-15. Structure of the interrupt register.

Interrupt Exchange from Job to Monitor Mode

An interrupt exchange from job to monitor mode occurs as a result of the central processor being interrupted by any one of the sources listed above in figure 5-15.

When the interrupt is recognized, the following actions take place:

- The process's register file contents are stored in its register file package.
- The process's invisible package is stored in the process structure.
- The monitor register file contents are loaded into the register file; it had previously been stored starting at physical address zero.
- The central processor mode is switched to monitor mode.
- The interrupt enable is cleared; interrupts are disallowed in monitor mode.
- To perform the next instruction, the central processor branches to the address specified in the register file at register 03.

Interrupt Network

System-wide interrupts are managed through the interrupt network. Any system processor can issue and receive interrupts from any other system processor through the interrupt network. The network includes interrupt registers in all system processors; central processors each have a 64-bit register, I/O units and service unit nodes each have a 48-bit interrupt register.

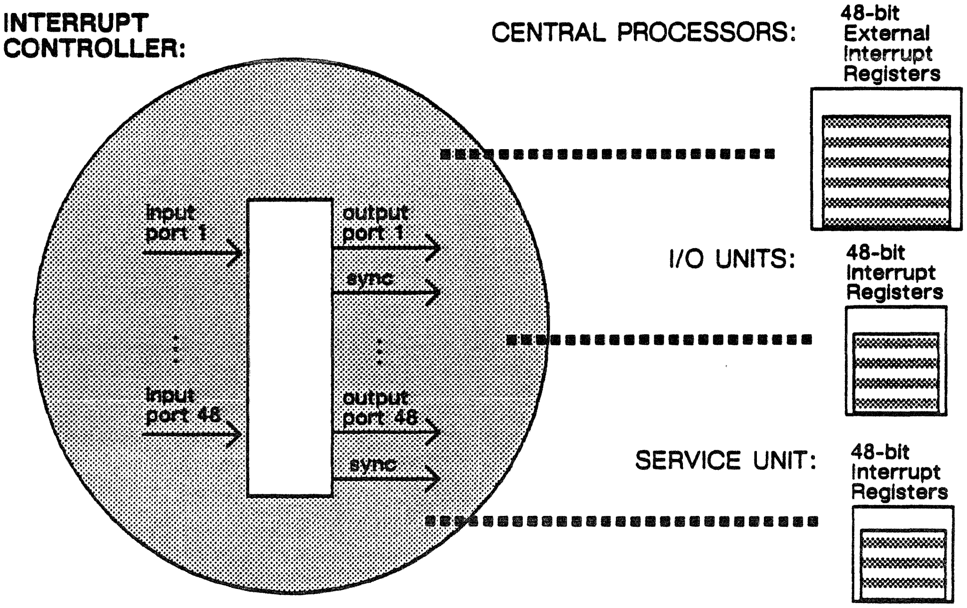


Figure 5-16 . The system-wide interrupt network.

The interrupt controller has 48 input ports and 48 output ports, the output ports are each matched with a synchronization port. Because the network can receive up to 48 interrupt requests simultaneously, the reception of interrupts is sequenced to ensure each is forwarded to the correct processor.

Control lines communicate with each input and output port. When an interrupt is clocked in from an input port, control lines shift interrupt data to the correct output port. Some cycles later the interrupt signal is sent through the correct output port to the target processor's interrupt register to be handled by processor software.

The interrupt network is controlled by a single chip located on the communication buffer interface board. In a redundantly configured ETA10 system, there are two communication buffer interface boards, and two interrupt networks. Only one interrupt network is active at any time. The interrupt network is synchronized by the master clock.

Machine Addressing

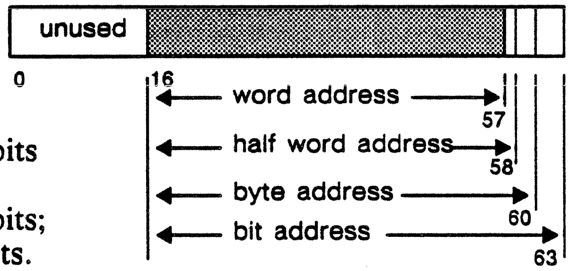
The ETA10 is a bit-addressed machine. All addresses are 48-bit quantities and contain enough information to reference a specific bit. Bits, bytes, half words, words, and pages are always addressed from left to right; the first bit of the addressed entity is the left-most bit of that entity. While the processor is in job mode, those bits are mapped through the virtual-to-physical addressing mechanism. In monitor mode, the addresses are real and the upper bits are ignored. Addressing is in modulo with respect to the 4 million word address limit in central processor memory.

Bit-Level Addressing

The lowest bit in an address can address to a specific bit in memory, and so lower bits are reserved for bit addressing. All addresses in the ETA10 are bit level except indexes and offsets that address to the word, half word, or byte level. As shown below, groups of bits address various units of storage:

The numbers indicate the bit position in a register or in an instruction word.

A full word has the lowest 6 bits of an address set to zero; a half word has the lowest 5 bits; and a byte has the lowest 3 bits.



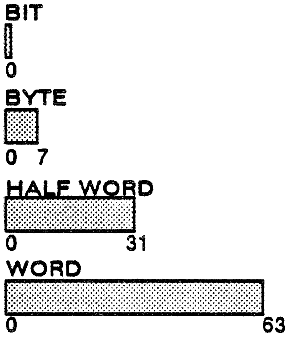
The addressable range is 2^{48} bits (281,474,976,710,656 bits), or 2^{42} words (4,398,046,511,104 words).

Words, Half Words, and Bytes

A byte is defined as an 8-bit quantity; the address of the left-most bit is always a multiple of 8_{10}.

A half word is defined as a 32-bit quantity; the address of the left-most bit is always a multiple of 32_{10}.

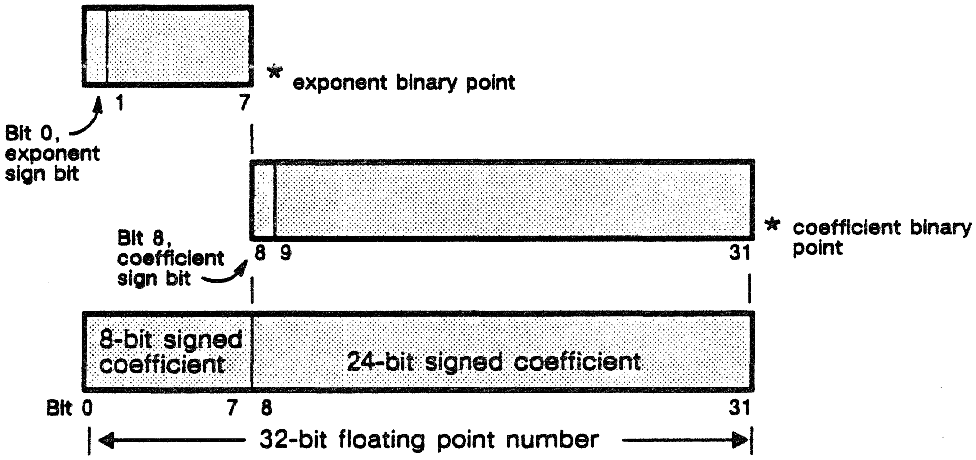
A word is defined as a 64-bit quantity; the address of the left-most bit is always a multiple of 64_{10}.



Using Floating Point Numbers and Arithmetic

32-Bit Floating Point Format

There are two 32-bit half words in each 64-bit word. A 32-bit floating-point number occupies a half word. Bit zero holds the sign: 0 for positive, 1 for negative.


* exponent binary point
1
7
Bit 0, exponent sign bit
Bit 8, coefficient sign bit
8 9
31
* coefficient binary point
8-bit signed coefficient
24-bit signed coefficient
Bit 0
7 8
31
32-bit floating point number

Both coefficient and exponent are expressed as twos complement, signed integers. Numbers are of the form $c \cdot 2^x$, where c is the 24-bit signed coefficient, x is the 8-bit signed exponent, and the base is 2.

The range of coefficients is from 800000_{16} to $7FFFFF_{16}$, which is from $-8,388,608_{10}$ to $+8,388,607_{10}$. The range of useful exponents is from 90_{16} to $6F_{16}$, which is from -112_{10} to $+111_{10}$.

The values of 70 through $8F_{16}$ all fall into a special end case as shown here:

Element:	Representation:
Machine zero	$8XXXXXXX_{16}$
Indefinite	$7XXXXXXX_{16}$

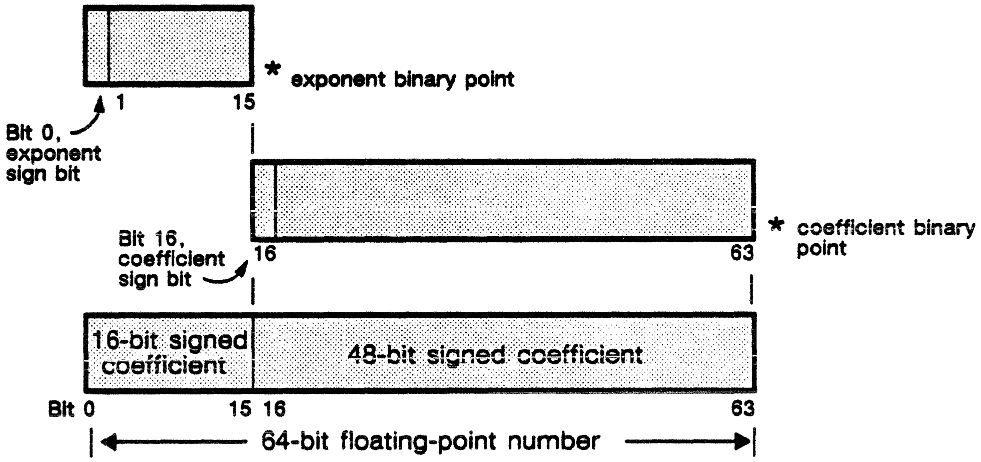
These are examples of 32-bit floating-point format represented in base 16:

+1	00	000001
+1 normalized	EA	400000
−1	00	FFFFFF
−1 normalized	E9	800000
$+256_{10}$	00	000100

As can be seen, the coefficient is an integer, the binary point is to the right of the least significant bit. A floating-point number is normalized if the coefficient sign bit is different from the next bit to the right. This condition implies that the coefficient has been shifted to the left as far as possible. An all-zero coefficient requires special attention for normalized operations.

64-Bit Floating Point Format

A 64-bit floating-point number is contained in a 64-bit word. Bit zero holds the sign: 0 for positive, 1 for negative.


* exponent binary point
1
15
Bit 0, exponent sign bit
* coefficient binary point
Bit 16, coefficient sign bit
16
63
16-bit signed coefficient
48-bit signed coefficient
Bit 0
15 16
63
64-bit floating-point number

Both coefficient and exponent are expressed as twos complement, signed integers. Numbers are of the form $c \cdot 2^x$, where c is the 48-bit signed coefficient, x is the 16-bit signed exponent, and the base is 2.

The range of coefficients is from 8000 0000 0000_{16} to 7FFF FFFF FFFF_{16}, which is from $-140{,}737{,}488{,}355{,}328_{10}$ to $+140{,}737{,}488{,}355{,}327_{10}$. The range of useful exponents is from 9000_{16} to 6FFF_{16}, which is from $-28{,}672_{10}$ to $+28{,}671_{10}$.

The values of 7000 through 8FFF_{16} all fall into a special end case as shown here:

Element:	Representation:
Machine zero	$\text{8XXXXXXXXXXXXXXX}_{16}$
Indefinite	$\text{7XXXXXXXXXXXXXXX}_{16}$

These are examples of 64-bit floating-point format represented in base 16:

Element:	Representation:
+1	0000 0000 0000 0001
+1 normalized	FFD2 4000 0000 0000
−1	0000 FFFF FFFF FFFF
−1 normalized	FFD1 8000 0000 0000
$+256_{10}$	0000 0000 0000 0100

As can be seen, the coefficient is an integer, the binary point is to the right of the least significant bit. A floating-point number is normalized if the coefficient sign bit is different from the next bit to the right. This condition implies that the coefficient has been shifted to the left as far as possible. An all-zero coefficient requires special attention for normalized operations.

End Cases

When indefinite is used as an operand in a floating-point instruction, both the upper and the lower results are indefinite. For the cases listed below, 0 represents machine zero and N represents an operand that is neither machine zero nor indefinite.

0 ± 0 = 0	0 * 0 = 0	0 / 0 = Indefinite
0 ± N = ±N	0 * N = 0	0 / N = 0
N ± 0 = N	N * 0 = 0	N / 0 = Indefinite

Precision

Full and half words undergo operations with differing degrees of precision. Precision reflects the number of fully accurate bits in an answer, and depends somewhat on the number of bits in the answer. As a result, operations using full 64-bit words produce answers that have higher degrees of accuracy, or precision, than those using half words because there are twice the number of bits in full words. In the 24-bit coefficient of a half word, numbers are precise to seven decimal digits. In the 48-bit coefficient of a full word, numbers are precise to 14 decimal digits.

Twos Complement Arithmetic

The twos complement binary number system is used in central processor arithmetic operations because it simplifies computing. In the twos complement system, the left-most bit holds the sign (0 for plus, 1 for minus), and the remaining bits hold the number itself. Positive numbers are represented by their binary equivalent. Negative numbers are formed by replacing the 1s in the positive number with 0s, the 0s with 1s, and adding 1 to the result; a carry from the left-most bit is discarded.

Error Management in the Central Processor

Hardware errors and detectable software errors in the central processor are processed through system control software polls of the hardware maintenance registers and by the central processor Monitor software. Depending upon their severity, errors cause the system to halt or the executing program to be interrupted.

Hardware Detected Events

Hardware detected events cause the central processor to be halted or an interrupt to be issued. When the central processor halts, the service unit performs diagnostic and recovery operations, the error is logged and the system restarted. Some hardware errors require removal and repair of a component. When the system is interrupted, the recovery activity depends upon the origin of the error. If the error is caused by a user's program, the operating system terminates the program and forwards a report of the termination status to the user. If the error is within the operating system, then the system control features report the error, perform a dump to capture the state at the time of failure, and attempt to restart.

Instruction Stack Parity Error

This error indicates a hardware failure and is identified by the service unit system control software polling the hardware maintenance registers. The central processor stops. Instruction stack parity is generated just before the data is written into the stack. Parity is checked at the time a valid instruction is put in the instruction stack output register.

Multiple Match Error

This error occurs when the hardware detects duplicate associative register entries. It causes the central processor to halt. The error indicates that either the hardware incorrectly read the virtual address translation data, a software failure has occurred in system domain mode, or system tables are corrupt due to software or hardware failure.

Illegal Monitor Mode Instruction

This fault occurs when the hardware detects an attempt to execute an illegal instruction in monitor mode. When it happens, the central processor stops. The error is detected by the service unit through polling of the hardware maintenance registers. It indicates that the hardware incorrectly read or decoded an instruction, that a software failure has occurred in monitor mode, or that monitor code is corrupt (a software or hardware failure has occurred).

Central Processor Memory Faults

Faults that occur while the service unit is reading central processor memory are reported via the processor maintenance registers. If the central processor is also accessing its memory, any fault indication in the maintenance registers may result from either the service unit or the central processor access. Single error correction-double error detection (SECDED) is provided within central processor memory. Parity, or single error detection, is provided between the memory interface and the shared memory port.

Central Processor Memory Double SECDED Error

This fault indicates central processor hardware has failed, and it includes double, odd multiple, and various other memory faults. It is detected at the service unit through polling of the hardware maintenance registers. When it happens, the error is logged and the central processor halts.

Single Bit SECDED Error

This fault is detected at the service unit through polling of the hardware maintenance registers. A correctable memory failure is indicated, and is recovered by hardware. When this happens, the error is logged and the central processor does not halt.

Software Detected Events

Communication Buffer Access Lockout Error

This fault occurs when execution of a communication buffer instruction would violate communication buffer access privileges. This fault indicates that either central processor hardware has failed, a software failure has occurred in system domain code, or that system tables are corrupt due to hardware or software failure.

Communication Buffer Base/Limit Addressing Error

This fault occurs when execution of a communication buffer instruction references an area of communication buffer outside that specified by the base/limit address pair.

Access Interrupt

Access violations result in an exchange to monitor mode; monitor software processes these violations. One such violation occurs when a process attempts to access a page for which it has no access rights, the page is not in the process's domain. Another violation occurs when a process attempts to illegally use a page, for example, attempting to write to a page when it has only read rights. These violations may cause a process to be terminated. When a process requests a page external to central processor memory, it faults for that page. A page fault causes an access interrupt with the result that the process is blocked and an exchange occurs. Monitor calls the pager feature to resolve the page fault.

Illegal Job Mode Instruction

This fault occurs when the hardware detects an attempt to execute an illegal instruction in job mode. When system code illegally runs in job mode, an exchange to monitor mode occurs. The fault indicates that hardware incorrectly read or decoded an instruction (a central processor failure), that a software failure has occurred in system domain code, or that system domain code is corrupt (a hardware or software fault has occurred elsewhere). This fault is caused by a type 1 or type 2 illegal instruction. Illegal instructions in user code are considered a problem with the code, and not with the system. This fault causes an exchange to monitor mode and the user job to be terminated.

Illegal Domain Change

This fault occurs when the hardware detects an attempt to enter the system domain without the proper permissions. An exchange to monitor mode occurs. This fault indicates that either the hardware incorrectly read or decoded an instruction or address (a central processor failure), a software failure has occurred in system domain code, or that system domain code is corrupt (a hardware or software fault has occurred elsewhere). This fault is displayed as an illegal job mode instruction; it is a special case of a type 2 illegal instruction. When the fault is reported to the service unit, an operator alarm is sent; eventually the central processor stops.

Shared Memory Transfer Hardware Errors

These errors include double SECDED errors either in central processor memory or shared memory, and data or address parity errors between the central processor and the shared memory interface. When one of these errors occurs, there has been either a central processor memory hardware failure, a shared memory interface hardware failure, a shared memory hardware failure, or a reference to a shared memory location containing a double SECDED fault written by the hardware in response to a prior data parity error.

A SECDED error could occur outside the area of shared memory that was read because the SECDED check is enabled on 128-bit boundaries. A data parity error detected by the shared memory interface during a store to shared memory writes a double bit SECDED error so that the next read of that area gets a double SECDED fault.

Master Clock

There is no system-wide synchronization of ETA10 components; the processors, memories, and I/O are not all synchronized together. While groups of related components are synchronized, there is no synchronization with other groups of synchronized components.

As shown in figure 5–17, central processors, shared memory, and the major interfaces are synchronized by the master clock. Through the I/O interface on the communication buffer, there is an asynchronous connection to the data pipe interface. Data pipe interfaces at both ends of the data pipe are synchronized. The data pipe connects asynchronously to the I/O unit, and enables I/O "on demand" rather than during certain cycles. Within the I/O unit itself, the I/O processors are synchronized.

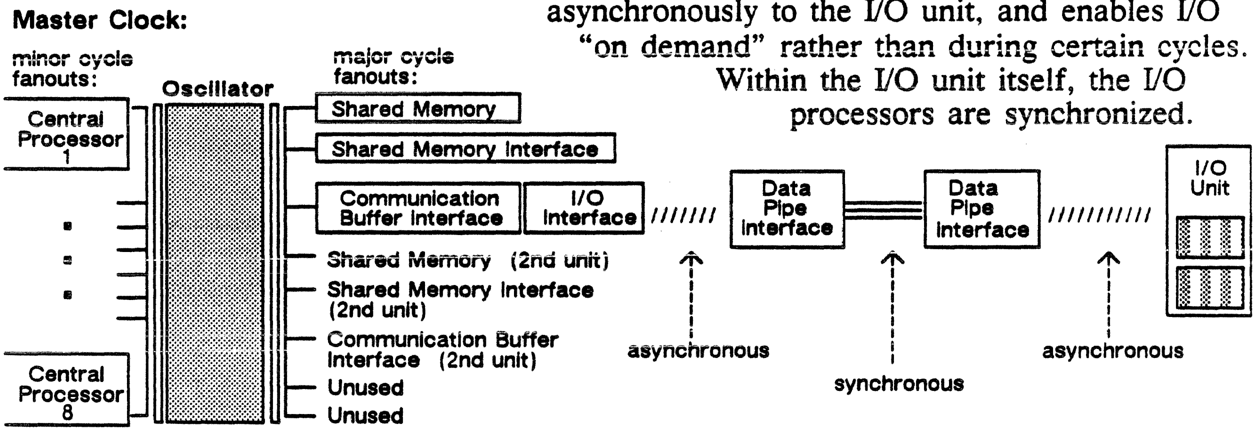


Figure 5–17. System synchronization in the ETA10.

Master Oscillator

The master clock uses the master oscillator to perform the clocking function. They reside in the shared memory SCMI cabinet; signals from the master oscillator are fanned out to clocks on these mainframe components:

- central processors
- shared memory
- shared memory interface
- communication buffer interface

Major and minor cycle times are fanned out from the master oscillator. Super-cooled boards are set to minor cycle times because the minor cycle is half the major cycle time. Super-cooled central processors are set to the minor clock cycle since their speed is nearly double that of air-cooled processors. All central processors are synchronized to within a nanosecond of each other. Components that run "warm" (at room temperature) are set to the major clock cycle. These are the shared memory and system interface boards.

Timers and Counters

The Real-Time Clock

The real-time clock is actually a counter, counting at a 1 megahertz rate. It is a free-running, 47-bit (plus a positive sign bit) counter, not set by any instruction. The real-time clock starts when the system is powered on and the master clock is started. Its function is to mark real time in the system. The master real-time clock is kept on the communication buffer interface, and sent out to all processing units. A special instruction, TRANSMIT REAL TIME CLOCK, stores the clock to a specified register. The real-time clock is incremented at a rate proportional to the mainframe clock speed.

The real-time clock is also stored in the transfer request block at the end of a shared memory transfer. An I/O processor may read or copy the real-time clock through the execution of a communication buffer instruction.

Monitor Interval Timer

The monitor and job interval timers are counters that count at the same rate as the real-time clock, but which are programmed by machine instructions. Both are 32-bit timers.

The monitor interval timer is read by the system when it exchanges to job mode. In case a job hangs or goes into a condition such as an endless loop, this timer causes an interrupt and allows the system to regain control.

When the processor is in monitor mode, the monitor interval timer can be loaded from the register using the TRANSMIT R TO MONITOR INTERVAL TIMER instruction. The timer is activated by loading it with anything but all zeros; once activated, it decrements until it reaches zero or is deactivated. It is deactivated by a master clear, by being loaded with all zeros, or by decrementing to zero. When it decrements to zero, it causes an interrupt which must be processed like any other interrupt. If the counter is halted by loading it with zeroes, no interrupt is generated.

Job Interval Timer

The job interval timer is used by application programs to time intervals. This timer is loaded in job mode only using the TRANSMIT R TO JOB INTERVAL TIMER instruction. Once loaded, the timer decrements until an exchange to monitor mode or a domain change occurs, or until the timer decrements to zero or is loaded with a value of zero. When an exchange or domain change occurs, the decrementing stops and the contents of the timer are stored in the appropriate invisible or stacked domain package. When execution is resumed, the job interval timer is loaded from the package and begins decrementing. The contents of this timer are sampled using the TRANSMIT JOB INTERVAL TIMER TO T instruction without deactivating the timer.

When the timer decrements to zero, it sets a bit in the data flag branch register; if the corresponding mask bit is also set, a data flag branch occurs at the next read-next-instruction time. The job interval timer is deactivated when it is loaded with a value of zero, or by a master clear. When it is loaded with a value of zero, it does not cause the data flag bit to be set.

Instrumentation Counters

Instrumentation counters count events within a central processor. There are eight 48-bit counters and their associated select and function codes. The counters are stored in the invisible package and domain package by an exchange operation or by a domain change. Two types of counters, fixed and selectable, are available. Results from the counters are available to users.

Fixed Counters

Counters 7 and 8 are fixed counters; they are permanently assigned to specific events. Counter 7 counts central processor clock cycles (similar to the real-time clock) that are used to determine how a program runs. It is stored but not cleared by an exchange to monitor mode. This counter is read by the TRANSMIT INSTRUMENTATION COUNTER TO T instruction. Counter 8 counts the number of central processor clock cycles the vector unit is busy.

Selectable Counters

A set of counters allows the user, via the service unit, to set the hardware up to measure the efficiency of the user's program. The items measured include the number of space table searches, time spent waiting for memory, number of instructions issued, number of instruction issued, and so on.

Up to six counters can be directed to count one of these specific events:

- issued instructions whose function codes are equal to the function code in the invisible or domain package (only done by counter 5)
- number of central processor cycles when the issue of scalar instructions is waiting for vector completion
- number of cycles vector data is processed
- number of space table searches
- number of central processor clock cycles of space table searches
- number of in-stack branches issued
- number of out-of-stack branches issued
- number of failed branches
- number of vector instructions issued
- number of vectors that used the shortstop path

- number of central processor clock cycles in which issue is delayed because no register file store cycle is available
- number of central processor clock cycles in which issue is delayed because no result address register is available
- number of forward and backward domain changes

Section 3: System Memories

This section describes the system memories and their interfaces.

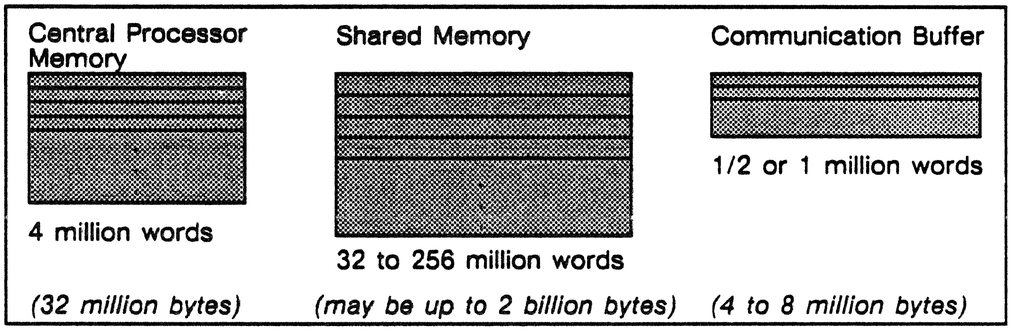


Figure 5–18. The system memories.

A central processor has unique access to its local 4 million word memory. The function of central processor memory is to provide a good-sized memory with fast access close to each central processor. Shared memory ranges from 32 to 256 million words, and is available to all central and I/O processors. Shared memory provides a substantial, accessible memory resource for the system. The communication buffer is used to store small amounts of data shared among the central processors. Communication buffer enables high speed communication between processes running on any of the central and I/O processors.

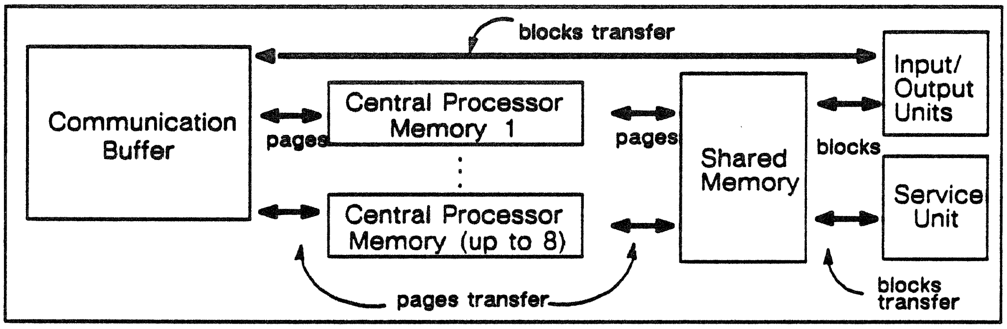


Figure 5–19. Transfer paths between the types of system memories.

Data is transferred from the I/O subsystem in blocks to shared memory or the communication buffer. Pages are exchanged between shared memory and the central processor memories.

Central Processor Memory

Central processor memory is the local memory dedicated to each central processor. It has a standard capacity of 4 million words. Central processor memory is managed and allocated in 1K blocks (1024 words).

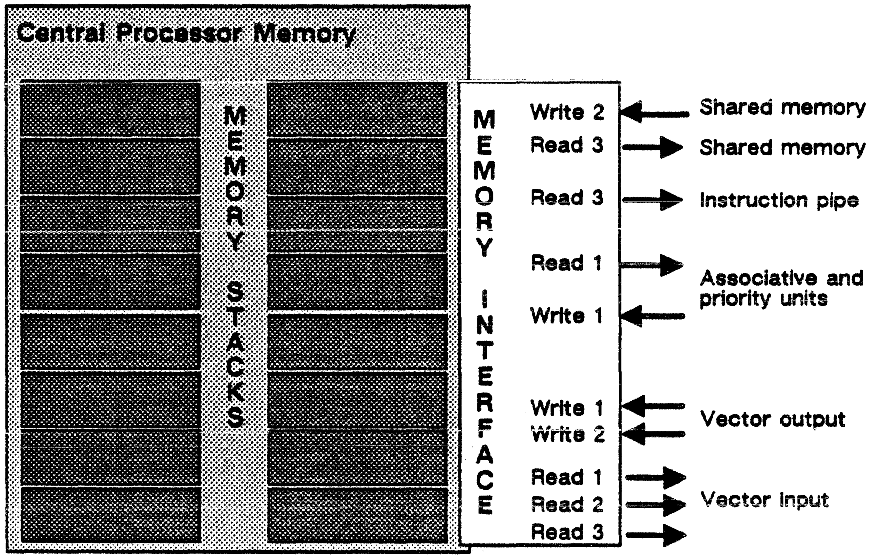


Figure 5–20. Functional diagram of central processor memory.

Structure and Organization

The memory is structured in 16 quarter-million word stacks that provide a total of 4 million words. Each stack contains eight banks of memory, 32 bits wide (plus seven SECDED bits).

The memory interface has ports that are used by the scalar processor, and the vector processor's input and output units, for internal transfers. External transfers come through the shared memory port. The service unit has maintenance access to the central processor via the system maintenance interface and does not access central processor memory.

Addressing

Central processor memory is addressed as 32-bit half words even though the memory addresses in instructions specify a bit address.

Bandwidth

The bandwidth allows eight 64-bit words (512 bits) to be transferred per clock cycle, but only one read or write port can be active in one cycle. Three of every four clock cycles accommodate the vector pipes. During each of these three cycles, either Read 1, Read 2, or Write 1 can transfer 8 words. During the fourth cycle, either Read 3 or Write 2 can transfer.

The processing pipes use two words per cycle. When Read 1, for example, loads eight words on cycle 1, it metes out two words per clock to the vector pipes for cycle 2, 3, 4. On the fourth cycle, while Read 1 sends out the last two words from the previous load, it then loads eight more words.

Error Detection and Correction

Central processor memory has single error correction-double error detection (SECDED) on each 32-bit half word for storage integrity. SECDED code is checked on all reads of data from the central processor memory. If a single SECDED error occurs, the error is corrected, and the correct data is passed on to the central processor. Both a fatal error and a single SECDED error may be reported simultaneously because more than one word at a time is transferred from memory.

Data Transmission Parity Error The central processor memory data is checked on the transfer path from the shared memory to the central processor memory interface. This path is four half words wide and parity is checked on each half word. If a parity error occurs, the possibly corrupt data is written into the central processor memory. The error is reported to the service unit via the maintenance interface for the central processor.

Memory Errors in Transfers with Shared Memory The central processor handles errors occurring on references to the central processor memory or shared memory. If a fatal error occurs, it is noted in the central processor's internal interrupt register and an error flag is set. The error flag does not prevent the offending data transfer from completing, but it does prevent the next data transfer from starting. The error flag does not prevent the central processor from executing instructions, but the error flag must be cleared before the shared memory port can restart new transfers. If a double SECDED error occurs when the transfer request block is being read from the central processor memory, the error flag is set as described above.

In addition, this type of error blocks all memory writes on data transfers to shared memory or central processor memory, and prevents the transfers from updating the completed transfer status. Memory writes are blocked because when the transfer request block is corrupt, the shared memory port might start transferring data to the wrong address and corrupt the rest of the system. The shared memory port goes through all the motions of executing the transfer request block data it receives even though no data is written.

Memory Errors in Transfers with the Service Unit When a double SECDED error occurs on a service unit read of central processor memory, it is recorded in the processor's maintenance registers and in the service unit port channel status. Single SECDED errors only show up in the central processor maintenance registers. If bad parity is detected on data sent from the service unit on a central processor memory write, a transmission parity error is set in the service unit port channel status.

Software Manager

An independent software feature resides in each central processor and allocates and deallocates memory for that central processor. Central processor memory manager is software that provides these basic management functions:

- page fault processing
- implicit file access functions
- physical memory functions
- process management support
- resource management support

Refer to chapter 4, "Operating System Kernel", for a description of this software.

Central Processor Memory Stack Assembly

Stack features:

- 4 million 64-bit words
- 64K CMOS static RAM
- 7-bit SECDED on half words
- 16 independent stacks
- 256K words per stack

The assembly is composed of one input board, eight memory boards, one output board, and two edge boards. The input, output, and memory boards are interconnected in a stacked fashion via connectors to the edge boards, forming the three-dimensional stack assembly as shown.

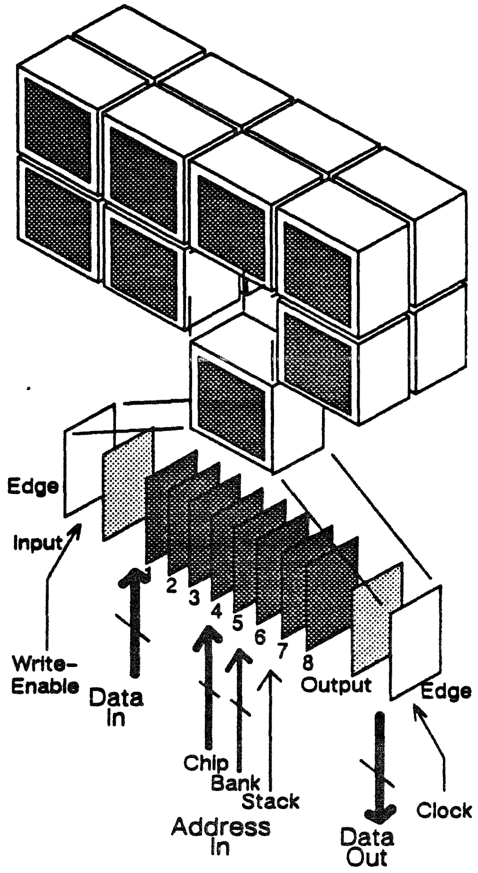


Figure 5–21. Expanded view of central processor memory stack assembly.

Stack Operations

The stack assembly is a stand-alone memory function capable of providing 512K x 39 bits of random access memory at a maximum random access rate of 49 nanoseconds, and a maximum banked or phased rate of 7 nanoseconds.

In addition, the stack accepts address input changes at a maximum rate of seven nanoseconds provided that the bank address bits (which enter the stack one cycle before all other signals) are the least significant bits, and an identical bank address field (three bits) must occur at a rate no faster than 49 nanoseconds. Three different clock adjustments tune out variations in device processing. One clock is tuned to latch in address and write data. A second clock tunes the gate array control signals. The third clock is tuned to latch in the read data sent out the stack output connector.

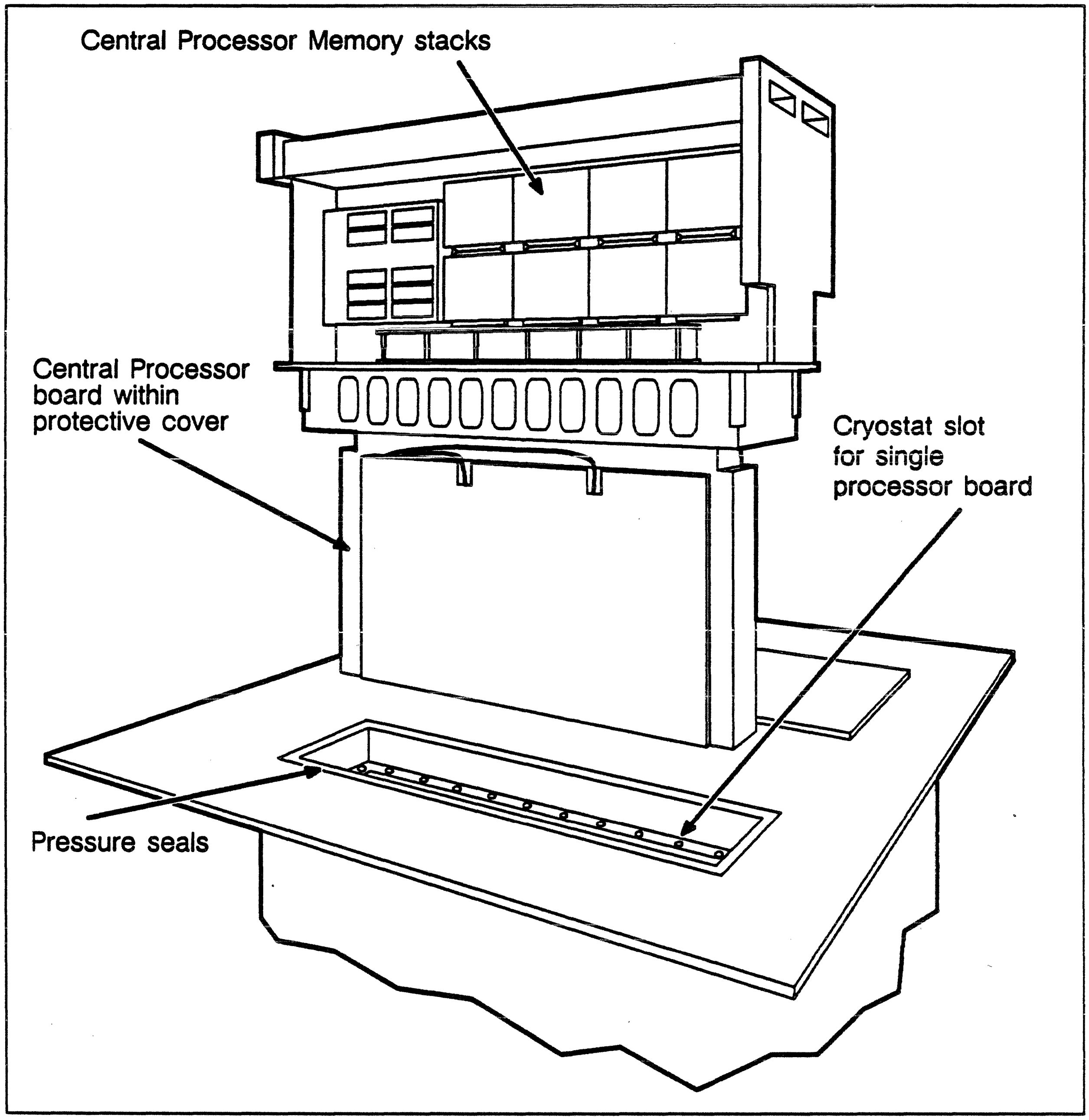

Central Processor Memory stacks
Central Processor board within protective cover
Cryostat slot for single processor board
Pressure seals

Figure 5–22. Central processor unit assembly for super-cooled operation; boards, interface, memory.

Central Processor and Memory Assembly

The central processor memory sits above the central processor boards. It is housed with the boards in the cryogenically cooled processor cabinet, the cryostat. The memory and the memory interface are sealed off from the liquid nitrogen bath in which the processor boards are immersed. Refrigerated air cools the memory stacks.

Virtual-to-Physical Addressing

When the central processor is running in job mode, its memory is virtually addressed; in monitor mode, memory is physically addressed. Addressing is accomplished by using a high speed mapping technique to convert a virtual address to an absolute storage address. This allows programs to appear logically contiguous to the user even though they are not physically contiguous in memory.

User programs provide virtual, logical addresses that the hardware maps to physical memory addresses. The associative unit contains 16 associative registers that are used for virtual-to-real address translation. These registers maintain a record of the most recently used virtual pages in memory, and are filled by the first 16 entries in the space table.

Page Size

Central processor memory pages are 2048 words (2K).

Space Table

The space table maintains a list of all pages in memory, each page described by an associative word. The associative unit searches the 16 registers in a single clock cycle and reads two space table entries per cycle. As shown below, the first 16 entries of the space table are also resident in the associative registers as associative words:

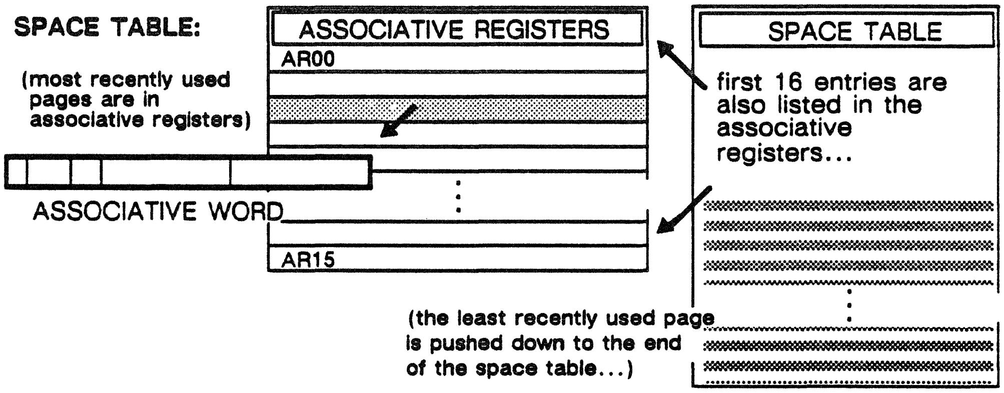

SPACE TABLE:
(most recently used pages are in associative registers)
ASSOCIATIVE REGISTERS
AR00
ASSOCIATIVE WORD
AR15
SPACE TABLE
first 16 entries are also listed in the associative registers...
(the least recently used page is pushed down to the end of the space table...)

Figure 5-23. Structures in the translation mechanism: associative words, the space table, the 16 associative registers.

Associative Word

One associative word is provided for each virtual page in memory. Figure 5-24 shows the associative word format:

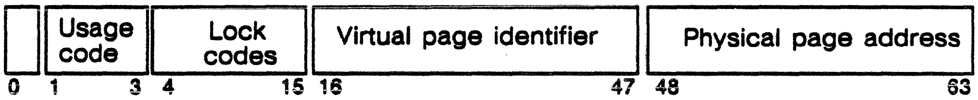


Figure 5–24. Format of the associative word.

The usage code indicates the page state or how a page has been used - if it has been read or modified, for example. Hardware sets this code according to how processes have used the page. If the page has been read or executed, the code indicates it has been referenced; if the page was written to, the code indicates the page has been modified. (This code also tells the software when and if a page is to be written out to a file; modified pages are prevented from being re-allocated until they are saved to a file.) One bit in the usage code indicates page size. The lock code indicates to which domain or domains the page belongs. To access a page, a process must hold the key matching a page's lock codes. (An attempt to access a page without a matching key results in the hardware issuing an access interrupt, the process may be terminated.) The physical and virtual addresses comprise the rest of the associated word.

If two associative words are found to exist for one virtual page, a multiple match fault occurs and the central processor stops.

Searching the Space Table

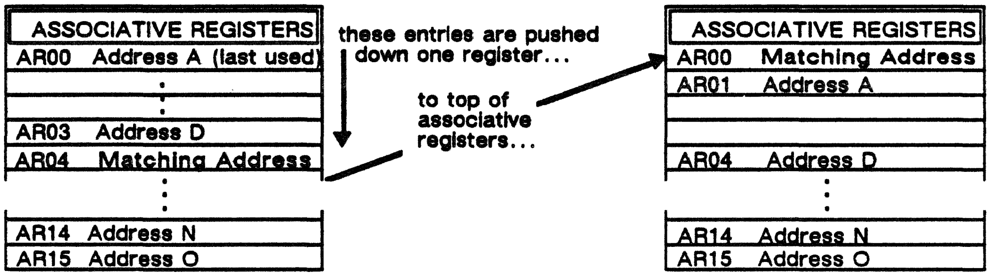
When a user process makes a page request, the associative unit searches the space table for the page's associative word by starting with a read of the associative registers. If the address is not found in the registers, the space table is searched at the rate of two entries per cycle until the match is found or until an end-of-table code is encountered. An end-of-table code ends the search. When the requested address is found, it is moved into the associative registers, and the entries ahead of it in the space table are rippled down (pushed down one). As a result, the space table maintains a list, in descending order, of last recently used pages with the least recently used page at the very end of the table.

Associative Registers

The sixteen associative registers are labeled AR00 through AR15, and are each one word in length. They are loaded from absolute bit addresses 4000_{16} through $43FF_{16}$, (word addresses 100_{16} through $10F_{16}$) by the LOAD ASSOCIATIVE REGISTER instruction. The registers are stored into the same absolute addresses by the STORE ABSOLUTE REGISTER instruction.

Reading the Associative Registers

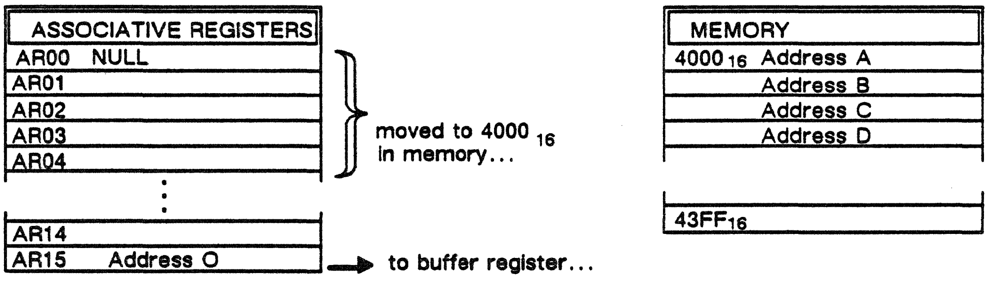
The registers are read in one cycle. When a virtual address match is found in the associative registers, the contents of the register containing the match is moved into an associative register as shown below:



The match may or may not be moved into AR00; this example assumes that it is. Simultaneously, the content of each register (except the new AR00) is pushed down one space to the next register. As a result, the 16 most recently used pages are always contained in the associative registers. When an end-of-table is encountered in the registers and no address match is found, the register contents remain unchanged and the hardware issues an access interrupt (an exchange to monitor mode occurs).

Reading the Space Table

When no match is found in the associative registers, the associative unit searches the space table using a ripple method to move space table entries into the registers to be read. The contents of AR15 are moved to a buffer register and a NULL (vacant location, no entry) is placed into one of the registers, pushing the registers down one. In the example below, the NULL is put into AR00, the contents of AR01 through AR04 are moved into memory to free register space for reads of space table entries. With the contents of the associative registers as shown below, entries from the space table are moved two at a time into the registers and read until a match or an end-of-table is encountered.



To start, the first associative word from the space table is moved into the registers and read; its place in memory is filled by the contents of the register buffer, once residing in AR15. If there is no match, word one is returned to memory, following old AR15, and the second word is brought into the register to be read. This method preserves the original order of associative words in the space table, and maintains the address from AR15 as its first entry.

After a match is found by reading the space table in this way, contents of registers that had been moved into memory are reinstalled into the registers.

Null Entries

A NULL entry indicates that the table or register location is vacant.

Two other conditions may occur during a space table search. One is when the matching address is not found in the associative registers but a NULL entry is encountered in the registers. When a match is found through clearing the registers and reading the space table as described above, the matching address moves from the space table to AR00 in the registers, with the NULL entry replacing it in the space table. There is no push down of space table addresses.

The second condition occurs when once more a match is not found in the associative registers but a NULL entry is found, this time, in the space table. If no match is found in reading the space table, the NULL moves to AR00. If a match is found, the matching address moves to AR00, and the NULL takes the match's old place in the space table.

Unsuccessful Search: Page Fault

When a match is not found in the associate registers and the space table is read until the end-of-table code is reached without finding a match, the entire space table is pushed down one position and a NULL is entered into one of the registers. If the unsuccessful search is initiated by a memory request in job mode, the hardware issues an access interrupt and an exchange-to-monitor mode is performed; the user's process has faulted for a page. An unsuccessful page search is reported to monitor, and the paging feature is called to continue the search and resolve the page fault.

Making the Virtual to Physical Translation

The associative registers translate virtual addresses into physical memory addresses. The top of figure 5–25 shows a virtual page address being compared to another page address in the associative registers; comparisons continue until a virtual match is found. Once a virtual match is made, the virtual-to-real translation starts. Fields in the page's associative word provide verification (lock), frame, and index information to obtain a fully translated physical address.

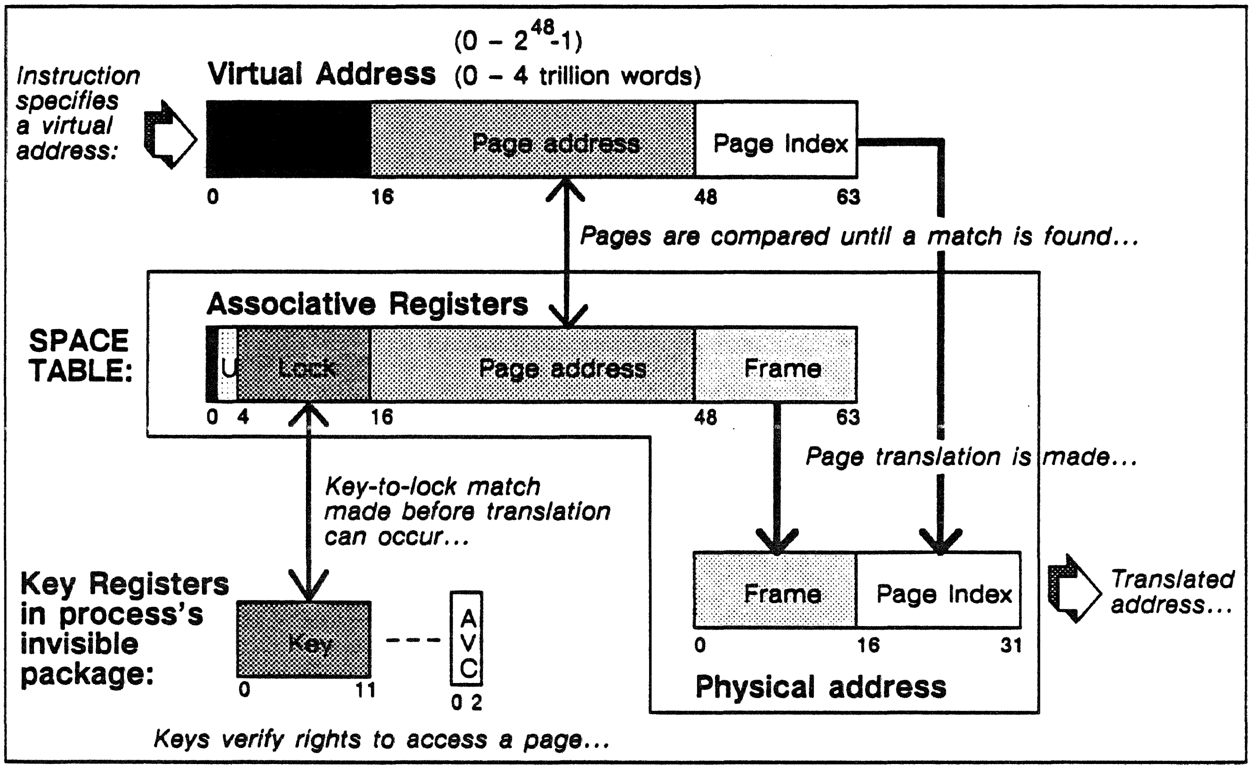

(0 - 2^48-1)
Virtual Address (0 - 4 trillion words)
Instruction specifies a virtual address:
Page address
Page Index
0
16
48
63
Pages are compared until a match is found...
SPACE TABLE:
Associative Registers
U
Lock
Page address
Frame
0
4
16
48
63
Page translation is made...
Key-to-lock match made before translation can occur...
Key Registers in process's invisible package:
Key
0
11
AVC
0 2
Frame
Page Index
0
16
31
Physical address
Translated address...
Keys verify rights to access a page...

Figure 5-25. Diagram of the virtual-to-physical addressing mechanism.

Verification of the user's right to access a page in memory is through the lock and key mechanism. Associated with each process is its invisible package, in the package are a set of key registers containing up to twelve keys. These keys specify domains in which the process has page access rights. Each key is linked to an access violation code (AVC) that specifies the type of access the process has for a page; read, write, and/or execute. The associative word describing a page in memory contains a set of lock codes. Accordingly, one of the process's key codes must match a lock code associated with each page. Hardware compares the page's lock code with the process's key code to verify the translation. Domains, keys, and locks are further described in the next section. The invisible package is described earlier in the "Monitor Mode and Job Mode" section.

Central Processor Memory Protection

A combination of hardware and software structures provide central processor memory protection and control user access to pages in memory. These structures include locks and keys, access rights, and a set of user and system domains.

Domains

A domain is the set of virtual memory pages unlocked by a process's current set of keys. Because a domain restricts software access to those pages, it provides protection against one domain violating the address space of another.

In job mode, the hardware accepts virtual addresses from a process and converts them into physical addresses. To perform this conversion, hardware requires a key matching the lock associated with the virtual page requested by the process. Hardware acquires keys from key registers in the process's invisible package. Hardware cannot access virtual pages for which a process does not hold a key; in this way, domains protect memory against illegal access and protect a process's data from other processes.

A process's address space is partitioned into one or more domains; a domain defines a set of virtual memory pages. A process executes in only one domain at a time, its current domain. A current domain consists of all virtual pages unlocked by the 12 keys currently held by the process. To expand or change a process's access to memory, instructions enable a process to change domains. For example, to use a library contained in the system domain, the process has embedded in its code a forward domain instruction. This instruction causes a domain change that substitutes new keys, the keys for the system domain, in the process's invisible package. New keys are acquired during a process switch or through a domain change.

The hardware acquires keys from the process's invisible package key registers and compares them to the virtual address locks. Software controls domain protection by establishing a lock on each virtual page, and by recording for each process their available domains and keys.

Additional information about domains is found in chapter 4, in the "Domain Management" section.

Locks and Keys

Each virtual page is identified with a lock code that protects a page from illegal accesses. The user process must contain the key that matches a page's lock in order for the process to have access rights to the page. Keys carried by a process define the area, or domain, of memory resources available to that process. Up to twelve keys are stored in the key register contained in the process's invisible package.

Access Rights

Access rights are linked to each key and control how the page is used by a process. If a process attempts a use without having the specific right, an access interrupt occurs; an exchange to monitor mode is made and the process may be terminated. Any of three permission codes are associated with each key:

- *read* permits virtual addresses to be read with the key.
- *write* permits the virtual address to be written with the key.
- *execute* permits virtual addresses to be executed with the key.

Protective Mechanisms at Work

This example shows a process legally requesting virtual address 619. Key 39 in the key register verifies address 619 is in the user's domain as are all virtual pages with a lock matching keys 39 and 43. Address 619 now moves to the top of the associative registers, pushing down the entries above it. (This process cannot access address 209 or 346.)

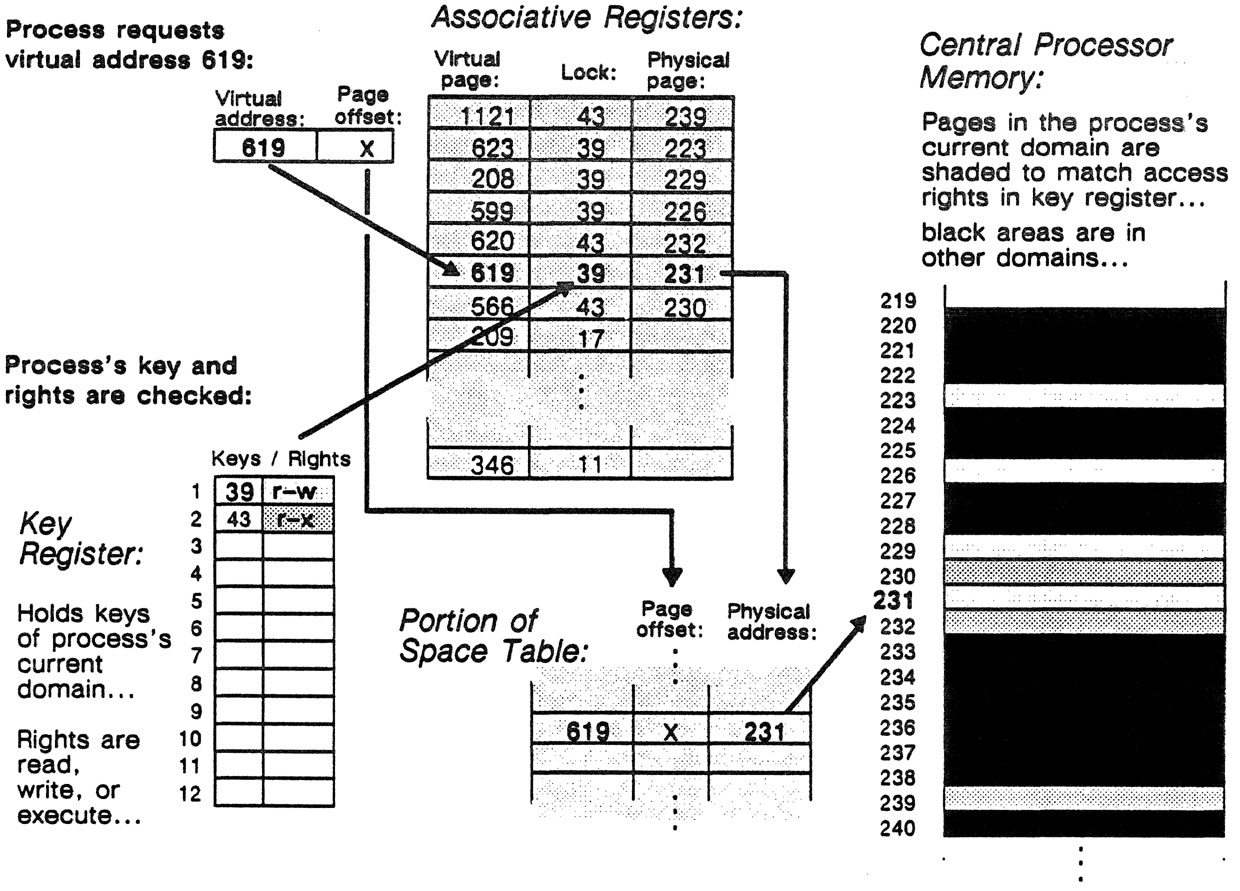


Figure 5–26. An example of virtual-to-real translation, including the protective locks and keys,

Domain Changes

A process may be allowed to use other domains and to move from one domain to the other to access previously protected pages. To move a process into a new domain, a FORWARD DOMAIN CHANGE instruction is executed to the desired domain. This type of instruction functions as a jump to an entry point in a new domain. As a result, the process's domain package is also changed and hardware loads 12 new keys from the altered process image. New keys change the set of virtual pages the hardware can access on behalf of the process. When resources in the new domain have been used, a backward domain instruction causes the process's original keys to be reinstated into its package, or the process may move forward into another new domain. A BACKWARD DOMAIN CHANGE instruction gives up new keys and restores the old, and the process returns to its prior domain.

A domain change is an efficient way to allow a process access to additional, required system functions without resorting to a process switch. Multiple domains are available to each process and the execution of a domain change instruction is simpler than a process switch.

Processes share virtual pages by sharing one or more keys. When a domain change causes the keys to be entirely replaced, the virtual pages of one domain are protected from operations of the process while it is in another domain. Keys may be shared or retained across domain changes by having the same key values in both domains.

Central Processor Memory Interface

The central processor memory interface provides memory access for the scalar and vector units as well as for the shared memory ports. Data is transferred to and from the memory ports in 32-bit half words, 64-bit words, or 512-bit blocks (8 words).

Data streams access central processor memory through ports. Each port moves data to or from memory with separate controls for addressing and data movement. Data is transferred to and from the memory ports in 32-bit half words, 64-bit words, or 8-word blocks. The memory supports five ports, one port may be active each clock cycle. Three ports, Read 1, Read 2, and Write 1, support the vector pipelines. Two other ports, Read 3 and Write 2, support the scalar pipelines and shared memory requirements.

Single error correction-double error detection (SECDED) is generated and checked by the memory interface for data within central processor memory. Write 2 checks parity on data coming from the shared memory port. Read 3 generates 1 parity bit on transfers to the shared memory port.

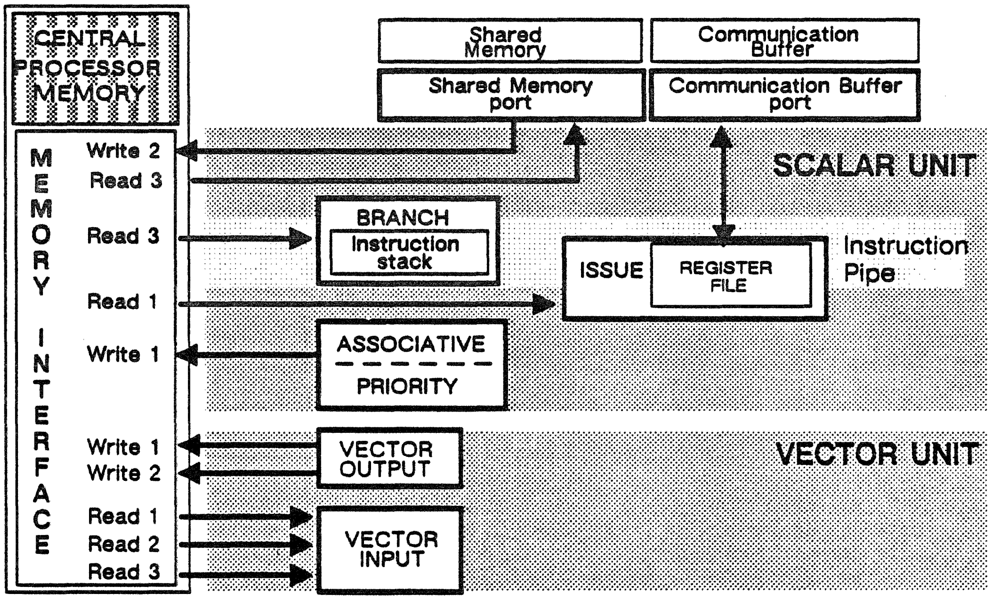


Figure 5-27. Components that read and write to the processor memory interface.

Write Ports

The memory interface provides two write ports. Write 1 receives write data from the scalar and vector units, Write 2 receives data from the vector unit and from shared memory. Write 2 also performs parity checks on the read data, and provides one parity bit for each 32 bits of data transmitted to the shared memory port. Both write ports generate their own check bits for SECDED, and provide logic to write both data and check bits into the appropriate location in central processor memory.

Read Ports

The memory interface has three read ports, each is 128 bits in size. Each read port performs its own error correction (SECDED) for each 32 bits of read data. Read 3 is used to fetch instructions. Scalar load data moves through Read 1. Vector instructions use Read 1 and Read 2 primarily; Read 3 is used for control vector access and for shared memory reads.

Memory and Signal Transfers

For a write request, write data is transmitted to the central processor memory stack on the next cycle following the stack request.

Each central processor reference busies its required bank for seven clock cycles. The minimum memory read or written is a half word, 32-bits; the maximum is eight consecutive words, or 512 bits. In an eight-word transfer,· the first word taken from or sent to memory comes from an address with 000 in the lowest part of the word address.

Shared Memory

Shared memory provides major storage resources for all system processors. It is the repository for work as it comes into the system, executes among the processors, and then leaves the system. Super-cooled ETA10 systems have from 32 to 256 million words of shared memory in one or two units.

Structure and Organization

A redundant ETA10 system has two units of shared memory, and both are housed in the shared memory (SCMI) cabinet. The two units have independent memory and memory control logic, though each unit can work with the other unit to coordinate the access ports.

Non-redundant ETA10 models have one unit of shared memory, a shared memory interface, and an I/O interface.

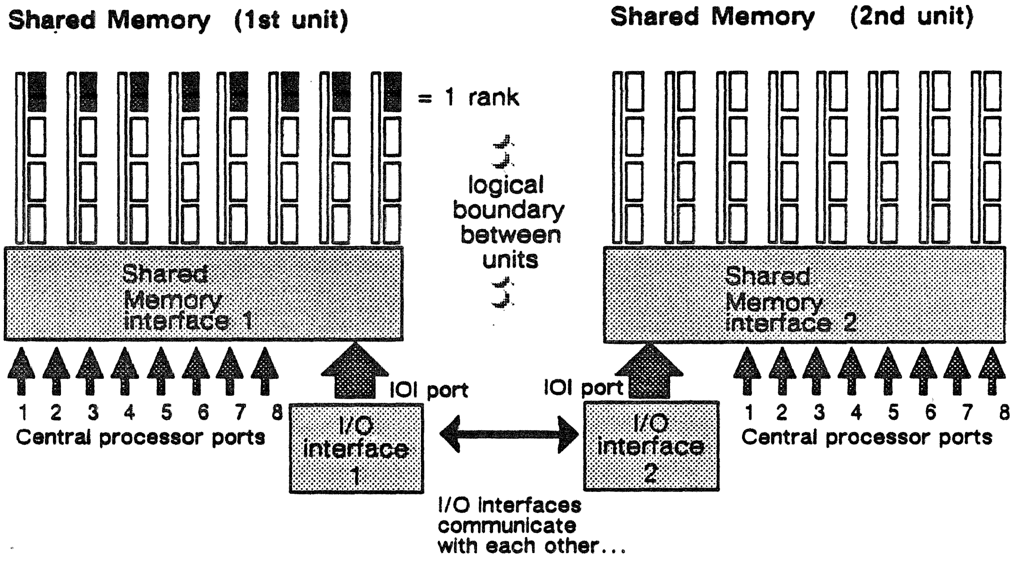


Figure 5-28. Shared memory units in a redundant system.

A logical boundary separates one set of ranks from the other. This is significant because a data request may only specify data located on one side of the boundary, in one shared memory unit or the other.

Size

Shared memory is configured in increments of 32 million words: the minimum size is 32 million, maximum size is 256 million words.

Access

Each unit of shared memory can perform five simultaneous transfers to central processors at the rate of one 64-bit word per processor cycle.

Addressing

Addressing is on half word boundaries. The minimum transfer unit size for central processor ports is one half word. Addressability and minimum transfer unit size for the I/O and service unit ports is one 64-bit word. Two 64-bit words transfer from the shared memory interface to the I/O interface per shared memory interface cycle.

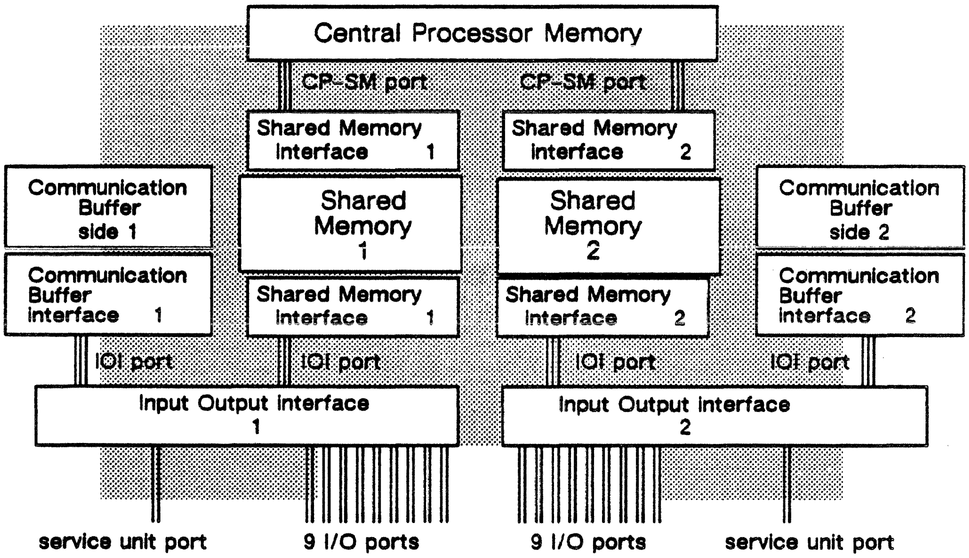

Central Processor Memory
CP-SM port
CP-SM port
Shared Memory Interface 1
Shared Memory Interface 2
Communication Buffer side 1
Shared Memory 1
Shared Memory 2
Communication Buffer side 2
Communication Buffer Interface 1
Shared Memory Interface 1
Shared Memory Interface 2
Communication Buffer Interface 2
IOI port
IOI port
IOI port
IOI port
Input Output Interface 1
Input Output Interface 2
service unit port
9 I/O ports
9 I/O ports
service unit port

Figure 5–29. Central processor and I/O access to shared memory.

As shown in figure 5–29, each shared memory unit provides eight central processor ports and one port to the I/O interface. Each I/O interface provides nine ports for I/O units and one service unit port. The I/O interfaces communicate with each other and with their paired shared memory interface. Shared memory interfaces do not communicate with each other. Two ports enable central processors to access both shared memory units. An I/O unit connected to I/O interface 1 can access the second shared memory unit through I/O interface 2.

Error Detection and Correction

The shared memory interface reports these types of status errors:

- double SECDED errors
- write data parity errors

- address parity errors
- address bounds error

The requesting unit reports read data parity errors.

SECDED Errors

Single error correction-double error detection (SECDED) is on each 32-bit half word. Information for SECDED is generated by the shared memory interface for each 32-bit half word of shared memory.

SECDED syndrome codes are checked on all reads of data from shared memory. If a single bit error occurs, the bit is corrected and the correct data is sent to the requester. If a double bit error occurs, the data cannot be corrected. The bad data is sent to the requester with a fatal error signal (the SECDED code tagged). The error is reported to the service unit along with the syndrome code via the maintenance interface. Both a fatal error and a single SECDED error may be reported simultaneously because multiple words per clock period are being read from shared memory.

Central Processor or I/O Interface Transfer Hardware Errors
These faults include double SECDED errors in shared memory, and data or address parity errors between the central processor or I/O interface and the shared memory interface. Such a fault indicates either a central processor or I/O interface hardware failure, a shared memory interface hardware failure, a shared memory hardware failure, or a reference to a shared memory location containing a double SECDED fault written by the hardware in response to a prior data parity error. A SECDED error could occur outside the area of shared memory that was read because the SECDED check is enabled on 128-bit boundaries.

A data parity error detected by the shared memory interface causes the hardware to write a double bit error such that the next read of that area gets a double SECDED fault.

Parity Errors

Parity checking is performed on all data and address transmission paths to and from the shared memory interface ports. This includes shared memory-central processor transfers and transfers to the I/O interface.

Data Transmission Parity Errors The port interface paths between processors or the I/O interface and the shared memory interface are all four half words in size and parity is checked on each half word. When a parity error occurs, a fatal error is reported back to the requesting unit. The corrupt data on a write to shared memory is written into shared memory with bad checkbit code. Reading this data causes a fatal SECDED error to be sent to the requesting port. This protects other processors that may read this incorrect data.

Address Transmission Parity Errors Parity is checked on the 30-bit address field transmitted to the shared memory interface. When an address parity error occurs, it is reported back to the requesting unit and there is no alteration of data in memory. The error address is also reported to the service unit via the maintenance interface.

Address Boundary Errors

A boundary error occurs when the shared memory interface detects a request made to an invalid area of memory (an inactive shared memory rank), or if a request attempts to transfer data across the boundary between shared memory units.

When a boundary error occurs, it is reported to the requesting unit. If the error is on a read of the shared memory from a central processor, the data sent to the requesting unit is all zeros. This error is reported to the service unit.

This fault indicates either a hardware failure in reading the transfer request block, a software failure in constructing the transfer request block, a central processor hardware failure, or corrupt system tables due to hardware or software failure.

This fault may also indicate a runaway transfer (a failure in the central processor or shared memory interface). The shared memory interface works with the current memory address and the central processor sends a terminate signal. If the terminate is missed, or not sent, then shared memory is destroyed out to the end of contiguous shared memory.

Service Unit–Shared Memory Transfers

The service unit accesses shared memory through an I/O port. When the service unit requires a communication link to communicate with shared memory, a connection is made through the service unit port on the I/O interface as shown in figure 5-29.

Service Unit to Shared Memory Reads and Writes

The service unit port in the I/O interface transfers data to and from shared memory. When the service unit tries to read data from shared memory that crosses the boundary between memory units, the service unit port does not respond. If the service unit tries to write data across the boundary, the service unit port does not initiate the service unit transfer. The same result occurs when the service unit attempts to read or write to an inactive memory rank. A status bit indicating a boundary error is always returned if any of the above occur.

I/O Unit–Shared Memory Transfers

I/O Operations

A data pipe controller in each I/O unit connects that unit to the data pipe and controls the flow of data to and from the data pipe. The data pipe connects the I/O unit to its port on the I/O interface. I/O transfers are temporarily stored in the I/O channel's data pipe buffer to await transfer across the data pipe.

When the data pipe buffer in an I/O channel processor wants access to the I/O unit's data pipe controller, it makes a bus request to the data pipe controller. The data pipe controller arbitrates bus requests using a round-robin method and grants use of the bus to a data pipe buffer. The data pipe buffer then issues a request header instructing the data pipe controller to execute an I/O operation.

I/O Interface Ports

Each of the 20 I/O interface ports shares access via two dedicated slots to both units of shared memory. (In non-redundant systems, ten I/O interface ports share access via one dedicated slot to shared memory.) All 20 ports can access the same shared memory unit. The I/O interface ports have only one access restriction in the shared memory address space; a single data transfer cannot cross the boundary between the units of the shared memory.

I/O to Shared Memory Read Operation

The shared memory read operation starts when the data pipe buffer issues a read shared memory header to the data pipe controller. The data pipe controller examines the header, then transfers data from the shared memory in a series of smaller messages to match the I/O interface port buffer size (512 byte buffers). The data pipe controller passes the data to the data pipe buffer as it is received over the data pipe. The data pipe controller updates status between each block of data by writing status information to the data pipe buffer.

The shared memory read operation terminates when the number of half words specified in the receive length field of the header have been transferred from the shared memory to the requesting data pipe buffer.

I/O to Shared Memory Write Operation

The shared memory write operation starts when the data pipe buffer issues a write shared memory header to the data pipe controller. The data pipe controller examines the header, then subdivides the write operation down into a number of smaller writes to match I/O interface port buffer size (512 bytes). The data pipe controller reads data from the data pipe buffer with each of these data transfers and passes it to the data pipe. The data pipe controller updates status between each block of data by writing status information to the data pipe buffer. The shared memory write operation terminates when the number of half words specified in the transmit length field of the header has been transferred from the data pipe buffer to the shared memory.

Data transfers to and from the shared memory start on a full 64-bit word boundary. The transfer length is an integral number of full words. Retries exist only for shared memory write operations from the I/O unit.

The Shared Memory Stack Assembly

A single shared memory stack assembly provides two million half-words of memory. Memory stacks are put together on shared memory boards called maxiboards. Each maxiboard functions independently, and holds stacks providing 4 to 16 million words.

A stack assembly consists of one input board, eight memory boards, one output board, and two edge boards. The input, memory, and output boards are interconnected via connectors to the edge boards, forming a three-dimensional stack assembly.

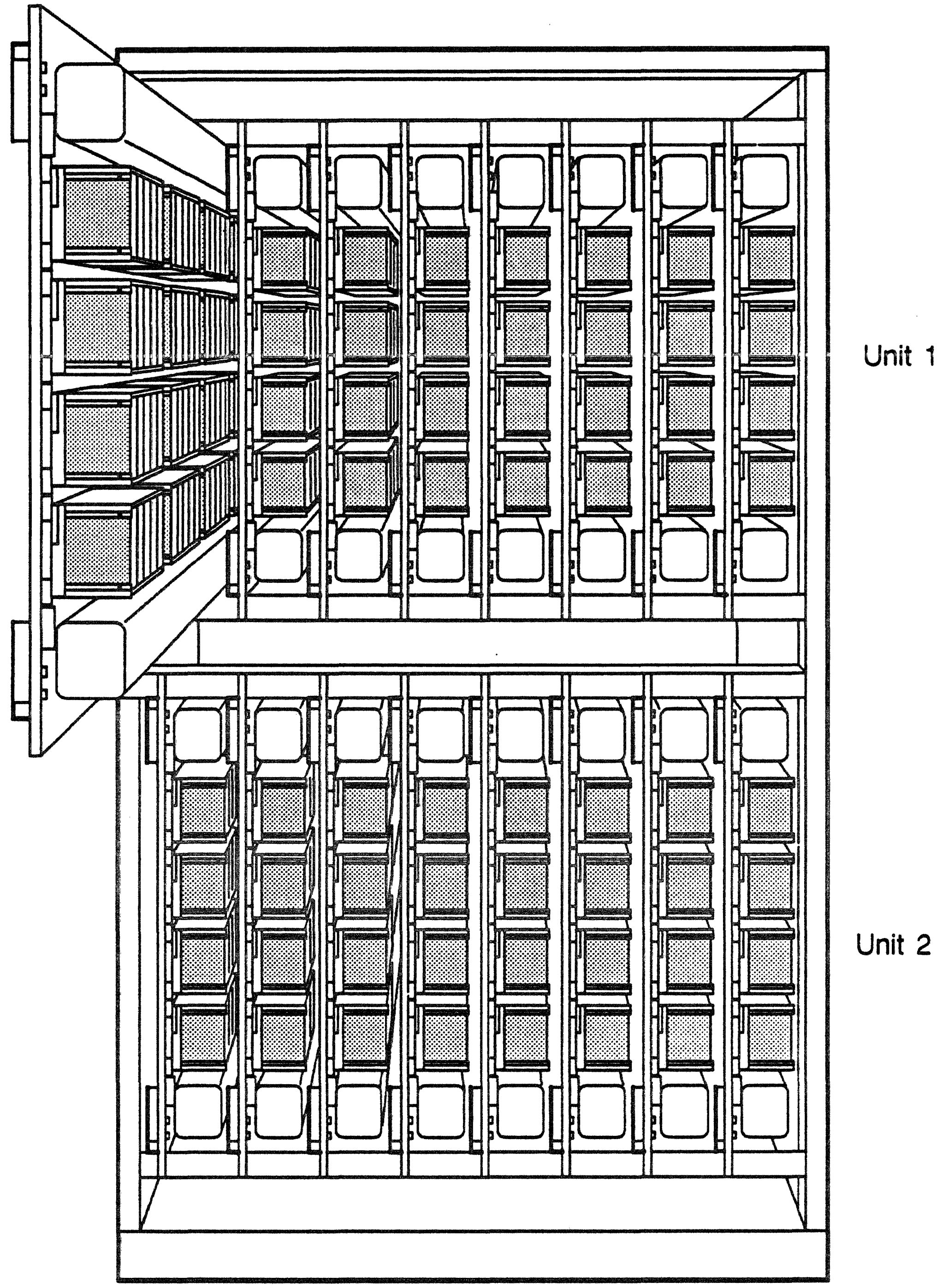


Figure 5-30. Two shared memory units housed in the SCMI cabinet.

Figure 5-30 illustrates two units of shared memory in the cabinet, one unit in the upper half, the second unit in the lower half. Each unit has eight shared memory maxiboards. The boards are vertical, hung from a slide assembly that moves the board out for any maintenance.

Shared Memory Interface

The shared memory interface provides the communication link between the shared memory and the memory interface's nine high-speed ports. Eight ports are reserved for central processors, one is dedicated to the I/O interface port. Data is transferred in blocks between shared memory and the I/O units and service unit. Data is transferred in pages between central processor memory and shared memory.

The requesting unit initiates and controls transfers between shared memory and central processor memory, and between shared memory and an I/O unit. The shared memory interface provides protective lockout features to validate and control access to the memory.

Single error correction-double error detection (SECDED) is generated and checked by the memory interface for data within shared memory. The shared memory interface checks parity on data coming from the central processor memory and generates one parity bit on transfers to central processor memory.

Hardware Characteristics

The shared memory interface board is manufactured in the same manner as the central processor board. It uses CMOS chips and is air-cooled. It is housed in the shared memory SCMI cabinet in the SMI frame.

Transfer Requests to the Interface

The I/O interface and shared memory ports in the central processors initiate and terminate transfer requests to the shared memory interface. When a request is initiated, the shared memory interface is signaled as to the impending transfer. A transfer terminates when field length counters in the I/O interface or shared memory port indicate that all requested data is transferred. The initiating port sends a terminate signal to cancel further transfer of data.

Shared Memory Interface Error Reporting

Error information is returned to the I/O interface and central processors for write and read operations. For write operations, information on address parity errors, write data parity errors, and address boundary errors is returned. For read operations, information on address parity errors, read data multiple SECDED errors, and address boundary errors is returned.

Error information is returned to the service unit via the shared memory interface maintenance interface for read and write operations.

Shared Memory Writes

As the write operation starts, the requesting port sends write data, parity, and write enables on a 136-bit path to shared memory. Before the data is stored in memory, a parity check is done on each 32-bit half word and seven bits of SECDED information is generated for the half word. Each of the four 32-bit half words on the data path is then sent with its write enable bit over a single 160-bit trunk to shared memory where it is stored.

A write data transfer, once started, must access consecutive addresses of memory for field lengths from one half word to 65536 64-bit words.

Shared Memory Reads

As the read operation starts, a data move signal is sent back to the requesting port. This signals the port that read data and parity are started from the shared memory. The read data and parity travel on a 160-bit path from shared memory. After the data is read from memory, a SECDED check is done on each 32-bit half word, and parity information is generated for the half word. Each of the four 32-bit half words on the data path is then sent with its parity bit on a single 132-bit trunk to the requesting port.

A read data transfer, once started, must access consecutive addresses of memory for field lengths from one half word to 65536 64-bit words.

Access Slot Operation

The shared memory interface has eight central processor ports which compete for five access slots to one unit of shared memory. A priority mechanism handles conflicts when a number of central processor requests arrive at the shared memory interface simultaneously. A redundant system with two shared memory interfaces provides ten access slots for up to eight central processors. If transfers are directed to both units of memory, access to memory is enhanced.

A maximum of five transfers to central processors can occur at one time through one shared memory interface. If five central processor requests are active, requests from other central processor ports must wait until a slot becomes available.

Priority is arranged from port 1 to port 8 in descending order. When more than one request arrives at the same time, request status for all involved ports is frozen. Shared memory interface then processes the requests by the priority scheme until all requesting ports have been serviced. Requests from other ports during the freeze time are recorded but are not allowed access to the priority network until the original group has finished.

I/O interface transfers take place in a dedicated slot, and are unaffected by central processor slot operations.

Reserving an Access Slot

When a central processor's shared memory port makes a request to the shared memory interface, the signal is recorded in a register during the request set-up period. The request is not set up until a slot is available. Set-up controls read the starting address to determine which logical block of data in shared memory the central processor requires first. The slot which will have access to this block of memory first is reserved for the requesting processor if the slot is available. If the desired slot is not free, the next closest free slot is chosen and reserved.

Making the Transfer

Once a slot is reserved for a shared memory request, as an example, the write data path from the central processor is connected to the reserved slot's data path. The slot rotates through each logical block in succession until it reaches the block containing the starting address. When the slot has reached the access point to start the transfer, a data move signal is sent to the central processor's shared memory port indicating the shared memory interface is ready to receive write data. The processor forwards write data as the memory interface sends it to consecutive addresses until a terminate signal arrives with the last cycle of data. The shared memory interface releases the slot to other central processors.

Validating the Transfer

After a requesting port has set up a request in a slot for data transfer, several checks are done to detect operational errors. An address parity check is done on the starting address for reads and writes to shared memory. For writes to shared memory, data parity is checked and SECDED check bits are generated for each half word. For data read from shared memory, SECDED is checked for errors. Boundary errors are reported if a request attempts to write to or read from an invalid area of memory, or attempts to cross to the address space of the other shared memory unit. Parity is generated for the read data transfer.

Central Processor Ports

Every central processor has two shared memory ports. A central processor may not transfer to more than one shared memory unit at a time; the two ports are not active simultaneously.

Transfer Rates

Field lengths for shared memory interface data transfers are held in the central processor's shared memory port. These field lengths may be from one half word (32 bits) to 65536 64-bit words. Once the transfer is started, the continuous transfer rate is one 64-bit word per central processor minor cycle. Two 128-bit wide data paths, one read and one write, run at half the central processor cycle rate between the central processor and the shared memory interface. This is an effective data rate of one 64-bit word per clock cycle; the rate is continuous for the entire transfer.

I/O Interface Access

The I/O interface services data transfers between shared memory and the service units and I/O units. One active I/O interface read or write request is allowed at one time to shared memory. All I/O ports can be moving data simultaneously; there are data buffers that enable the I/O requests to wait in turn for access to shared memory without any loss of bandwidth due to conflicts. I/O interface transfers can be issued to the shared memory interface every 64 minor cycles.

I/O interface transfers take place in a dedicated shared memory interface slot unaffected by demands at the processor slots. The I/O interface is synchronized with the dedicated slot so that requests fall within the logical address space currently available to the rotating slot. As requests come, they are organized in the I/O interface according to the order of logical block locations in shared memory. Requests are serviced in order as the slot rotates.

Transfer Rates

I/O interface requests to shared memory are a minimum of one 64-bit word. Since the shared memory interface has no field length counters, there is no real limit on the maximum request length. In general, 64 64-bit words are the most data transferred at one time.

I/O interface transfers may be issued to the shared memory interface every 64 minor cycles. The shared memory interface can transfer two 64-bit words of read or write data to the I/O interface every two minor cycles. This rate may be maintained continuously if there is sufficient demand for transfers from the I/O units and service units.

Communication Buffer

Communication buffer is a memory accessed by all system processors; it is used for certain data transfers, and primarily for transmission of high-speed synchronization messages and signals among the system processors. The communication buffer provides communication between all central processors, I/O processors, and the service unit. Its functions are to coordinate activities among central processors, perform data fetching and storing, and maintain system tables.

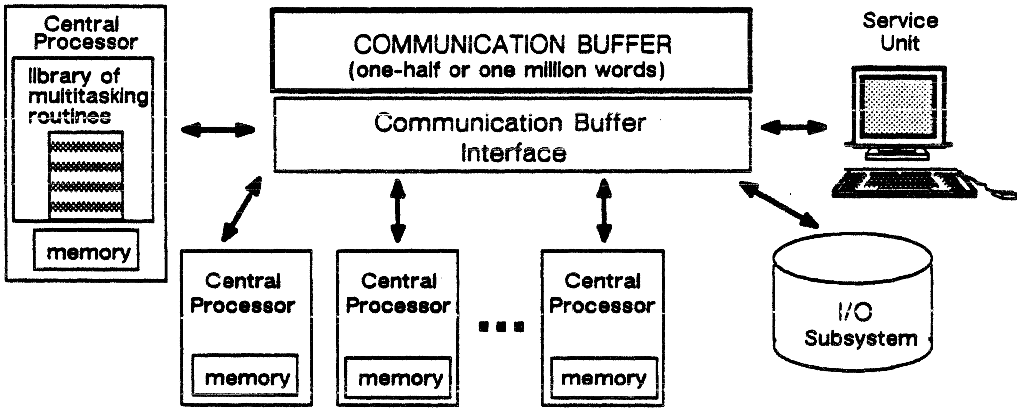


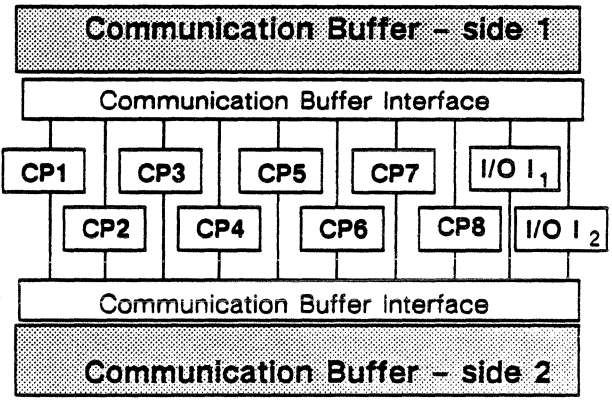
Figure 5-31. Processors/components connecting to the communication buffer.

Structure and Organization

The communication buffer is structured in discrete, independent units. In a redundant system, there are two units of communication buffer (one million words) and two communication buffer interfaces. Between units is a boundary. When there are two units, they are usually called side 1 and side 2. Each side is one-half million 64-bit words. To talk to each other, sides must go through the I/O interface or through a central processor. (For example, in a redundant system, communication buffer units communicate with each other to maintain duplicate system files.)

Each side has its own communication buffer interface that connects the communication buffer to other components in the system.

Each side also has eight ports for central processors, and two I/O interfaces for I/O units and the service unit.


Communication Buffer - side 1
Communication Buffer Interface
CP1
CP3
CP5
CP7
I/O I1
CP2
CP4
CP6
CP8
I/O I2
Communication Buffer Interface
Communication Buffer - side 2

Addressing

All addressing is absolute, there is no relative addressing.

Access time is about 64 clock cycles with no contention, and up to 128 clock cycles with full contention.

Bandwidth

Maximum bandwidth at the communication buffer memory is one word per two minor cycles (or one major cycle).

Word Transfers

LOAD, STORE, BIT BRANCH AND SWAPS, and BIT BRANCH AND LOAD/STORE instructions can operate on full word or half word operands. Addressing for full word operations must be on word boundaries and addressing for half word operations must be on half word boundaries.

POST SEMAPHORE and WAIT SEMAPHORE instructions operate only on full word operands. These instructions must be addressed on word boundaries.

Error Detection and Correction

Address Transmission Parity Errors

Parity is checked on each half word of the control word transmitted to the communication buffer. If a parity error occurs, the error is reported back to the requesting unit at terminate time as a fatal error. There is no alteration of data in the communication buffer. If the error is on an instruction that would return data, the data sent to the requesting unit is all zeros. The error is reported to the service unit via the maintenance interface for the communication buffer.

Data Transmission Parity Error

Parity is checked on each half word transmitted to the communication buffer. If a parity error occurs, the error is reported back to the requesting unit at terminate time as a fatal error. The corrupt data is written into the memory with the upper three bits of its SECDED code complemented. This causes a fatal SECDED error when the corrupt data is read out of memory. If the data parity error is on an instruction that would return data to the requester, the data returned is all zeros. The error is reported to the service unit via the maintenance interface for the communication buffer. For semaphore or bit branch instructions, a branch signal is returned to the requesting central processor.

Communication Buffer SECDED Errors

Seven bits of SECDED code are stored with each 32-bit half word of data stored in the central processor memory, shared memory, and communication buffer. The SECDED code gives the memories the ability to detect and correct all single-bit errors and to detect all double bit errors.

SECDED code is checked on all reads of data from the communication buffer. If a single SECDED error occurs, the error bit is corrected and the correct data is then sent to the requester. If a double SECDED error occurs, then data cannot be corrected. The bad data is sent to the requester at terminate time along with a fatal error signal. The error is reported to the service unit via the maintenance interface for the communication buffer. Both a fatal error and a single SECDED error may be reported simultaneously.

Address Bounds Error

The central processor checks for access lockouts and bounds errors when accessing the communication buffer in job mode. Checks for these faults are done on the address sent with the instruction and the address of the process word sent with a WAIT SEMAPHORE instruction. When the central processor detects either of these errors, an Address Bounds Error signal is transmitted to the communication buffer with the failing instruction.

An access lockout error occurs when execution of a communication buffer instruction violates communication buffer access privileges. This fault indicates that either central processor hardware has failed, a software failure has occurred in system domain code, or that system tables are corrupt due to hardware or software failure.

A bounds error occurs when execution of a communication buffer instruction references an area of communication buffer outside that specified by the base/limit access pair.

When the communication buffer receives an Address Bounds Error signal, there is no alteration of the data in the communication buffer. Data of all zeros and a Read Data Full is returned to the requesting central processor on all communication buffer instructions except for STORE instructions. A Branch signal is sent to the requesting central processor for semaphore and bit branch instructions.

The communication buffer's I/O ports operate only in monitor mode and thus do not check for access or bounds errors.

I/O Interface Ports on the Communication Buffer

There are two identical I/O interfaces. Each I/O interface contains 10 ports with a dedicated slot to each communication buffer interface. To improve performance, the I/O interface is capable of sending multiple commands to communication buffer at one time. A set of 16-word (64 bits per word) communication buffer command registers is provided in each I/O interface port. In addition, a set of 16-word registers holds the responses to the communication buffer commands; the response message contains the response to all communication buffer commands issued in the previous communication buffer command message.

Communication Buffer Command Buffers in the I/O Interface

Communication buffer commands stacked in the communication buffer command buffer can cross the boundary between the two sides of the communication buffer.

Each I/O interface services the communication buffer command buffers of its port in a round-robin fashion. Once execution begins, all communication buffer commands in that buffer are processed before service is passed to the next buffer. The I/O interface port cannot add to its communication buffer command buffer until a response message that contains the results to all of the communication buffer commands posted in the requesting message has been returned to the source.

I/O Operations to the Communication Buffer

Commands are sent to the communication buffer when the data pipe buffer in the I/O unit issues a communication buffer header message to the data pipe controller. The header message specifies the number of half words in the communication buffer command data field. The minimum value is two half words, which equals one communication buffer word; the maximum value is 32 half words. The data pipe controller reads the communication buffer command data field and sends it to the I/O interface. The I/O interface issues the commands to the communication buffer and returns the communication buffer responses back to the data pipe controller which writes it into the data pipe buffer memory.

Transfers from the Service Unit

The service unit access to the communication buffer is the same as a normal I/O channel. When the service unit requires a link to communication buffer, it is made through the I/O interface -communication buffer interface connection.

Communication Buffer Instructions

During multiprocessing, the communication buffer synchronizes the operations of the central processors, and controls their access to small amounts of data which they need to share. Communication buffer instructions support these activities by enabling:

- word and half word transfers to and from the communication buffer and the central processor's register file
- semaphore post and wait primitive operations
- conditional word and half word swaps from the communication buffer to the register file
- conditional test and sets with word and half word loads from the communication buffer to the register file

Descriptions of these instructions are included in the "ETA10 Instruction Set" section later in this chapter.

Communication Buffer Interface

The communication buffer interface manages all of the communications that take place via the communication buffer, and directs the coordination of activities between the central processors. The interface provides system-wide communication and exchanges between central processors, I/O processors, and the service unit processors.

Hardware Description

Communication buffer is built with the same CMOS chip and board technology as that of the central processors. An air-cooled component, it is housed in the shared memory SCMI cabinet.

The interrupt network logic is on the communication buffer interface board. Through the interrupt network, all central processors, I/O processors, and service unit processors can send and receive interrupts to and from every other processor. The interrupt network is described earlier in this chapter in the "Central Processor Architecture" section.

Section 4: The ETA10 Instruction Set

This section is an introduction to the set of ETA10 machine instructions. It describes the instruction formats, and provides a list of the instructions grouped according to their function. At the end of the chapter is a discussion of scalar and vector instructions.

Background

Iverson's *APL, A Programming Language,* was the original philosophical and design basis for the ETA10 instruction set. The use of vectors and vectorization is inherent in APL; this feature was adopted into the ETA10 instruction set. This emphasis on vectors is implemented in the set's many vector instructions, instructions tailored to the requirements of processing streams of vectors. As a result of APL, this instruction set is designed around vector processing rather than extended to accommodate vector processing.

Characteristics

The ETA10 instruction set is model independent, and is compatible with Control Data Corporation's CYBER 205 instruction set. It has 256 function codes, 40 of which are unused.

The set incorporates many three-operand instructions in which two source operands and one result operand may be specified in one instruction. Three-operand instructions are important because fewer instructions are issued to obtain the same result. Execution is efficient, and processing performance improves.

Additional options for some instructions give added functionality to operations. One instruction, for example, can take all the A input operands and make them all negative.

Or, one instruction can add A and B with the option to complement all the B operands,

Another variant of the same instruction can add A and B with the value of B a constant. In a 20 by 20 matrix where all numbers are to be multiplied by 2, one instruction sets B as a constant in the register file to be used as a source operand with each of the 400 matrix elements (instead of duplicating 2 the 400 times needed to multiply each member of the matrix).

The set includes instructions for floor operations that find the nearest integer that is less than the present integer, as well as instructions for ceiling operations that find the next greater integer than the present integer.

There are instructions that reorder data. A vector organized into rows can be reorganized into columns.

The richness of this set is extended by the inclusion of a subfunction field in several of the instruction formats. The subfunction field enables the function of those instructions to be further specified. Depending upon the basic function field, the subfunction field can give the instruction additional refinement. For example, the BIT BRANCH AND ALTER instruction has a subfunction field containing four modifiers, one of which can be appended to the basic branch function of that instruction. Another four G-bits are available to specify operations upon the object bit. These permutations expand the richness of the ETA10 instruction set.

In summary, this instruction set offers flexibility and extended functionality to programmers.

Instruction Formats

There are 13 instruction formats; six of them require a full word, seven fit into a half word. Each instruction contains three or more fields, or designators, that are one or more bytes in length. The F designator is the first byte in all instructions, and determines the basic function of the instruction. The instructions use the remaining fields in a variety of ways.

All fields are 8 bits unless otherwise specified.

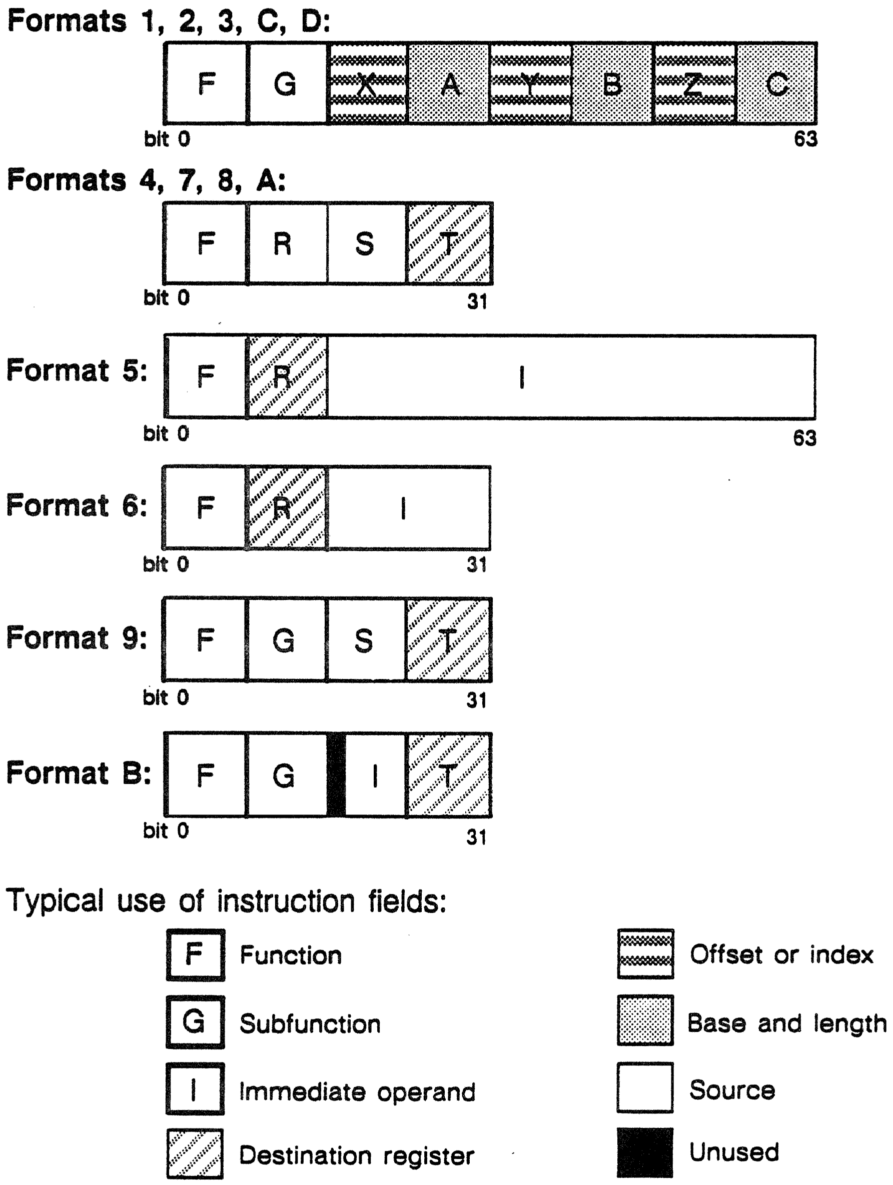


Figure 5–32. Formats for the ETA10 instruction set.

Vector Instruction Format

F	G	X	A	Y	B	Z	C
Function code	Subfunction code	address offset for source operand A	base address & field length of source operand A	address offset for source operand B	base address & field length of source operand B	base address of control vector	base address & field length for the result operand C+1

Figure 5–33. General format for vector instructions.

Vector instructions are in format 1 as shown above. Vector operands are specified in terms of registers pointing to their location in memory. The 8-bit designators in the instruction fields are as follows:

F specifies the instruction's function code

G specifies the instruction's subfunction code

X specifies the register that contains the address offset for source operand A

A specifies the register that contains the base address and field length of source operand A or, in the case of a broadcast constant, holds the operand.

Y specifies the register that contains the address offset for source operand B

B specifies the register that contains the base address and field length of source operand B or, in the case of a broadcast constant, holds the operand.

Z specifies the register holding the base address of the control vector

C specifies the register holding the base address and field length of the memory location where the result goes or, for vector instructions returning single results, the results.

C+1 is the register that contains the offset for both the control vector and the result field (usage of C+1 depends upon bit 2 of the G field; if C+1 is used, C must be even)

Register Instruction Format

F	R	S	T
Function code	source operand A	source operand B	destination of result operand

Register instructions specify registers that actually hold the source operands and are destinations for the result operands (except for load/store instructions).

Figure 5–34. General format for register instructions.

F specifies the instruction's function code

R specifies the register containing source operand A

S specifies the register containing source operand B

T specifies the result destination for the register operand

Addressing

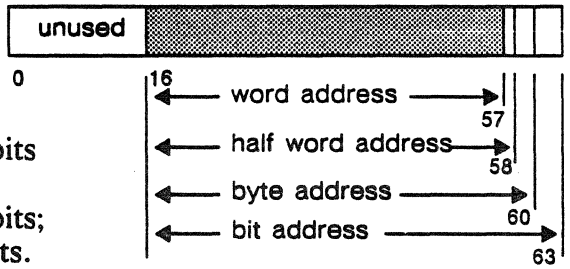
The ETA10 is a bit-addressed machine. All addresses are 48-bit quantities and contain enough information to reference a specific bit. Bits, bytes, half words, words, and pages are always addressed from left to right; the first bit of the addressed entity is the left-most bit of that entity. While the processor is in job mode, those bits are mapped through the virtual-to-physical addressing mechanism. In monitor mode, the addresses are real and the upper bits are ignored. Addressing is in modulo with respect to the 4 million address limit in central processor memory.

Bit-Level Addressing

The lowest bit in an address can specify a particular bit in memory, and so lower bits are reserved for bit addressing. All addresses in the ETA10 are bit level except indexes and offsets that address to the word, half word, or byte level. As shown below, groups of bits address various units of storage:

The numbers indicate the bit position in a register or in an instruction word.

A full word has the lowest 6 bits of an address set to zero;
a half word has the lowest 5 bits;
and a byte has the lowest 3 bits.



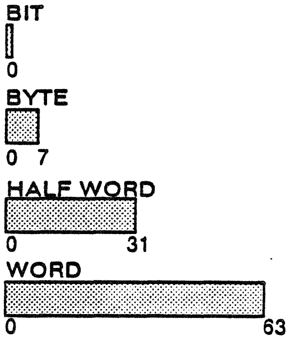
The addressable range is 2^{48} bits (281,474,976,710,656 bits), or 2^{42} words (4,398,046,511,104 words).

Words, Half Words, and Bytes

A byte is defined as an 8-bit quantity; the address of the left-most bit is always a multiple of 8_{10}.

A half word is defined as a 32-bit quantity; the address of the left-most bit is always a multiple of 32_{10}.

A word is defined as a 64-bit quantity; the address of the left-most bit is always a multiple of 64_{10}.



Instruction Addressing

Instructions are addressed on full-word and half-word boundaries. The instruction address counter is incremented by a half word after executing a 32-bit instruction and by a full word after executing a 64-bit instruction. This incrementation allows instructions to be packed contiguously in storage.

Instructions are packaged within 64-bit words in various ways:

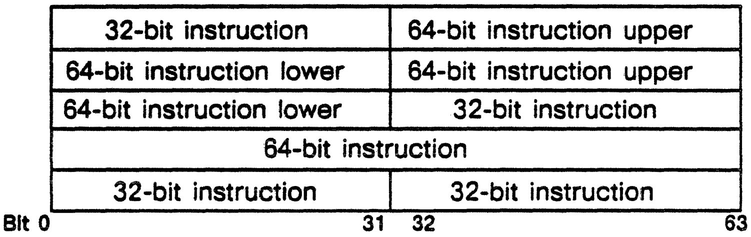
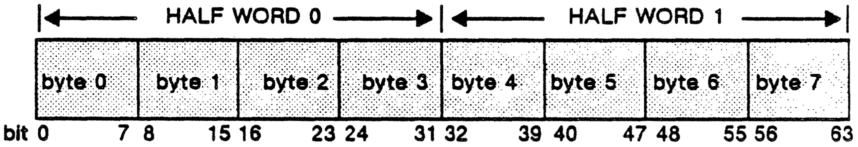


Figure 5-35. Instruction packing options.

A branch is possible to an instruction beginning on any half word. The five right-most bits in any branch address are always interpreted as zeros.

Depending on the type of instruction operand, a number of the right-most bits in the address are ignored. For example, if a byte is being read, the right-most three bits are ignored. Depending on the instruction, operands are counted on a bit, byte, half word, or word basis. Within a word, the bits, bytes and half words are always numbered from left to right, starting at the lowest addressed bit, byte, or half word in the word.

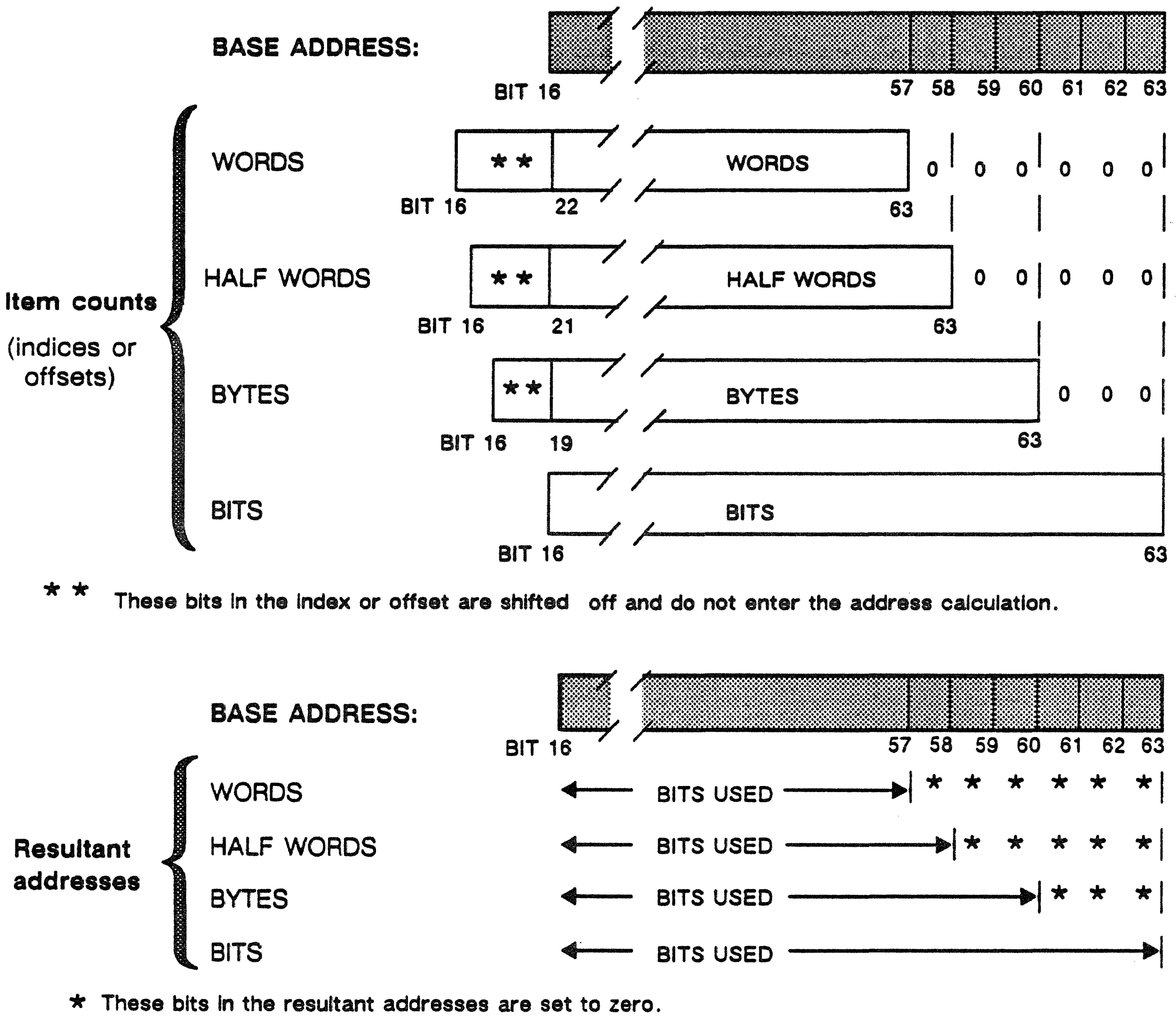
These are the relative locations of each bit, byte, and half word within a 64-bit word:



Item Counts

When the instructions specify that addresses and item counts such as indices and offsets are to be added to base addresses, the item count is shifted left, end-off until it is properly aligned with the address. Zeros are appended to the right end of the quantity being shifted.

The result of the addition always addresses a quantity having the same unit as the item count. For instance, if a byte count is added to any address, the result references a byte. In this case, the right-most three bits of the address are ignored. Figure 5-36 summarizes the process of adding an item count to an address and shows which bits are ignored in the resulting address.



** These bits in the index or offset are shifted off and do not enter the address calculation.

* These bits in the resultant addresses are set to zero.

Figure 5-36. As item counts are added to base addresses, bits in resultant addresses are set to zero.

Illegal Instructions

There are two types of illegal instructions; type one can occur in either monitor or job mode, type two can occur only in job mode. In monitor mode, type one illegal instructions are not a user's problem. Type two is an attempt to issue a valid instruction, but one that is not allowed to execute in the current domain.

Type One Illegal Instruction

Type one includes two kinds of illegal instructions; issuing unassigned function codes and the incorrectly aligned register swap instruction. An unassigned function code is not a valid instruction; a listing of these appears at the end of the "Functional Categories" section. The register swap instruction is set to quickly load the register file with even addresses and even operand lengths. It becomes an illegal instruction when it is set to odd addresses and odd operand lengths.

Type Two Illegal Instruction

The second type of illegal instruction occurs when some monitor mode instructions attempt execution in job mode. There are three instances when this type of illegal instruction can occur:

- a 'monitor mode only' instruction is issued in job mode
- a program includes a FORWARD DOMAIN CHANGE instruction that is not allowed by the domain change mask in the program's current domain package.
- a program includes an instruction that is not allowed by the illegal instruction mask in the program's current domain package; the instruction is attempted in a locked-out domain

The second type of illegal instruction occurs only in job mode. It sets a bit in the interrupt register that waits until the non-zero value in the register causes an exchange to monitor mode.

The illegal instruction mask is a 32-bit number that is a defined quantity in the invisible package and the domain package. The mask allows a particular instruction or set of instructions to be selected as legal or illegal instructions for different domain packages.

When instructions are executed in monitor mode, the illegal instruction mask has no effect. If a type one illegal instruction occurs in job mode, a bit is set in the interrupt register and an exchange to monitor mode occurs. When an illegal instruction occurs in monitor mode, the interrupt is issued but no exchange occurs.

No-Op Instructions

No-op instructions are defined as those instructions and special cases of instructions that do not alter the data flag register, the register file, central processor memory, shared memory, communication buffer, or any instruction defined hardware registers. An unnecessary space table search and/or access interrupt may be caused by a no-op instruction.

Overlap of Operand and Result Fields

Undefined results can occur due to overlapping when source elements are both original source elements and newly-stored result elements. Such overlaps occur when the results field overlaps a source field so that elements of the result are stored in a portion of the source field before elements in this portion are read. An instruction's results may become undefined when this happens. Certain instructions specifically prohibit any overlap of source and destination fields.

Scalar Instructions

In scalar instructions, the instruction designators point to registers that are the sources and destinations. This is in contrast to vector instructions, for example, in which the instruction designators point to registers containing a description of the sources and destinations that are located in memory.

Most scalar instructions contain R, S, and T fields that are used to designate the contents of one of 256 registers. R and S fields indicate registers containing source operands, source registers 0-255. The T field indicates the destination register containing the result operand, destination register 0-255. For scalar instructions, a register may be used to hold one or both source operands as well as the result.

In register-to-register operations, the contents of the source registers are usually unchanged, and the destination register is usually cleared before the result is transferred into it. Accordingly, an instruction that returns a 48-bit result will change, for example, the left 16 bits of the result register.

Vector Instructions

The vector instructions perform operations on ordered scalars.

In vector instructions, the designators point to registers that describe the sources and the destination. The sources and the destination themselves are in memory and are vectors rather than single quantities. All the vector instructions are in instruction format 1 as shown earlier in this section.

Base addresses, which define the location of the first element of the vector, and vector field lengths are obtained for the source and destination operands. Fields in the vector instruction designate the registers that contain the location in memory of the source operands of the hardware instructions for both vector streams. These fields are used to arrive at the vector's start address which is the base address and an offset. Another field designates a register that gives the location in memory of the destination of the hardware instruction. The offsets are item counts. When used as an offset, the item count is multiplied by a factor that adjusts for the size of the operands before it is added to the base address. For example, for a vector with 64-bit operands, the offset is shifted left six places before being applied to the base address. For a vector with 32-bit operands, the offset is shifted left five places.

Vector instructions terminate when the result vector is exhausted. Source vectors that are exhausted before the result vector is exhausted are extended with machine zeros in additive operations or normalized ones in multiplication or division operations. Result lengths are affected by offsets, as explained below.

Control Vector

A control vector is a bit vector. When control vectors are specified, a single unique bit from the control vector is associated with the storing of each result element in the output field and the setting of data flags for that result. When a bit with a control vector prohibits the storing of a result element, the previous contents of the associated result vector element are not altered nor is the data flag register modified. The *n*th bit read from the control vector prohibits or allows the storing of the *n*th result of the result vector.

Indexing and Offsetting

There are three basic differences between indexing and offsetting:

1. For indexing, the field length found in the register containing the base address is the same as the length of the data field. For

offsetting, the field length in the register is equal to the offset plus the length of the data field.

2. Indexing does not affect the field length used. Offsetting requires that the offset be subtracted from the field length to get the length of the data field.
3. For indexing, the field length starts at the base address plus index. For offsetting, the field length starts at the base address.

Vector instructions are offset and logical string instructions are indexed.

Vector Termination Rules

Vector instructions terminate upon exhausting the result length of a data field, a data string, or a vector. A vector here is any operand or result that consists of either multiple elements (100 32-bit floating point numbers, for example), or strings of elements (a 100-bit boolean, for example). A string, field, or vector is exhausted prior to the first operand fetch if its length is zero, or if the field length is read from register 00. These strings, field, and vectors are exhausted at this point when the result of subtracting the number of elements from the field length is zero. When the string, field, or vector is exhausted prior to the first operand fetch, the instruction becomes a no-op. No memory location, register file location, or data flags are altered.

A vector operand is exhausted if the result of subtracting the offset from the field length is zero or negative. Such a vector is assigned a length of zero. A vector that has an offset is also exhausted when the result of subtracting both the number of operands encountered and the offset from the field length is zero.

Vector instructions terminate when the result vector is exhausted. Source vectors that are exhausted before the result vector is exhausted are extended with machine zeros in additive operations or normalized ones in multiplication or division operations.

Sparse vector instructions terminate when the result order vector is exhausted. If the result order vector designator or length is zero, no data flags are set and the instruction is a no-op. If the order vector has a non-zero length and the result vector designator is zero, the results of the instruction are undefined.

A string instruction terminates when the result string is exhausted. Source strings that are shorter than the result string are extended with zeros unless otherwise specified.

Sparse Vector Instructions

Due to arithmetic operations, many elements of a vector may be reduced to near zero. These elements need not be carried along as floating-point numbers except for their positional significance. The sparse vector instructions make possible the expansion and compression of vectors with many elements of this type in order to conserve both storage space and calculation time. Sparse vector instructions conserve both storage space and calculation time by not calculating with or having storage for (near) zero elements.

A sparse vector consists of a vector pair, an order vector, and a data vector. A data vector is a floating-point array containing the significant data elements. The order vector is a bit vector used to determine the positional significance of the elements of the data vector.

A sparse vector is formed when the compare instructions generate an order vector. Then, a compress instruction can reduce a normal vector to a sparse vector by using the order vector as a means to clear out the near-zero elements.

To determine if the arithmetic operation on the data vectors is meaningful, a logical operation is performed on the order vectors. When the operation is an add or a subtract, the OR of the order vectors determines if the corresponding data vectors are to be added or subtracted. When the operation is a multiply or a divide, the AND of the order vectors determines if the corresponding data vectors are to be multiplied or divided.

A sparse data vector is indistinguishable in format from any other vector. However, it does have an associated order vector which determines the positional significance of the elements of the sparse data vector. The sparse vector includes both the data vector and the order vector. The order vector enables the original vector to be regenerated from the data vector. Offsetting and indexing are not performed by sparse vector instructions. All sparse vector instructions are in the first instruction format (2) shown earlier in this section.

Sparse vector instructions terminate when the result order vector is exhausted. If the result order vector designator or length is zero, no data flags are set and the instruction is a no-op. If the order vector has a non-zero length and the result vector designator is zero, the results of the instruction are undefined.

Functional Categories of Instructions

This section lists the complete set of ETA10 instructions grouped according to their function and usage. The instructions are grouped into these five categories:

- Data motion instructions:
  - general
  - enter immediate
  - replace (transmit)
  - shared memory
  - communication buffer
- Data calculation instructions:
  - add
  - subtract
  - multiply
  - divide
  - type conversion
  - bit operations
  - shifts
  - logical
  - other
- Branch instructions:
  - domain change
  - general
  - atomic branch and alter
- Search and comparison instructions
- Other:
  - general
  - timers and counters
  - monitor mode only
  - unused function codes

Data Motion Instructions

General Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
12	LOAD BYTE; (T) PER (S), (R)	7
13	STORE BYTE; (T) PER (S), (R)	7
5E	LOAD HALF-WORD; (T) PER (S), (R)	7
5F	STORE HALF-WORD; (T) PER (S), (R)	7
7C	LENGTH; (R) TO (T)	A
7D	SWAP; S ---> T AND R ---> S	7
7E	LOAD WORD; (T) PER (S), (R)	7
7F	STORE WORD; (T) PER (S), (R)	7
B7	SCATTER ---> INDEXED C	1
B8	TRANSMIT REVERSE; A ---> C	1
BA	GATHER ---> C	1
BB	MASK; A, B, ---> C PER Z	2
BC	COMPRESS; A ---> C PER Z	2
BD	MERGE; A, B, ---> C PER Z	2
CF	ARITHMETIC COMPRESS; A ---> C PER B	1

Enter Immediate Instructions

FUNCTION CODE:	INSTRUCTION NAME:		FORMAT TYPE:
2A	ENTER LENGTH OF (R) WITH I	(16 BITS)	6
3E	ENTER (R) WITH I	(16 BITS)	6
4D	HALF-WORD ENTER R WITH I	(16 BITS)	6
BE	ENTER (R) WITH I	(48 BITS)	5
CD	HALF-WORD ENTER (R) WITH I	(24 BITS)	5

Replace Instructions (Transmit)

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
38	TRANSMIT (R) BITS (0-15) TO (T) BITS (0-15)	A
58	TRANSMIT HALF-WORD; (R) TO (T)	A
78	TRANSMIT WORD; (R) TO (T)	A
98	TRANSMIT VECTOR; A ---> C	1

Central Processor Memory/Shared Memory Transfers

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
18	SHARED MEMORY: CQTA TO (T), (S) TO CQTA	7
19	SHARED MEMORY: (S) TO IQHA, (T) TO IQHA, START TRANSFER (OR TRANSFERS)	7
1A	SHARED MEMORY: IQHA TO (S), IQVF AND IQTA TO (T)	7
1B	SHARED MEMORY: IQVF, TRANSFER BUSY FLAG FATAL ERROR STATUS AND TRBSA TO (T)	7

Transfers between central processor memory and shared memory are memory-to-memory. Shared memory instructions manipulate shared memory queues, setting up the input queue (IQ) and the completion queue (CQ). The shared memory port processor reads the queues and makes the transfer according to data found in the queues. Instructions specify head, tail, and start addresses (HA, TA, SA) as well as transfer request blocks (TRBs).

Central Processor to Communication Buffer Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
FA	POST SEMAPHORE	D
FB	WAIT SEMAPHORE	D
FC	BIT BRANCH AND SWAP	D
FD	BIT BRANCH AND LOAD/STORE	D
FE	LOAD HALF-WORD OR WORD; (C) PER (X)	D
FF	STORE HALF-WORD OR WORD; (C) PER (X)	D

Central processors share access to the communication buffer to synchronize their operations and share data during multiprocessing. The communication buffer instructions enable:

1. word and half-word transfers to and from the communication buffer and the central processor's register file
2. semaphore post and wait operations
3. conditional word and half-word swaps from the communication buffer to the register file
4. conditional tests and sets with word and half-word loads from the communication buffer to the register file

When the shared memory-communication buffer port receives a communication buffer instruction, the port compares the function code and address with those belonging to the requesting process's invisible package to detect any access errors. Only the central processors use these instructions; other system processors interface to the communication buffer through a hardware controller which issues instructions to the communication buffer interface.

Data Calculation Instructions

In the instructions that follow:

U	signifies the upper result
L	signifies the lower result
S	signifies the significant result
N	signifies the normalized result

The size and type of the result are noted with the instruction; 24- and 48-bit results are the right-most 24 and 48 bits of 32- and 64-bit operand results.

Add Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
2B	ADD TO LENGTH FIELD	4
3F	INCREASE (R) BY I (48-bit scalar int)	6
40	ADD U; (R) + (S) TO (T) (32-bit scalar fp)	4
41	ADD L; (R) + (S) TO (T) (32-bit scalar fp)	4
42	ADD N; (R) + (S) TO (T) (32-bit scalar fp)	4
4E	HALF-WORD INCREASE (R) BY I (24-bit scalar int)	6
60	ADD U; (R) + (S) TO (T) (64-bit scalar fp)	4
61	ADD L; (R) + (S) TO (T) (64-bit scalar fp)	4
62	ADD N; (R) + (S) TO (T) (64-bit scalar fp)	4
63	ADD ADDRESS; (R) + (S) TO (T) (48-bit scalar int)	4
80	ADD U; A + B ---> C (32/64-bit vector fp)	1
81	ADD L; A + B ---> C (32/64-bit vector fp)	1
82	ADD N; A + B ---> C (32/64-bit vector fp)	1
83	ADD ADDRESS; A + B ---> C (48-bit vector int)	1
A0	ADD U; A + B ---> C (32/64-bit sparse vector)	2
A1	ADD L; A + B ---> C (32/64-bit sparse vector)	2
A2	ADD N; A + B ---> C (32/64-bit sparse vector)	2
BF	INCREASE (R) BY I (48-bit scalar)	5
CE	HALF-WORD INCREASE (R) BY I (24-bit scalar	5

Subtract Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
44	SUBTRACT U; (R) - (S) TO (T) (32-bit scalar)	4
45	SUBTRACT L; (R) - (S) TO (T) (32-bit scalar)	4
46	SUBTRACT N; (R) - (S) TO (T) (32-bit scalar)	4
64	SUBTRACT U; (R) - (S) TO (T) (64-bit scalar)	4
65	SUBTRACT L; (R) - (S) TO (T) (64-bit scalar)	4
66	SUBTRACT N; (R) - (S) TO (T) (64-bit scalar)	4
67	SUBTRACT ADDRESS; (R) - (S) TO (T) (48-bit scalar)	4
84	SUBTRACT U; A + B ---> C (vector)	1
85	SUBTRACT L; A + B ---> C (vector)	1
86	SUBTRACT N; A + B ---> C (vector)	1
87	SUBTRACT ADDRESS; A - B ---> C (48-bit scalar)	1
A4	SUBTRACT U; A - B ---> C (sparse vector)	2
A5	SUBTRACT L; A - B ---> C (sparse vector)	2
A6	SUBTRACT N; A - B ---> C (sparse vector)	2

In the instructions that follow:	U	signifies the upper result
	L	signifies the lower result
	S	signifies the significant result
	N	signifies the normalized result

Multiply Instructions

FUNCTION CODE:	INSTRUCTION NAME:		FORMAT TYPE:
3C	HALF-WORD INDEX MULTIPLY (R)*(S) TO (T)	(24-bit scalar)	4
3D	INDEX MULTIPLY (R)*(S) TO (T)	(48-bit scalar)	4
48	MULTIPLY U; (R) * (S) TO (T)	(32-bit scalar fp)	4
49	MULTIPLY L; (R) * (S) TO (T)	(32-bit scalar fp)	4
4B	MULTIPLY S; (R) * (S) TO (T)	(32-bit scalar fp)	4
68	MULTIPLY U; (R) * (S) TO (T	(64-bit scalar fp)	4
69	MULTIPLY L; (R) * (S) TO (T)	(64-bit scalar fp)	4
6B	MULTIPLY S; (R) * (S) TO (T)	(64-bit scalar fp)	4
88	MULTIPLY U; A * B ---> C	(vector fp)	1
89	MULTIPLY L; A * B ---> C	(vector fp)	1
8B	MULTIPLY S; A * B ---> C	(vector fp)	1
A8	MULTIPLY U; A * B ---> C	(sparse vector fp)	2
A9	MULTIPLY L; A * B ---> C	(sparse vector fp)	2
AB	MULTIPLY S; A * B ---> C	(sparse vector fp)	2

Divide Instructions

FUNCTION CODE:	INSTRUCTION NAME:		FORMAT TYPE:
4C	DIVIDE U; (R) * (S) TO (T)	(32-bit scalar fp)	4
4F	DIVIDE S; (R) / (S) TO (T)	(32-bit scalar fp)	4
6C	DIVIDE U; (R) / (S) TO (T)	(64-bit scalar fp)	4
6F	DIVIDE S; (R) / (S) TO (T)	(64-bit scalar fp)	4
8C	DIVIDE U; A / B ---> C	(vector fp)	1
8F	DIVIDE S; A / B ---> C	(vector fp)	1
AC	DIVIDE U; A / B ---> C	(sparse vector fp)	2
AF	DIVIDE S; A / B ---> C	(sparse vector fp)	2

Also; see instruction 56, under *General Miscellaneous*, for linking vectors.

Conversion Instructions

FUNCTION CODE:	INSTRUCTION NAME:		FORMAT TYPE:
10	CONVERT BCD TO BINARY, FIXED LENGTH	(48-bit scalar)	A
11	CONVERT BINARY TO BCD, FIXED LENGTH	(64-bit scalar)	A
5C	EXTEND; 32-BIT (R) TO 64-BIT (T)	(64-bit scalar)	A
5D	INDEX EXTEND; 32-BIT (R) TO 64-BIT (T)	(64-bit scalar)	A
76	CONTRACT; 64-BIT (R) TO 32-BIT (T)	(32-bit scalar)	A
77	ROUNDED CONTRACT; 64-BIT (R) TO 32-BIT(T)	(32-bit scalar)	A
96	CONTRACT; 64-BIT A ---> 32-BIT C	(32-bit scalar)	1
97	ROUNDED CONTRACT; 64-BIT A ---> 32-BIT C	(32-bit vector)	1
9C	EXTEND; 32-BIT A ---> 64-BIT C	(64-bit vector)	1

Bit Operation Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
14	BIT COMPRESS (bit string)	7
15	BIT MERGE (bit string)	7
16	BIT MASK (bit string)	7
1C	FORM REPEATED BIT MASK WITH LEADING ZEROS (bit string)	7
1D	FORM REPEATED BIT MASK WITH LEADING ONES (bit string)	7
1E	COUNT LEADING EQUALS (64-bit scalar)	7
1F	COUNT ONES IN FIELD R, COUNT TO (T) (64-bit scalar)	7
6D	INSERT BITS; (R) TO (T) PER (S) (64-bit scalar)	4
6E	EXTRACT BITS; (R) TO (T) PER (S) (64-bit scalar)	4

Shift Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
30	SHIFT (R) PER S TO (T) (scalar)	7
34	SHIFT (R) PER S TO (T) (scalar)	4
8A	SHIFT; A PER B ---> C (vector)	1

Logical Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
2C	LOGICAL EXCLUSIVE OR (R), (S),TO (T) (64-bit scalar)	4
2D	LOGICAL AND (R), (S), TO (T) (64-bit scalar)	4
2E	LOGICAL INCLUSIVE OR (R), (S), TO (T) (64-bit scalar)	4
9D	LOGICAL; A, B ---> C (32/64-bit vector)	1
F0	LOGICAL EXCLUSIVE OR A, B ---> C (bit string)	3
F1	LOGICAL AND A, B ---> C (bit string)	3
F2	LOGICAL INCLUSIVE OR A, B ---> C (bit string)	3
F3	LOGICAL STROKE A, B ---> C (bit string)	3
F4	LOGICAL PIERCE A, B ---> C (bit string)	3
F5	LOGICAL IMPLICATION A, B ---> C (bit string)	3
F6	LOGICAL INHIBIT A, B ---> C (bit string)	3
F7	LOGICAL EQUIVALENCE A, B ---> C (bit string)	3
F8	MOVE BYTES LEFT; A ---> C (byte string)	3

The logical bit string instructions perform manipulations on strings of bits. The bit string operations (F0-F7) are performed as bit operations on bit boundaries.

The string instruction F8 has indices and fields identical to those of the logical string instructions except that the item count and indices are in bytes instead of bits. The string operation is performed as a byte operation on byte boundaries.

Other Calculation Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
	The 5x codes are 32-bit scalar instructions:	
50	TRUNCATE; (R) TO (T) **(32-bit scalar)**	A
51	FLOOR; (R) TO (T) **(32-bit scalar)**	A
52	CEILING; (R) TO (T) **(32-bit scalar)**	A
53	SIGNIFICANT SQUARE ROOT; (R) TO (T)	A
54	ADJUST SIGNIFICANCE; (R) PER (S) TO (T)	4
55	ADJUST EXPONENT; (R) PER (S) TO (T)	4
59	ABSOLUTE; (R) TO (T)	A
5A	EXPONENT; (R) TO (T)	A
5B	PACK; (R), (S) TO (T)	4
	The 7x codes are 64-bit scalar floating-point instructions:	
70	TRUNCATE; (R) TO (T)	A
71	FLOOR; (R) TO (T)	A
72	CEILING; (R) TO (T)	A
73	SIGNIFICANT SQUARE ROOT; (R) TO (T)	A
74	ADJUST SIGNIFICANCE; (R) PER (S) TO (T)	4
75	ADJUST EXPONENT; (R) PER (S) TO (T)	4
79	ABSOLUTE; (R) TO (T)	A
7A	EXPONENT; (R) TO (T)	A
7B	PACK; (R) TO (T)	A
	9x and some Dx codes are vector floating-point instructions:	
90	TRUNCATE; A ---> C	1
91	FLOOR; A ---> C	1
92	CEILING; A ---> C	1
93	SIGNIFICANT SQUARE ROOT; A PER B ---> C	1
94	ADJUST SIGNIFICANCE; A PER B ---> C	1
95	ADJUST EXPONENT; A PER B ---> C	1
99	ABSOLUTE; A ---> C	1
9A	EXPONENT; A ---> C	1
9B	PACK; A, B ---> C	1
D0	AVERAGE (A(N) + B(N)) / 2 ---> C(N)	1
D1	ADJACENT MEAN (A(N+1) + A(N)) / 2 ---> C(N)	1
D4	AVERAGE DIFFERENCE (A(N)-B(N)) / 2 --> C(N)	1
D5	DELTA (A(N+1) - A(N)) ---> C(N)	1
D8	MAXIMUM OF A TO (C), ITEM COUNT TO (B) ☜	1
D9	MINIMUM OF A TO (C), ITEM COUNT TO (B) ☜	1
DA	SUM (A0+A1+A2+...+AN) TO (C) AND (C+1) ☜	1
DB	PRODUCT; (A0*A1*A2*...*AN) TO (C) ☜	1
DC	DOT PRODUCT TO (C) AND (C+1) ☜	1
DF	INTERVAL; (A) PER (B) ---> C	1
	☞ (These instructions return scalar results)	

Branch Instructions

Domain Change Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
17	BACKWARD DOMAIN CHANGE	7
36	BRANCH OR FORWARD DOMAIN CHANGE	7

General Branch Instructions

FUNCTION CODE:	INSTRUCTION NAME:		FORMAT TYPE:
20	BRANCH IF (R) EQ (S)	(32-bit fp)	8
21	BRANCH IF (R) NE (S)	(32-bit fp)	8
22	BRANCH IF (R) GE (S)	(32-bit fp)	8
23	BRANCH IF (R) LT (S)	(32-bit fp)	8
24	BRANCH IF (R) EQ (S)	(64-bit fp)	8
25	BRANCH IF (R) NE (S)	(64-bit fp)	8
26	BRANCH IF (R) GE (S)	(64-bit fp)	8
27	BRANCH IF (R) LT (S)	(64-bit fp)	8
B6	BRANCH TO IMMEDIATE ADDRESS; (R) + I	(48-bits)	5

Atomic Branch and Alter Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
05	VOID STACK AND BRANCH	4
2F	REGISTER BIT BRANCH AND ALTER	9
31	INCREASE (R) AND BRANCH IF (R) NE 0	7
32	BIT BRANCH AND ALTER	9
33	DATA FLAG REGISTER BIT BRANCH AND ALTER	B
35	DECREASE (R) AND BRANCH IF (R) NE 0	7
FC	BIT BRANCH AND SWAP	D
FD	BIT BRANCH AND LOAD/STORE	D

These instructions cause the program to execute elsewhere in the code, they break the normal sequence of execution. The following items can be examined to determine a branch condition:

- a single bit
- a 24- or 48-bit integer
- a 32- or 64-bit integer
- a 32- or 64-bit floating-point operand

Two special branch instructions go between different job programs: FORWARD DOMAIN CHANGE and BACKWARD DOMAIN CHANGE.

An item count is a count that is independent of the size of the units counted. An item count of five, for example, could be five 64-bit operands, five 32-bit operands, or five single bits. Counting items in this way enables a vector instruction to operate on vectors with 32-bit operands and vectors with 64-bit operands using the same item count to specify the offset. In other words, when the offsets are item counts, the same offset can be used for vectors having different sizes of operands.

In branch instructions, all item counts are in half-words. In the BIT BRANCH AND ALTER instruction, for example, the T designator is used as the item count. Since the item count is in half words, the contents of the T register are shifted left five places to form a new address.

Search and Compare Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
28	SCAN EQUAL	7
B0	COMPARE INTEGER, BRANCH IF(A) + (X) EQ (Z) (64-bit scalar)	C
B1	COMPARE INTEGER, BRANCH IF(A) + (X) NE (Z) (64-bit scalar)	C
B2	COMPARE INTEGER, BRANCH IF(A) + (X) GE (Z) (64-bit scalar)	C
B3	COMPARE INTEGER, BRANCH IF(A) + (X) LT (Z) (64-bit scalar)	C
B4	COMPARE INTEGER, BRANCH IF(A) + (X) LE (Z) (64-bit scalar)	C
B5	COMPARE INTEGER, BRANCH IF(A) + (X) GT (Z) (64-bit scalar)	C
C0	SELECT EQ; A EQ B, ITEM COUNT TO (C)	1
C1	SELECT NE; A NE B, ITEM COUNT TO (C)	1
C2	SELECT GE; A GE B, ITEM COUNT TO (C)	1
C3	SELECT LT; A LT B, ITEM COUNT TO (C)	1
C4	COMPARE EQ; A EQ B, ORDER VECTOR ---> Z	1
C5	COMPARE NE; A NE B, ORDER VECTOR ---> Z	1
C6	COMPARE GE; A GE B, ORDER VECTOR ---> Z	1
C7	COMPARE LT; A LT B, ORDER VECTOR ---> Z	1
C8	SEARCH EQ; INDEX LIST ---> C	1
C9	SEARCH NE; INDEX LIST ---> C	1
CA	SEARCH GE; INDEX LIST ---> C	1
CB	SEARCH LT; INDEX LIST ---> C	1
CC	MASKED BINARY COMPARE; A EQ/NE (B) PER (C)	D

Miscellaneous Instructions

General Miscellaneous Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
03	KEYPOINT - MAINTENANCE	7
04	BREAKPOINT ON ADDRESS - MAINTENANCE	4
07	SERIAL/PARALLEL EXECUTION MODE SELECT - MAINTENANCE	7
09	EXIT FORCE	4
3B	DATA FLAG REGISTER LOAD/STORE (64-bit scalar)	A
56	SELECT LINK	7
57	READ DOMAIN REGISTERS; SPECIAL REGISTER PER R TO 64 BIT (T) (64-bit scalar)	7

Timer and Counter Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
29	TRANSMIT INSTRUMENTATION COUNTER TO (T) (48-bit scalar)	A
37	TRANSMIT JOB INTERVAL TIMER TO (T) (32-bit scalar)	A
39	TRANSMIT REAL-TIME CLOCK TO (T) (48-bit scalar)	A
3A	TRANSMIT (R) TO JOB INTERVAL TIMER	A

Monitor Mode Only Instructions

FUNCTION CODE:	INSTRUCTION NAME:	FORMAT TYPE:
00	IDLE	7
06	FAULT TEST - MAINTENANCE	7
08	TRANSMIT EXTERNAL INTERRUPT, (R)	4
0A	TRANSMIT (R) TO MONITOR INTERVAL TIMER	4
0C	STORE ASSOCIATIVE REGISTERS	4
0D	LOAD ASSOCIATIVE REGISTERS	4
0E	READ INTERRUPT REGISTER TO (T)	4
0F	LOAD KEYS FROM (R), TRANSLATE ADDRESS (S) TO (T)	4

Monitor instructions perform when the central processor is in monitor mode. When the central processor is not in monitor mode, these instructions perform as illegal instructions.

Unused Function Codes

FUNCTION CODE:	
01	ILLEGAL
02	ILLEGAL
0B	ILLEGAL
43	ILLEGAL
47	ILLEGAL
4A	ILLEGAL
6A	ILLEGAL
8D	ILLEGAL
8E	ILLEGAL
9E	ILLEGAL
9F	ILLEGAL
A3	ILLEGAL
A7	ILLEGAL
AA	ILLEGAL
AD	ILLEGAL
AE	ILLEGAL
B9	ILLEGAL
D2	ILLEGAL
D3	ILLEGAL
D6	ILLEGAL

FUNCTION CODE:	
D7	ILLEGAL
DD	ILLEGAL
DE	ILLEGAL
E0	ILLEGAL
E1	ILLEGAL
E2	ILLEGAL
E3	ILLEGAL
E4	ILLEGAL
E5	ILLEGAL
E6	ILLEGAL
E7	ILLEGAL
E8	ILLEGAL
E9	ILLEGAL
EA	ILLEGAL
EB	ILLEGAL
EC	ILLEGAL
ED	ILLEGAL
EE	ILLEGAL
EF	ILLEGAL
F9	ILLEGAL

CHAPTER 6

Chapter 6

The Input/Output Subsystem

Chapter Contents

List of Figures in Chapter 6

Chapter 6

The Input/Output Subsystem

In This Chapter . . .

I/O for the ETA10 is provided through a set of independent, multifunctional channels. Chapter 6 describes the I/O hardware and software, and its interfaces to the data pipe, to shared memory, and to the service unit.

- Section 1: Overview of the I/O Subsystem
  - Brief profile of I/O components
  - I/O architecture
- Section 2: I/O Software
  - Overview, relation to system software
  - Channel software
  - Disk transfers
- Section 3: I/O Hardware
  - Channel hardware
  - I/O connections and interfaces
  - I/O processor hardware

Section 1: Overview of the Input/Output Subsystem

The I/O subsystem provides the physical path from the ETA10 mainframe to the system disks and networks, and directly connects the networks and high-speed disks to shared memory.

The management of I/O resources and transfers is distributed between the central processors and the input/output units (IOUs). Using intelligent device processors and file servers, the IOUs independently manage I/O devices and processes. These intelligent processors offload low-level I/O operations from the central processors. The mainframe manages most logical I/O, the IOU manages physical I/O.

The IOU itself is a multi-processor, bus-connected computer in which channel processors and operating system software provide comprehensive services for I/O channel applications.

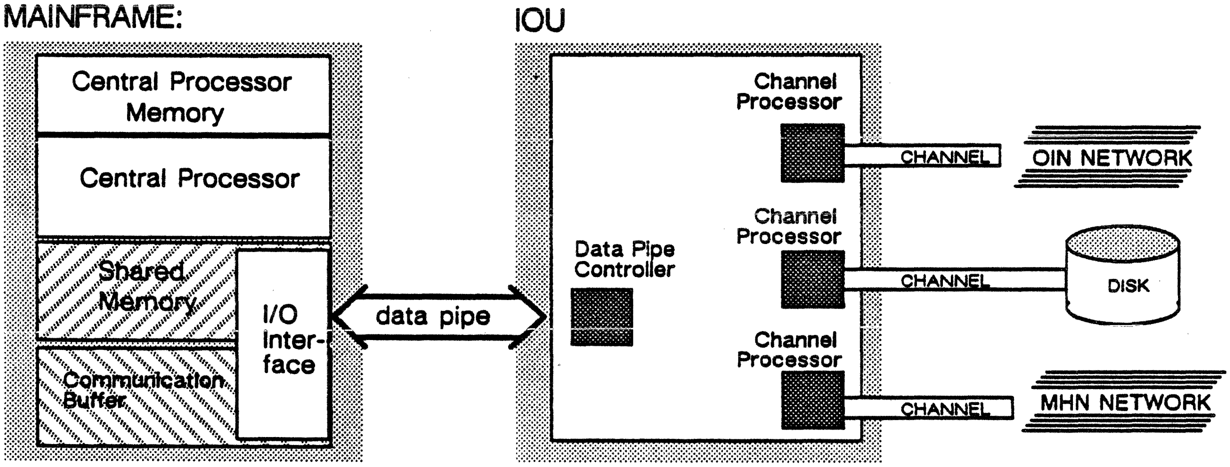


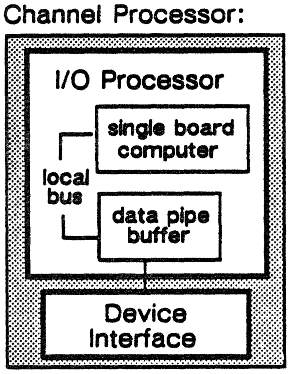
Figure 6-1. The data path between the I/O subsystem and the ETA10 mainframe.

Channel processors coordinate data transfers between shared memory and the system peripherals and networks. As shown in figure 6-1, the basic IOU hardware components are a set of channel processors and the controller for a data pipe. A channel processor controls the connection between a data pipe and a particular network or peripheral device channel. The data pipe controller manages data transfers between the channel processors and the I/O interface to shared memory.

System Configuration Options

The number of IOUs in a particular system depends upon site requirements rather than the number of central processors in the system configuration. An ETA10 system can be configured with as few as two IOUs, but it can accommodate up to eighteen.

Channel Processor



A channel processor contains an I/O processor paired with a specific type of network or device interface. The I/O processor contains two hardware components: a single board computer and a data pipe buffer, linked by a local bus. The choice of device interface and I/O processor software determine the functionality of the channel processor - that is, whether it is a channel for a local area network, a loosely coupled network, or a high-speed disk device. These are the three types of channel processors and their associated device interfaces:

- a FIPS 60-2 channel processor housing a FIPS channel interface
- a VME channel processor housing a VME channel interface
- a disk channel processor housing a disk channel interface

Figure 6–2 shows the types of channel processors, each with an internal I/O processor and a device interface that accommodates peripheral and network channels. The data pipe controller and the global memory are also shown. A common bus connects all single board computers to the global memory; all data pipe buffers are linked by the data pipe bus to the data pipe controller.

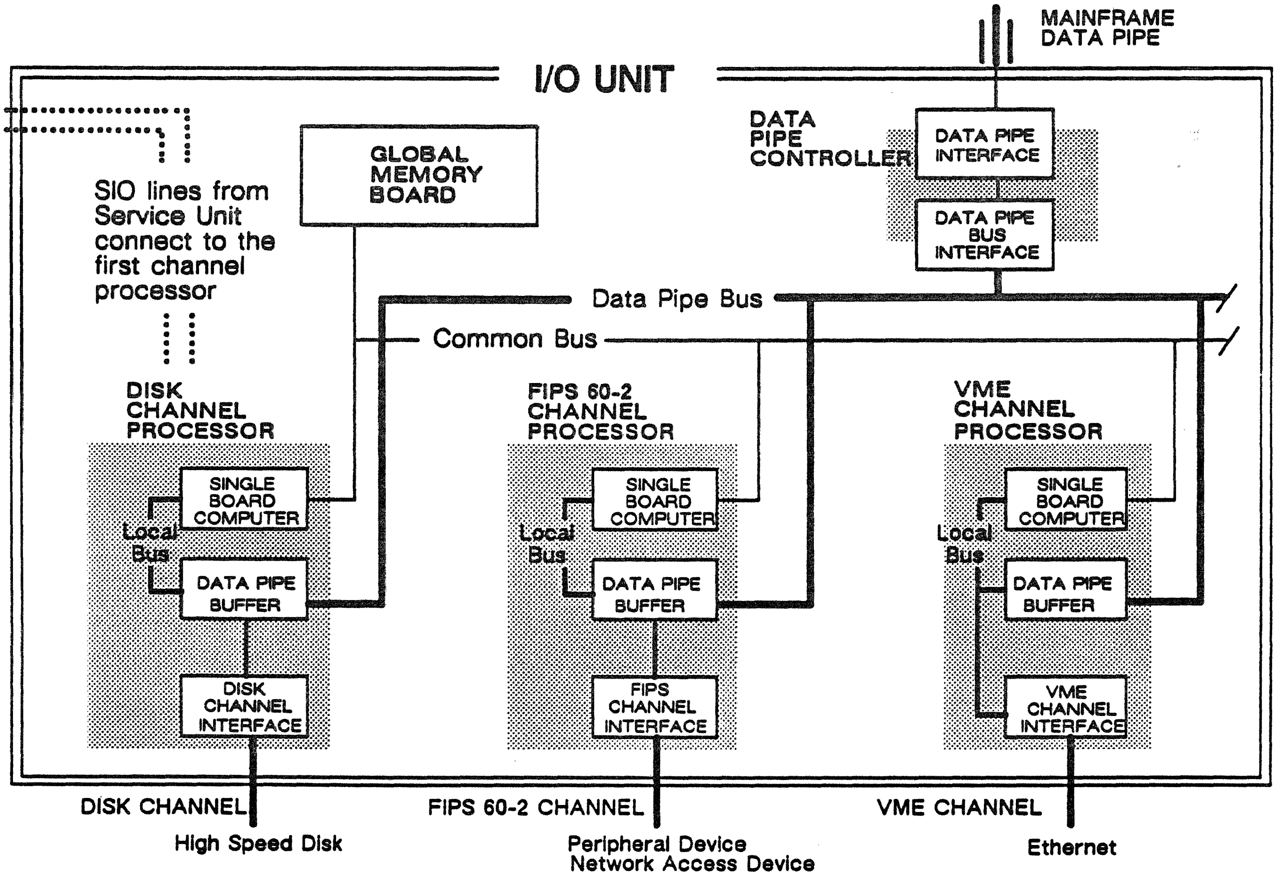


Figure 6–2. Internal components of an I/O unit.

In this internal view of an IOU, each type of channel processor and interface is shown: the disk processor with the disk channel interface, the FIPS processor with a FIPS interface, and a VME processor with a VME interface.

Note that the first I/O channel processor receives the serial input/output (SIO) lines that connect the IOU to the service unit. First, in this case, refers to the first processor board slot in the IOU chassis. The SIO lines provide the service unit maintenance connection to each IOU; within the IOU, the common bus extends the maintenance connection to each I/O processor.

Disk Channel Processor

The disk channel processor provides management for data transfers between the I/O unit and the system's physical disk devices. The disk channel transfers data at a sustained rate of 86.5 megabits/second. It supports disk drives that conform to ISI standards.

FIPS 60-2 Channel Processor

The Federal Information Processing Standard (FIPS) channel processor provides the FIPS 60–2 channel management capabilities for the IOU. The FIPS channel connects to the network access devices (NADs) for the Multi-Host Network. The ETA10 connection to the Multi-Host Network is through the FIPS channel. The Multi-Host Network is ETA Systems' version of CDC's Loosely Connected Network. The FIPS channel is capable of transferring data at a peak rate of 24 megabits per second.

VME Channel Processor

The VME channel processor provides management of the channel connecting to the Open Interconnection Network, the ETA Systems version of the local area network. Within the processor, the VME channel interface provides a connection compatible with the VME bus standard and supports the TCP/IP protocol. Data is transferred over the VME channel from the Open Interconnection Network at a peak rate of 10 megabits per second. The VME channel is also accessible to the smaller class of disk drives.

Data Pipe Controller

The data pipe controller manages the transfer of data between the mainframe I/O interface and I/O channel processors via a fiber optic data pipe. It interfaces between data pipe and data pipe buffers, accepting control information from the buffers. The primary components of the data pipe controller are the data pipe interface and the data pipe bus interface.

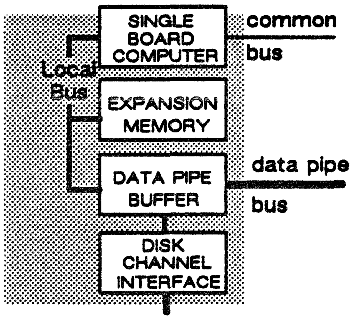
Global Memory Board

The IOU's global memory - 9 Mbytes of RAM and 1 Mbyte of PROM - is used to store disk allocation data. It is accessible to all the single board computers by way of the common bus. Memory accesses of 8-bits, 16-bits, and 32-bits are supported. Two I/O registers provide memory protection as well as detection and correction of single-bit errors and detection of dual-bit errors (SECDED).

Expansion Memory Board

Some IOUs are configured with a 4-megabyte expansion memory installed in one disk channel processor.

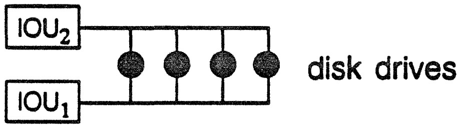
As shown, the single board computer in that channel is the only one to access the memory, and it does so via local bus. Expansion memory is used to store disk allocation data.



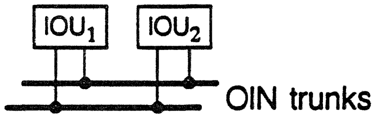
I/O Reliability

To provide a reliable I/O subsystem, an ETA10 system typically requires two IOUs. This requirement ensures that I/O operations are maintained during power supply or other failures in a single IOU. Dual access paths provide basic reliability in the event of an I/O component problem. The dual paths are maintained as a part of system configuration so that when I/O hardware or software problems are detected on the primary path, the system may automatically switch to the second access path.

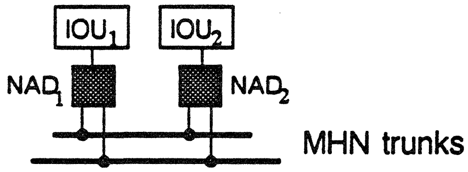
Dual I/O paths to disk units are maintained through two separate channels with dual-access drives.



Dual paths to the Open Interconnection Network cables are set up with separate IOUs as shown:



Dual paths to the Multi-Host Network trunks are set up with redundant network access devices (NADs), NADs do not have dual channel connections.



Dual access paths to shared memory and the communication buffer are created when redundant connections are established through two separate IOUs. For maintenance and operations access from the service unit, the first channel processor in each IOU is connected to service unit server nodes by a pair of SIO lines. The service unit-IOU connection is discussed in more detail later in this chapter.

IOU Architecture

The I/O units (IOU) provide for the controlled transfer of data between shared memory and attached peripherals. Each I/O unit is individually connected to shared memory through its own data pipe and I/O port. Within the IOU, I/O processors coordinate with peripheral device interfaces to transfer data to and from the data pipe. How I/O software and hardware transfer data and manage data transfers is discussed in this section.

I/O Transfer Path

Figure 6-3 illustrates the path over which data is transferred to/from the mainframe. When a process executing on a central processor requests data not in central processor memory, the request is sent to software that manages shared memory. If the data already resides in shared memory, the transfer is immediately made into processor memory.

I/O Data Transfer Path:

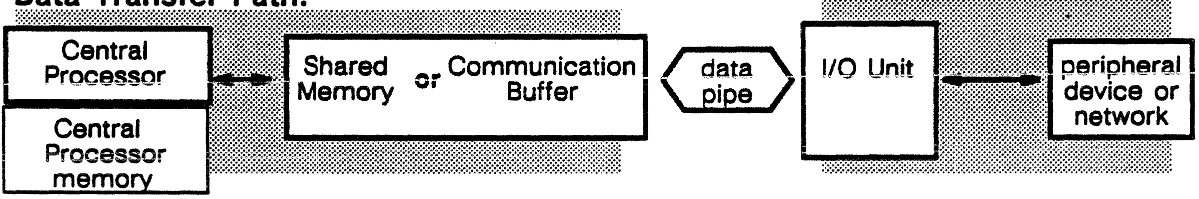

Central Processor
Central Processor memory
Shared Memory or Communication Buffer
data pipe
I/O Unit
peripheral device or network

Figure 6-3. The path of I/O data transfers between the mainframe and the IOU.

When the requested data is not in shared memory, a message requesting the data is sent to the IOU. The IOU retrieves the data from the peripheral device or network, and transfers the data to shared memory via the data pipe.

I/O Control Model

The basic I/O transfer control model is illustrated in figure 6-4. When a user application running on the central processor requests specifically-addressed data, it may call a network application (also residing on the central processor) to transfer data from a network source. An example is when a process calls a network application to request a file residing on a remote mainframe. The network application uses the operating system software on the central processor to direct and issue the request to the IOU operating system.

Operating system software on the IOU receives the request, processes it, and forwards the request to the proper device driver software. The device driver processes the request and turns the transfer management back to the operating system software. When the network controller passes the file to the IOU, the operating system software on the IOU sends it through the data pipe to shared memory where it is available for the next process request.

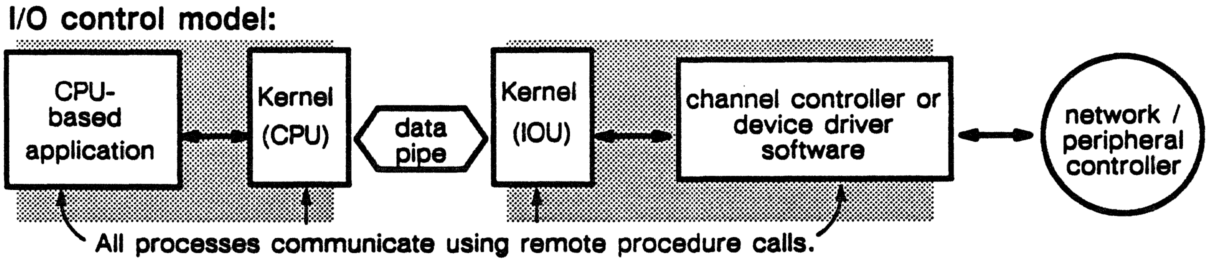
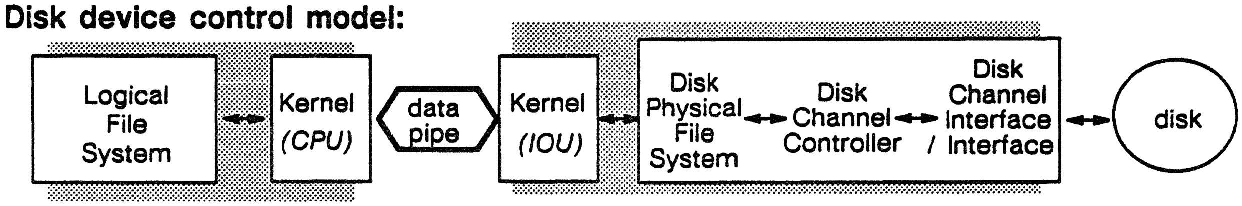

I/O control model:
CPU-based application
Kernel (CPU)
data pipe
Kernel (IOU)
channel controller or device driver software
network / peripheral controller
All processes communicate using remote procedure calls.

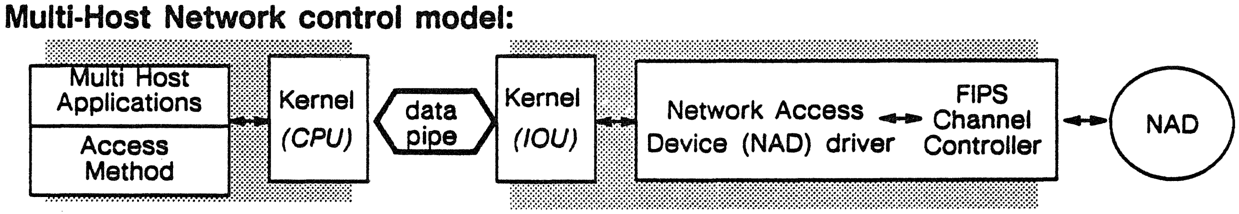
Figure 6–4. Control model for I/O transfers.

The reverse of this process is illustrated when a user logs on to a network and the logon is transmitted by the network controller to the operating system software on the IOU. This software relays the logon request to central processor-based features in the operating system software that establish the user session with the ETA10.

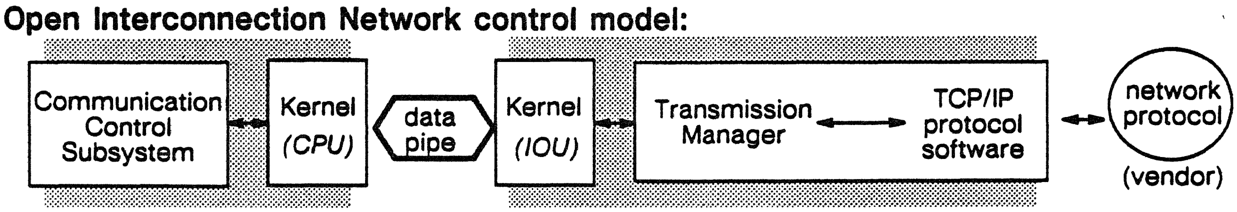
Following the general model in figure 6–4, figure 6–5 diagrams specific I/O control models for disk transfers, Multi-Host Network transfers, and Open Interconnection Network transfers. The software components are described in the "I/O Channel Software" section.


Disk device control model:
Logical File System
Kernel (CPU)
data pipe
Kernel (IOU)
Disk Physical File System
Disk Channel Controller
Disk Channel Interface / Interface
disk

The logical file and disk physical file systems manage disk transfers supported by disk channel software components.



The Multi-Host Network software provides commands and coordinates user connections with support from FIPS channel software.

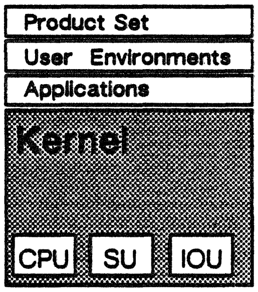


The communication control subsystem and transmission manager software coordinate the exchange of control and data packets between users on the network and the ETA10.

Figure 6–5. I/O control models between central processors and I/O devices.

Section 2: I/O Software

This description of I/O software begins with the operating system kernel and the place of I/O software within it. As shown on the right, the operating software includes the kernel, layers of applications, user environments, and a product set. Some portions of kernel are distributed across the three types of system processors, other portions are processor-specific.



Kernel Operating System

The diagram below shows processor-specific as well as distributed kernel features.

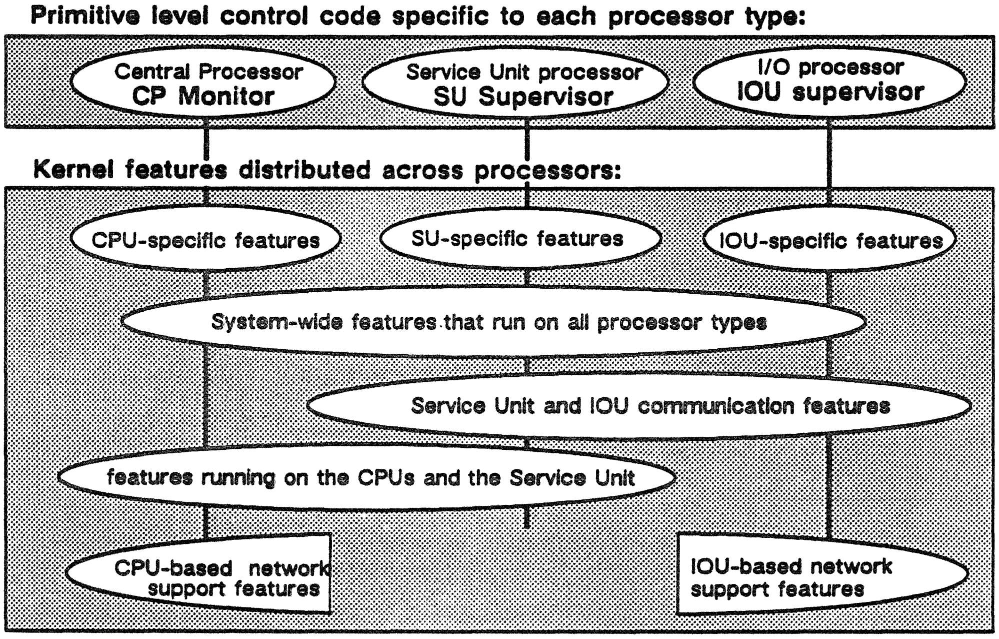

Primitive level control code specific to each processor type:
Central Processor
CP Monitor
Service Unit processor
SU Supervisor
I/O processor
IOU supervisor
Kernel features distributed across processors:
CPU-specific features
SU-specific features
IOU-specific features
System-wide features that run on all processor types
Service Unit and IOU communication features
features running on the CPUs and the Service Unit
CPU-based network support features
IOU-based network support features

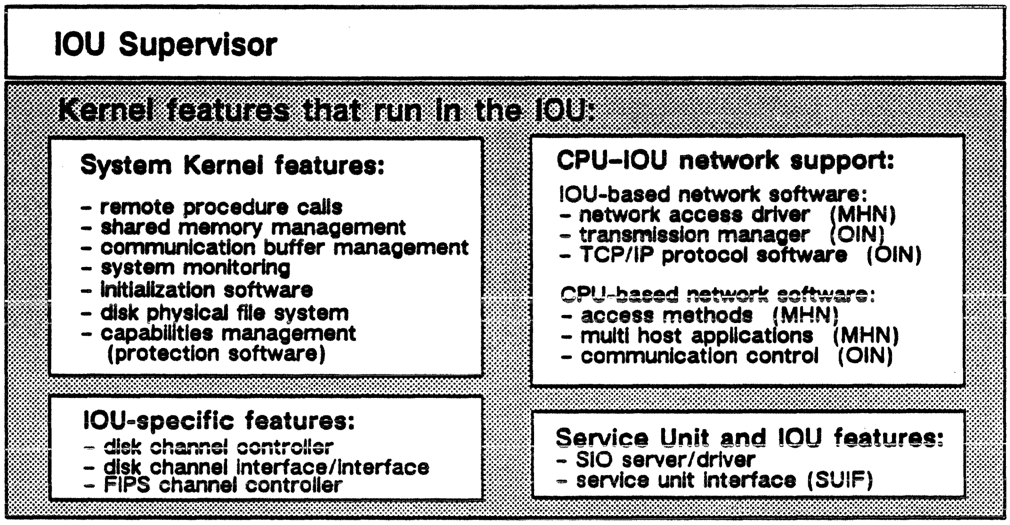
Figure 6-6. Kernel features and feature locations.

The top portion of figure 6-6 shows each type of system processor and its hardware-specific control code; central processors run CPU monitor (in monitor mode), I/O processors run IOU supervisor (in supervisor mode), service unit processors run SU supervisor (in supervisor mode). The lower portion shows distributed kernel features as they reside on one or more system processors. All kernel features run in job (user) mode.

Refer to chapter 4, "Operating System Kernel", for more information about the system software.

Kernel Features Residing on the IOU

This diagram focuses on the IOU portions of figure 6–6.



IOU Supervisor

Kernel features that run in the IOU:

System Kernel features:
- remote procedure calls
- shared memory management
- communication buffer management
- system monitoring
- initialization software
- disk physical file system
- capabilities management (protection software)

CPU–IOU network support:

IOU-based network software:
- network access driver (MHN)
- transmission manager (OIN)
- TCP/IP protocol software (OIN)

CPU-based network software:
- access methods (MHN)
- multi host applications (MHN)
- communication control (OIN)

IOU-specific features:
- disk channel controller
- disk channel interface/interface
- FIPS channel controller

Service Unit and IOU features:
- SIO server/driver
- service unit interface (SUIF)

IOU Supervisor

Supervisor is the base on which the kernel operating system is built. The IOU supervisor runs in supervisor mode which allows it to execute privileged instructions. All I/O processors run IOU supervisor.

System Kernel Features

These are features from the kernel operating system that also reside elsewhere in the system. Some are memory management features that enable the IOU to read/write to communication buffer and shared memory. Others are part of system maintenance and monitoring software.

IOU-specific Features

These features run only on the IOUs. They are I/O device driver and channel controller software that make calls to the IOU supervisor while managing and interfacing I/O channels and peripheral devices.

Central Processor and IOU Network Features

These features provide the CPU and IOU software counterparts to support two types of networks. The network features residing on the IOU side are interfaces and device drivers for network devices.

Service Unit and IOU Features

These features provide service unit / IOU communication options.

The next three sections in this chapter describe the IOU software: IOU supervisor, system kernel features, and I/O channel software.

IOU Supervisor

The IOU supervisor is the primitive level code running in the IOU. It is common to all I/O processors and resides in both random access and read only memories of the single board computer (MC68020) in each processor. The specific I/O function of an I/O processor is determined by the additional device driver software it carries and the device interface board to which it connects. In mirroring the functions of CPU monitor, IOU supervisor provides a multitasking environment that coordinates processes running within the device drivers.

The IOU supervisor operates on three basic system objects: processes, mailboxes, and messages. System-wide, these objects are used as communication mechanisms. IOU processes access supervisor services through procedure calls. The IOU supervisor provides a set of system calls that allow the device drivers to create/delete mailboxes and send/receive messages through mailboxes. These system calls are provided through a single board computer TRAP instruction.

IOU Supervisor Functions

IOU supervisor runs in supervisor mode and provides these functions:

- IOU process management
- IOU driver support
- IOU interprocess communications
- IOU memory management
- interrupt handling
- I/O processor initialization

IOU Process Management

Supervisor supports processes as the basic functional element of program execution and provides a set of service processes to assist the device driver applications. Processes in the IOU are either running, ready to run, or waiting for an event. Only one process runs at a time. Ready to run processes are scheduled via a priority ordered scheduling queue. When a process makes a request that cannot be filled immediately, it goes into the wait state. When the requested event occurs, the waiting process is put into the scheduling queue.

Processes communicate with each other via mailboxes; many processes get their work from a mailbox. Memory protection hardware prevents non-supervisor processes from accessing supervisor memory and data pipe register memory.

Driver Support

The IOU supervisor supports drivers for two hardware interfaces common to all I/O processors. Both drivers are interrupt-driven.

Serial Input/Output Driver

The serial input/output (SIO) driver is accessed using the system mailbox mechanism. The SIO driver enables communication between the service unit and IOU processes, allowing a process to send/receive data via the SIO connection. It also performs primitive read, write, and initialization functions for the I/O processors. This software interface enables the execution and monitoring of diagnostics, passes messages to the system error log, and provides for utility processing.

Data Pipe Driver

The data pipe driver allows a process to send/receive data to/from communication buffer and shared memory. The driver performs these functions: read shared memory, write shared memory, transmit communication buffer command, and transmit interrupt.

IOU Interprocess Communications

IOU supervisor provides mailboxes and mailbox-support functions to transfer messages and replies between processes throughout the IOU. The mailbox structures enable IOU processes to send and receive messages and to exchange synchronization signals between channel processors.

Mailboxes

Mailboxes are system objects that can be created and deleted by processes for passing messages. A mailbox holds messages; processes send messages to mailboxes and receive messages from mailboxes. When created, a mailbox is assigned a unique mailbox ID that identifies the mailbox location for its system calls. Each mailbox has a unique name, and it also has access rights, determining the operations that can be performed on it and the processes that can use it. Processes may check for the arrival of a message in a mailbox (polling), or processes may wait for notification that a message has been delivered to their mailbox. Some processes receive their work through their mailbox.

I/O System Mailboxes

IOU supervisor provides a set of system mailboxes that enable device drivers and IOU processes to use system services. System mailboxes are used by a restricted set of system features. These are the I/O system mailboxes supported by IOU supervisor:

- serial input/output driver
- data pipe controller

- service unit interface driver to server
- service unit interface driver from server
- service unit interface server from application
- maintenance interface-I/O processor diagnostic monitor
- maintenance interface-I/O processor debug monitor
- maintenance interface-I/O processor error log processor
- maintenance interface-I/O processor utilities
- remote procedure requests
- shared memory manager requests
- system initialization
- disk physical-file system

Messages and Replies

A message is a variable length memory-resident data structure consisting of a structured header and a data block. Messages and replies are always directed toward a mailbox. A process forms a message and then directs a supervisor feature to deliver the message. A message of null length, carrying no data, would function much as an interrupt if it caused the receiving process to begin an activity.

IOU Memory Management

Memory protection is at 4 kilobyte boundaries. I/O memory has a common memory mapping that controls access to:

- local data on the single board computer
- registers in the data pipe buffer
- unit-wide I/O data on the global memory
- data in the device interfaces.

Data Pipe Buffer

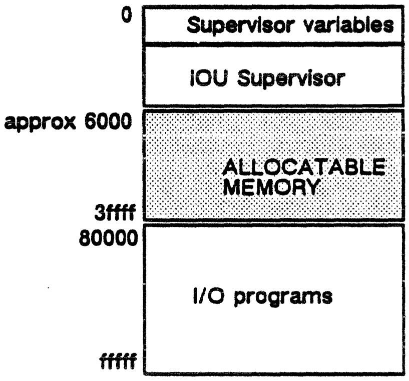
The data pipe buffer is a 128 kilobyte memory used primarily for temporary storage during data transfers between a data pipe and an I/O channel. The data pipe buffer is a memory used by the single board computer to store into. Each single board computer is directly linked to a data pipe buffer by the local VME bus. The data pipe buffer contains source registers that support data pipe activity such as transfers to shared memory.

Global Memory

The global memory is 9 megabytes of memory accessed over a common bus by an IOU's single board computers. Single board computers can move data from global memory into their own on-board memory as a means of communicating with each other. The 1-megabyte PROM holds the I/O processor boot loader and some IOU supervisor relocateable code that is loaded into the single board computers. Copies of the free space maps for each disk drive are maintained in global memory. Device drivers can reference global memory to move disk-based tables into the memory, but they do not store their device data there. The disk channel processor (via the disk physical-file system) frequently accesses global memory, since it maintains long-term storage of disk-based files.

Single Board Computer Memory

The single board computer has one megabyte of on-board memory.



Memory on the single board computer is used for the major components of IOU software and for maintaining some IOU tables. It is also used for channel procesor message passing.

It is a statically loaded memory. During processor initialization, programs are loaded into predefined areas.

Interrupt Handling

IOU supervisor supports interrupts from the peripheral devices and also passes control to a device driver. An interrupt handler receives control when the single board computer detects an internal exception (instructions, address errors, tracing, breakpoints) or an external exception (interrupts, bus errors, reset). The interrupt handling pre-empts "normal" device driver execution.

I/O Processor Initialization

The I/O processor boot (IOP boot) is ROM-resident software on the single board computer. During system initialization, the boot is called by a system initialization feature to activate the first I/O processor (this processor is connected to the system disk). The IOP boot causes the IOP initialization image to be moved from shared memory to the I/O processor, thereby activating the I/O processor. Then a system initialization feature initializes the processor and brings it to the on-line state. When the remaining I/O processors are initialized, they are loaded with the IOP initialization image, activated, and promoted to the on-line state. See the "System Initialization Software" section for more information about system initialization features on the IOU.

Operating System Kernel Features

The operating system software residing on the IOU also includes a set of system kernel features that function in job (user) mode. These features are common to all I/O processors and reside in the single board computer in each I/O processor.

- shared memory manager
- communication buffer memory manager
- remote procedure call
- system initialization software
- protection software
- system monitoring software

Shared Memory Manager Features

Features of the shared memory manager software enable the IOU to read and write shared memory objects. Shared memory objects are locked-down regions in the shared memory that the operating system uses to store data which must be shared across the system. Shared memory objects that directly relate to the IOU are the large buffers used by the remote procedure call feature to store messages.

Communication Buffer Manager Features

The communication buffer is a memory set aside for system-wide communication among system processors. Communication buffer memory management features on the IOU allow the IOU to access the communication buffer, issue communication buffer commands, and read and write to the communication buffer memory. The IOU uses these features to enable it to transfer data to the communication buffer and to talk to central processors, the service unit, and other IOUs.

Data that the IOU transfers across the data pipe is usually sent to shared memory for use by a process running on a central processor. When the requested data is needed for several processes running on different central processors, the IOU reads that data into communication buffer memory where it is managed for concurrent use by the several processes. The IOUs also communicate with other IOUs and with the service unit through the communication buffer itself.

The IOU issues commands to the communication buffer through the data pipe buffer. The data pipe controller processes the messages and sends them to the I/O interface on the communication buffer. The I/O interface issues the commands to the communication buffer and returns responses to the data pipe controller. The data pipe controller writes the responses into data pipe buffer memory, where they are available to the I/O processor.

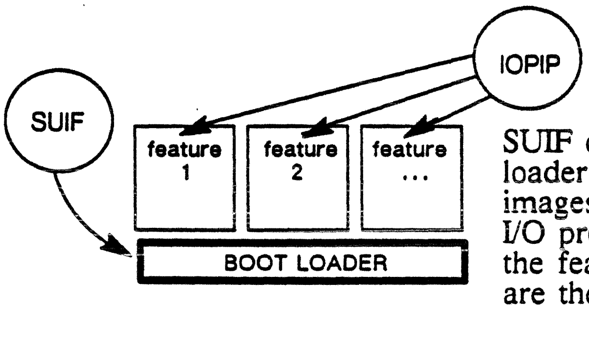
Remote Procedure Calls

This operating system feature provides functions that transfer messages and replies between processes throughout the ETA10 system. Mailboxes are used to hold messages upon their delivery from a message buffer. A message buffer contains a copy of the original message and space for any requested reply; one message buffer is maintained for each message. Processes send messages to a specific mailbox address, then the remote procedure call feature moves the message to the intended mailbox. The receiving process checks its mailbox for messages and uses the remote procedure call feature to return a reply to the mailbox or memory indicated by the first process. The remote procedure call software requires the use of shared memory and the communication buffer, and is supported by their memory management software.

A kernel feature establishes an IOU kernel mailbox so that communication buffer replies can be received by an IOU process. The feature uses mailbox access rights to protect mailboxes, messages, and replies from inadvertent access. The communication buffer and shared memory managers are used to protect messages and replies stored in communication buffer and shared memory. During system initialization, system features create their own set of mailboxes and dictate which other features and processes may access the mailboxes. Using these mailboxes, processes export/import and receive/send messages to system wide mailboxes. Before sending a message, the process must import (connect to) a mailbox; before receiving a message, a process must export (enable a connection to) a mailbox.

System Initialization Software

System initialization is the name of a set of features located system-wide that are used to initialize system components. Two features are directly involved in IOU initialization activity: the I/O processor initialization process (IOPIP) and the service unit/IOU interface (SUIF).

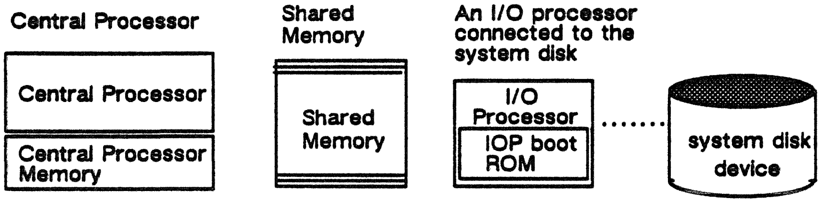


SUIF calls the ROM-resident boot loader to load the IOP initialization images from shared memory to the I/O processor. The boot loads the features in the image and they are then initialized by IOPIP.

Role of I/O Processor in System Initialization

The service unit is used to build the central processor and I/O processor initialization images. Initialization images are the operating and applications software that runs on the processor. Figure 6–7 shows the components required to start the initialization process:

Components used to start system initialization...



Status of components prior to initialization:
- the I/O processor is connected to the system disk device
- the IOP boot program is ROM-resident in the I/O processor
- the system disk device is started

System Initialization software then:
- master clears the central processor and its memory
- patterns shared memory

Figure 6–7. Components used in the system installation process.

After the images are built, they are loaded into shared memory. The service unit now calls the central processor and I/O processor boot programs. The I/O processor boot is ROM-resident on each I/O processor, and, when called, moves the I/O processor initialization image from shared memory into all the I/O processors. IOP initialization software (IOPIP) initializes one I/O processor that runs disk physical file software and brings it to a ready state.

The remaining central processors are initialized, and the now-active I/O processor is used to install the operating system and system files, completing central processor startup. Then the remaining I/O processors are initialized, activated, and promoted to a ready state.

Service Unit / IOU Interface Software

During system initialization, direct communication with the service unit is through the service unit/IOU interface (SUIF). SUIF routes messages from the service unit to the intended I/O processor through either the SIO server/driver or the remote procedure call mechanism.

The SUIF is a synchronous protocol that operates at a transfer rate of 9600 bits per second. After the I/O processor boot starts up the I/O processor, the SUIF is also used for the primary wake-up commands from the service unit that direct the processor to load the IOU software from shared memory.

Protection Software

A generalized resource protection scheme is provided via capabilities management, a low-level software feature. The capability feature provides a general protection mechanism that kernel features can use to authenticate access to the system resources they manage. Throughout the system, this feature is used to control access to system resources. On the IOU, shared memory management software uses the capabilities features to protect access to shared memory objects. Shared memory objects are those areas of the shared memory accessed only by the system. Capability features provide access protection to the connections established for users on the networks. A capability uniquely names an object or resource and the rights required to access the object or resource.

System Monitoring Software

The system monitor is a feature that collects and manages error reports for the ETA10 system. While the system monitor is based largely on the service unit, it interfaces to other system processors. The IOU interface resides on each I/O processor and functions much as a server. This system monitor interface queues error reports from all IOU software including the I/O supervisor, channel software, and device driver software. When the service unit-based error probe makes a polling request, the system monitor interface sends the error reports to the system monitor server for processing and logging. Refer to the "IOU Connections to the Service Unit" section later in this chapter.

I/O Channel Software

Each type of I/O channel provides I/O control and management for a specific type or types of peripheral device or network. Software running in the channel works in conjunction with software running on the central processor. This section describes the software required to provide the specific channel functions, and includes software residing on the central processor as necessary. Software for the three channels includes:

- disk channel software
- FIPS 60-2 channel software
- VME channel software

Disk Channel Software

Software on the disk channel processor tracks disk sectors and addresses per instructions from mainframe software. The disk channel processor executes disk physical-file system requests through the disk channel interface/interface software. The disk channel interface/interface software communicates with the disk channel interface board. Figure 6–8 diagrams the layers of central processor- and IOU-resident software supported by kernel features.

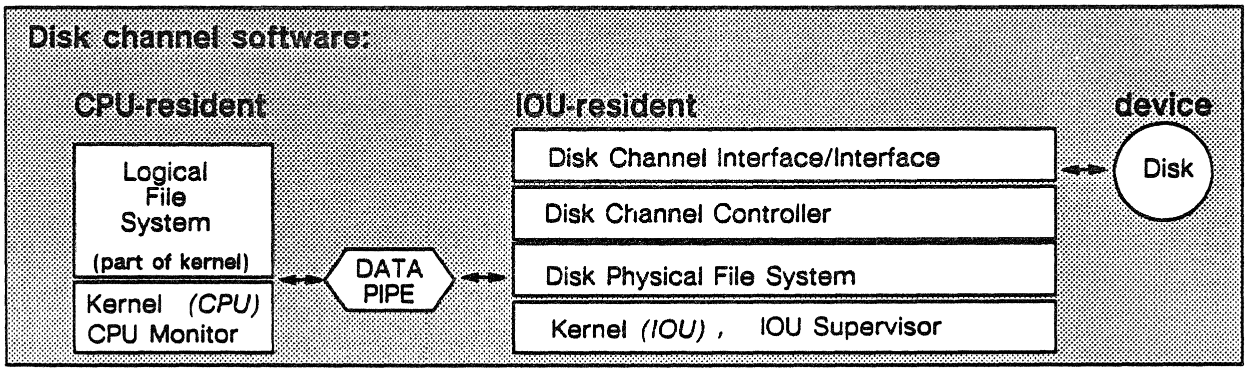


Figure 6–8. Diagram of disk channel – central processor software components.

Logical File System

The logical file system executes on the central processor and provides a system-wide set of file services. It provides uniform file interfaces between user environments and makes file management and identification easier for users. It has the ability to create/destroy and extend/reduce files. As a program spawns files, the logical file system links or groups the physical file identifiers into a user-named file. When users requests files, the logical file system tracks a user-given file name to a physical file identifier.

Disk Physical-File System Software

The disk physical-file system executes primarily on the single board computer in the disk channel processor, although some portion executes on the central processors as well. The disk physical-file system manages access to disk drives and partitions the space on them into physical disk files. When a file is created, the disk physical-file system returns the file's physical file identifier to the logical file system and maintains the record of the file's physical disk location.

Service Unit Requests

The disk physical-file system gets maintenance and configuration status requests from the service unit via the service unit/IOU interface (SUIF). Examples of service unit requests include requesting disk physical-file system to assign a new logical device, to reformat a physical device, or to initialize an I/O processor. SUIF is a sub-feature of the disk physical-file system residing on the service unit.

Disk Channel Controller

The disk channel controller software executes on the single board computer and processes instructions it receives from the disk physical-file system. It directs the disk channel interface/interface software to read and write to/from a physical disk device. It also sends reset and initialization calls to the disk channel interface/interface.

Disk Channel Interface/Interface Software

The disk channel interface/interface software runs on the single board computer of the disk channel processor. Its primary activity is to communicate with the disk channel interface board, hence its name.

During normal I/O operations, the disk channel interface/interface software acts in tandem with the disk channel controller software to provide the low-level interface to the disk physical-file system. As a result, communications between the disk physical-file system and the command modules on the disks are unimpaired.

During diagnostic activity, the disk channel interface/interface acts in tandem with the intelligent disk storage modules to provide the low-level interface between the diagnostic system interface and the disk channel interface board.

FIPS 60-2 Channel Software

The FIPS channel connects the Multi-Host Network, ETA Systems' version of Control Data Corporation's Loosely Coupled Network, to the ETA10.

The Multi-Host Network (MHN) is a communications network with network access devices (NADs) connected to a set of trunk lines. Mainframes attach to the network access devices and communicate with other similarly attached mainframes through the trunk lines. The MHN is used mainly for batch processing. The FIPS channel software is shown in figure 6–9.

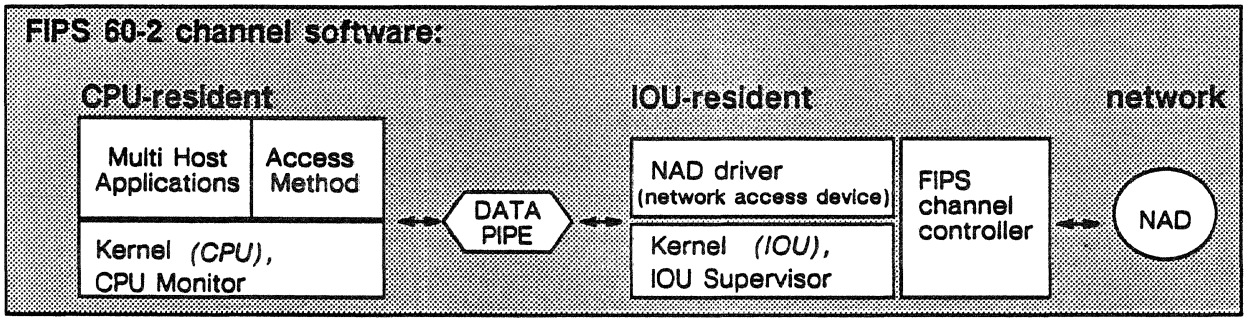


Figure 6–9. FIPS channel software components with central processor-resident software.

Access Method

The access method software establishes logical connections to the multi-host applications software that resides on the central processors. It manages user connections so that outgoing data transfers are sent to the correct IOU and the correct network access driver (NAD) driver on the IOU. For incoming transfers, the access method ensures the data transferred by the NAD is routed to its correct destination. When the access method determines there is too much network activity on the CPU side, it disallows more transfers.

In establishing the user's session, the access method feature is called to set up interfaces between the multi-host application commands and the underlying kernel operating system. When a Multi-Host Network user begins a logon to establish a session on the ETA10, the common login processor feature verifies the user and creates the user's environment shell; this establishes the user's session.

Multi Host Applications

Multi host applications are a set of commands available to any user in a VSOS user environment session. The MFLINK command executes transfers for users logged onto the ETA10 mainframe. MFLINK transfers files to/from a directory on a remote mainframe out on the network. For remote host users to transfer files to/from an ETA10 directory, there must be a server counterpart for the ETA10 command. PTFS is the server used in remote host mainframes. PTFS transfers files to/from the remote host permanent file directory. When the MFLINK command is entered on the remote host, the ETA10 provides the servers.

NAD Driver

The Network Access Device (NAD) driver software has four programs that interface to the NADs on the network. Two programs provide initialization and connection capabilities for the channel. The other programs set up and manage the mailboxes used for system communications, the path mailbox server and the connection mailbox server.

The NAD driver manages network connections from the IOU side. The NAD connected to the ETA10 can accommodate up to 64 paths to the ETA10. If the NAD becomes overloaded, the NAD driver refuses new connections; they are not queued, and remote users are returned a rejection error. The NAD driver transfers incoming files across the data pipe to the shared memory, and queues notifications of same to the Access Method until the process is ready to receive the file.

The object code for the NAD is vendor-supplied.

FIPS 60-2 Channel Controller

The FIPS channel controller software provides the Federal Information Processing Standard 60-2 channel interface and management capabilities for the IOU.

Refer to chapter 7, "Networks", for more information on the Multi-Host Network.

VME Channel Software

The VME channel connects to the Open Interconnection Network, the ETA Systems version of the local area network. Within the channel processor, the VME channel interface provides a connection compatible with the VME bus standard and supports the TCP/IP protocol. The VME channel is also accessible to the smaller class of disk drives.

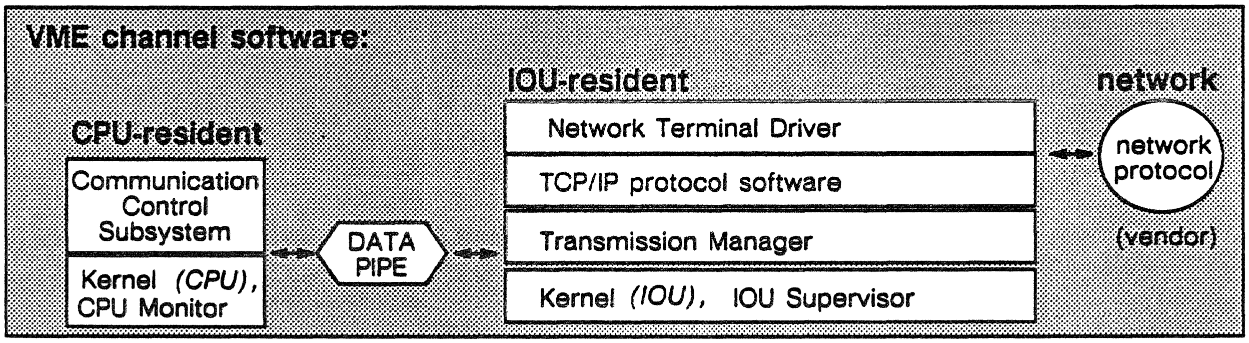


Figure 6-10. VME channel software components with central processor-resident software.

Communication Control Subsystem

The communication control subsystem (CCS) resides and executes on the central processors. CCS interfaces with the user environment shells and controls the communication interface to the IOUs.

CCS is called by the operating system on the CPU to manage the user connections made from an Open Interconnection Network. When a user begins a session via the network, an operating system feature (the common login processor) establishes the connection through CCS and creates the user's session. The CCS provides support for both dumb terminals connected to a terminal server and for intelligent workstations directly connected to the network. The terminal traffic through the TCP/IP protocol is monitored and managed by CCS.

CCS on the central processors and transmission manager software on the IOU function interdependently, exchanging control packets and data packets. A packet is a group of bits that include data and source/destination addresses, packets are sent to and from the network. Using the remote procedure call feature, CCS moves the data packets between the CPU and the IOU. CCS uses the system configuration table to maintain tables of user connections and network load leveling.

Transmission Manager

The purpose of the transmission manager (TM) is to provide the communication control subsystem a method for communicating with the Open Interconnection Network. Transmission manager software provides communication services for the communication control subsystem to fulfill requests to establish and terminate a network connection, and to transfer data over the connection. The set of primitives provided by the transmission manager allows the communication control subsystem to communicate with remote systems that follow the Transmission Control Protocol / Internet Protocol (TCP/IP) standard. The communication control subsystem resides in the central processor.

Transmission manager also provides initialization primitives to aid in the management of the Open Interconnection Network.

FTP and Telnet/Rlogin Commands

FTP and Telnet/Rlogin commands are available from the VSOS user environment. FTP is the file transfer command for the file transfer protocol (FTP) that allows network users to transfer files between network terminals, and from network terminals to ETA10 disks.

Telnet/Rlogin is a Department of Defense protocol that provides a standard method of interfacing terminal devices and terminal-oriented processes to each other. Rlogin is a Berkeley 4.2 facility which allows a terminal on a Berkeley 4.2 machine to connect to a remote machine.

TCP/IP Protocol

The transmission control protocol/internet protocol (TCP/IP) are protocols defined by the Department of Defense and supported by ETA Systems for the Open Interconnection Network (OIN). The transmission control protocol (TCP) is a connection-oriented transport protocol for use in packet-switching communications networks. TCP provides connections and communications between pairs of processes in logically distinct hosts on a single network as well as in hosts on interconnected networks. The internet protocol (IP) supports the interconnection of networks to form an internetwork. IP transmits blocks of data from source hosts to destination hosts located on the same network or located on internetworks. IP does not create connections or logical circuits, and has no mechanism to monitor or control data flow. The connections between networks are provided by gateway devices; IP moves data through these gateways. The object code for the TCP/IP protocol processor board is supplied by the vendor.

TCP/IP Protocol Processor

The TCP protocol processor is a board that plugs into ETA Systems' standard VME bus board, usually referred to as a VME channel interface board. The I/O processor downloads the TCP/IP software onto this board. Thus configured, the TCP/IP protocol processor controls access to the Ethernet and provides the transport and network protocols to route packets through the OIN.

Network Terminal Driver

This software is called by the common login processor to open user connections requested by users running Telnet on a device out on the network. Once the logon is validated, the common login processor starts the environment shell which uses the connection established by the network terminal driver. When a user logs out, the environment shell closes the connection and ends the session. These connections are what the VSOS environment and common login processor use to read/write to the user's terminal.

Refer to chapter 7, "Networks", for more information on the Open Interconnection Network.

Disk Input/Output

The disk input/output system is designed to read at least one complete revolution of the disk without stopping.

Disk File Transfers

To initiate a transfer from disk, the logical file system calls a central processor-resident disk physical-file system feature. This feature, in turn, uses the remote procedure call mechanism to call the IOU-resident disk physical-file system. When the IOU-resident disk physical-file system receives calls from the logical file system, it sorts the logical file calls and passes them to the correct disk channel processor.

To process the transfer, disk physical-file system collects the response from the disk channel controller software, and notifies the logical file system that the files are valid and the transfer can be made. The disk channel controller software verifies file location and other control information. During the transfer, shared memory manager features are extended to the I/O, as shown in figure 6–11.

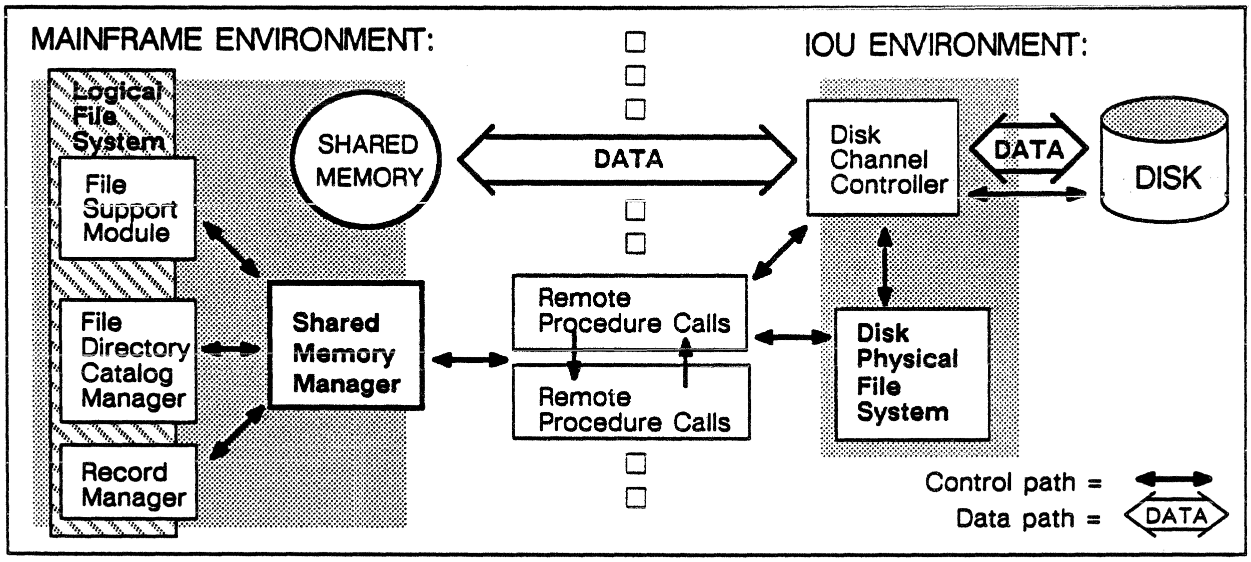
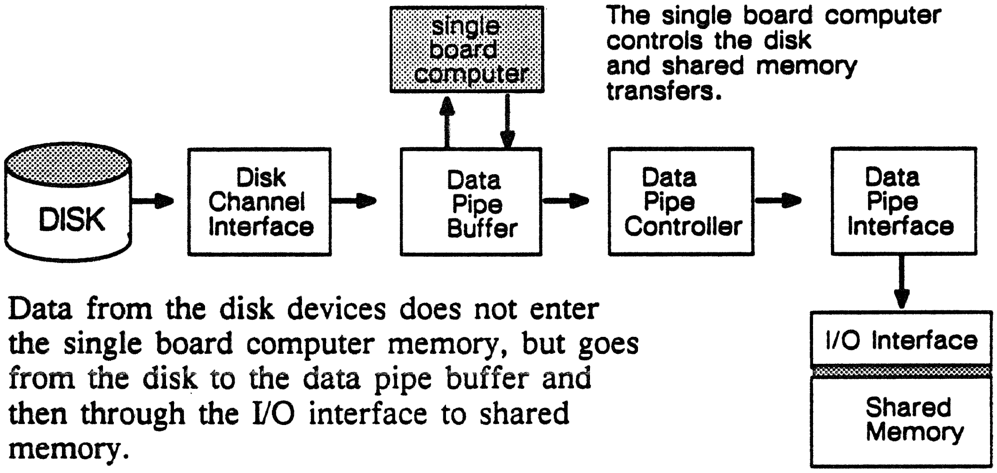


Figure 6–11. Software components involved in file transfers between disk and shared memory.

Communicating via remote procedure calls, disk physical-file system manages the data transfers to shared memory. To read/write files, the disk physical-file system translates the file-relative logical read/write requests into absolute disk device requests, and then commands the appropriate disk channel processor to read from/write to the absolute sectors specified. The shared memory manager features are in control once the the data is transferred over the data pipe. The disk physical-file system provides a primitive file system to the operating system that is little more than a set of allocated spaces.

From this, a sophisticated logical file system is built and provided to system features and users.



The single board computer controls the disk and shared memory transfers.

Data from the disk devices does not enter the single board computer memory, but goes from the disk to the data pipe buffer and then through the I/O interface to shared memory.

Physical File

A physical file created by the disk physical-file system is a specific, ordered list of physical disk sectors which contain data. When sectors are assigned to a physical file, they are not available for any other physical file. Sectors that are not assigned are recorded in a free space map.

Disk Physical-File System Software

The disk physical-file system executes primarily on the single board computer in the disk channel processor although some portion executes on the central processors as well. The disk physical-file system manages access to disk drives and partitions the space on them into physical disk files. When a file is created, disk physical-file system returns the file's physical file identifier to the logical file system and maintains the record of the file's physical disk location.

Physical File Creation

When a user's program requests the creation of a physical file, the request goes to the logical file system (LFS). The logical file system calls the disk physical-file system feature residing on the central processor to create a physical file of a certain length, on a certain logical device. (A physical file is one or more segments of data, stored on a disk, and identified by a physical file identifier.) Using a remote procedure call mailbox, this message is sent to the disk physical file software residing on the I/O processor that manages the specified logical device. This disk physical-file system feature allocates space on the device, updates the free space map, and assigns the file a physical file identifier. The disk physical-file system on the IOU returns a message containing the space allocation data and the physical identifier to disk physical-file system software on the central processor. This feature forwards the physical file identifier to the logical file system. The physical file identifier links to the file's physical location and length. The I/O global memory keeps copies of all of the free space maps for the disks assigned to each IOU.

Writing to Physical Files

When the user's program issues a file write, the data is usually written to shared memory. As shared memory fills, the shared memory manager directs the file to be written out to disk by sending disk physical-file system (CPU) a message indicating the length of the file and the file's physical identifier. The disk physical-file system (CPU) sends this message to the disk physical-file system (IOU) which initiates a shared memory transfer to the appropriate data pipe buffer. Then disk physical-file system (IOU) gets the data written from the buffer out to the location on the disk and updates the free space map for that disk.

Logical Device

Current software supports a single physical device configured as a logical device. A logical device is the grouping of a set of physical disk devices. From one to four rotating mass storage devices may be identified or configured to be one logical device. Together the devices are treated as one addressable device. All members of a logical device are connected to a single IOU, and to its IOU partner when the IOU is paired. From 0 to 100 percent of the physical device's space may be assigned to the logical device to which it belongs. When logical devices are entered in the system configuration, they are assigned a mailbox which is the access path to that device for operating system features.

The disk physical-file system assigns a file to a logical device based upon the user's device class and the space available on the logical devices that correspond to that device class. The disk physical-file system records, updates, and returns the device classes to which a particular logical device belongs. The free space table for each logical device is maintained in the communication buffer. It is kept updated by the disk physical-file system through the system configuration control feature.

Device Class

The device class is a set of performance attributes and allocation permissions or descriptors that is kept in the label of a logical device. When the disk physical-file system constructs a logical device, the device class information is recorded in the device label according to the formats of internal system configuration software. The disk physical-file system assigns a file to a logical device based on the user's device class validation and the space available on the logical device with the corresponding device class. The disk physical-file system records, updates, and returns the device classes to which the logical device belongs. Each logical device has a label that includes a header, a free space map, and allocator tables.

File Characteristics

User Files

Files are grouped by being linked to a directory. Users link files to directories on the basis of granted access permissions. Each directory and file pathname is owned by an individual user name. The primary protection for user files is through the establishment and verification of user access permissions for files. Users may not access a file without specific permission.

File Size

The system does not currently limit file size; file size is limited by disk size. The allocation of space to files is dynamic. The system automatically extends a file while it is being written as long as there is space for it to grow on its logical device (logical devices are described in this section).

File Access and Protection

Users protect and share access to their file pathnames and directories by using access permissions. As the file system receives requests to open, read or write to files, it verifies the requester's access permissions for each type of access request before granting access.

The file system always uses shared memory as a disk cache, and protects the files it moves into and out of shared memory. As needed, the files are mapped for write or read-only into a central processor memory. Within a central processor memory, two user processes do not access the same section of memory at the same time.

Disk Characteristics

The ETA10 system supports non-removable rotating mass storage devices with a capacity of 1.2 gigabytes. These are ISI compatible disk drives, with sustained transfer rates of 86.5 megabits/second.

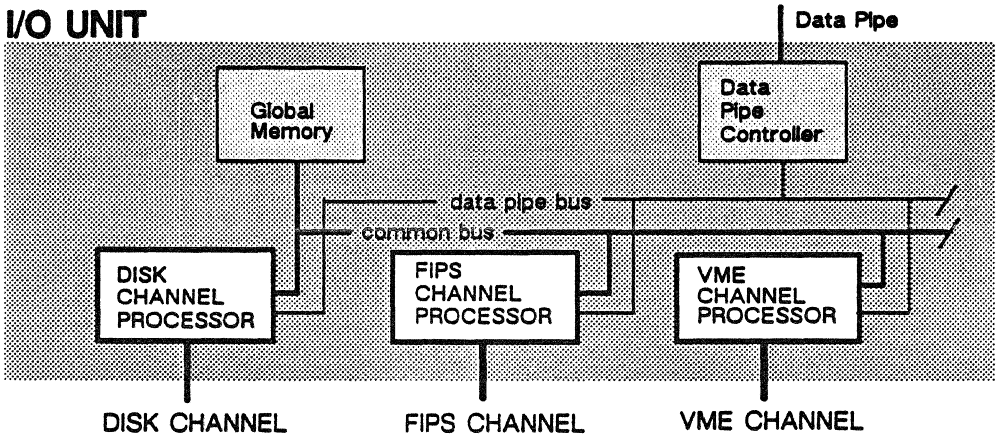
ISI requirements include the following:

- dual channel access
- 30 millisecond (or less) access time
- connection capability of up to eight drives per ISI channel
- an internal and independent device controller
- host controlled self-diagnostic capability

Refer to chapter 8, "Peripherals", for more information about disk characteristics.

Section 3: I/O Hardware

This section describes the channel and device interface hardware, as well as I/O interfaces and connections to other system components.

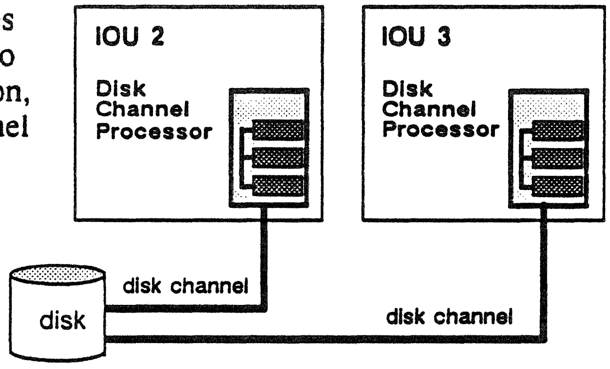


The Disk Channel

The disk channel is the physical control and data path between a set of disk devices and a disk channel processor installed in an I/O unit. Each disk channel processor controls from one to eight disk devices; the eight devices can be daisy-chained to one disk channel. The disk drives are ISI compatible and sustain 86.5 megabits/second transfer rates.

Disks are dual access devices and may be connected to two disk channels and, in addition, controlled by two disk channel processors.

Dual disk connections must be made to two disk channel processors located in different IOUs.



RMS Device 1

While up to eight disk devices can be connected to one disk channel, only one device can be addressed at one time; data can be transferred to or from only one of the devices in a disk channel at any one time. Each step in a transfer is buffered.

Disk Channel Interface

The disk channel interface (DCI) board manages I/O data that is sent to and from the disk channel. It interfaces to ISI-compatible disk devices. The disk channel interface supports absolute sectoring, used by ETA Systems applications where possible.

The disk channel operates at a peak transfer rate of 12 megabytes per second when the disk is in non-interlocked mode. Non-interlocked is a mode of operation of the disk where sync out and sync in signals are pulses, with the information exchanged on the disk channel being synchronized with the leading edge of the appropriate pulse.

The disk channel interface board is mounted with an ALSI 20K gate array chip that handles disk channel interface or ISI protocols.

The FIPS 60-2 Channel

The FIPS channel is the cable that provides the connection between the FIPS 60-2 channel interface and the network control devices attached to the IOU. The FIPS bus and tag cables connect to every peripheral and network control device attached to the channel. A termination is installed at the end of the chain. The FIPS channel is capable of a peak transfer rate of 24 megabits per second. The network access devices transfer at about half that rate.

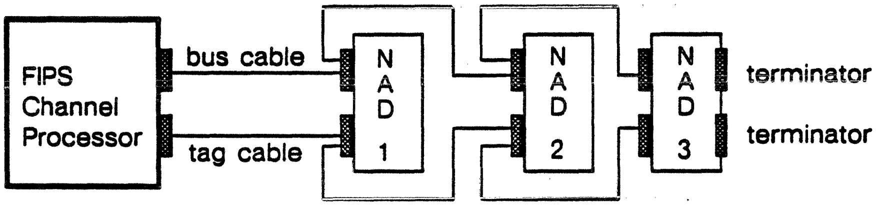


Figure 6–12. Typical attachment of NADs to the FIPS channel processor.

FIPS 60-2 Channel Interface

The FIPS channel interface (FCI) board provides for the transfer of I/O data and activity that conforms to the FIPS 60–2, Federal Information Processing Standards. This standard includes and is maintained over data transfers in streaming mode.

The FIPS channel interface board provides the standard FIPS 60–2 interface between the IOU and the peripheral device or network connected to the FIPS channel. The FIPS channel interface directs the flow of information between I/O devices and the data pipe buffer, and relieves the single board computer of communicating directly with the devices. The FIPS channel interface accepts the control information coming from the single board computer and changes the format into a sequence of signals acceptable to the target control unit and I/O device. When the I/O device sends signals for the single board computer, the FIPS channel interface changes the signals into a format acceptable to the single board computer. The single board computer issues commands to the FIPS channel interface by placing commands in the data pipe buffer.

The VME Channel

The VME channel provides the physical connection through the protocol processor to the Open Interconnection Network (OIN), ETA System's local area network. Protocol processor boards mount in the IOU backplane. The VME cable assembly connects the Ethernet transceiver to the IOU unit connector panel.

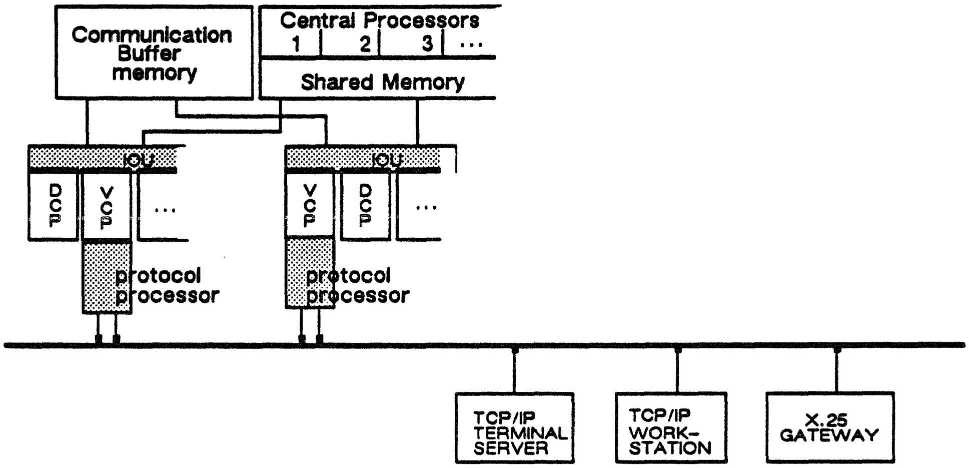


Figure 6–13. Typical attachment of an OIN to the VME channel processor.

The VME channel provides a peak transfer rate of 10 megabits per second from the Open Interconnection Network.

VME Channel Interface

The VME channel interface (VCI) provides management and control of the Open Interconnection Network, a local area network, to which the ETA10 is connected. It manages the I/O data transfers coming through the VME channel from the Open Interconnection Network. If it is installed in I/O processor slot 0, the VME channel interface can also be connected to the service unit SIO maintenance interface lines.

The VME channel interface can consist of up to three standard double-height Eurocard boards able to operate in either master or slave modes.

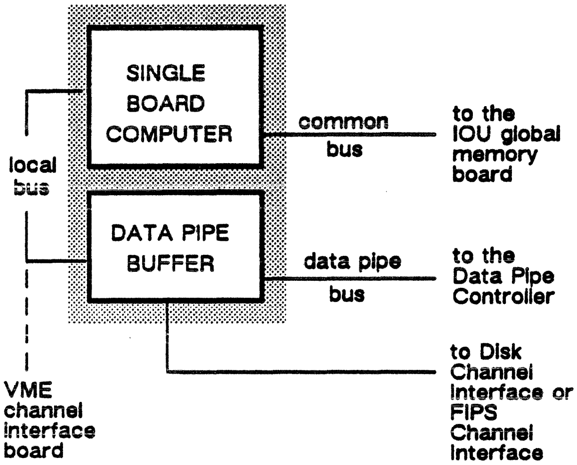
I/O Processor Components

Each IOU channel processor has an I/O processor and a specific type of peripheral or network device interface.

An I/O processor (IOP) is a single board computer and a data pipe buffer linked by a local bus.

The I/O processor works with the device interface to coordinate and manage I/O channel transfers between the mainframe and the system networks and peripherals.

I/O Processor:


SINGLE BOARD COMPUTER
DATA PIPE BUFFER
local bus
common bus
to the IOU global memory board
data pipe bus
to the Data Pipe Controller
to Disk Channel Interface or FIPS Channel Interface
VME channel interface board

The Single Board Computer

The single board computer (SBC) is a standard component of each I/O processor. The IOU supervisor and some system kernel software features provide the basic operating system for the IOU and run on this board. Device driver software such as transmission manager, disk channel controller, and the network access device driver also run on the single board computer.

The IOU uses a high performance single board computer that supports a dual external bus architecture and is optimized for use in a multiprocessor environment. The single board computer provides one megabyte of on-board memory. The board is driven by an MC68020 microprocessor in a pin grid array package. It has two asynchronous serial I/O channels and eight ROM sockets. Device interfaces do not access single board computer internal memory.

The single board computer installed in the "first" I/O processor board slot also connects to a set of serial input/output lines coming from two service unit server nodes. This provides a redundant maintenance path between the service unit and the I/O subsystem.

The Data Pipe Buffer

The data pipe buffer is a standard component of each I/O processor. The data pipe buffer is a memory buffer used as a temporary storage site during data transfers. It compensates for differences in data flow rates between the I/O channel and the fiber optic data pipe.

Its memory is 128K bytes, 16K bytes long by 8 bytes wide. The data pipe buffer interrupts the single board computer whenever a memory parity error is detected.

The data pipe buffer has two control features. One is a set of registers that the single board computer can read or write in order to set up transfers on the data pipe buffer. The other feature is a path for registers that allows the single board computer to enable the channel interfaces to control transfers to and from the devices on their lines. The data pipe buffer communicates with the data pipe controller through transmitted messages across the data pipe bus.

The data pipe buffer board is mounted with a number of ALSI 20K gate array chips to provide memory management capability.

Global Memory Board

The global memory is accessed by all I/O processors over the common bus.

The global memory board provides a 9-megabyte memory accessible by all single board computers via the common bus. In addition, the global memory provides 1 megabyte of PROM accessible from the common bus. Two I/O registers provide detection and correction of single-bit errors, and detection of dual-bit errors (SECDED). Memory accesses of 8-bits, 16-bits, and 32-bits are supported.

The IOU Connection to the Data Pipe

Each IOU is connected to one of the shared memory ports via a data pipe. From the IOU side, the direct access to the data pipe is provided by the IOU's data pipe controller. The data pipe controller controls the flow of data intended for the shared memory or the communication buffer from the I/O processors. From the mainframe side, the I/O interface controls this flow of data into shared memory or the communication buffer. The I/O interfaces communicate with each other. Figure 6–14 shows the data pipe connections to a redundant ETA10 system with two shared memory interfaces:

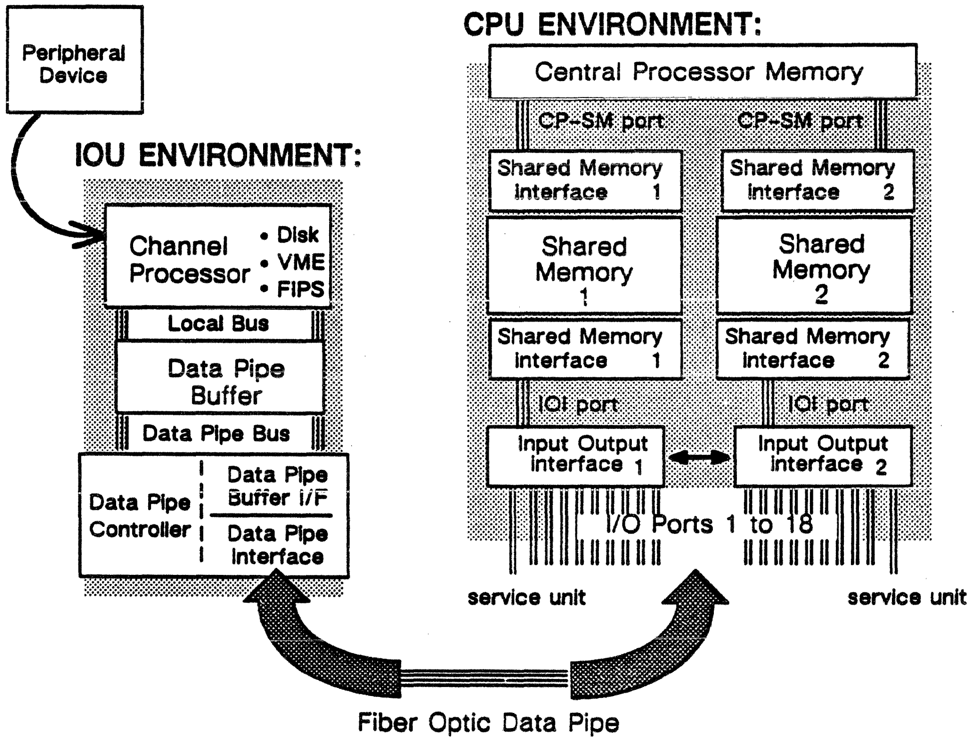


Figure 6–14. Data path provided by the data pipe between an IOU and the shared memory.

Data Pipe

The data pipe is a fiber optic cable that provides the physical connection between an IOU and the I/O interface to shared memory and communication buffer.

Recovery is provided for in the event of a data pipe error. The data pipe controller re-transmits the error block twice. To do this, the data pipe controller instructs the data pipe buffer to "rewind" its data and byte count pointers to the values prior to the error condition. Then the data pipe controller restarts transmission from that point.

Data Pipe Controller

Within the IOU, the data pipe controller controls the flow of information to and from the data pipe.

The data pipe controller is the IOU component that directly connects an IOU to its data pipe. That connection extends across the data pipe to the data pipe interface on the I/O interface (figure 6–15). The I/O interface provides the IOI ports to shared memory and the communication buffer.

All communications and data transfers between the ETA10 mainframe and I/O channel processors go through the data pipe controller. The data pipe buffer in each channel processor connects to the data pipe controller via the data pipe bus.

The data pipe controller arbitrates data pipe buffer bus requests using the round-robin method, and grants use of the bus accordingly. When a transfer header is sent, the data pipe controller receives the header, reformats it, and adds a verification check known as a cyclic redundancy check. As shown in figure 6–15, the data pipe controller receives data from the data pipe buffer memory and transfers it to the I/O Interface across the data pipe. All transfers to the data pipe controller are in 8-byte increments.

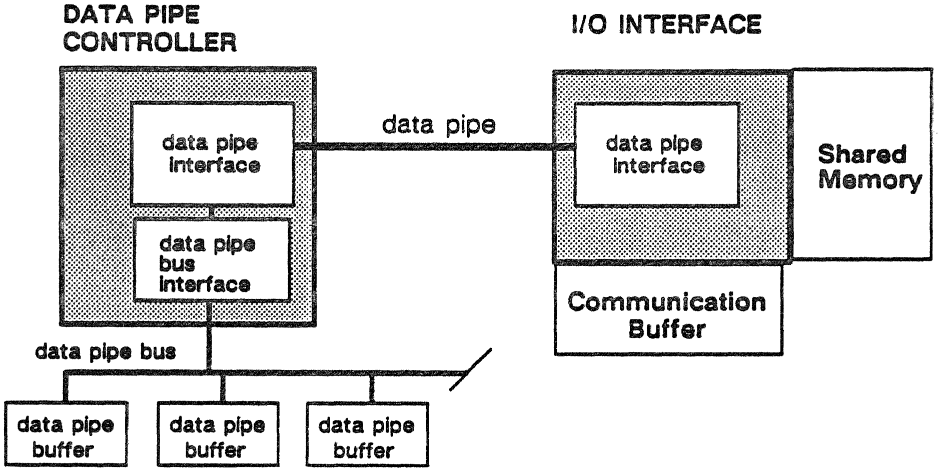


Figure 6–15. Components of the data pipe controller and its connection to the I/O interface.

Data Pipe Interface

The data pipe interfaces are the two boards connected to either end of the fiber optic data pipe cable. One board is on the IOU side in the data pipe controller and a second board is on the mainframe side on the I/O interface assembly.

The data pipe interface serially transports data over the data pipe using separate transmitting and receiving lines. Source data is transferred over a 32-bit data path, across the pipe over a 4-bit data path, and at the destination over a 16-bit data path. This transfer scheme takes advantage of the available bandwidth at the source and ensures data integrity at the destination.

Data Pipe Bus Interface

The data pipe bus interface is circuitry that interconnects and provides compatibility between the data pipe controller and the data pipe buffers. It is a duplex, 32-bit synchronous bus circuit board that plugs into the I/O unit backpanel. It maintains a data transfer rate of one 32-bit word every major cycle. All data pipe buffers are attached to this interface through the data pipe bus. The data pipe bus interface board is mounted with ALSI 20K chips that manage the protocol associated with the fiber optic data pipe.

Data Pipe Buffer

This board is a memory buffer used as a temporary storage site during data transfers, it is connected to and controlled by a single board computer. It compensates for differences in data flow rates between the I/O channels and the data pipe.

One of its main functions is to send control information to the data pipe controller. When an I/O device driver wants to use the communication buffer, shared memory, or the interrupt net, the data pipe buffer composes a header message in a data pipe buffer register file. When a data pipe buffer wants access to the data pipe controller, it makes a bus request to the data pipe controller. The data pipe controller arbitrates bus requests using the round-robin method and grants use of the bus to a data pipe buffer. The data pipe buffer then issues a header instructing the data pipe controller to execute an I/O operation.

Shared memory read and write operations are started when the data pipe buffer issues the appropriate shared memory headers to the data pipe controller. Shared memory read and write operations terminate when the header-specified number of half words are transferred.

The data pipe controller and the data pipe buffer have two modes of operation that relate to each other's timing; transmit mode and receive mode. Transfers between these components occur by timing rather than with handshaking.

The data pipe buffer sends an interrupt to the single board computer when a data pipe operation completes. An error condition associated with a data pipe operation causes that operation to complete.

Data pipe buffer has 128K bytes of memory; 16k bytes long by 8 bytes wide. The data pipe buffer memory address is 16 bits in length and addresses an 8-byte segment. Byte control is provided to allow the single board computer to access bytes, words, and long words.

The I/O Interface to Shared Memory

The I/O interface provides the interface between the I/O subsystem and the ETA10 mainframe, and coordinates all mainframe I/O activity. The I/O interface provides three capabilities for service unit and IOU processes:

- reading and writing shared memory
- performing operations on the communication buffer
- receiving and sending interrupts via the interrupt network

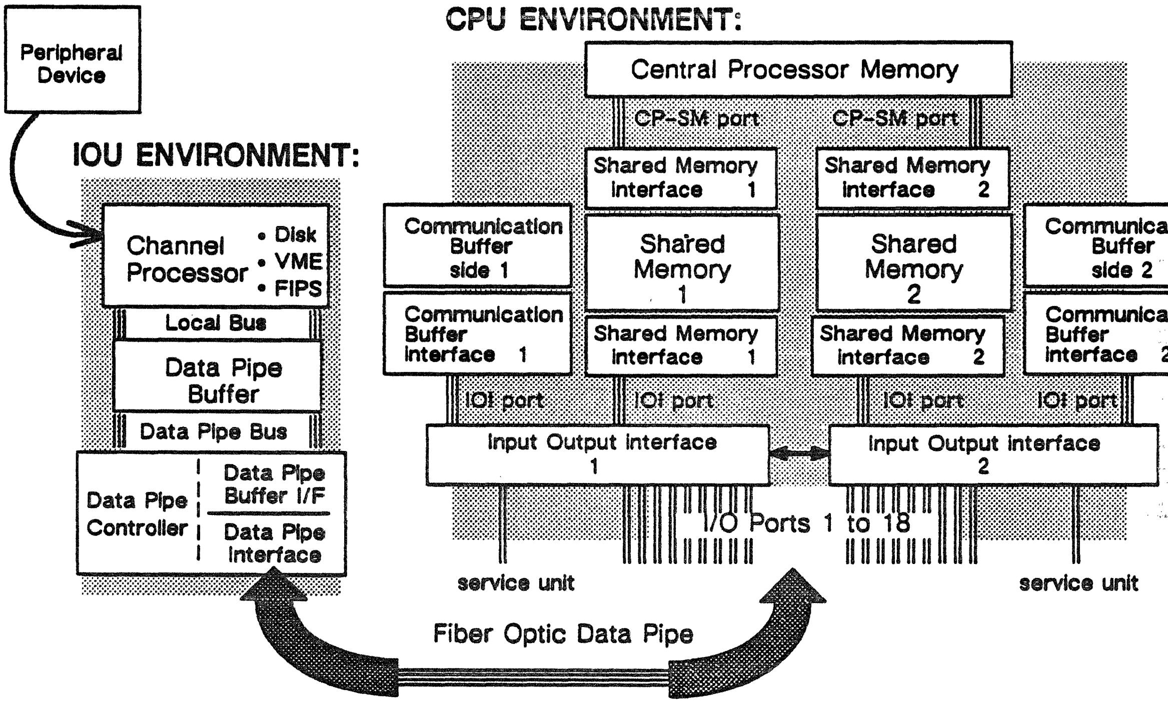


Figure 6-16. IOU access to shared memory and communication buffer.

I/O Interface (IOI)

The I/O interface (IOI) connects the ports from the IOUs to the ETA10 mainframe components. There are two IOIs in a redundant system, they communicate with each other and are powered by a separate power supply to ensure reliable I/O connections to the mainframe. There are 20 ports in a redundant ETA10 system. Each IOI has 10 ports; one port on an IOI is dedicated to the service unit, nine are reserved for IOUs. Each IOI directly interfaces to one shared memory unit and can access the other unit; all 20 ports can access each shared memory unit. In an IOI there are eight shared memory request queues, one for each shared memory board. Each IOI has a dedicated interface to each side of communication buffer.

The IOI passes interrupts between any of the central processors and any of the 20 I/O and service unit ports.

Access to Shared Memory

Shared memory interfaces each have a set of five timing slots that control I/O and central processor access to shared memory units. Four memory slots are reserved for central processor access. One slot is dedicated to the I/O Interface and provides uninterrupted I/O access.

Each IOI board has ten I/O connections; nine go to IOUs across the fiber optic data pipes connecting the IOUs, one goes to the service unit.

Figure 6-16 shows a redundant ETA10 system configured with two units of shared memory, two shared memory interfaces, and two I/O interfaces: SM_1/SM_2, SMI_1/SMI_2, and IOI_1/IOI_2. The memory interfaces, SMI_1 and SMI_2, do not communicate with each other. Because the I/O interfaces do communicate, all twenty I/O connections are able to access either shared memory unit. Buffers in the IOI ports allow concurrent transfers on all I/O connections.

The I/O interface provides a write command to make transfers to shared memory and a read command to make transfers from shared memory. Shared memory diagnostic read and write commands allow the IOU to test the data buffers in the I/O interface ports.

Access to Communication Buffer

The communication buffer is a one or a one-half million word memory reserved for the management of system-wide interprocessor communication. I/O processors access the communication buffer via the I/O interface port as shown in figure 6-16.

Communication buffer units each have a set of five timing slots that control I/O and central processor access to these memories. Four memory slots are reserved for central processor access. One slot is dedicated to the I/O interface and provides uninterrupted I/O access.

The communication buffer command buffers in the I/O interface ports are serviced in a round-robin fashion. Once execution of these commands begins, all of the communication buffer commands in that buffer are processed before service is passed to the next buffer.

Interrupt Network

The interrupt network is a means by which any processor can interrupt any other processor in the system. Any IOU, service unit, or central processor can cause an interrupt to itself or to any other IOU, service unit, or central processor. Interrupts are exchanged through a

48-bit wide central interrupt distribution network. Interprocessor interrupts are distinguishable from internal interrupts.

If an IOU, or service unit, wants to interrupt another processor in the system, the IOU (or service unit) sends the interrupt to its own I/O port in the I/O interface. Each I/O port has a 48-bit interrupt output register that holds the interrupt until the interrupt network says the unit can send it. The interrupt network receives the interrupt and sends it to the intended processor. Each I/O port has another 48-bit interrupt input register that receives interrupts sent to that IOU from other processors. When the I/O interface detects an interrupt waiting in this register, it passes the interrupt as soon as the IOU's or service unit's data pipe permits.

The system-wide interrupt network includes interrupt registers in all system processors; central processors, I/O units, and service unit nodes each have 48-bit interrupt registers.

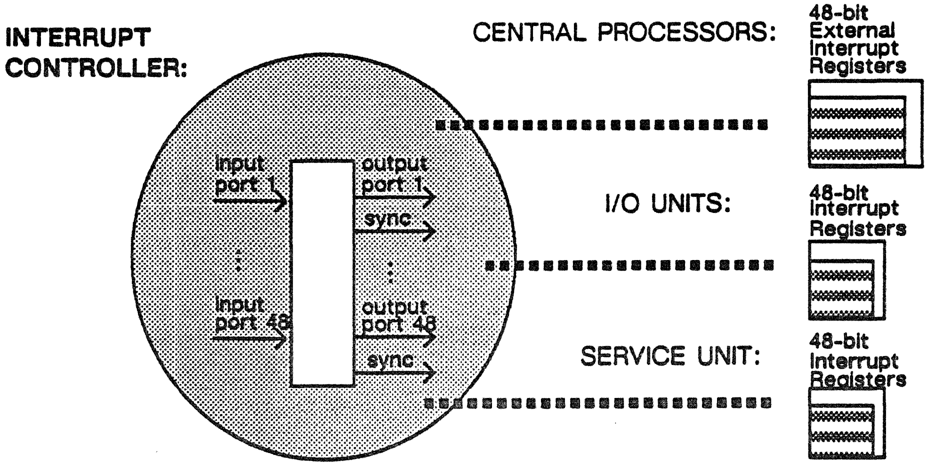


Figure 6–17. The system-wide interrupt network and interrupt controller.

The interrupt controller has 48 input ports and 48 output ports, the output ports are each matched with a synchronization port. Because the network can receive up to 48 interrupt requests simultaneously, the reception of interrupts is sequenced to ensure each is forwarded to the correct processor. Control lines communicate with each input and output port. When an interrupt is clocked in from an input port, control lines shift interrupt data to the correct output port. Some cycles later the interrupt signal is sent through the correct output port to the target processor's interrupt register to be handled by processor software.

The interrupt network is controlled by a single chip located on the communication buffer interface board. In a redundantly configured ETA10 system, there are two communication buffer interface boards, and two interrupt networks. Only one interrupt network is active at any time. The interrupt network is synchronized by the master clock.

IOU Connections to the Service Unit

There are two connections between the IOU and the service unit. One is the IOU maintenance path over which the units directly connect via a serial input/output line. The other is the connection over the system maintenance path which is made through the communication buffer, a memory provided for system communications.

IOU Maintenance Path

The IOU maintenance path provides the diagnostic and communication connection between the service unit and the IOU. A pair of serial input/output (SIO) lines directly connect each IOU with the service unit server nodes to provide the path. Using the SIO lines, the service unit communicates with the IOU and runs IOU-based diagnostics. IOU-based diagnostics execute on the I/O processor and test both I/O and mainframe components. Figure 6–18 shows how diagnostics are run on the IOU from the service unit:

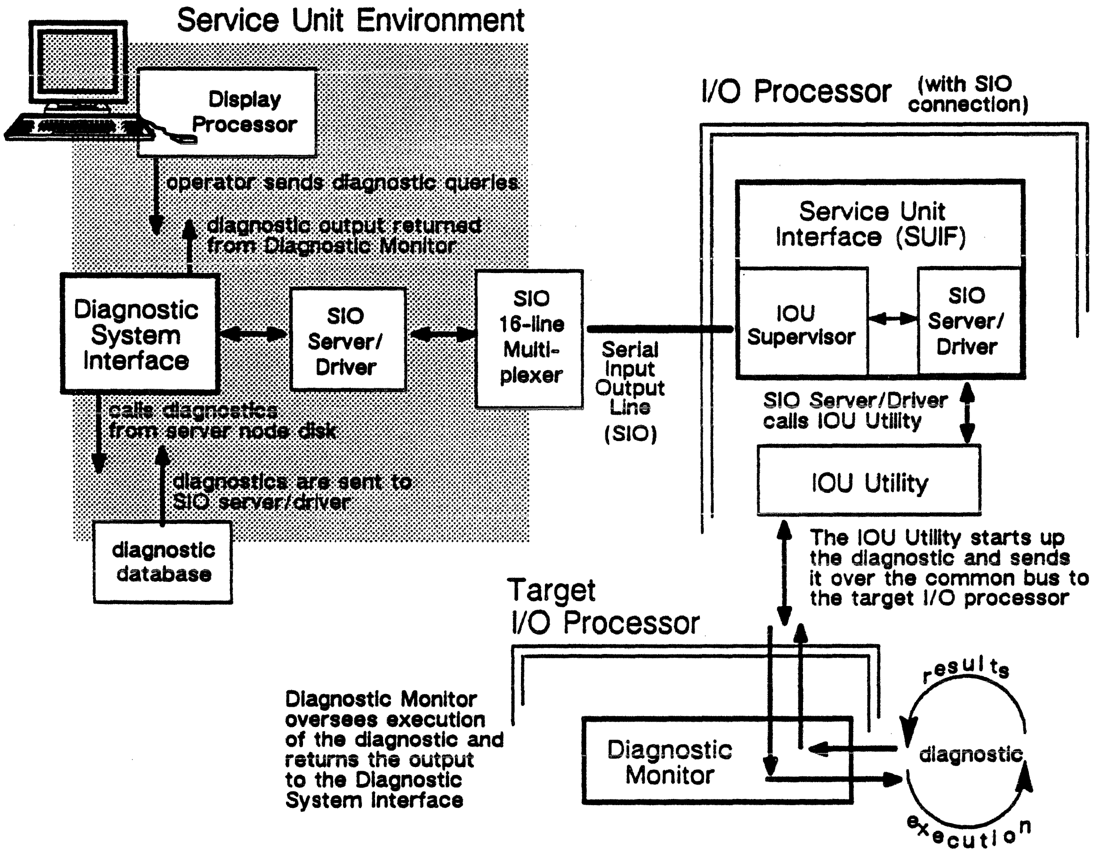

Service Unit Environment
Display Processor
operator sends diagnostic queries
diagnostic output returned from Diagnostic Monitor
Diagnostic System Interface
SIO Server/ Driver
SIO 16-line Multi- plexer
calls diagnostics from server node disk
diagnostics are sent to SIO server/driver
diagnostic database
Serial Input Output Line (SIO)
I/O Processor (with SIO connection)
Service Unit Interface (SUIF)
IOU Supervisor
SIO Server/ Driver
SIO Server/Driver calls IOU Utility
IOU Utility
The IOU Utility starts up the diagnostic and sends it over the common bus to the target I/O processor
Target I/O Processor
Diagnostic Monitor oversees execution of the diagnostic and returns the output to the Diagnostic System Interface
Diagnostic Monitor
results
diagnostic
execution

Figure 6–18. Service unit-based diagnostics using the IOU maintenance path.

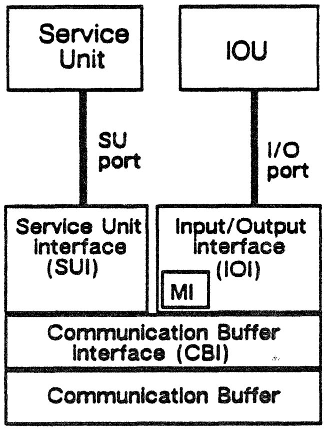
In this example, the sequence starts at the service unit display node when the operator selects a diagnostic to check out an erratic processor. The display processor sends the operator request to the diagnostic system interface, the interface moves the diagnostic from the server node disk and moves it to the SIO server/driver. The diagnostic is transmitted over the SIO line to the service unit interface in the IOU. The SIO server/driver calls the IOU utility feature to start up the diagnostic and send it to the target I/O processor. At this point, the processor's diagnostic monitor oversees the execution of the diagnostic and collects the results. The results are returned back through the maintenance path to the diagnostic system interface and then to the display processor where an operator alarm is issued.

On the IOU, the SIO lines connect to the single board computer in the first I/O channel processor. First in this case refers to the first processor board slot in the IOU chassis. The service unit communicates to the other I/O processors in the IOU over the common bus.

System Maintenance Path

The system maintenance path connects the service unit to the maintenance interface (MI) component built onto several mainframe boards. This maintenance path is used to run service unit-based and central processor-based diagnostics on the mainframe.

The IOU and the service unit can also communicate through the system maintenance path. The IOU sends a message to its IOI port, the message is processed by the communication buffer interface and passed through the SU port to the service unit.


Service Unit
IOU
SU port
I/O port
Service Unit Interface (SUI)
Input/Output Interface (IOI)
MI
Communication Buffer Interface (CBI)
Communication Buffer

These components have maintenance interface logic:

- communication buffer interface
- shared memory interface
- central processors

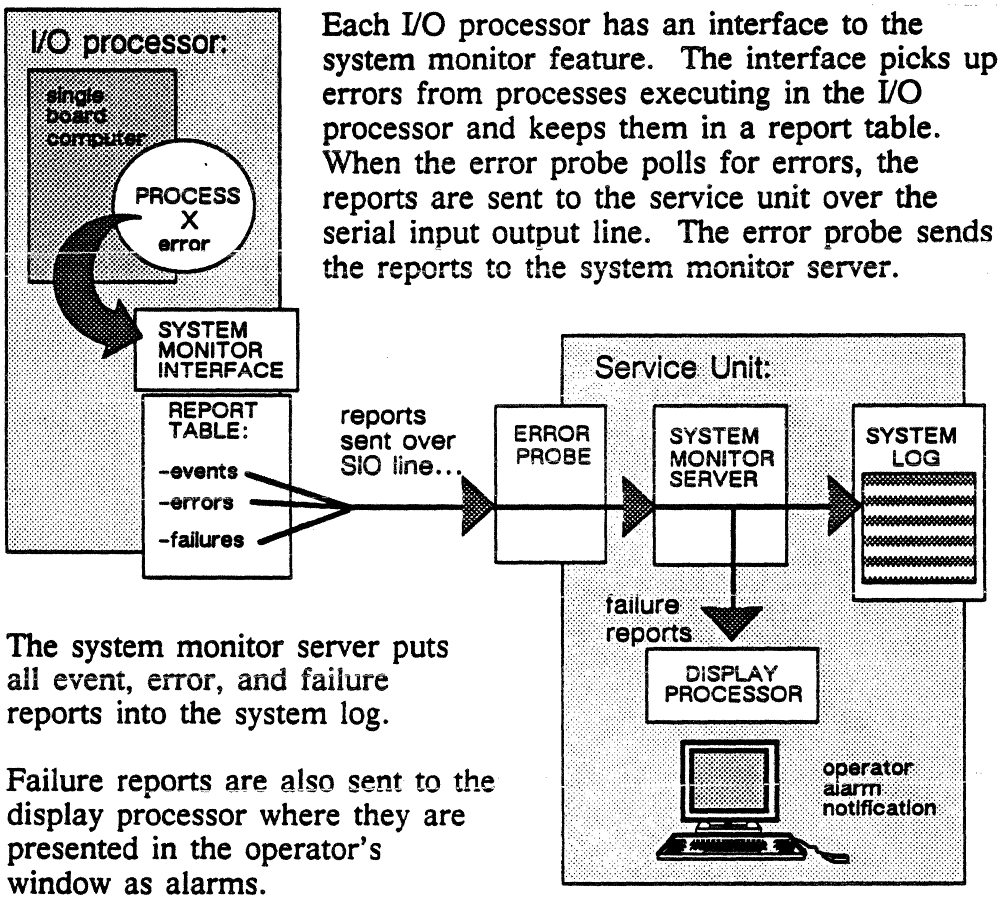
IOU Initialization and Reset Via the Service Unit

A reset of the IOU is command-driven from the service unit via the serial input/output (SIO) connection using the service unit/IOU interface. The reset signal goes to the IOU termination board and from there to each of the I/O processors. A reset signal is also forwarded to each of the device interface boards to initiate their internal reset.

Reset clears the hardware and leaves the IOU in a known condition. Reset does not significantly affect any of the I/O memories (memory in the single board computer, the data pipe buffer, or the I/O global memory). A reset causes the IOU to re-initialize, and the I/O processors to execute the boot loader program ROM-resident on the single board computer.

IOU Error Logging

Each I/O processor has an interface to the system monitor feature. The interface picks up errors from processes executing in the I/O processor and keeps them in a report table. When the error probe polls for errors, the reports are sent to the service unit over the serial input output line. The error probe sends the reports to the system monitor server.



The system monitor server puts all event, error, and failure reports into the system log.

Failure reports are also sent to the display processor where they are presented in the operator's window as alarms.

System features attach severity level flags to each report sent to a system monitor interface. When the flag indicates a failure, those reports are recorded and are also processed as operator alarms. An error report might report a disk sector problem or a single error correction-double error detection (SECDED) error. A failure report would indicate a failed disk device or a fatal double SECDED error.

IOU Bus Architecture

The IOU has three bus systems. All the I/O processors in an IOU share the common bus to access the global memory and for interrupts. Within each I/O processor, the single board computer and the data pipe buffer are connected by a local bus. All the data pipe buffers are connected to the data pipe controller via the data pipe bus.

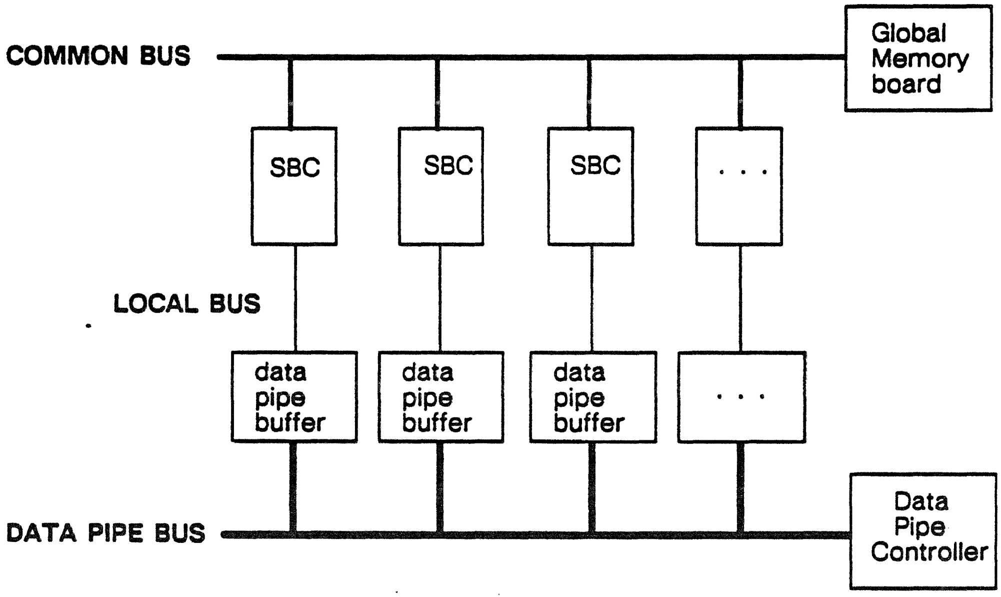


Figure 6–19. The three types of internal IOU buses.

Common Bus

The common bus is a VME bus with 24-bit addressing that connects all single board computers in an IOU and provides a communication path between them. All single board computers access the global memory through the common bus.

Local Bus

The local bus is a VME bus with 24-bit addressing that provides an individual connection between the single board computer and the data pipe buffer board in each I/O processor. In the VME channel processor, the local bus also provides a single board computer-to-VME channel interface connection.

Data Pipe Bus

The data pipe bus connects all data pipe buffers in an IOU to the data pipe controller. All data transfers between the data pipe and the I/O channel processors go through the data pipe bus. The data pipe buffers send and receive messages across the bus to the data pipe controller.

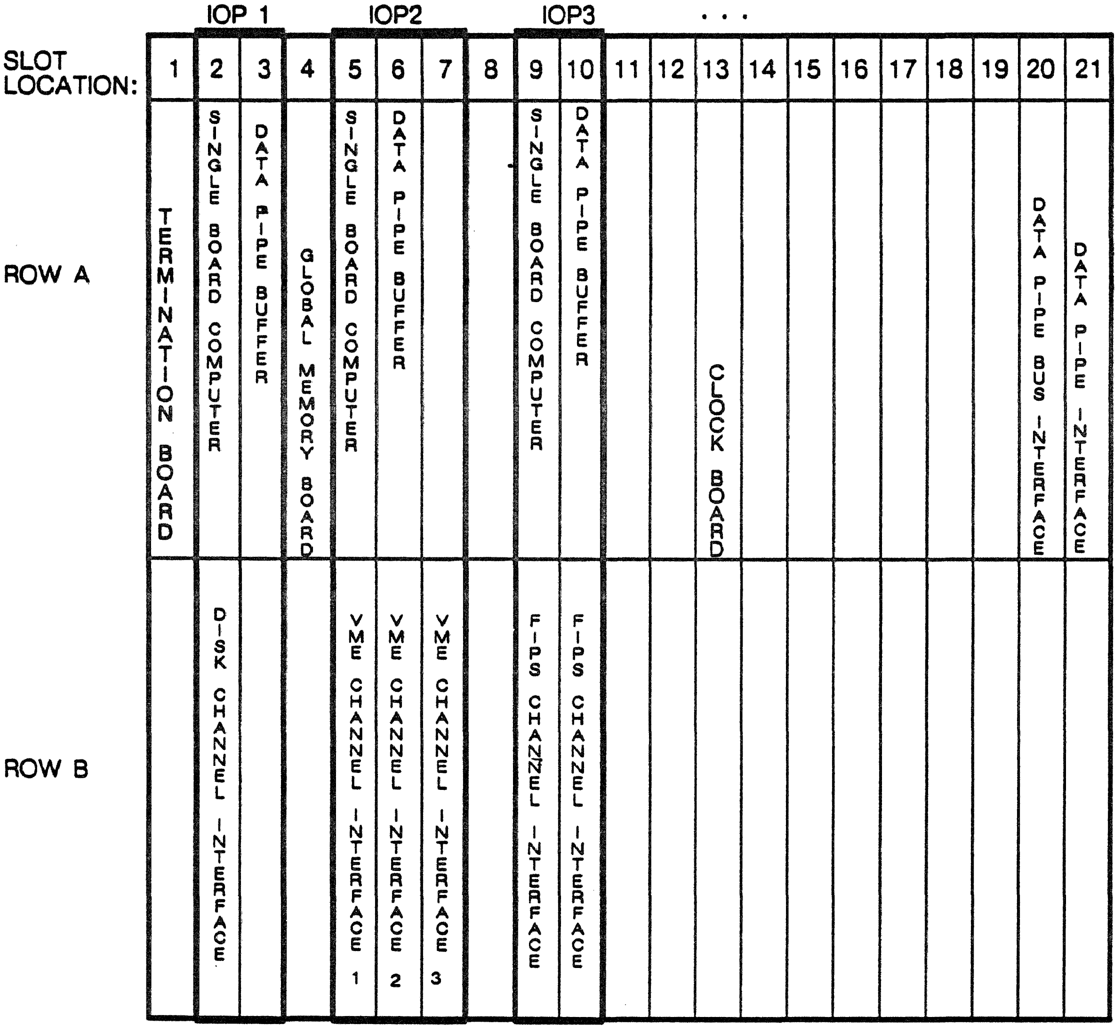
VME Bus Addressing

Words are defined as two bytes in accordance with the VME bus standard. Operand sizes are defined as follows: a byte equals eight bits, a word equals 16 bits, and a long word equals 32 bits. All word addresses are even. Bytes are individually addressable with the high order byte having the same even address as the word in which it resides. The low order byte has an odd address that is one count higher than the address of the word in which it resides. Long words of four bytes are accessed only on alternate word boundaries. If the word or byte address bits are set on a long word or word operation, an error indication appears.

A conversion from a VME address to a data pipe buffer memory address is necessary because the VME bus addresses single bytes and the data pipe buffer memory addresses are on 8-byte boundaries.

The IOU Chassis

The design of the IOU chassis and the IOU boards provides the flexibility to install a variety of device interfaces. The chassis itself is set up as two rows, or backplanes, connected by twisted pair wire wrap. The top row, row A, provides slots for the standard-sized I/O boards (single board computer, global memory, data pipe buffer boards, and so on). Row B provides slots for the device interface boards. The device interface boards are standard-sized also, but they are designed so that a variety of device protocol and other device-specific boards may be plugged into them. Figure 6-20 shows the backplanes with the type of boards installed in each row.



	SLOT LOCATION:	ROW A	ROW B
	1	TERMINATION BOARD	
IOP 1	2	SINGLE BOARD COMPUTER	DISK CHANNEL INTERFACE
	3	DATA PIPE BUFFER	
	4	GLOBAL MEMORY BOARD	
IOP2	5	SINGLE BOARD COMPUTER	VME CHANNEL INTERFACE 1
	6	DATA PIPE BUFFER	VME CHANNEL INTERFACE 2
	7		VME CHANNEL INTERFACE 3
	8		
IOP3	9	SINGLE BOARD COMPUTER	FIPS CHANNEL INTERFACE
	10	DATA PIPE BUFFER	FIPS CHANNEL INTERFACE
	11		
	12		
. . .	13	CLOCK BOARD	
	14		
	15		
	16		
	17		
	18		
	19		
	20	DATA PIPE BUS INTERFACE	
	21	DATA PIPE INTERFACE	

Figure 6-20. IOU backplane, rows A and B, with installed boards.

The basic IOU processor, clock, and memory boards plus the data pipe controller boards are installed in row A, including:

- single board computers
- data pipe buffers
- data pipe interface board
- data pipe bus interface board
- IOU clock board
- global memory board
- expansion memory board
- termination board

The device interface boards are installed in row B:

- VME channel interface boards
- FIPS channel interface boards
- Disk channel interface boards

The IOU cabinet houses the power supply, the cooling system, the two backplanes, the I/O processors, and the device interface boards. One I/O processor in each IOU is also connected to the serial input/output lines from the service unit.

IOU Clock

The IOU runs on a clock that pulses at regular intervals called major cycles. The IOU clock synchronizes data pipe buffer and data pipe buffer interface operations. It provides sequencing pulses that pace the logical gating operations of IOU components. The IOU clock circuit board installs in row A.

IOU clock signals (major cycle) are transmitted to data pipe buffer and data pipe controller logic. Minor cycle signals (1/5 of a major cycle) are transmitted to disk channel interface logic. Single board computers and FIPS interfaces have their own clocks.

CHAPTER 7

Chapter 7

Networks

Chapter Contents

List of Figures

List of Tables

Table *Title* *Page*

Chapter 7

Networks

In This Chapter

Users connect to the ETA10 through two types of networks: the Open Interconnection Network (OIN) and the Loosely Coupled Network (LCN).

The OIN is workstation-based and oriented towards interactive use. It lets users take advantage of the ETA10's ability to handle many small interactive sessions efficiently.

The LCN is mainframe-based, oriented towards large batch jobs. Its high-speed data transfer rate allows file transfers between large computer systems and the ETA10 for batch processing.

- How users access the ETA10 through the OIN.
- Typical ETA10 OIN configuration and hardware.
- The five levels of OIN communication protocols.
- The ETA10 software for the OIN.
- How users transfer files over the LCN.
- The five levels of LCN communication protocols.
- The ETA10 hardware and the LCN configuration.
- The ETA10 software for the LCN.

Section 1: The Open Interconnection Network

The Open Interconnection Network (OIN), which is a type of local area network (LAN), is the workstation-based network for the ETA10. By means of the OIN, users have interactive access to the ETA10 and the ability to transfer files back and forth between the ETA10 and Apollo workstations and other devices. The types of hardware that can be connected to the OIN are described later in this chapter in "OIN Hardware Components".

The Range of the OIN

The Ethernet cable used in an OIN has a maximum length of 2.8 kilometers. However, remote hosts can connect to the network via modems and gateways and their users can have both interactive access and the ability to transfer files. Gateways and modems are discussed later in "OIN Hardware Components", and how a remote login is achieved is discussed in the subsection called, "How Users Access the ETA10 Through the OIN".

OIN Data Transfer Rates

The data transfer rate of the OIN is limited by the capacity of the Ethernet cables, which have a peak transfer rate of 10 million bits per second in a single LAN. The system design goal is to provide an effective transfer rate of approximately 1.5 million bits per second through one TCP/IP protocol processor, which allows a reasonable performance and response time for interactive use.

Actual performance is highly dependent on the applications running on the ETA10 and the response times from other hosts.

TCP/IP and CSMA/CD Communication Protocols

Communication protocols and the methods of data transmission are fixed within each local area network.

Protocols determine how communications are addressed on the network. The OIN conforms its communication to the TCP/IP protocol suite, a standard set of protocols developed by the U. S. Department of Defense and is in wide use on UNIX systems. The TCP in TCP/IP stands for transmission control protocol, and the IP stands for internet protocol. IP provides the format for routing individual messages between internet hosts, and TCP defines the way datagrams are sequenced. The TCP and IP protocols are also discussed later in "Five Levels of OIN Communications Protocols", which appears later in this section, and in volume two of the *DDN Protocol Handbook*, which is listed at the end this document along with other OIN-relevant documentation. Together TCP and IP enable the ETA OIN to provide a TCP/IP stream service, which is used by the ETA Systems proprietary RETA remote login software and LANCOPY file transfer services. The RETA remote login and LANCOPY file transfer services are described in "How Users Access the ETA10 Through the OIN", later in this section.

The network software establishes a socket (port) in the initiating TCP/IP host and in a remote TCP/IP host to define the end points for a TCP/IP service called STREAMS. A connection is made between the two sockets, and communication commences. The ETA10 completes connections initiated by remote ETA10 login RETA software operating in Apollo gateways.

The actual delivery of messages over Ethernet is based on the Ethernet method of data transmission, reflected in the 802.3 standard of the Institute of Electrical and Electronic Engineers (IEEE), otherwise known as the IEEE 802.3 CSMA/CD. CSMA/CD stands for the type of arbitration mechanism: *carrier sensing multiple access with collision detection.*

The Ethernet serves as the delivery system for the TCP/IP and other networking protocols. The ETA10 transfers TCP packets to and from Apollo gateways within IP packets, which are in turn inserted into Ethernet frames.

Typical ETA10 OIN Configuration

In the sample ETA10 OIN configuration in figure 7-1, a TCP/IP protocol processor in the I/O unit (IOU) of the ETA10 computer connects to an Ethernet bus, which in turn connects to a gateway. The gateway can consist of an Apollo DSP90 node with a TCP/IP Ethernet board installed. The gateway in figure 7-1 connects to an Apollo Domain Ring and to an Ethernet network. Explanations of the network hardware and the TCP/IP protocols appear in other parts of this section.

Typical Apollo Domain Ring Configuration

The Apollo Domain Ring connected to the gateway in figure 7-1 connects to a box that stands for one or more Apollo workstations. A modem or modems allow communication with one or more remote Apollo workstations over telephone lines.

Typical Ethernet Network Configuration

Also as shown in figure 7-1, the gateway can be connected to a Sun workstation and any number of communication servers, file servers, and print servers, which, in turn, connect the ETA10 to personal computers, asynchronous terminals, modems and printers.

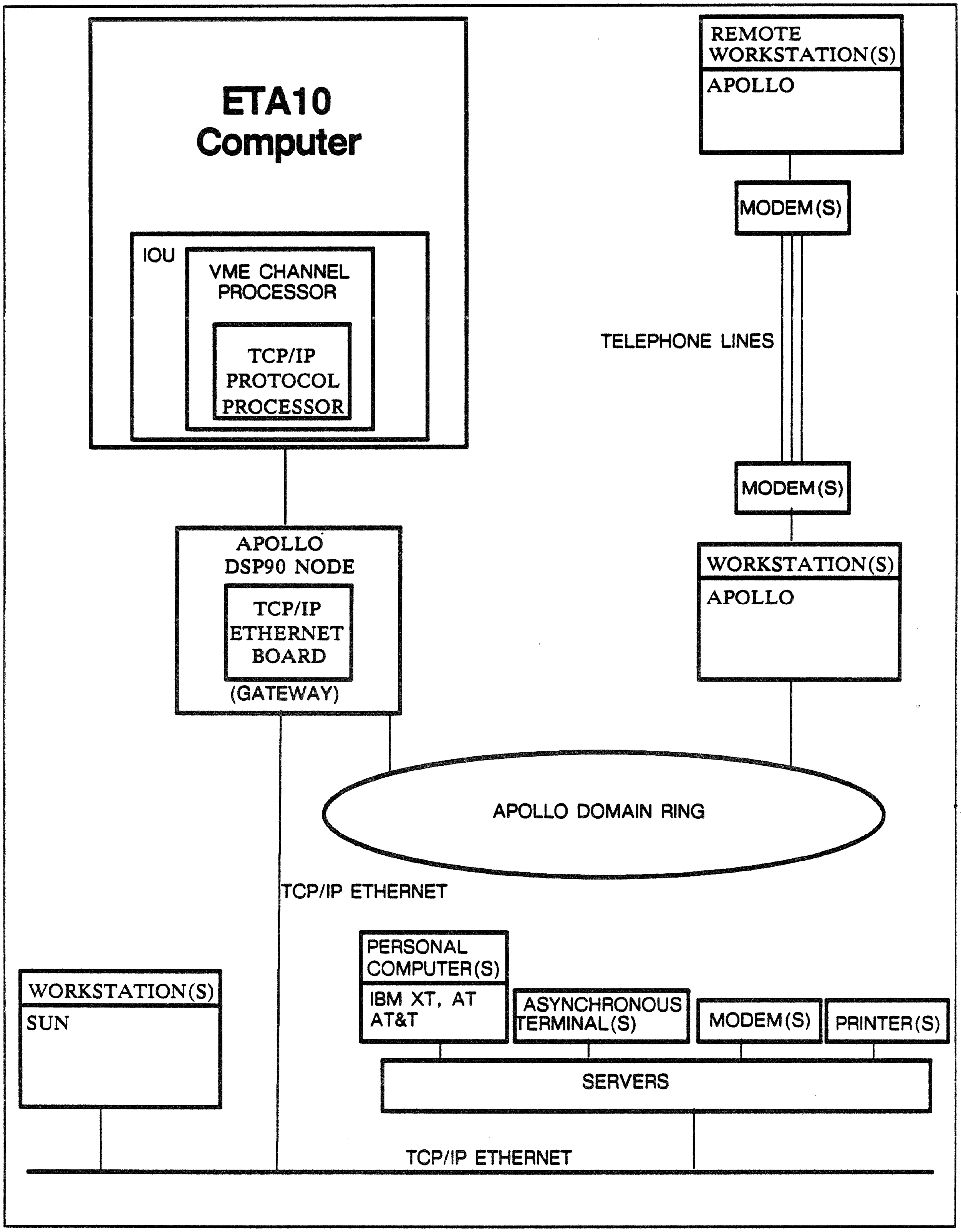


Figure 7-1. Sample Open Interconnection Network connections.

The OIN Hardware Components

Gateways

The term "gateway" means a device that makes communications possible between devices that have differing communication protocols. A gateway in the ETA10 OIN network interfaces the Apollo Domain Ring with the ETA10, and it interfaces devices on the Ethernet local area network with the ETA10. The gateway can also interface the ETA10 with remote hosts on other networks. A gateway in the OIN can consist of an Apollo DSP90 node with a TCP/IP Ethernet board.

An Apollo gateway supports a limited number of concurrent processes, typically 24. Two processes are required by each invocation of the remote login software (RETA), and this limits the number of concurrent users logged in to the ETA10 through a given Apollo gateway. RETA software is described later in this section.

Apollo Workstation Components and Capabilities

Each Apollo workstation includes a keyboard, CRT display, central processor, memory, and memory management hardware. Some workstations are connected to disks that are used for local file storage, while some workstations are without disks. Other Apollo nodes are free-standing and act as file servers. Nodes without disks make use of demand paging to obtain access to files stored on file servers or local storage nodes.

Apollo Workstation Connections to the ETA10

Apollo workstations are interconnected by means of a serial I/O Apollo Domain Ring. Remote Apollo workstations connect via telephone lines and modems through local Apollo workstations to the ETA10.

Other Devices Connected to the ETA10

In addition to the Apollo Domain Ring interface, the gateway provides the interface between the ETA10 computer and other types of devices connected to the Ethernet cable, including SUN workstations and servers. The servers make it possible for personal computers, asynchronous terminals, storage devices, modems and printers to be connected to the OIN.

How the VME Channel Processor Controls the OIN

The ETA10's CPUs run the communication control subsystem (CCS) software that directs OIN activity. The path linking the CPUs with the OIN's Ethernet leads through shared memory (SM) and the communication buffer (CB) to CB and SM interfaces. The path continues, by way of fiber optic data pipes, to the data pipe buffers in VME channel processors in the I/O units. Each VME channel processor has a data pipe buffer and single board computer. The single board computer runs the transmission manager (TM) software, and also interfaces to the service unit (SU). The purpose of this subsection is to describe the role of the VME channel processor and its components in controlling the OIN.

The CPUs, shared memory, communication buffer, SM and CB interfaces, data pipe controllers, and the VME channel processors are discussed in greater detail in chapter 5, "Mainframe Components" and chapter 6, "I/O Subsystem". The communication control subsystem, along with the transmission manager software that is mentioned in the following subsection, is presented in "ETA10 Software for the OIN", later in this section.

The VME Channel Processor is a Type of I/O Processor

As noted in chapter 6, the I/O unit cabinet can house any of a number of different types of I/O processors. The OIN requires that a certain type of I/O processor - a VME channel processor - be installed in the I/O unit.

Components of the VME Channel Processor

Besides the data pipe buffer and single board computer mentioned above, each VME channel processor also has a VME channel interface board with a TCP/IP protocol processor installed. The VME channel interface is controlled by the single board computer indirectly through the data pipe buffer. The TCP/IP protocol processor is directed through the VME channel interface. The TCP/IP protocol processor provides the physical and electrical interface to the Ethernet.

VME Channel Interface Board and the TCP/IP Protocol Processor

Data flow between the VME channel processor's data pipe buffer and the TCP/IP protocol processor is regulated by the IEEE P1014-standard VME bus protocol. The VME channel interface board in the VME channel processor physically holds the smaller TCP/IP protocol processor board. Figure 7-2 shows the TCP/IP protocol processor and its connection to the OIN.

The typical components of a TCP/IP protocol processor that are shown in figure 7-2 are described in the following subsection. The TCP/IP protocol processor is described in great detail in the vendor's *ENP-10 User's Guide,* from Communication Machinery Corporation.

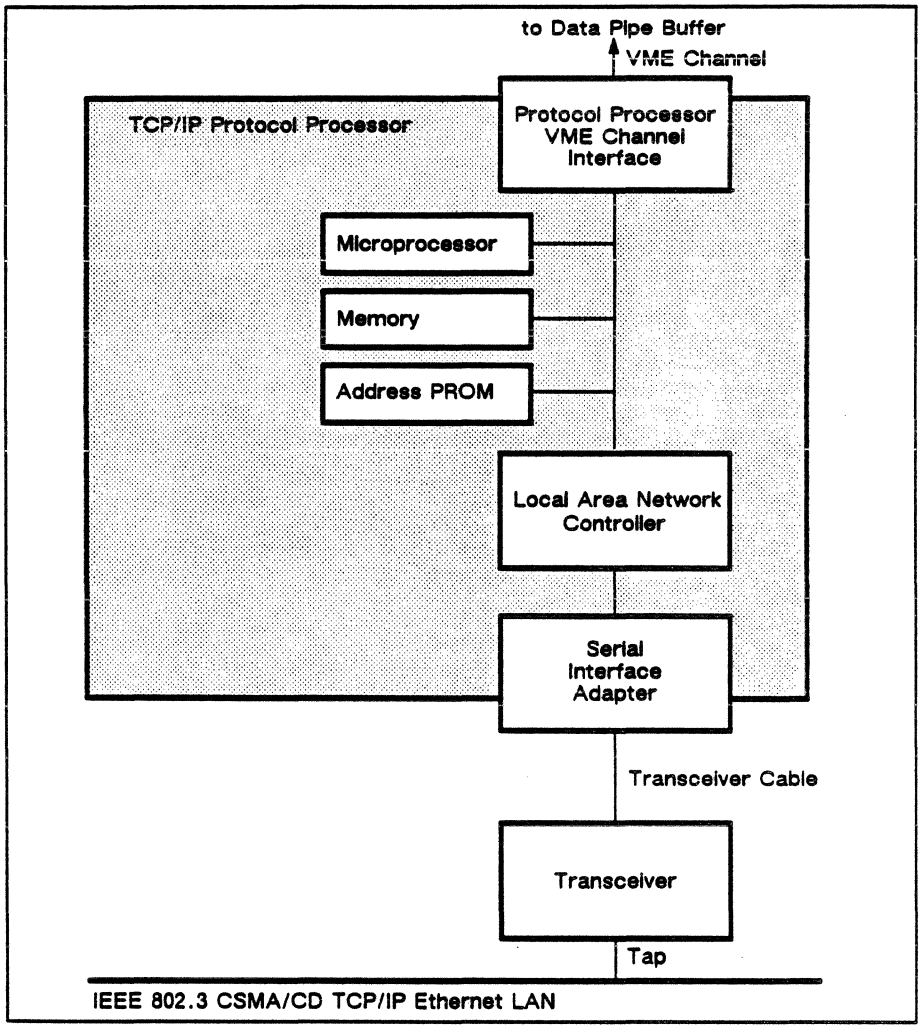

to Data Pipe Buffer
VME Channel
TCP/IP Protocol Processor
Protocol Processor VME Channel Interface
Microprocessor
Memory
Address PROM
Local Area Network Controller
Serial Interface Adapter
Transceiver Cable
Transceiver
Tap
IEEE 802.3 CSMA/CD TCP/IP Ethernet LAN

Figure 7-2. TCP/IP protocol processor with its connection to the OIN.

TCP/IP Protocol Processor Components

VME Channel Interface

The TCP/IP protocol processor board physically connects to the VME channel interface board through its own VME channel interface. The protocol processor is capable of acting as both master and slave on the VME bus.

Microprocessor

A microprocessor controls the TCP/IP protocol processor. It moves commands and data to and from memory. It implements Ethernet addressing protocol, and mediates communications between the VME channel and the local area network controller that also resides on the board.

Memory

The memory holds the Ethernet addressing protocol software and the buffered Ethernet data packets.

Address PROM

The Ethernet address for the ETA10 is held in the address PROM. The Ethernet address is a unique world-wide 48-bit address assigned to every Ethernet station by the Ethernet Address Administration Office at Xerox Corporation.

Local Area Network Controller

The local area network controller transmits and receives data packets from the local area network. It implements the CSMA/CD algorithm.

Serial Interface Adapter

The serial interface adapter performs encoding and decoding for all communications with the Ethernet local area network.

Transceiver

The TCP/IP protocol processor in the I/O unit cabinet is directly connected to the Ethernet local area network by a transceiver cable. At the other end of this cable, the transceiver performs electrical signal line driving and receiving functions. The transceiver taps directly to the Ethernet cable.

Five Levels of OIN Communication Protocols

TCP/IP and the additional standard protocols used by the ETA10 are defined at five levels as shown in figure 7-3.

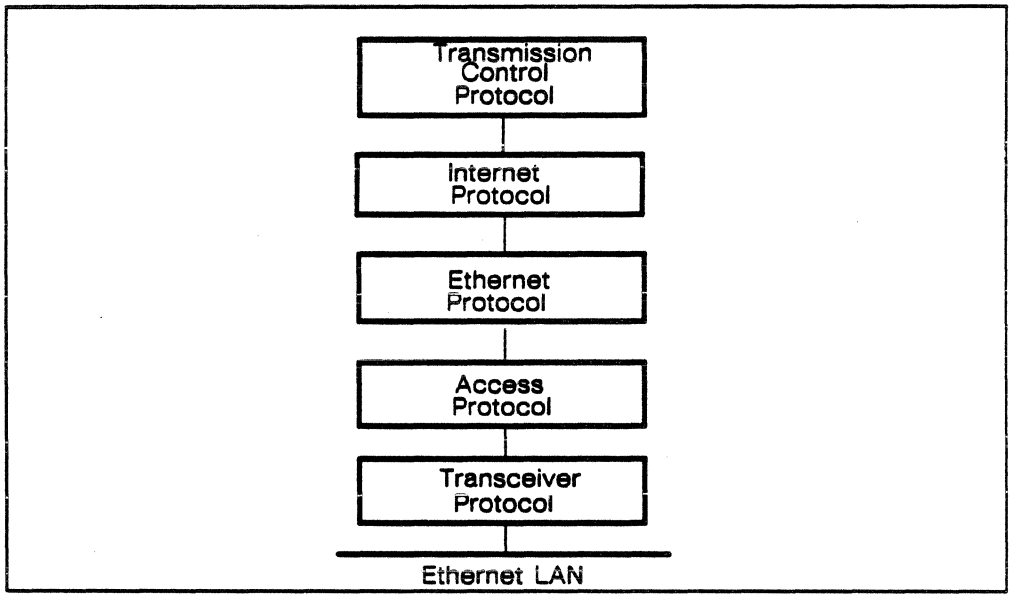


Figure 7-3. Five levels of OIN communication protocols.

Transmission Control Protocol and Internet Protocol

Together the Internet Protocol (IP) and Transmission Control Protocol (TCP) make up the TCP/IP protocol suite, which is discussed elsewhere in this section.

The ETA10 transfers TCP packets to and from Apollo gateways within IP packets inserted in Ethernet frames.

Internet Protocol

The IP packet used to transfer data between local area networks follows the United States Department of Defense Internet Protocol (IP) standards. It carries inter-network routing information.

Each IP packet holds a complete TCP packet or a fragment thereof. The IP packet contains 24 to 1476 bytes of TCP packet data in addition to 24 bytes of IP data fields. The IP data fields specify the transmitting and receiving host and network addresses, the maximum time allowed for packet transmission and acknowledgment, the TCP packet length and fragmentation, the TCP packet sequence number, and the frame check sequence used by the receiving host to validate the IP data fields.

Transmission Control Protocol

The TCP defines the way TCP/IP packets are sequenced to form a reliable TCP/IP byte stream service. Each IP packet is called a segment. TCP packets are inserted into IP segments.

In addition to data fields, the TCP packets include fields that specify transmitting and receiving host ports, packet sequence number, and synchronize, acknowledgment, and close connection flags. Other fields specify the way the packet data is buffered by transmitting and receiving hosts. The data fields of each TCP packet total 24 bytes of information. Up to 1452 bytes of data is permitted.

The Ethernet, Access, and Transceiver Protocols

The Ethernet protocol, the access protocol, and the transceiver protocol make up the bottom three layers in the five layer protocol model for the OIN.

Ethernet Protocol

The Ethernet protocol defines the frame used to transfer TCP/IP packets over Ethernet local area networks. The Ethernet frame serves as an envelope to carry the TCP/IP packet for delivery from a transmitting host to a receiving host. The ETA10 transfers TCP/IP packets to and from Apollo gateways in Ethernet frames.

Access Protocol

The ETA10 follows the CSMA/CD protocol for transmitting across the Ethernet. Because each local area network determines which access protocol it follow, differing access protocols are possible. Without a gateway, the ETA10 will not interface directly to other local area networks that use token ring or other access protocols.

Transceiver Protocol

The transceiver protocol defines the the Manchester format bit-serial encoding and decoding interface to the local area network, which determines the way data is transferred on the bit-serial transceiver cable to and from the Ethernet. There are three different Ethernet transceiver protocols supported by the ETA10. Each encodes the bit stream in Manchester format. A direct current offset is used to signal bit values while the serial line is active.

How Users Access the ETA10 Through the OIN

The user on the OIN communicates with the ETA10 by means of ETA System's proprietary remote ETA10 login (RETA) software in the Apollo gateways, which conforms to the TCP/IP communication protocols. The remote login service of RETA enables the user to login and issue commands to the ETA10.

The LANCOPY file transfer utility on the ETA10 enables the user to transfer files between the ETA10 and the Apollo Domain Ring.

The Apollo gateway extends the ETA10 login and file transfer services to other devices on the local area network and to remote hosts on other local area networks that are connected to the gateway. Remote users establish a connection to the Apollo gateway, transfer their files to the Apollo Domain Ring, and then invoke RETA to login, to enter ETA10 commands, and to relay their files to the ETA10.

The User Entry and Display Station

Apollo workstations and other types of workstations and terminals provide the human interface to the ETA10. A keyboard enables the user to enter commands. A display shows the user prompts that indicate when to enter commands and echos the commands as they are entered. The user may also be able to create and save source files at the workstation or other local device.

Steps for Users to Access the ETA10

Several simple commands enable the user to initiate an interactive session and transfer files to and from the ETA10. The following steps outline the use of these commands, which are described in more detail in chapter 2, "User Environment" in this manual.

Steps for Users at Apollo Workstations

To obtain interactive access and to transfer files to the ETA10 through the OIN, users at Apollo workstations must go through the following steps:

1. Create a process on a gateway node. Users may use the *crp* command to access the node and create the process. Users will have to consult their Apollo administrators for node numbers and information about how to proceed and refer to the Apollo documentation for other necessary information.

2. Enter the *reta* command to establish a connection and log in to the ETA10. The RETA utility prompts the user to enter a login name and password. The ETA10 validates the login and creates a user environment. After validation, commands entered by the user are treated as commands to the ETA10 operating system.

3. To transfer a file to the ETA10 from the Apollo Domain Ring the user enters the LANCOPY command. The user enters additional commands to run whatever applications are required during the session. When finished, the user enters *bye* to terminate the ETA10 session and connection, followed by *logout* to terminate the remote login and sever connection to the Apollo gateway.

Steps for Users at Other Devices

To obtain interactive access and transfer files to the ETA10 through the OIN, users at devices other than Apollo workstations must take the following steps:

1. Users at devices other than Apollo workstations typically use the TCP/IP-based *telnet* utility to open a virtual terminal connection to the gateway node. These users should consult their network administrators and the documentation that supports the device they are using. For example, users at SUN workstations should refer to the SUN networking documentation and the manual page for the *telnet* command.

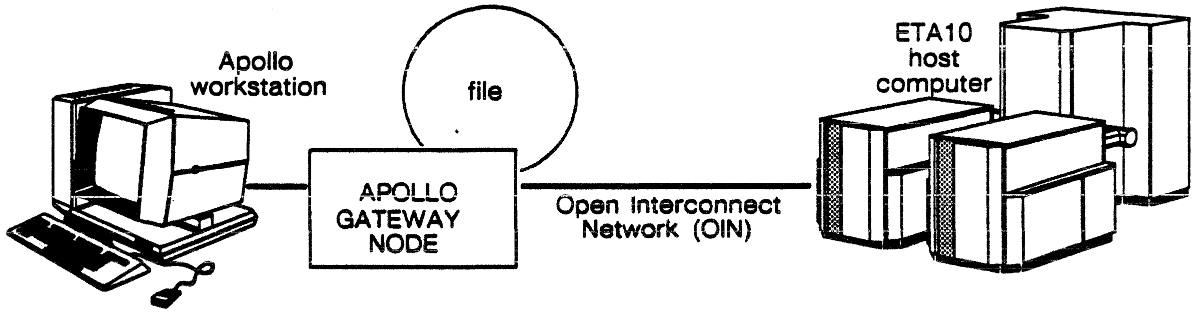
2. Enter the *reta* command to establish a connection and log in to the ETA10. The RETA utility prompts the user to enter a login name and password. The ETA10 validates the login and creates a user environment. After validation, commands entered by the user are treated as commands to the ETA10 operating system.

3. From other entry and display stations the user uses the available file transfer utilities to transfer the files to be processed on the ETA10 to the Apollo Domain Ring.

4. To transfer a file to the ETA10 from the Apollo Domain Ring the user enters the LANCOPY command. The file may be written to the ETA10 with the (T=W) argument or read from the ETA10 with the (T=R) argument.The user enters additional commands to run whatever applications are required during the session. When finished, the user enters BYE to terminate the ETA10 session and connection. When the telnet utility is used, the *logout* command terminates the remote login and breaks the connection to the Apollo gateway. The actual format of this command is dependent on the device used. The user should refer to the specific networking documentation for the device used.

Five Commands from an Apollo Workstation

Figure 7–4 shows five user commands entered at an Apollo workstation and the operations at the ETA10 which conduct the user session. User prompts and minor command options are not shown.



*1 `crp -on gateway_node_number -me`
(creates remote process in Apollo gateway node)

2 `reta ETA10_host_name` ——→ VSOS user environment
(username, password) (makes connection to ETA10)

3 `LANCOPY, T=W I=ETA10_filename, O=Apollo_filename` ——→ file

Intervening commands to the
ETA10 operating system
are entered here

4 `BYE` ——→ (closes user environment)
(closes connection to ETA10) (logs out of ETA10)

*5 `logout`
(closes remote process in Apollo gateway node)

*The first and last commands may or may not be required depending on the Apollo configuration.

Figure 7–4. Sample Apollo workstation dialogue to operate the ETA10.

ETA10 Software for the OIN

The software components essential for ETA10 service to the OIN are shown in figure 7-5, which follows. The ETA10 CPUs execute several of these software components, which are listed below:

- Communication control subsystem.
- User management.
- User environment.
- User programs.
- Operating system utilities.
- LANCOPY file transfer utility.

The single board computer and the TCP/IP protocol processor in the VME channel processor execute additional software components, which are also essential for ETA10 service to the OIN. These components are as follows:

- Transmission manager (executed by the single board computers).
- TCP/IP protocol software (executed by the TCP/IP protocol processors).

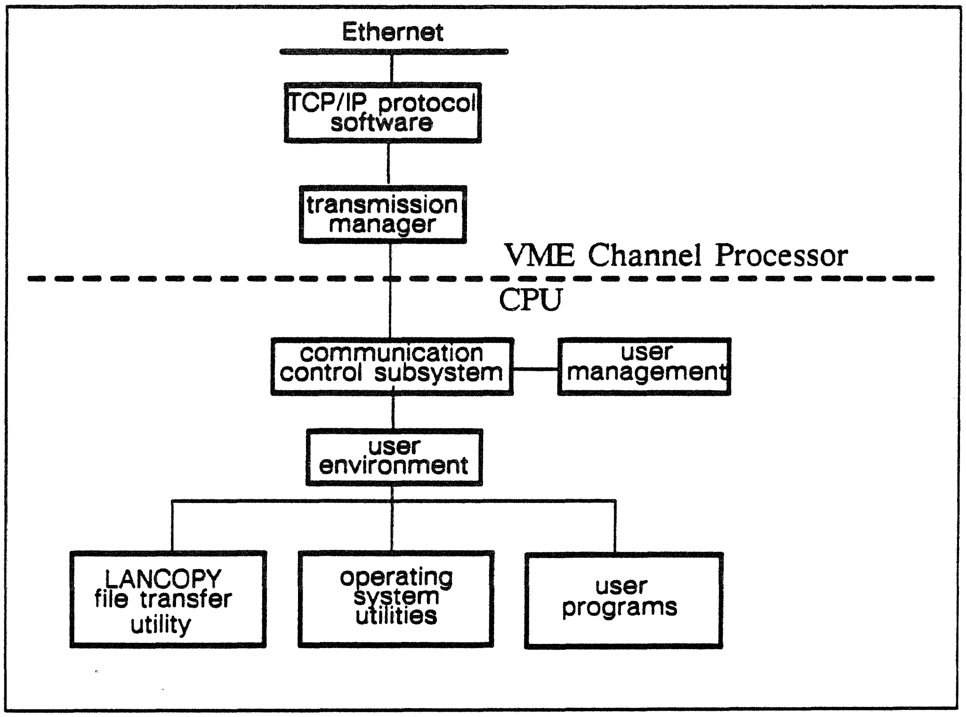


Figure 7-5. ETA10 software for the OIN.

Transmission Manager and TCP/IP Protocol Software

TCP/IP Protocol Software

TCP/IP protocol software provides the data packet format, inter-network packet address, and packet acknowledge protocol for ETA10 communications with the OIN.

The protocol software is loaded into the TCP/IP protocol processor from the single board computer during ETA10 system initialization. The protocol software communicates with the transmission manager in the single board computer while the ETA10 system is operating.

In the detailed implementation, several additional software units interface the TCP/IP protocol software to the transmission manager. These units reside with the transmission manager in the VME channel processor's single board computer. They are shown in figure 7-6.

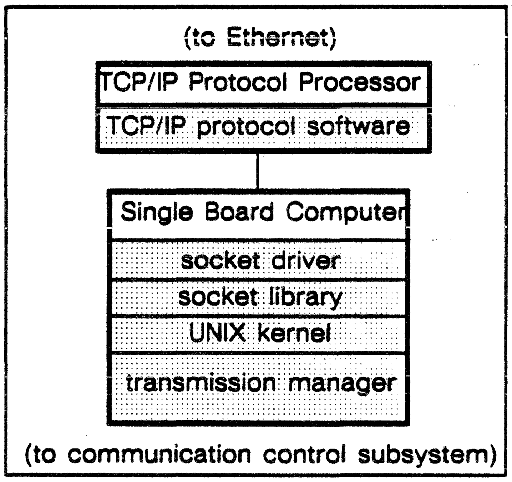


Figure 7-6. TCP/IP protocol software to transmission manager interface

The socket driver and socket library provide the TCP/IP interface for all TCP/IP connections. These software units and their interface to the transmission manager are supported by a UNIX kernel in the single board computer.

Transmission Manager Software

The transmission manager creates connections to and accepts connections from remote TCP/IP hosts for the ETA10. A copy of the transmission manager resides in the single board computer in each VME channel processor. Each copy of the transmission manager communicates with the communication control subsystem. The transmission manager provides the communication control subsystem with a reliable and efficient mechanism for communicating with the Ethernet. It shields the communication control subsystem from the details of how data is exchanged between hosts on the local area network.

The transmission manager provides the TCP/IP stream service and defines the facility called a socket for sending and receiving data. The transmission manager establishes the connection between the ETA10 socket and a remote TCP/IP host. Data is sent and received as a single byte stream of data in sequenced data packets. User record boundaries are not preserved. The TCP/IP stream service is used by remote ETA10 login (RETA) utility and LANCOPY file transfer service.

To implement the above service, the transmission manager software supports operations on two types of shared memory objects: network service access objects and TCP/IP service access objects. TCP/IP service access objects implement the TCP/IP stream service and are supported by network service access objects.

Network Service Access Objects

A network service access object is created for each TCP/IP protocol processor and is activated at system initialization time. Each network service access object is associated with a local area network by the TCP/IP protocol processor hardware connection. Network service access objects mask the details of the hardware operation while supporting up to 64 concurrent connections to each TCP/IP protocol processor.

TCP/IP Service Access Objects

A TCP/IP service access object defines each connection used by the RETA remote ETA10 login and LANCOPY file transfer to send and receive data.

Transmission Manager to Communication Control Subsystem Calls

The transmission manager accepts remote procedure calls from the communication control subsystem to send and receive data. The transmission manager does not queue these requests. Each TCP/IP stream connection can have only one send and receive request outstanding. The transmission manager provides procedures to control the connections.

Procedures that operate connections require a parameter called *ccs_id* which identifies the connection. The *ccs_id* can be thought of as the ETA10 socket for the connection. The *ccs_id* is created by the communication control subsystem when the connection is made. It provides unique identification for the lifetime of the connection.

The OIN–Related Software Residing in the CPU:

Communication Control Subsystem

The communication control subsystem (CCS) controls the ETA10 interface to the OIN. The communication control subsystem is informed of the OIN configuration at system initialization when the TCP/IP protocol processors are brought on–line. No dynamic reconfiguration is possible. During run time, the communication control subsystem sends requests to the transmission manager to establish sockets for sending and receiving data to and from remote TCP/IP hosts. These communications with the transmission manager are formatted as remote procedure calls, which are routed to the transmission manager in the correct VME channel processor by remote procedure call mailbox name. Mailboxes, which reside in the communication buffer, are ETA10's facility for passing procedure calls between the CPUs and the I/O units. Formatted communications are sent via the communication control subsystem buffer pool in shared memory.

ETA10 system software sends requests to the communication control subsystem through program procedure calls.

User Management

User management requests the new connection to the ETA10 for each RETA login from the Apollo gateway. User management queues a request for access to the next user connection at system initialization time, and again when each new connection is made. User management uses these connections to send and receive data that validates the user's login parameters with the ETA10 user registry. Each validated connection is then passed to the global scheduler which creates the user environment that operates the connection.

User Environment

The user environment software gives the ETA10 control of the user's connection. Each command received from the connection is interpreted by the operating system and processed within the user environment. Operating system utilities may be invoked by command.

LANCOPY File Transfer Utility

The LANCOPY utility is used to transfer files between the ETA10 and Apollo Domain Rings. The ETA10 provides only a LANCOPY initiator. The Apollo Domain Ring gateway provides the LANCOPY server. LANCOPY cannot be invoked in a batch job stream. It uses the RETA user login connection to transfer file data to or from the Apollo Domain Ring.

Management of OIN Equipment

System administrators manage ETA10-supplied OIN equipment on the service unit (SU) of the ETA10. They define the OIN equipment for the ETA10 system, analyze events that affect the system performance, and secure the ETA10 from unauthorized user access. The system administrator's network management functions at the service unit are outlined in the following subsections.

Other local area network equipment can be managed by personnel responsible for installing and maintaining network control servers, bridge servers, and gateway servers that are provided by suppliers other than ETA Systems. The procedures and the job classifications of the equipment managers vary from site to site and depend on the types of servers used. Possible management services include the control of local area network protocol filters, host device physical address filters, and configuration of host devices, bridges, and gateways.

Defining the ETA10 OIN Equipment

System administrators at the service unit define the ETA10 resources which service the OIN by means of a utility called Build System Configuration Table. The system configuration table is part of the communication control subsystem and contains definitions of the TCP/IP protocol processors and associated local area networks that are connected to the ETA10. The defined configuration goes into effect the next time the operators initialize the system.

Recording Events that Affect OIN Performance

System events that affect OIN reliability, availability, and serviceability are logged on the service unit disk. These events are also reported to the ETA10 operator on the service unit display as they occur. Maintenance and system administration personnel analyze events by scrolling through the system log history.

ETA10 Security

System administration personnel secure the ETA10 from access by unauthorized users. Validation of remote ETA10 user login requests protects the ETA10 files and other resources. System administrators set up the user registry and each user's environment profile. The user's login name and password are validated with the user registry before the ETA10 system creates a user environment. The user environment profile limits the number of files and resources that the user is able to access.

Reliability of ETA10 Network Services

Reliability of TCP/IP stream transmissions is built in to the TCP/IP protocol. The transport control protocol data packet contains a 16-bit checksum of all 16-bit half words supplied. The receiving host validates the checksum and returns an acknowledgment packet to the sender. If the sender times out before receiving an acknowledgment, the sender retransmits.

The communication control subsystem returns an exception status to the ETA10 system application software in the case of a connection failure. Exception error conditions are reported to the application software. Exception conditions include the breaking of a connection by the remote TCP/IP host.

The service unit monitors the events of the ETA10 service to the OIN and reports problem and error conditions to the ETA10 operator. Events are also logged on the service unit disk for reference by system administration and maintenance personnel.

Section 2: The Loosely Coupled Network

The Loosely Coupled Network (LCN) is part of ETA System's Multi-Host Network (MHN). The LCN is the batch-oriented, mainframe-based network for the ETA10. ETA Systems' LCN is based on Control Data Corporation's (CDC's) proprietary Loosely Coupled Network.

Effective ETA10 data transfer rates to the LCN have not yet been confirmed. Actual performance will be highly dependent on the RHF applications and the response environment from other LCN hosts. Important factors are the size of the files transferred, the file type, and the transfer block size.

Computers and Software that Connect to the LCN

The LCN provides high speed file transfer directly between the ETA10 and certain Control Data Corporation (CDC) computers. Certain models of IBM, Digital Equipment, and certain other models of CDC computer systems also may be indirectly connected to the ETA10 through the LCN, but the ETA10 can only communicate directly to the specific CDC computers listed here.

Computers and Software that Communicate with the ETA10

The ETA10 communicates directly over the LCN with the CDC computer systems and software in the following list:

- CDC CYBER 170 series computers running NOS (Network Operating System), release 2
- CDC CYBER 180 series computers, model 835s, the 850 series, and model 875s running NOS, release 2
- CDC CYBER 205s running VSOS (Virtual Storage Operating System)

Figure 7-7 shows an ETA10 Model P connected directly to a CYBER 205 and to a CYBER 850 through an LCN.

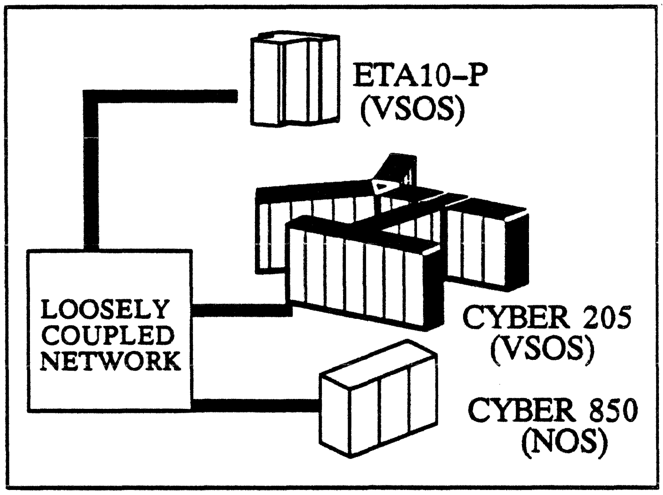


Figure 7-7. ETA10-P, CYBER 205, and a CYBER 850 on an LCN.

NOTE

Because initialization and maintenance of certain LCN equipment must be performed by NOS, release 2, software, each LCN must have at least one CDC system that runs NOS and is connected directly to the ETA10.

Other Host Computers Supported on the LCN

The following computer systems can also be connected to the LCN, and they can communicate indirectly with the ETA10 through one of the CDC computer systems in the previous list.

- IBM 370 computers
- Digital Equipment Corporation PDP-11, VAX 750 and 780 computers

Figure 7-8 shows a DEC VAX 780 connected to a CYBER 205 and to a CYBER 850 on an LCN.

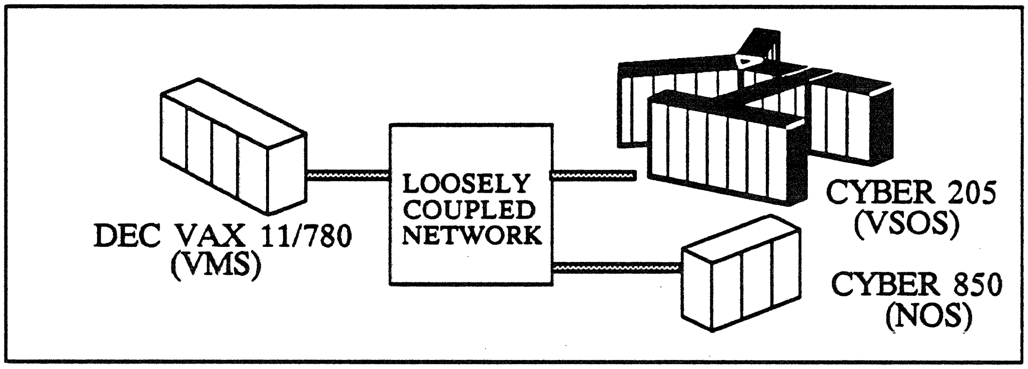


Figure 7-8. DEC VAX 11/780, CYBER 850, and CYBER 205 on an LCN

Figure 7-9 shows the equipment in the two previous figures as they are connected at one university supercomputing center.

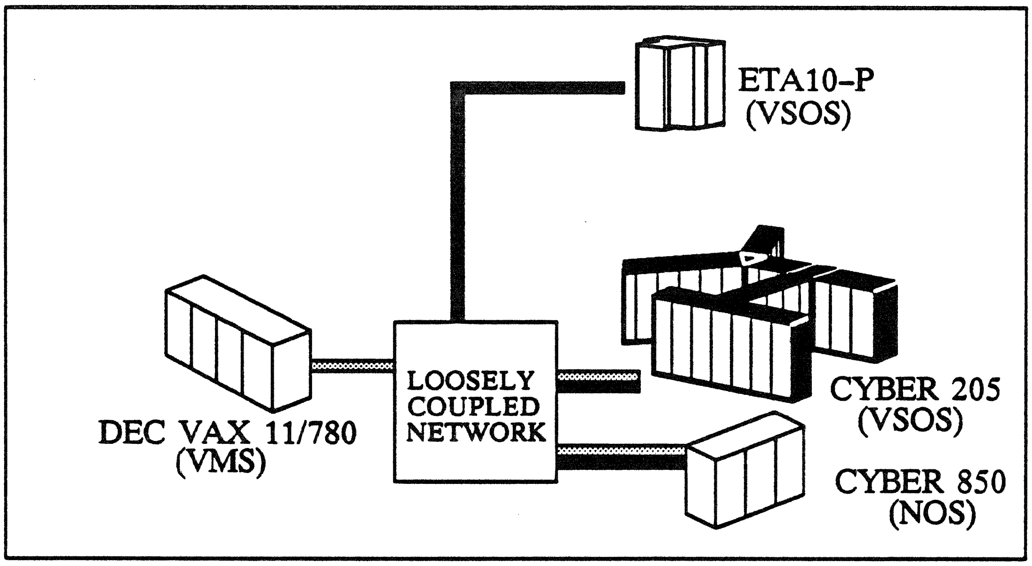


Figure 7-9. ETA10-P, CYBER 205, CYBER 850, and DEC VAX on an LCN.

CHAPTER 8

Chapter 8

System Peripherals

Chapter Contents

List of Figures

Chapter 8

System Peripherals

In This Chapter

- Disk physical-file structure

  Concepts defining the structure of disk physical files: sector, segment, minimum allocation units, free space allocation, logical and physical devices.

- Hardware components and the disk system

  The ETA10 disk channel interface and disk hardware.

- Interface to the disk system

  A description of the user interface to the Disk System, and the ETA10 software which services the disk system: where it resides and how it interacts.

- Disk system performance

  A discussion of disk seek and transfer time performance, and I/O unit data storage requirements.

- Disk system management

  How system administration personnel initialize and configure the system using the service unit. A discussion of the security, reliability, and administration of the logical and physical disk systems.

Disk Physical-File Structure

A disk physical file is organized data stored and maintained on the disk system. Disk physical-file system software (DPFS) manages access to a set of disk devices, and partitions space on them into disk physical files. Physical files reside on logical disk devices, which are portions of one or more physical disks treated as a single addressable device. In initial ETA10 software releases a logical disk device resides on a single physical disk unit. Each physical file has a *physical-file ID*, a unique identifier consisting of a logical device number and a disk physical-file map ordinal.

The logical-file system uses physical files to build the logical files used by user applications and operating system utilities. Physical files also hold the directories and catalogs used to manage these logical files.

Sector

Physical files consist of groups of physical sectors residing on logical disk devices. Each disk unit has over one billion bytes of storage capacity, divided into more than 70,000 16k–byte sectors. The sector is the smallest piece of data that can be physically transferred to or from a disk. Each sector contains a header field to verify its position on the disk, a data field, and a trailer containing a cyclic redundancy check field used to detect read errors.

Sectors are identified by a physical sector address. (Cylinder and track positions are unknown to the disk physical-file system). Sector numbering begins from zero on cylinder zero, head zero, sector zero, and consecutive numbers are assigned to all sectors for that head. Numbering continues with the next head on the same cylinder until all sectors in the cylinder have been numbered, and so on through every cylinder on the disk.

Minimum Allocation Unit

While the sector is the smallest piece of data that can be transferred to and from disk, a *minimum allocation unit* is the smallest piece of data that can be allocated on a drive. The minimum allocation unit is an even number of consecutive sectors, four on the ETA10 at Release 1.0. Use of the minimum allocation unit reduces the size of the disk's free space map, which is used to allocate disk space for disk physical files.

Free Space Map

The free space map designates which portions of the disk are available for allocation to disk physical files. It is implemented as a bit map, with each bit corresponding to a minimum allocation unit. Flawed portions of disk are *not* allocated. When the disk is initialized, minimum allocation units containing sectors listed in the disk factory flaw map are marked on the free space map as already in use.

Segment

A segment is a minimum allocation unit, or a set of consecutive minimum allocation units. DPFS allocates disk free space in segments. At the same time, it creates segment allocator information containing the following set of fields:

ld. Logical disk device selector.

pfid. Physical-file ID

fsa. First physical sector address.

f. Flags first segment allocator for physical file.

length. Length of the segment.

link. Link to the next segment allocator for the physical file.

ld	pfid	fsa	f	length	link

Figure 8–1. Segment allocator.

Disk Physical-File Map

The disk physical-file map is a set of linked lists of allocated segments, indexed by physical-file ID. Each linked list in a set is an allocation map of one physical file. The segment allocators for a given physical file are linked in the order the segments were assigned, with the physical-file ID pointing to the first link. Non–contiguous segments assigned to a physical file are tracked by their linked segment allocators.

Hardware Components and the Disk System

The path through the ETA10 system from the CPUs to the disk drives goes from shared memory and the communication buffer to their interfaces, then through fiber optic data pipes to data pipe buffers in disk channel processors in each I/O unit. Besides the data pipe buffer, each disk channel processor contains a single board computer which runs the disk physical-file system and disk channel controller software.

Disk Channel Interface

Each disk channel processor manages communications between its data pipe buffer and a disk channel interface. The disk channel interface connects with up to four disk units through disk control modules on each disk unit, as shown in figure 8-2.

The disk channel interface provides the physical connection between a disk channel processor and its disks. It controls the exchange of commands and data between the data pipe buffer memory and the disk control module in the disk unit. The single board computer directs the data pipe buffer to load the disk channel interface registers and supply the appropriate disk commands which control it.

A protocol similar to the CDC Intelligent Standard Interface protocol regulates communication between the disk channel interface and the disk control modules. The disk units are directed by communication protocol procedures.

The disk channel interface provides about 30 signal lines to the control modules. 16 data lines and several control lines connect to all the control modules. While these lines are not otherwise employed, the control modules use them to present *attention* status when data or status is ready for transmission, and *busy* status when they are in use.

Disk channel interface communication protocol prescribes the sequence in which procedures are executed. Five protocol procedures serve to read and write the function and data buffers of a control module.

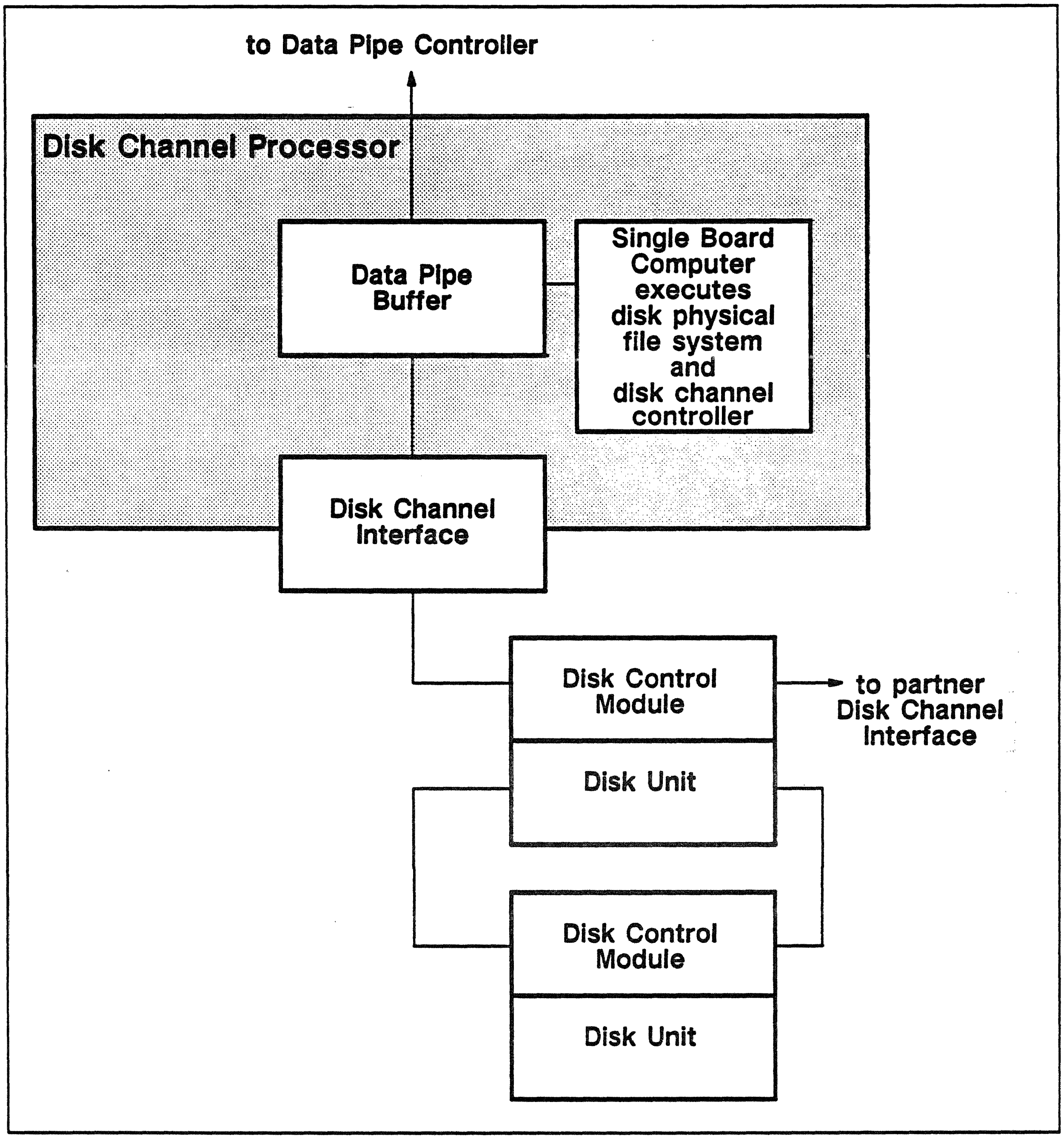


Figure 8–2. Disk channel processor and two disk units.

Disks

The ETA10 DSU10-1 unit contains one control module and one disk unit. Up to four disk units can be cabled through their control modules to one disk channel interface in one I/O unit. The same disk units can be cabled to to a second disk channel interface in a partner I/O unit.

Disk Unit

The disk unit provides over one billion bytes of data storage capacity. The peak data transfer rate is 10 million bytes per second. Average seek time is 16 milliseconds, average rotational delay is 8.5 milliseconds.

The disk is formatted with 11 16k-byte data sectors per track. There are 8 data tracks per cylinder and 886 cylinders per disk. Average seek time for a one cylinder move is 2.5 milliseconds.

Cylinders are separated from each other by a rotational delay of 5 milliseconds (the time required to transfer about 3 sectors). Disk read and write functions can span cylinders. When this happens, the 5 millisecond delay is realized between each cylinder.

A factory flaw map of unusable sectors is written on cylinder 1, redundantly in sectors 0, 9, 18, and 27 of each track. This flaw map is read by the disk channel interface and factory flaw sectors excluded from the disk free space map.

Error correction information is recorded at the end of each sector. The disk unit averages a data transfer error rate of no more than 2.5 in 10 billion bits when tested by reading one trillion bits. When an error is detected, error correction is attempted, and followed by application of the error correction circuitry.

The disk unit exhibits a seek error rate of no more than one in one million. Correct cylinder, head, and sector addresses are recorded in each sector for verification. A seek is retried if an error is detected.

On the disk unit control panel, switches are provided for power, write disable, and partner disk channel interface select. Indicators display command ready and disk unit fault conditions. There are supply, logic, analog, spindle, and blower power breakers.

Control Module

A control module provides logic and control circuitry for communicating with the disk channel interface and directing the disk unit. In the control module, a *function buffer* holds command, status, and operating mode information from the disk channel interface, and a *data buffer* transfers data between the disk and the disk channel interface. The control module can read from disk while the data buffer is transmitting, and write to disk while the data buffer is receiving, data from the disk channel interface. For a Read-Disk function, the control module sends a delayed attention signal to the disk channel interface, notifying it that enough data has been read from disk to start data transmission.

The control module provides automatic diagnostic tests to validate itself and the disk unit. When power is applied, the control module runs a self-diagnostic test to verify its own circuitry. When a power-up spindle command is executed, a disk unit diagnostic test is performed. If error conditions are detected, they are reported to the disk channel interface.

Control module commands include:

- Transmit Control Module Error Log
- Clear Control Module Error Log
- Recalibrate Disk Seek Actuator
- Seek
- Power Down Spindle
- Power Up Spindle
- Write Disk From Filling Data Buffer
- Write Disk And Verify From Filling Data Buffer
- Write Disk From Static Data Buffer
- Copy Data Area On Disk
- Diagnostic Write Simulated Error Condition
- Diagnostic Read Simulated Error Condition
- Read Disk To Emptying Data Buffer
- Read Disk To Static Data Buffer
- Salvage Disk Data Using Error Correction Codes
- Read Disk Header

- Read Disk Factory Flaw Map
- Transmit Data Buffer
- Receive Data Buffer
- Format And Certify Disk
- Receive Addressing, Error Recovery, and Attention Delay Modes
- Receive Attention Delay Parameters
- Perform Diagnostic Functions

Control modules return normal and error status information to the disk channel interface. Normal status includes:

- Power-up Complete
- Control Module Idle
- Read Data Available (sufficient data has been read from the disk to begin transmitting data to the disk channel interface)
- Data Buffer Space Available (sufficient data has been written to the disk to begin receiving more data from the disk channel interface)
- Command Complete (the control module has finished executing the command block)

Interface to the Disk System

Software Servicing the Disk System

Software components essential for service to the disk system reside on different hardware units of the ETA10 system (figure 8–3). The *logical-file system* and *shared memory manager* execute in the central processing units. The *DPFS* and *disk channel controller* software reside in the disk channel processors. The *open file table* and *physical-file ID map* are copied to the communication buffer and CP memory when logical files are opened.

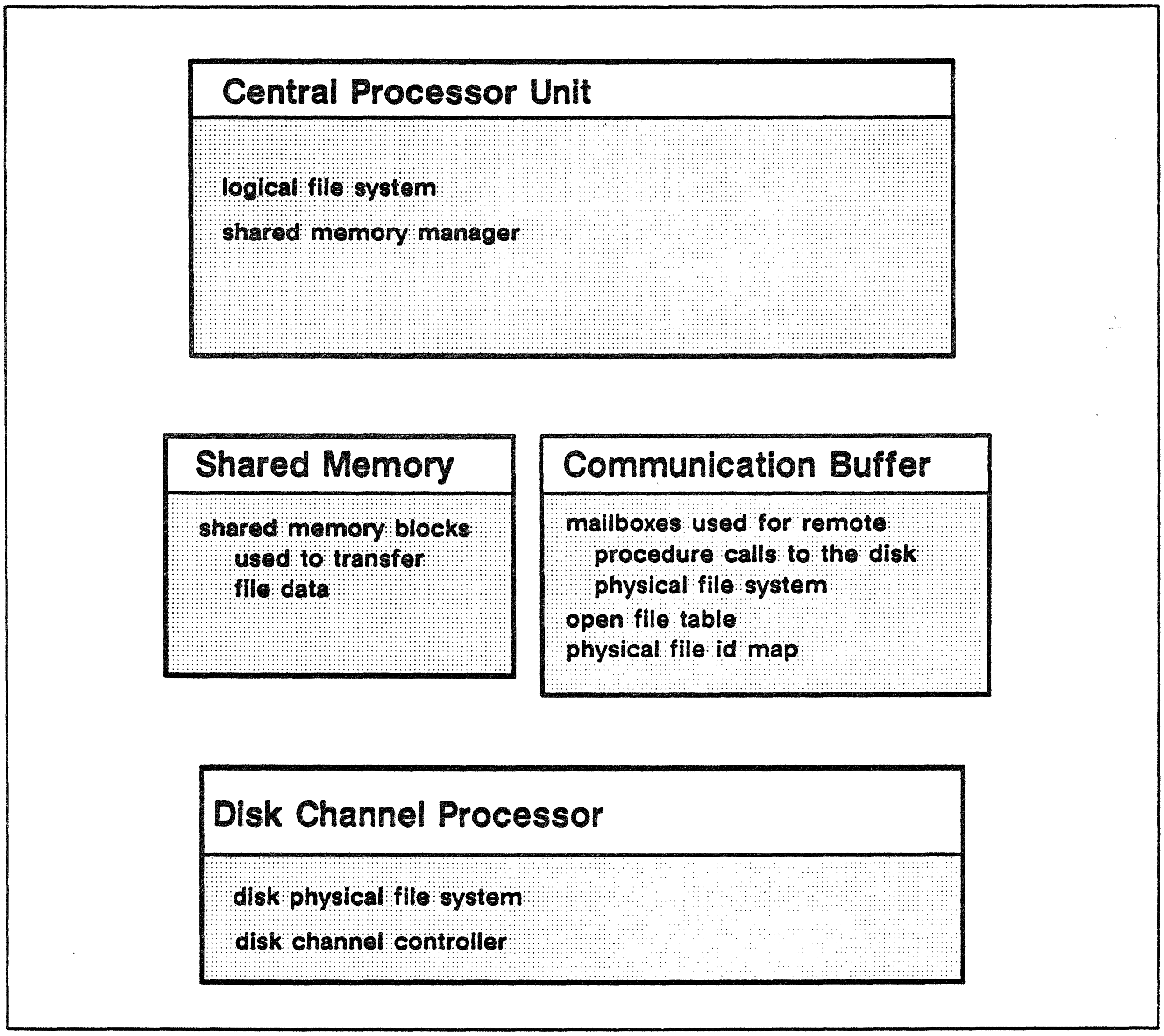


Figure 8–3. Residence of software servicing the disk system.

ETA VSOS provides the user with a set of commands to invoke operating system features that process logical files and interface with the disk system. These commands include:

- ATTACH. Attach permanent files.
- COMPARE. Compare file contents.
- COPY. Copy a file.
- DEFINE. Define a permanent file.
- FILES. List file information.
- GIVE. Transfer file ownership.
- LISTAC. List file access permission set.
- MFGIVE. Transfer a file to a remote host facility.
- MFLINK. Open file transfers for a remote host facility.
- MFTAKE. Accept file transfer from a remote host facility.
- PERMIT. Change file access permission set.
- PURGE. Delete permanent files.
- REQUEST. Create local file.
- RETURN. Delete local files and return permanent files.
- REWIND. Reposition to front of file.
- SKIP. Reposition within a file.
- SWITCH. Change file attributes.

User programs can also interface with the logical-file system by using SIL calls to open and close files, and perform explicit and implicit reads and writes.

The Software Interface to the Disk Physical-File System

The logical-file system manages logical files, storing information about them in catalogs and directories. To create a directory, the logical-file system creates a logical file and stores on it a directory holding access permissions for itself and its files. Directories reference the file catalog, which also is stored on a logical file.

The logical-file system handles explicit requests to create, open, read, and write logical files. To create a logical file it calls DPFS; to read one, it calls the shared memory manager. When requested to extend a logical file, the logical-file system directs DPFS to extend the supporting physical file, and if this is not possible, directs DPFS to create an additional physical file.

Implicit I/O - servicing page faults - is handled by the pager, which consults process page tables and directs the SM manager to move pages in and out of CP memory.

The shared memory manager handles all file data transfers between the CPUs, shared memory, and disk. It allocates shared memory in blocks of a specified number of disk sectors and directs DPFS to move data to and from them as required.

When a logical file that has been written is closed, the shared memory manager directs DPFS to move the modified shared memory blocks to disk.

To translate logical file requests to disk physical-file requests, the shared memory manager uses two tables: the *open file table*, which provides each logical file with a pointer to the physical-file ID map, and the *physical-file ID map*, which associates logical file addresses to physical addresses, holding physical-file IDs for all open files.

Figure 8–4 diagrams how software components interface to DPFS.

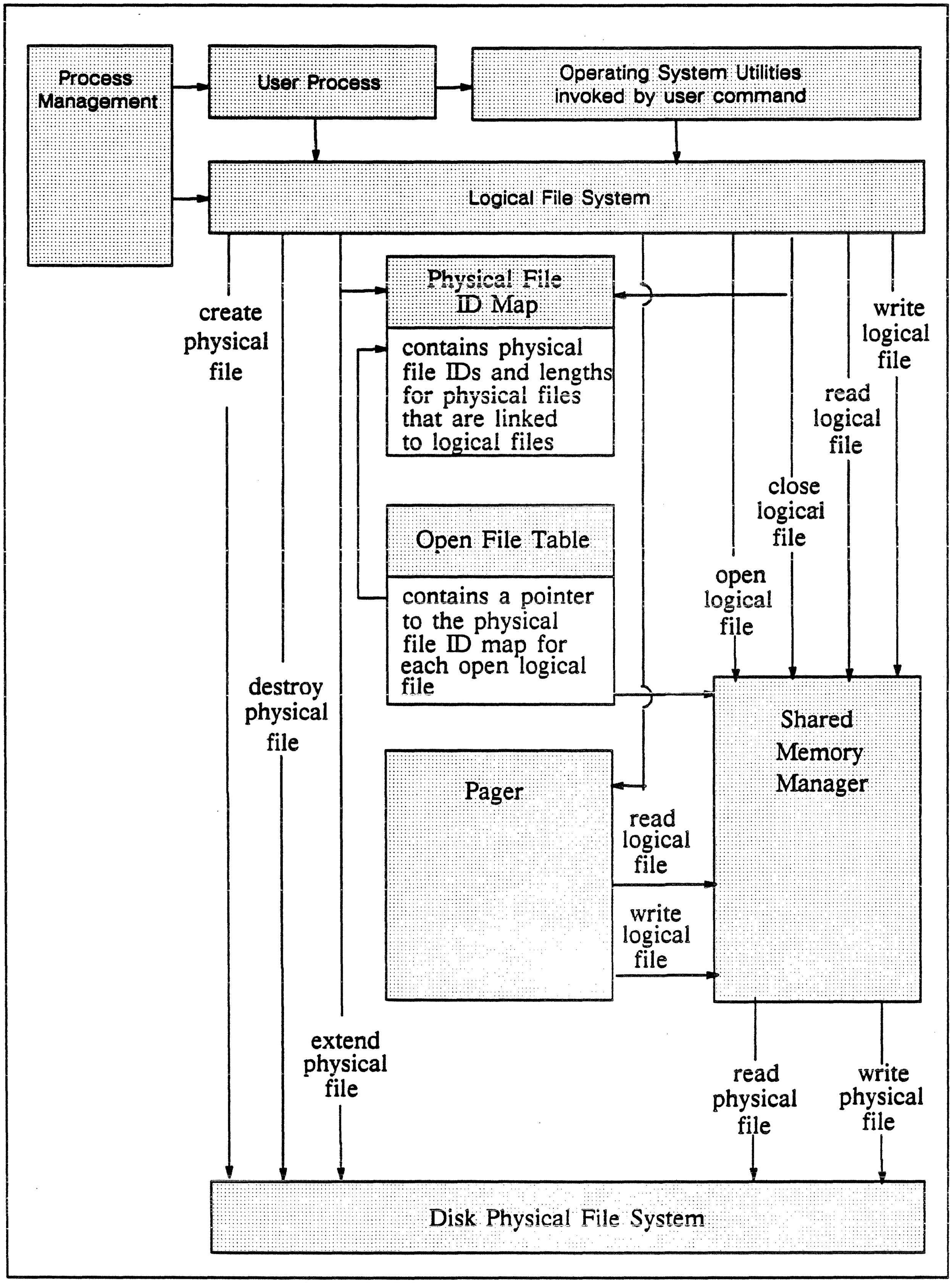


Figure 8–4. Software interface to the disk physical-file system (DPFS).

Disk Physical-File System (DPFS)

DPFS performs operations on disk devices, managing access to them and partitioning space on them into physical files. It executes in a single board computer on the IOU.

DPFS uses the interprocess communication facility (described in Chapter 4 of this manual) to communicate with the logical-file system and shared memory manager on CPUs. The shared memory manager and the logical-file system issue file requests through remote procedure calls directed to the physical-file system's mailbox.

The service unit and DPFS interact over the RS422 interface to perform configuration, system initialization, and system termination functions.

DPFS receives commands from CPUs and the service unit, communicates these commands to one or more disk channel controllers, waits for responses, and returns the results to the CPU or service unit which made the original request.

Physical-file requests are:

- Create physical file
- Extend physical file
- Reduce physical file
- Read physical file
- Write physical file
- Destroy physical file

When a physical file is created, extended, reduced, or destroyed, the disk free-space map and disk physical-file map for the supporting physical disk device must be rewritten to reflect the updated file information.

System configuration control also communicates with DPFS by making remote procedure calls. Since these requests involve updating device labels, they are made through the service unit while the ETA10 is not operating. (A disk label lists a logical device residing on the disk and up to 64 device classes to which the device can belong, depending on the configuration).

System configuration control's requests are as follows:

- Add logical device to the system
- Remove logical device from the system
- Initialize physical disks
- Create logical device
- Initialize logical device

System configuration and control also calls DPFS during ETA10 system initialization to:

- Initialize the disk physical-file system
- Create logical device mailbox

Disk Channel Controller

The disk channel controller runs on the single board computer in the disk channel processor in the I/O unit. Its primary function is handling input and output. It directly manages I/O over the disk channel between a disk device and shared memory.

The disk channel controller processes instructions received from DPFS to read and write data to and from the physical device. Each request from DPFS is directed through a disk channel controller to a specific disk unit control module.

Disk System Performance

The disk system's performance is affected by disk access time, data transfer rates, and by the storage capacities of the disk and the single board computer. Disk access time depends on the disk unit's characteristics and on whether the disk's free-space and physical-file maps are resident on the disk channel processor. If they need to be moved from disk to processor, the speed of the transfer depends on the availability of the single board computer bus. Transfer rates for disk data are governed by disk unit characteristics and the availability of the single board computer bus.

A single disk channel processor drives up to four disk units. Up to two disk channel processors can be configured within a single I/O unit. Each disk unit can provide a second interface to a partner I/O unit, so that a subset of the total disk units is controlled by each partner I/O unit. In case of an I/O unit failure, the ETA10 system is reconfigured so that the partner I/O unit controls all the disks. The number of disk units, the controlling I/O unit, and the minimum allocation unit are configured so disk labels, free space maps, and physical-file maps will fit on the single board computers and the expansion memory of each I/O unit.

The disk units and I/O units are configured to optimize the availability of the I/O unit data pipe channel to shared memory. Disk transfers to shared memory by way of the data pipe are very fast (about 625 sectors per second). If the transfer bridges more than one cylinder, transfer delays between cylinders of five milliseconds (about the time required to transfer three sectors) occur.

The average seek time of 16 milliseconds plus an average rotational delay of 8.5 milliseconds determines disk access time. Release 1.0 of the disk channel controller uses no disk queue selection algorithms to reduce average seek and rotational delay times.

When the disk is accessed, the disk label, free-space map, and physical-file map must also be accessible to the I/O unit single board computer and expansion memory. The transfer rate from the disk through the single board computer bus is four million bytes per second. In Release 1.0, the disk and I/O units are configured so all disk labels, free-space maps, and physical-file maps can be entered at the same time to eliminate transfer delays.

Disk System Management

System administration personnel manage the disk system through service unit display nodes at the ETA10 site, or via modem links at remote sites. The service unit initializes, monitors, and manages the disk hardware and software configuration. Administration personnel initialize and allocate the disk resources, configuring the physical disk system and the logical disk devices while the ETA10 system is not operating. Logical and physical device preparation requires writing the disk label and is done immediately. Disk configuration changes entered into the system configuration table on the service unit disk take effect the next time the ETA10 system is initialized.

Configuration

The disk system is configured with I/O unit data storage and data pipe transfer requirements in mind. The I/O unit holds a copy of the labels, free space maps, and physical-file maps for each connected disk unit. The I/O unit must transfer all disk data to and from shared memory through its single data pipe. The disk system is optimally configured when the requirements for I/O unit data storage are met and the requirements for data pipe transfer time are balanced.

System administration personnel define logical disk devices that reside on physical disk units. A logical device may be a member of up to 64 device classes, which are sets of logical devices. Files are created in a device class.

A summary of the SU configuration and maintenance utilities follows. Chapter 9 presents the service unit in detail.

Build System Configuration Table

The system configuration table specifies the disk channel processors, disk logical devices, and disk units in the ETA10 system. It is built and installed on the service unit. The specified configuration takes effect the next time the ETA10 system is initialized.

Format and Initialize Disks

Service unit programs verify disk labels, initialize disk units, and define logical devices. To initialize a disk unit, the disk factory flaw map is retrieved from the disk, and the disk free-space map is created, with minimum allocation unit segments containing flawed sectors marked as already in use. The label, disk free-space map, and an empty disk physical-file map are written to the disk.

Logical disk devices reside on separate physical disk units. Each may belong to up to 64 device classes. The physical disk label contains the device classes for each logical device.

System Log and Query

ETA10 system events that affect disk system reliability, availability, and serviceability are logged on the service unit disk, as well as being reported to the ETA10 operator on the service unit display as they occur. Maintenance and system administration personnel analyze events by scrolling through the system log history.

Security

Security and reliability of the disk system is maintained by ETA10 software. Disk files and other system resources are protected from user access by user login validation. The username and password must be validated with the user registry before a user environment is created. The user profile controls user access to disk files and other system resources. Both the user registry and the user profile are managed by system administration personnel.

Reliability

Recovery is attempted for disk hardware errors. Recovery is successful in almost all cases while the ETA10 system is operating. For Release 1.0, if error recovery is unsuccessful the system is halted.

CHAPTER 9

Chapter 9

Service Unit Features and Functions

Chapter Contents

Contents (continued)

List of Figures

List of Tables

Table *Title* *Page*

Chapter 9

Service Unit Features and Functions

In This Chapter . . .

This chapter describes the capabilities and operating features of the ETA10 service unit (SU).

The major topics covered in this chapter are:

- The SU Network - An Overview
- The SU Operator Environment
- SU Hardware Components
- The SU Applications
- The SU Support Processes

The SU Network – An Overview

The service unit (SU) provides system operators with a unique tool for *controlling* and *monitoring* system activity. Often referred to as if it were a single workstation, the Service Unit (figure 9–1) is really a network of display and server nodes that, together with applicable software, form an integral part of the computer system.

To control and monitor the system, operators use a variety of application programs that run on the SU nodes. These applications help the operators perform many of the maintenance and management functions required to keep the computer system operating smoothly and efficiently. In addition to these "display" applications, the SU nodes run background processes that interact with hardware and software subsystems of the computer to collect information for a system-wide database. Information stored in this database is available to the application programs and is also available to SU operators in the form of reports and logs.

Control and Monitor Functions of the Service Unit

The operations that can be controlled and monitored through the service unit's applications are listed below.

- Control and monitor system activity.
- Configure or reconfigure system hardware and software.
- Run diagnostics to evaluate hardware operation.
- Run utilities to get a more detailed look into system components.
- Monitor the progress of system components during startup and shutdown.
- Monitor the operation of the hardware and software.
- Monitor the system log and prepare reports.

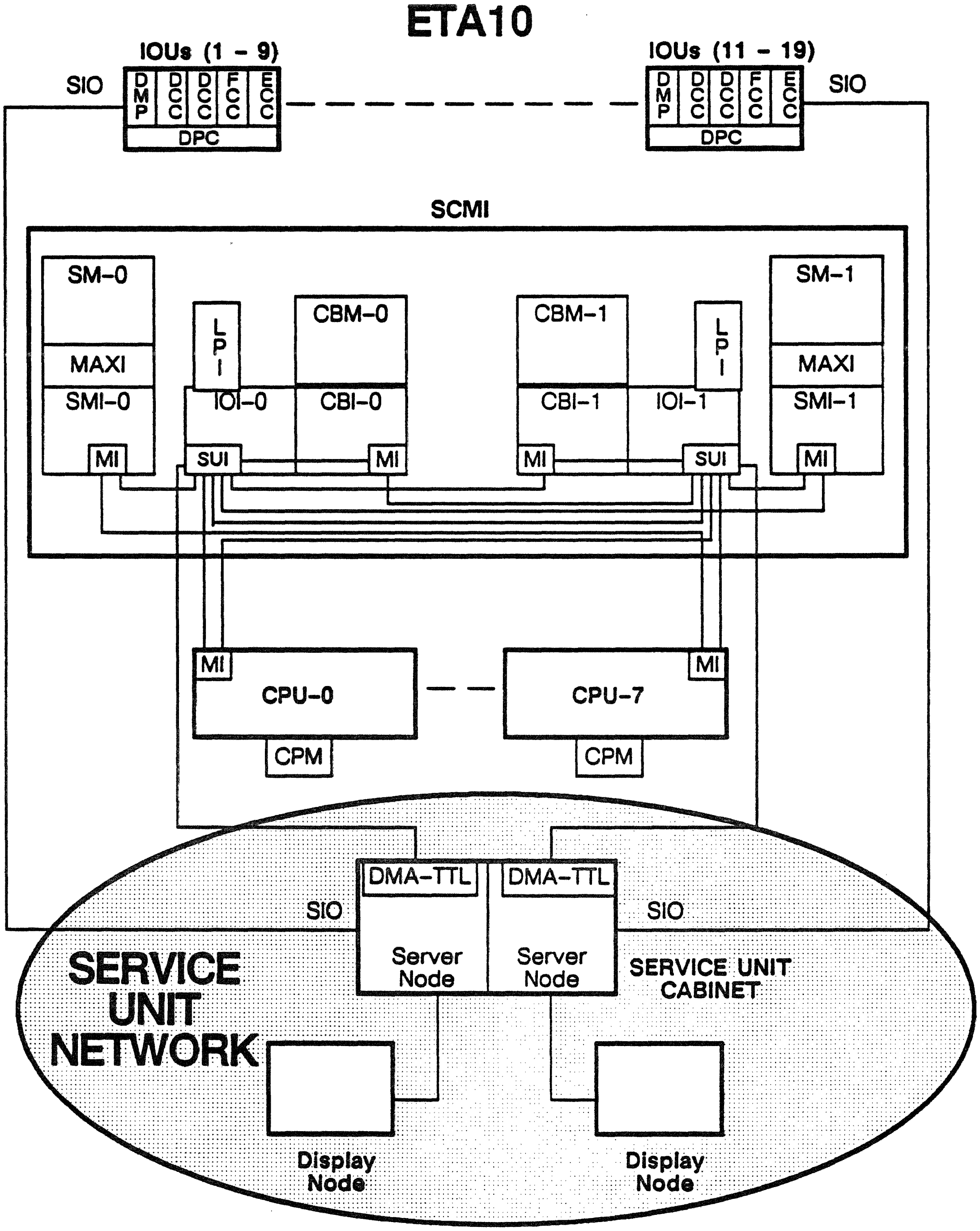


Figure 9-1. The SU network and the ETA10.

Communications with the ETA10

As shown in figure 9–1, the service unit communicates with the ETA10 though a maintenance interface (MI) and a serial I/O (SIO) line. A third method of communicating, to power and cooling supervisor (PCS) components only, is also available (see Chapter 10).

Each server node is connected through its MI to a service unit interface (SUI) in the SCMI cabinet. MIs then connect the SU interface to most other components of the ETA10. It is through the SU interface and MIs that the service unit reads or writes to shared memory, the communication buffer, and to CP memories.

The serial I/O lines allow access to the I/O units (IOUs) only, and enable direct diagnostic and communications access to the I/O units.

The maintenance interfaces are the chief vehicle of communication and together with the serial I/O lines provide the service unit with the following capabilities:

- Read and write to CPU and I/O unit memories.
- Load clocks.
- Access the B.E.S.T. circuitry on high-tech boards in the CPUs, the shared memory interface, and the I/O interface.
- Load and store the maintenance registers (MIDR, MODR); stop and start the CPUs and the I/O processors.
- Read addresses and syndrome bits for SECDED errors detected on CPU memories, shared memory, and the communication buffer.
- Degrade system elements.

The communication process between the service unit and the ETA10 computer system using the MI connection proceeds as follows (refer to figure 9–2):

Assume an application running on the display node wants to send data to shared memory, the CP memory, or the communication buffer. The procedure begins with the display node accessing its MI server. The MI library (part of a library that exists on the display node) enables the application to make a request to a remote server on the server node. The request goes to the MI server mailbox on the server node. Multiple applications may be making requests to this mailbox, either from the same display node or other display nodes in the network.

The server node monitors the requests coming into the mailbox and calls the MI driver that provides an interface across the MULTIBUS to the direct memory access processor (DMAP) board.

The application on the display node passes the data to be transmitted to the MI server on the server node what data to write. This data is buffered in the MI server, which buffers it on the server node, and tells DMAP where it is and where it should go. Once that message is delivered, DMAP takes it from there, using its own internal protocol to send the data through the DMA-TTL hardware channel, through the SU interface and to the appropriate memory.

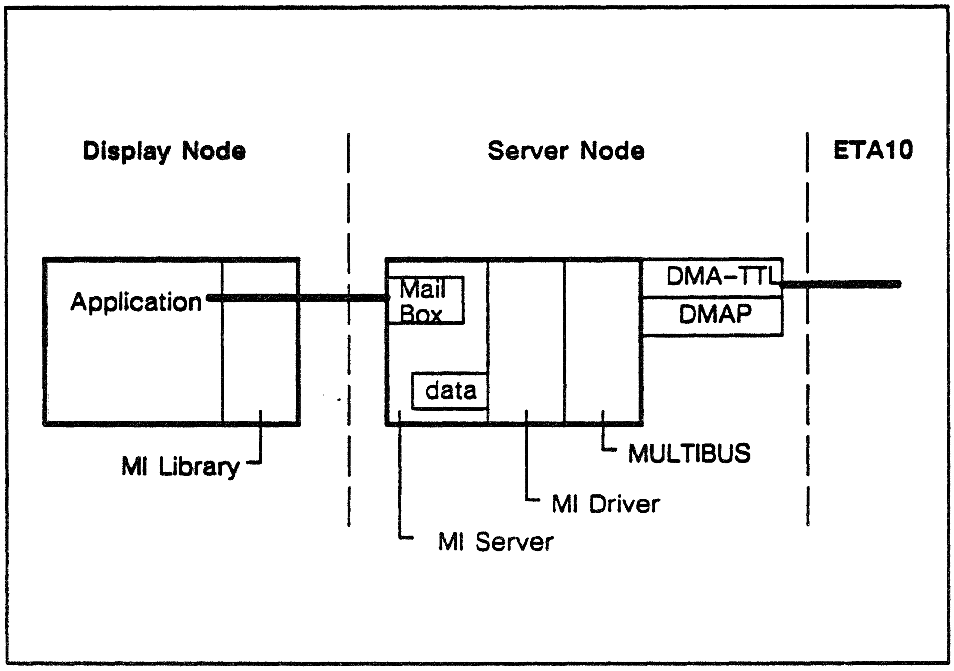


Figure 9-2. Communications between the service unit and the mainframe.

The Service Unit Operator Environment

Computer operators, maintenance engineers, system administrators, and other SU operators access SU applications through a mouse-and-menu interface, which uses color graphics, windowing, and menu-oriented displays to present information and permit operator input and control actions.

The **main** menu shown in figure 9-3, provides access to all applications available on the service unit (see figure 9-4 for a list of menu items). All applications are represented as menu items, and they are selected using a mouse-controlled cursor. The operator simply moves the cursor to the desired menu item and presses the 'select' key on the mouse.

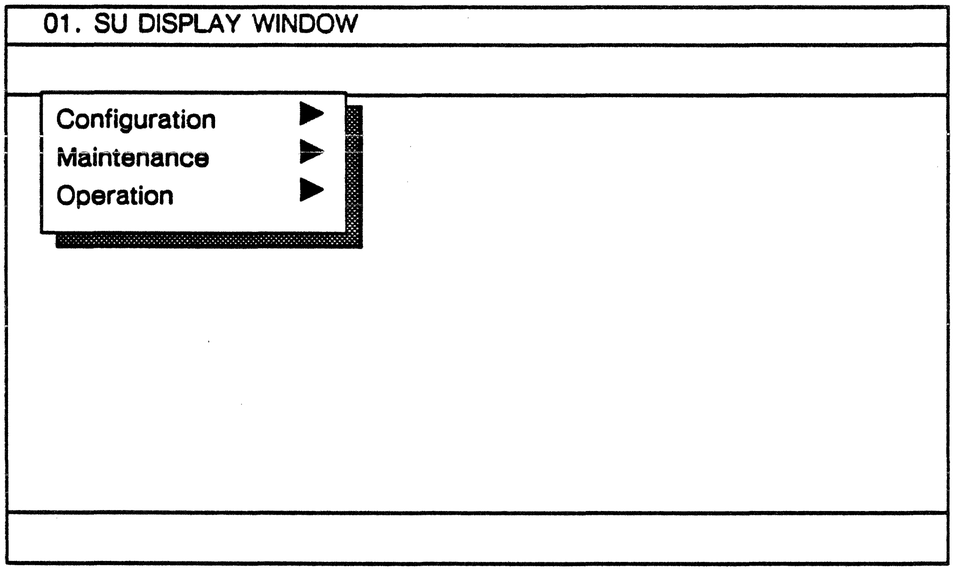


Figure 9-3. Main menu for the SU applications.

When an application is selected through the menu, a *window* is opened into the application. These windows provide access to the application and allow the operator to view and modify information, start and terminate processes, and so on.

An application menu item (see figure 9-5 for an example) can have sub-menus that describe other features and options available for use with that application. Application windows that perform an operation may use additional menus to interact with the operator once the application is running. This structure is known as a *menu tree*, since conceptually it resembles the manner in which trees grow with each branch growing from another branch.

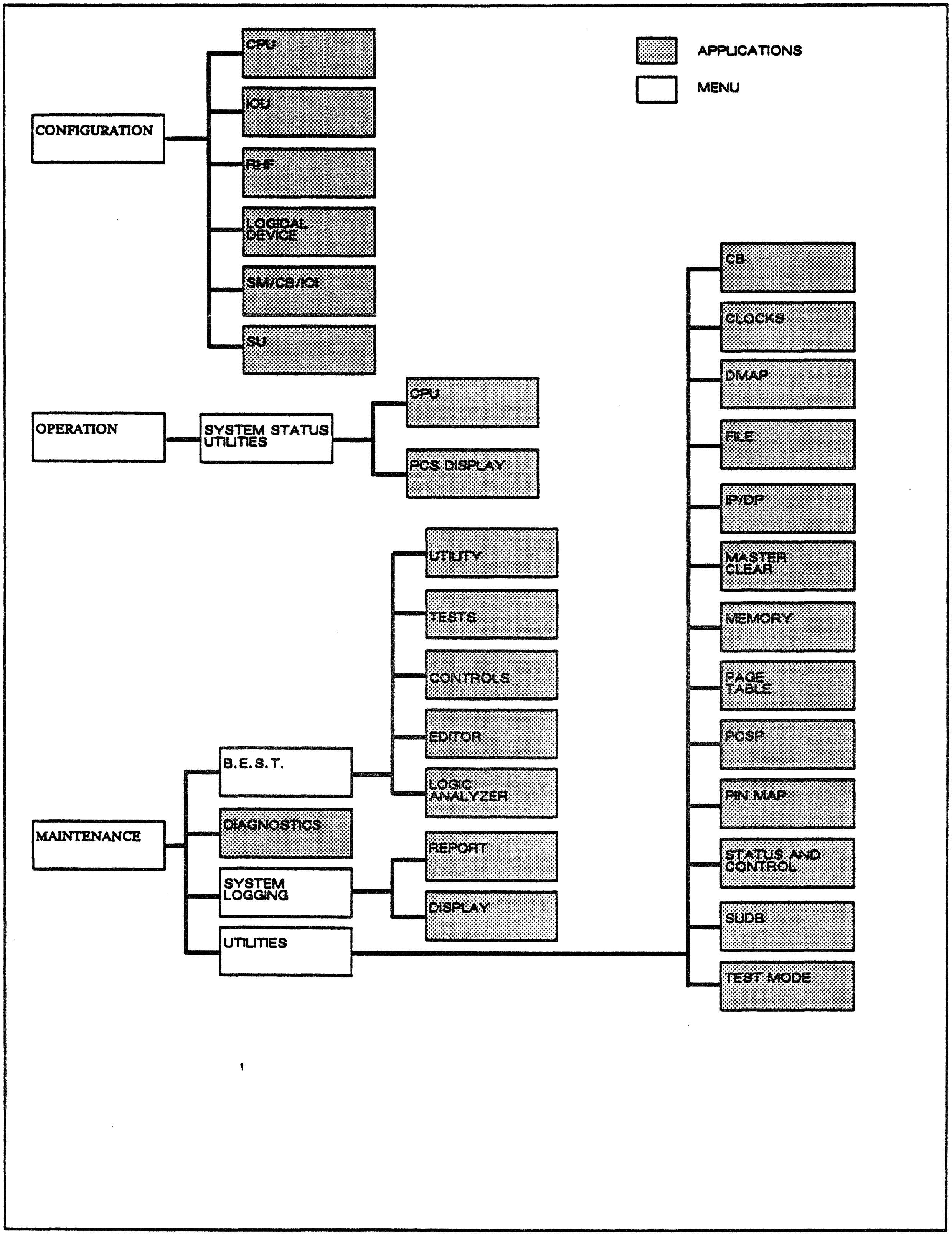
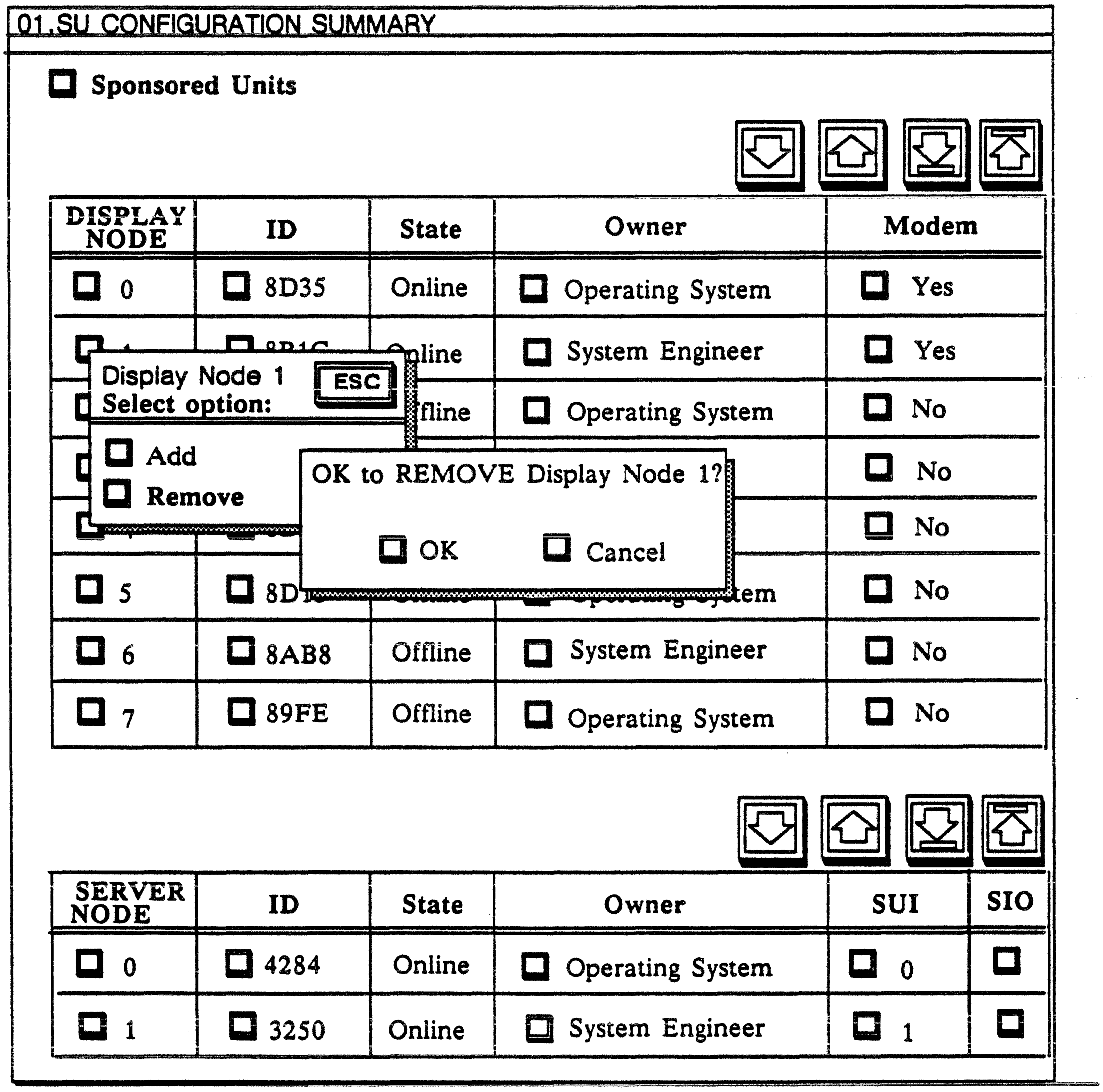


Figure 9-4. The menu-tree of SU applications and menus.

A number of windows may be open to the operator at any time, and each window may have several associated processes running that affect the contents of the display within the window. The operator can control window placement and collapse windows to icons.

A 'Snap' utility allows the operator to take a snapshot of the information currently displayed and print it.

Certain events within the system set off alarms to alert the operator of the event and request some action. When an alarm is issued, a long tone sounds at the display node and a message flashes in the clock window. To view the message associated with the alarm, the operator calls up the System Logging application and searches the database for the most recent entries.



01.SU CONFIGURATION SUMMARY

□ **Sponsored Units**

DISPLAY NODE	ID	State	Owner	Modem
□ 0	□ 8D35	Online	□ Operating System	□ Yes
□ 1	□ 8B1C	Online	□ System Engineer	□ Yes
□		Offline	□ Operating System	□ No
□				□ No
□				□ No
□ 5	□ 8D		Operating System	□ No
□ 6	□ 8AB8	Offline	□ System Engineer	□ No
□ 7	□ 89FE	Offline	□ Operating System	□ No

Display Node 1
Select option:

□ Add
□ **Remove**

OK to REMOVE Display Node 1?

□ OK □ Cancel

SERVER NODE	ID	State	Owner	SUI	SIO
□ 0	□ 4284	Online	□ Operating System	□ 0	□
□ 1	□ 3250	Online	□ System Engineer	□ 1	□

Figure 9-5. Window into an SU application.

SU Hardware Components

The service unit network consists of up to eight display nodes, a service unit cabinet, and a graphics printer (see figure 9-6).

The SU Cabinet contains the two server nodes, two 500 Mbyte disk drives, and interface hardware. Each display node is equipped with an 86 Mbyte disk drive, a tape cartridge, and a modem (for connecting the service unit to remote site support). The graphics printer is generally connected to one of the display nodes.

The SU hardware arrangement provides complete redundancy so that the service unit remains available during equipment maintenance and/or troubleshooting. Server nodes and display nodes are connected by manual ring switches that allow individual components or display node/server node pairs to be isolated from the network.

The SU cabinet and the graphics printer are located near the mainframe. Typically, one of the display nodes is also located very near the mainframe for the convenience of the maintenance operators. The other display nodes can be located up to 1000 feet from the SU cabinet, depending on site requirements.

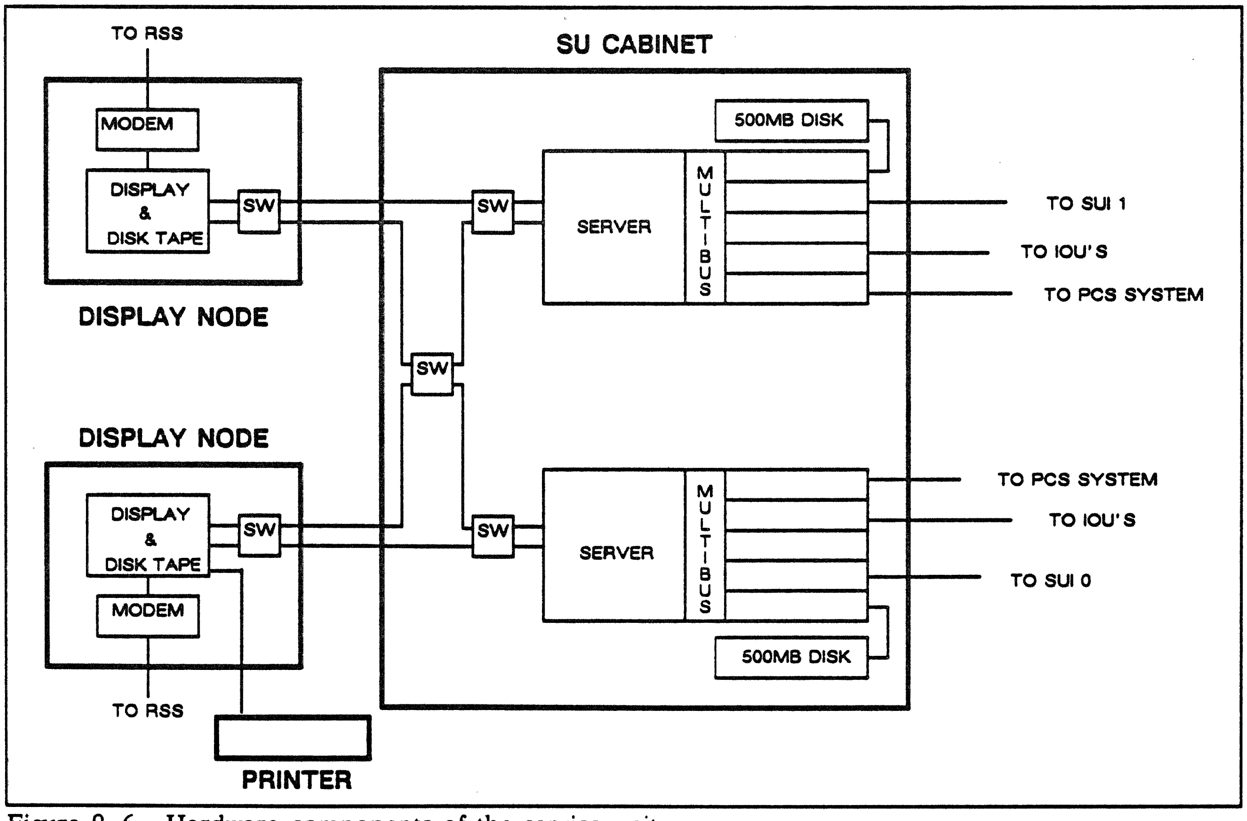


Figure 9-6. Hardware components of the service unit.

The SU Cabinet and its Server Nodes

The bulk of the monitoring and control work of the service unit is done by the server nodes housed in the SU cabinet.

Each of the two server nodes (figure 9-7) consist of a computer and the following add-on boards (the boards plug into the MULTIBUS):

- FSD-CNTRL board to control the server node's disk drive.
- DMA-TTL and DMAP boards to provide a link to the service unit interface (SUI) in the SCMI cabinet for communication with the CPU memory, SM memory, and the communication buffer.
- SSIO board to interface with the I/O units for diagnostic and maintenance access.
- PCSP board to connect the SU to the system-wide power and cooling supervisor sensors and data acquisition hardware. (Refer to *Chapter 10* for information about the power and cooling supervisor.)

The computer contains a 32-bit processor with 2 Mbytes of internal memory. Each server node can be independently removed from the network ring using its manual ring switch.

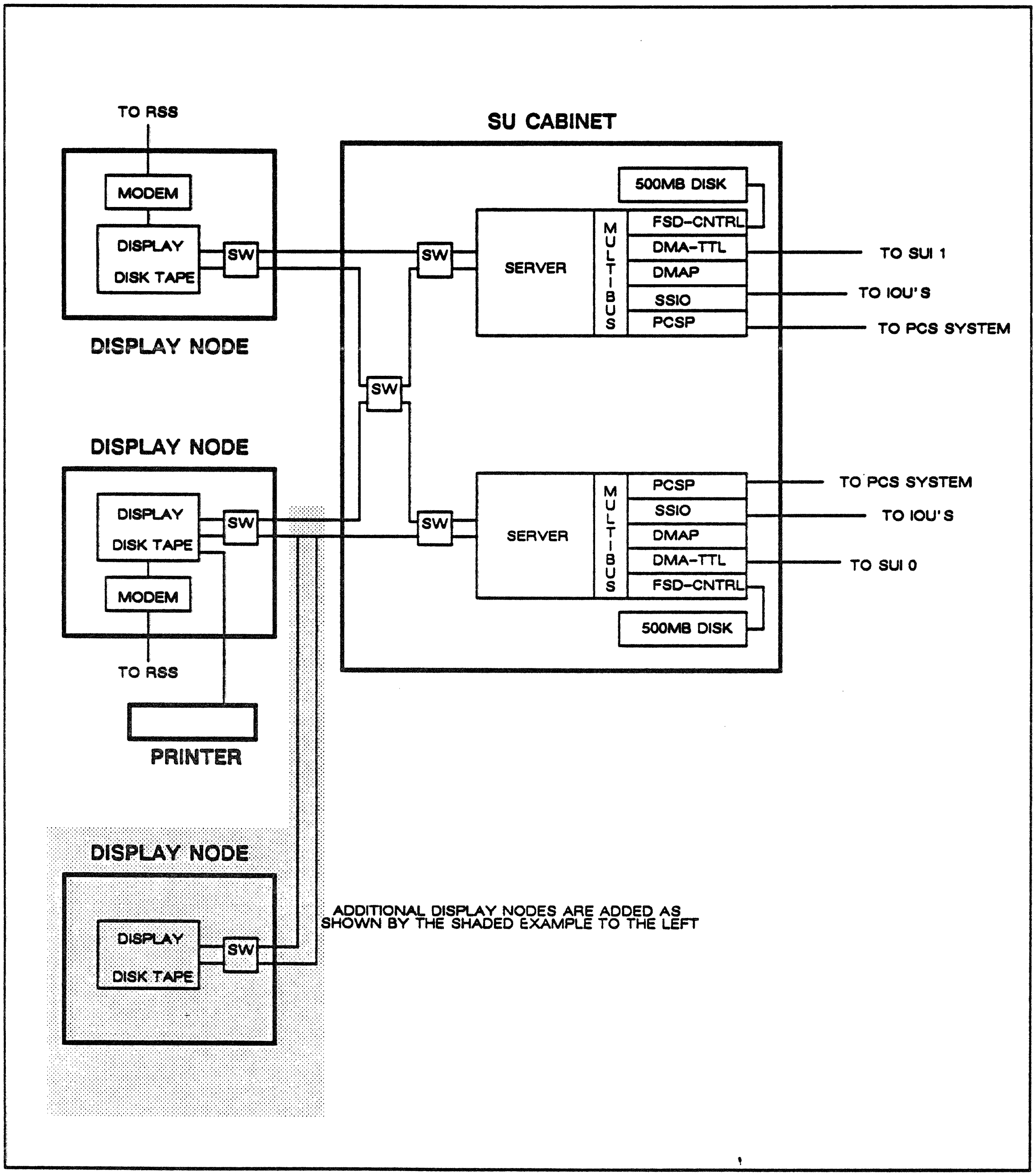


Figure 9–7. Hardware configuration of the SU.

The Graphics Printer

The graphics printer provides a printing capability for the SU network. It is typically used to print reports based on information generated by application programs running on the SU or snapshots (*snaps*) of the display node screen.

Operator Display Nodes

Display nodes are the *operator workstations* of the service unit and are used by applications that require an operator interface. They are the center of system operations, maintenance, and support.

The service unit has at least one, and up to eight, display nodes. Each display node, figure 9-8, includes a desk-like cabinet and a high-resolution, color display. The cabinet houses an 86 Mbyte disk drive and a 60 Mbyte tape cartridge drive. Software for both the service unit and the rest of the ETA10 may be loaded and updated using this 1/4-inch tape cartridge capability. A modem, for connecting the system to remote system support, is also housed in the display node cabinet and connects to a serial I/O port on the display unit. A manual ring switch is housed in each display node cabinet to take the node out of the network if necessary.

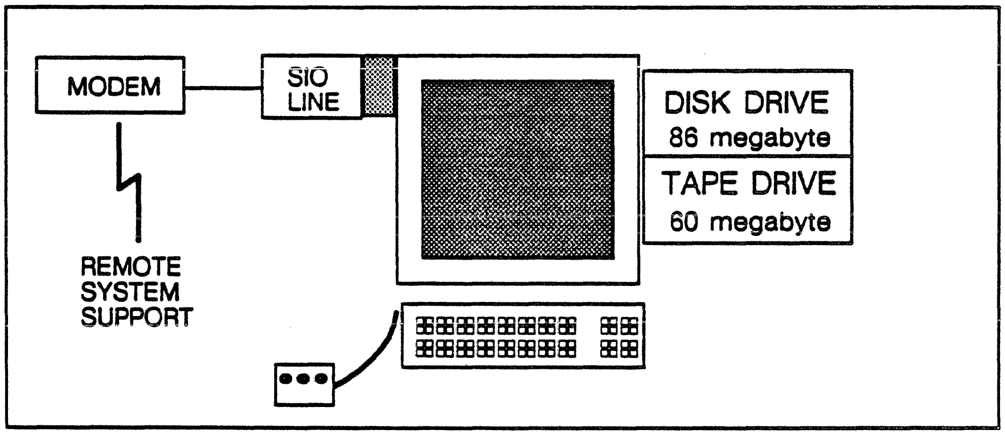


Figure 9-8. Display node components.

Service Unit Applications

The main menu of the SU display provides access to all applications. The three main entries are: Configuration, Operation, and Maintenance.

A list of the available applications is shown in figure 9-9. The following paragraphs describe each application in general terms. For more information, refer to the manuals referenced in the discussion.

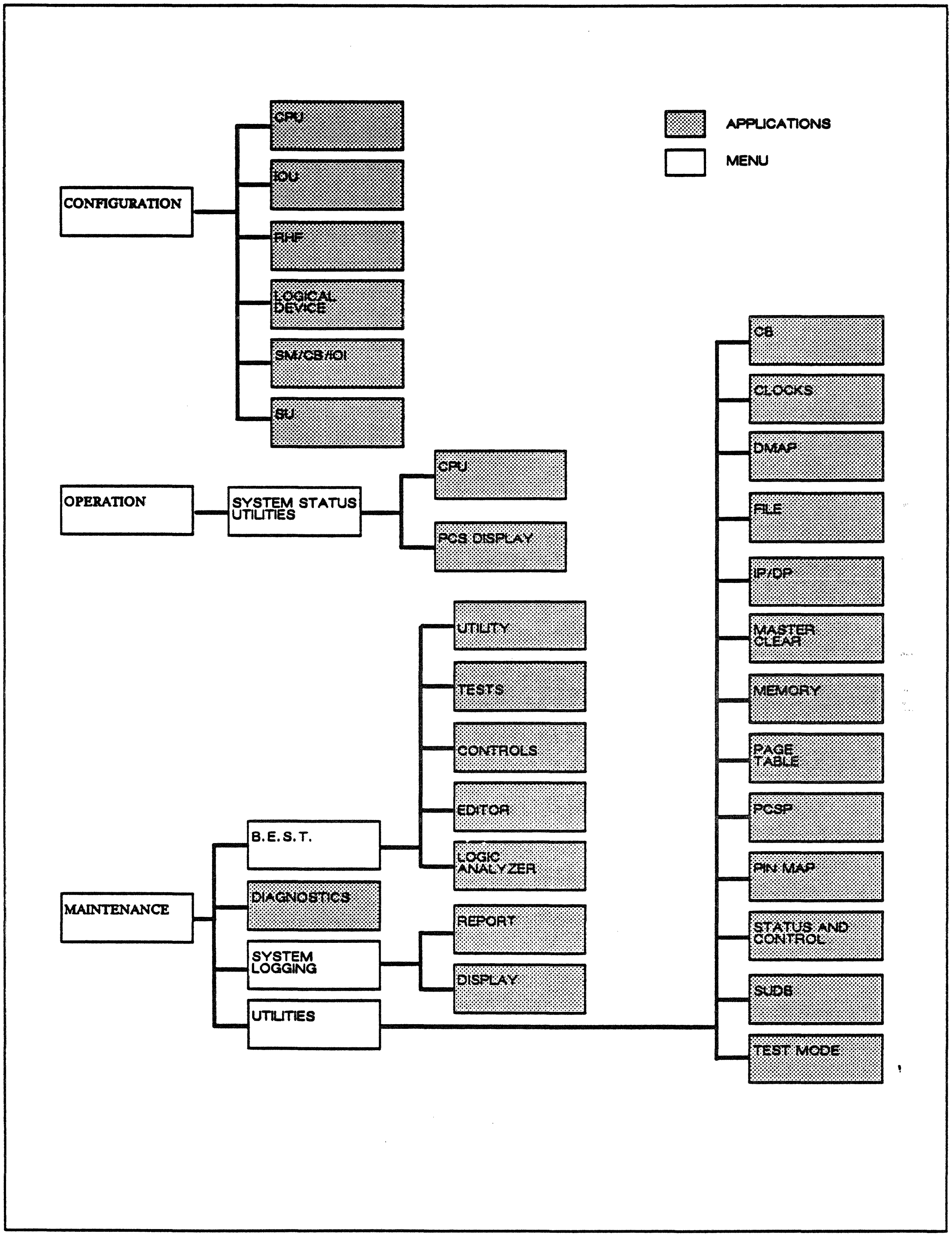


Figure 9–9. The menu-tree of SU applications and menus.

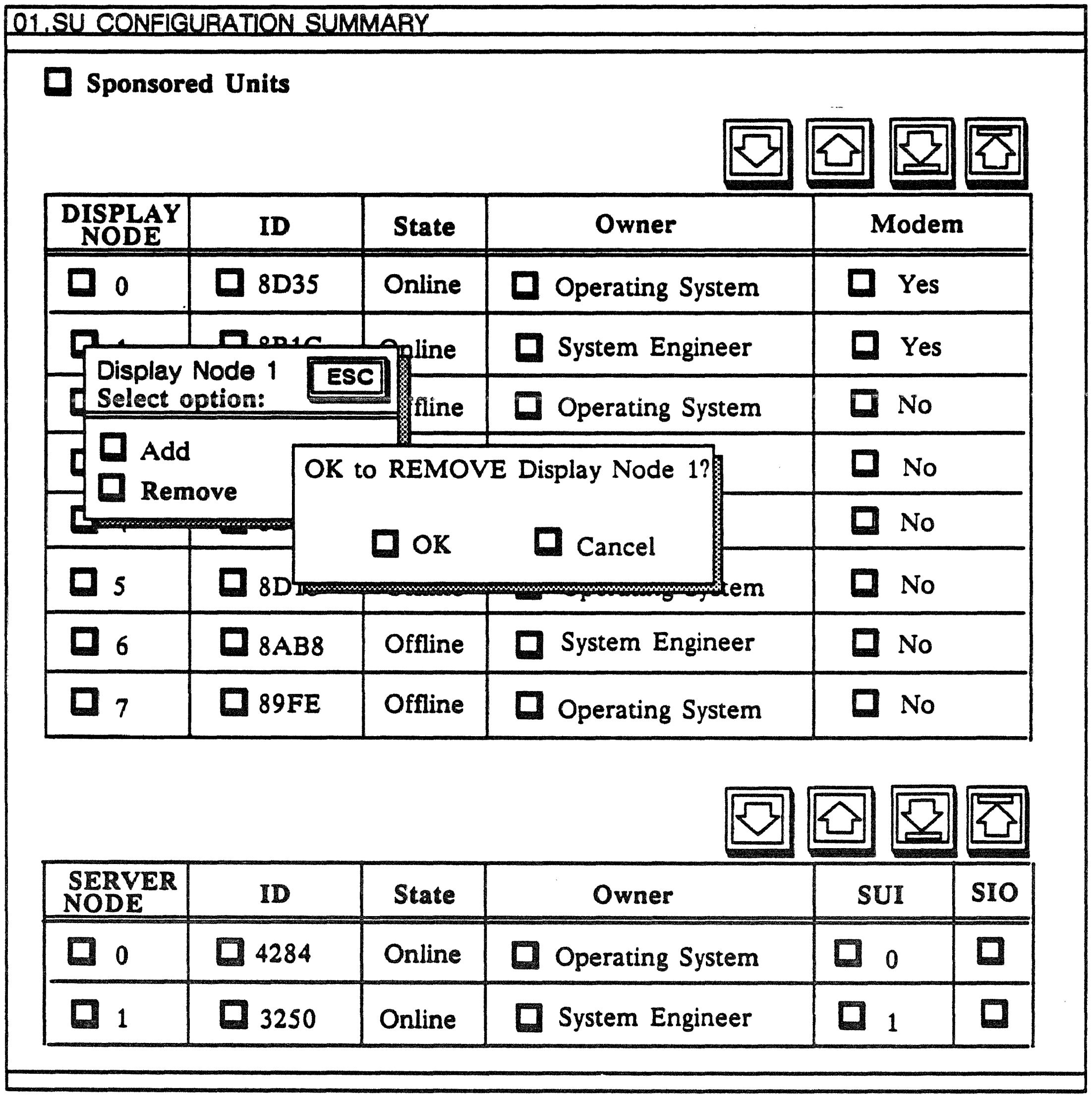
Configuration Related Applications

Configuration related applications allow the operator to install, configure, and reconfigure hardware and software units of the ETA10 System. These applications are generally used only by the system administrator or the lead operator.

The hardware configuration displays (see the example shown in figure 9-10) basically allow the operator to add or remove hardware units from system use, and change various parameters for those units.

Six types of hardware units can be configured from the main Hardware Configuration menu:

- Central processing unit (CPU)
- I/O unit (IOU)
- Remote Host Facility (RHF)
- Logical device
- SM and CB/IOI (SM/CB/IOI)
- Service unit (SU)



01.SU CONFIGURATION SUMMARY

□ **Sponsored Units**

DISPLAY NODE	ID	State	Owner	Modem
□ 0	□ 8D35	Online	□ Operating System	□ Yes
□	□	nline	□ System Engineer	□ Yes
□		ffline	□ Operating System	□ No
□				□ No
□				□ No
□ 5	□ 8D		em	□ No
□ 6	□ 8AB8	Offline	□ System Engineer	□ No
□ 7	□ 89FE	Offline	□ Operating System	□ No

Display Node 1 ESC
Select option:

□ Add
□ Remove

OK to REMOVE Display Node 1?

□ OK □ Cancel

SERVER NODE	ID	State	Owner	SUI	SIO
□ 0	□ 4284	Online	□ Operating System	□ 0	□
□ 1	□ 3250	Online	□ System Engineer	□ 1	□

Figure 9–10. Window into the SU Configuration Summary application.

CPU Configuration

The CPU Configuration display shows the current physical CPU configuration and allows the operator to modify it. The operator can:

- Add or remove a cryostat (vessel) from the configuration.
- Change the cryostat (vessel) serial number.
- Add, move, remove, or swap a CPU from a cryostat (vessel).
- Change the owner of a CPU.
- Change the system start source for a CPU. The *system start source* is a hardware control that allows the operator to change how the system synchronizes itself.
- Change a CPU serial number.
- Monitor the logical state of the CPUs.
- Monitor which server node is connected to which CPU.

All changes to the display are automatically sent to the system configuration table (SCT).

IOU Configuration

The IOU Configuration display application allows the operator to display the current I/O unit configuration and make changes to the configuration.

The information displayed for each I/O unit includes its owner, its current state, and the types of I/O processor boards (IOPs) it contains. The operator can add new I/O units, change ownerships, remove a unit from the system, swap one unit's contents with another's, move the contents from one unit to a different one, and edit the number and type of IOPs in an I/O unit.

In the same manner, information about each I/O processor board can be displayed and its configuration modified.

SU Configuration

The SU Configuration display shows the current physical service unit configuration and allows the operator to modify it. The operator can:

- Remove or add server nodes and display nodes.
- Change display and server node ID numbers.
- Change the ownership of display and server nodes.

- Add or remove an RSS modem connection for the display nodes.
- Change the service unit interface (SUI) number for server nodes.
- Monitor the logical state of display and server nodes.
- Display and change the current sponsor for any existing sponsored unit.
- Change which node the printer is attached to.
- Change the number of available SIO ports.
- Change the SU-to-IOU SIO connections. Two connections are available, one from each server node.

All changes to the display are automatically sent to the system configuration table.

SM/CB_IOI Configuration

The SM/CB_IOI Configuration display allows the operator to monitor and control the configuration of shared memory (SM), the communication buffer (CB), and the input/output interfaces (IOI) of the ETA10 System. Through the SM/CB_IOI Configuration display, the operator can:

- Add or remove an SM or CB/IOI unit.
- Change the SM rank configurations and memory chip size for both SM and CBM.
- Change the ownership of SM or CB/IOI units.
- Change the system start source for SM or CB/IOI units.
- Change the serial number of SM or CB/IOI units.
- Monitor the logical state of SM and CB/IOI units.
- Monitor the SU sponsorship of the SM and CB/IOI units.

All changes to the display are automatically sent to the system configuration table.

RHF Configuration

The RHF (Remote Host Facility) Configuration display allows the operator to configure the network facilities for the computer system. All changes to the display are copied to the system configuration table (SCT). The operator can:

- Monitor current network configuration. Descriptions for each host includes: host description, site (local or remote), host type (CY200, CY280, DEC, ETA, and IBM), physical identifier, status (enable/disable), NAD assignment, and trunk controller unit (TCU) connections.
- Add, delete, or modify the parameters for both local and remote hosts
- Add, delete, or modify parameters for trunk control unit (TCU) identifiers
- Add, delete, or modify the parameters for both local and remote network access devices (NADs)
- Enable or disable each network host and monitor the logical identifier.

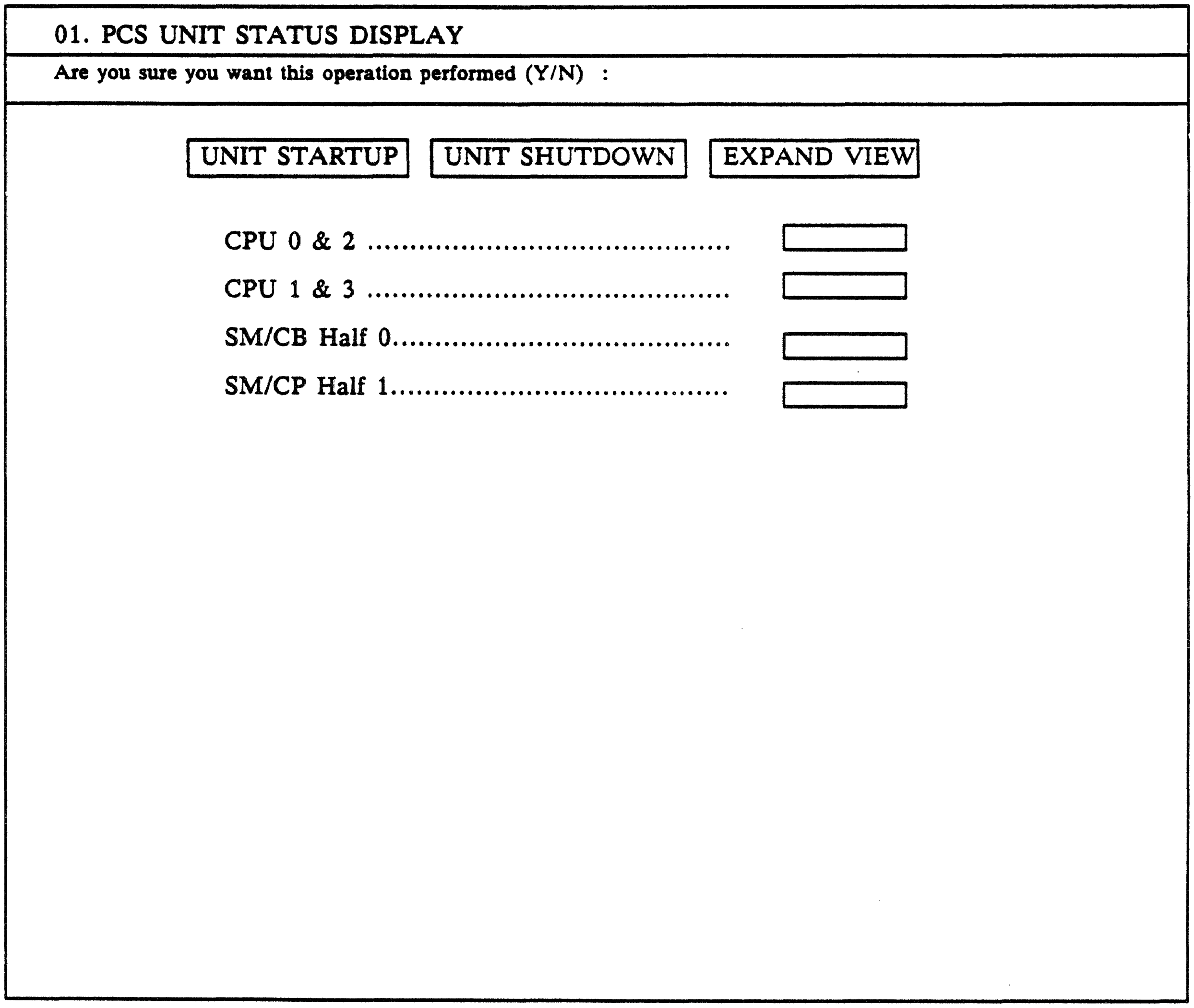
Logical Device Configuration

Logical Device Configuration display allows the operator to configure the logical disk devices for the computer system. All changes to the display are copied to the system configuration table (SCT). Using the Logical Device Configuration display the operator can:

- Create or delete a device class (i.e., system backup, system primary, etc.).
- Add or remove a logical device from a device class.
- Display a summary of all logical devices by logical device number or logical device set.
- Create new logical sets by configuring new physical drives into logical devices or combining small slice sets into larger slice sets.
- Display the logical set containing the selected logical devices in order to switch logical device access from one I/O unit to another or add a logical device to the logical set.

Operation Related Applications

Operation related applications (see example display in figure 9-11) allow the SU operator to monitor the status of and control the system's CPUs and the Power and Cooling Supervisor.



01. PCS UNIT STATUS DISPLAY

Are you sure you want this operation performed (Y/N) :

CPU 0 & 2 .... □

CPU 1 & 3 .... □

SM/CB Half 0....................................... □

SM/CP Half 1....................................... □

Figure 9-11. PCS System Status and Control display.

System Status Utilities

The System Status Utilities menu allows the operator to control and monitor the status of the CPUs and the PCS (Power and Cooling Supervisor).

CPU allows operators to view the CPU configuration and some of the attributes associated with each CPU including the state of the CPU, the current instruction address register contents, the operating mode, any fatal faults, and the stop/idle flags. Through submenus of the CPU main display, the operator can also view the hardware status and process status for each CPU. Information available through these displays include:

- Hardware status, including: associative register status, invisible package base address, address parity errors, and so on. Detailed descriptions are given for SECDED single errors, SECDED double errors, address parity errors, and data parity errors.
- Process status, including: process ID, state, and total number of processes.

PCS Display allows operators to monitor all points reporting to the Power and Cooling Supervisor (PCS) and control the operation of many power and cooling elements. (See Chapter 10, Power and Cooling Systems, for a complete description of the power and cooling supervisor.)

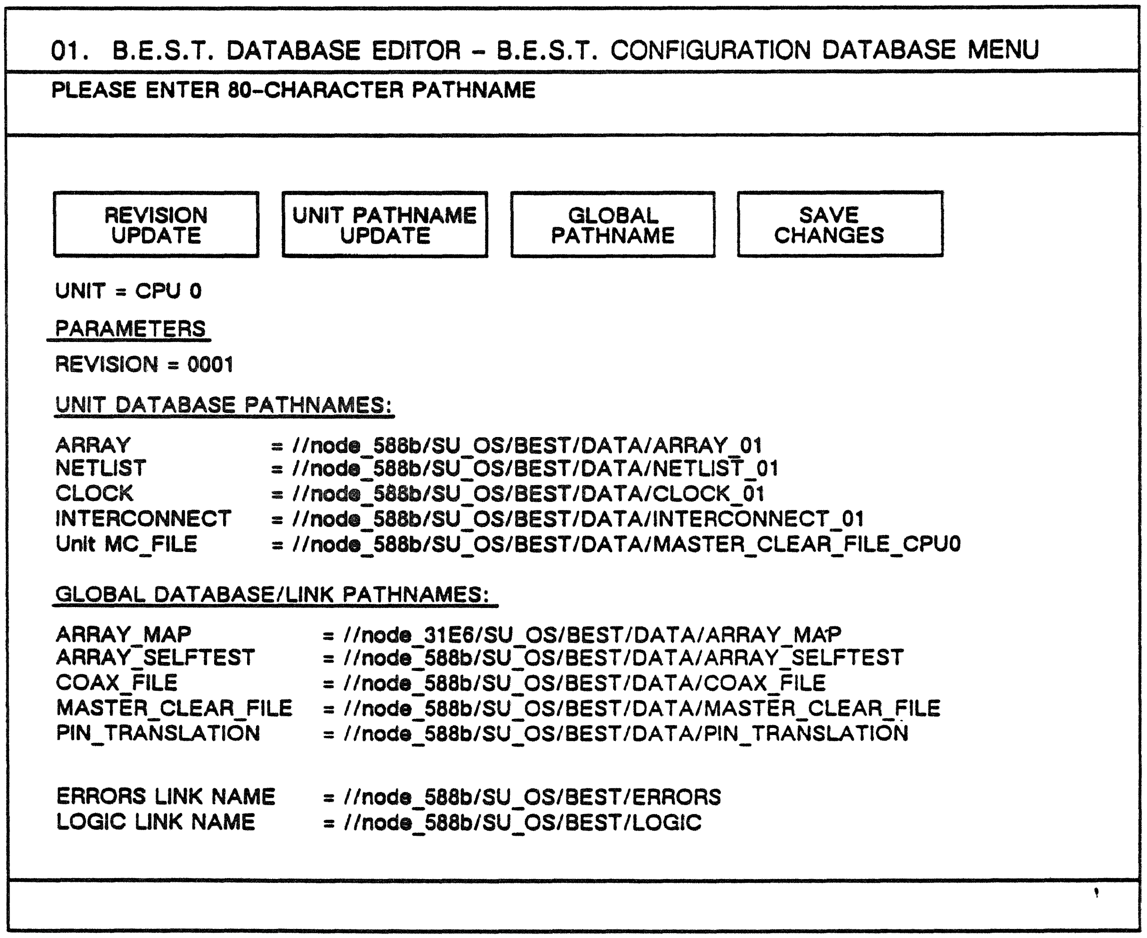
Through the PCS displays the operator can monitor temperature, voltage, current, liquid nitrogen (LN_2) level, and so on. Range indicators help operators determine whether points are moving away from their normal positions or remaining stable. Operator control of functions such as power on/off, setting voltage margins, adjusting freon valves, enabling and disabling the MI connection to each data acquisition processor (DAP), and so on are also a part of the PCS display.

Maintenance Related Applications

These applications provide a wide range of maintenance functions by allowing the operator to:

- Run B.E.S.T. tests on the high-tech boards.
- Run diagnostics. All diagnostics are controlled through the SU.
- Run utilities
- Log maintenance events and query the event database

An example of a typical maintenance display is shown in figure 9–12. In general, only the maintenance engineer uses these displays.



```
01. B.E.S.T. DATABASE EDITOR - B.E.S.T. CONFIGURATION DATABASE MENU
PLEASE ENTER 80-CHARACTER PATHNAME

UNIT = CPU 0
PARAMETERS
REVISION = 0001
UNIT DATABASE PATHNAMES:
ARRAY              = //node_588b/SU_OS/BEST/DATA/ARRAY_01
NETLIST            = //node_588b/SU_OS/BEST/DATA/NETLIST_01
CLOCK              = //node_588b/SU_OS/BEST/DATA/CLOCK_01
INTERCONNECT       = //node_588b/SU_OS/BEST/DATA/INTERCONNECT_01
Unit MC_FILE       = //node_588b/SU_OS/BEST/DATA/MASTER_CLEAR_FILE_CPU0
GLOBAL DATABASE/LINK PATHNAMES:
ARRAY_MAP            = //node_31E6/SU_OS/BEST/DATA/ARRAY_MAP
ARRAY_SELFTEST       = //node_588b/SU_OS/BEST/DATA/ARRAY_SELFTEST
COAX_FILE            = //node_588b/SU_OS/BEST/DATA/COAX_FILE
MASTER_CLEAR_FILE    = //node_588b/SU_OS/BEST/DATA/MASTER_CLEAR_FILE
PIN_TRANSLATION      = //node_588b/SU_OS/BEST/DATA/PIN_TRANSLATION

ERRORS LINK NAME     = //node_588b/SU_OS/BEST/ERRORS
LOGIC LINK NAME      = //node_588b/SU_OS/BEST/LOGIC
```

Figure 9–12. An example of a maintenance application display (for B.E.S.T.).

Built–in Evaluation and Self–Test (B.E.S.T.) Functions

ETA System's Built–in Evaluation and Self–Test (B.E.S.T.) System provides dedicated software and hardware that allows the SU operator to detect problems in the ALSI–20K gate array boards of the mainframe.

The B.E.S.T. system uses on-chip maintenance logic to perform array level, board level, and board interconnection tests. B.E.S.T. includes features for testing arrays, editing test output, controlling some aspects of array operation, and analyzing array and board logic.

Figure 9–13 shows the system components covered by B.E.S.T., and the following paragraphs briefly describe the four B.E.S.T. functions. For a full description of B.E.S.T., see *PUB–1118, Support Tools: Utilities, Debugger, B.E.S.T.*

Tests is a series of tests of the circuit boards to verify the B.E.S.T. logic, the array matrix, the interconnect paths between the arrays on each board, and the interconnect paths between the boards. It ensures that the hardware is in place, is at the correct revision level, and is operating properly.

Editor allows the SU operator to read and write to the database maintained on each of the ALSI 20K gate array boards in the system. The database includes a description of each array, its interconnects, its clock tuning parameters, master clear data, checksum self-test, and pin translation tables. Installing and updating engineering change orders is done through the B.E.S.T. editor.

Controls allows the operator to check or modify the operations of the B.E.S.T. logic on each array. Options enable the write chain to be edited, the read chain data to be displayed, and the B.E.S.T. function registers to be edited. Using this application, you may also read and write the chain and read and write the function registers.

Logic Analyzer allows the operator to examine the logic states of the input/output pins on a board populated with the ALSI 20K gate arrays. It captures the states of all I/O pins for a specified number of clock cycles and can display the state of selected pins. Event cycles and cycle counts may be specified by the operator. Trace data captures include unconditional, conditional, and trigger search options. The display shows the logic states as either a trace or as hexadecimal data.

Utility consists of five features that make interfacing to B.E.S.T. software easier and provide additional help in using B.E.S.T. to troubleshoot the ALSI-20K boards. The features allows the SU operator to define the B.E.S.T. software database configuration, display the engineering change order record for a specified board, translate real pin to virtual pin numbers, and write the clock and B.E.S.T. chains with specific patterns during troubleshooting procedures.

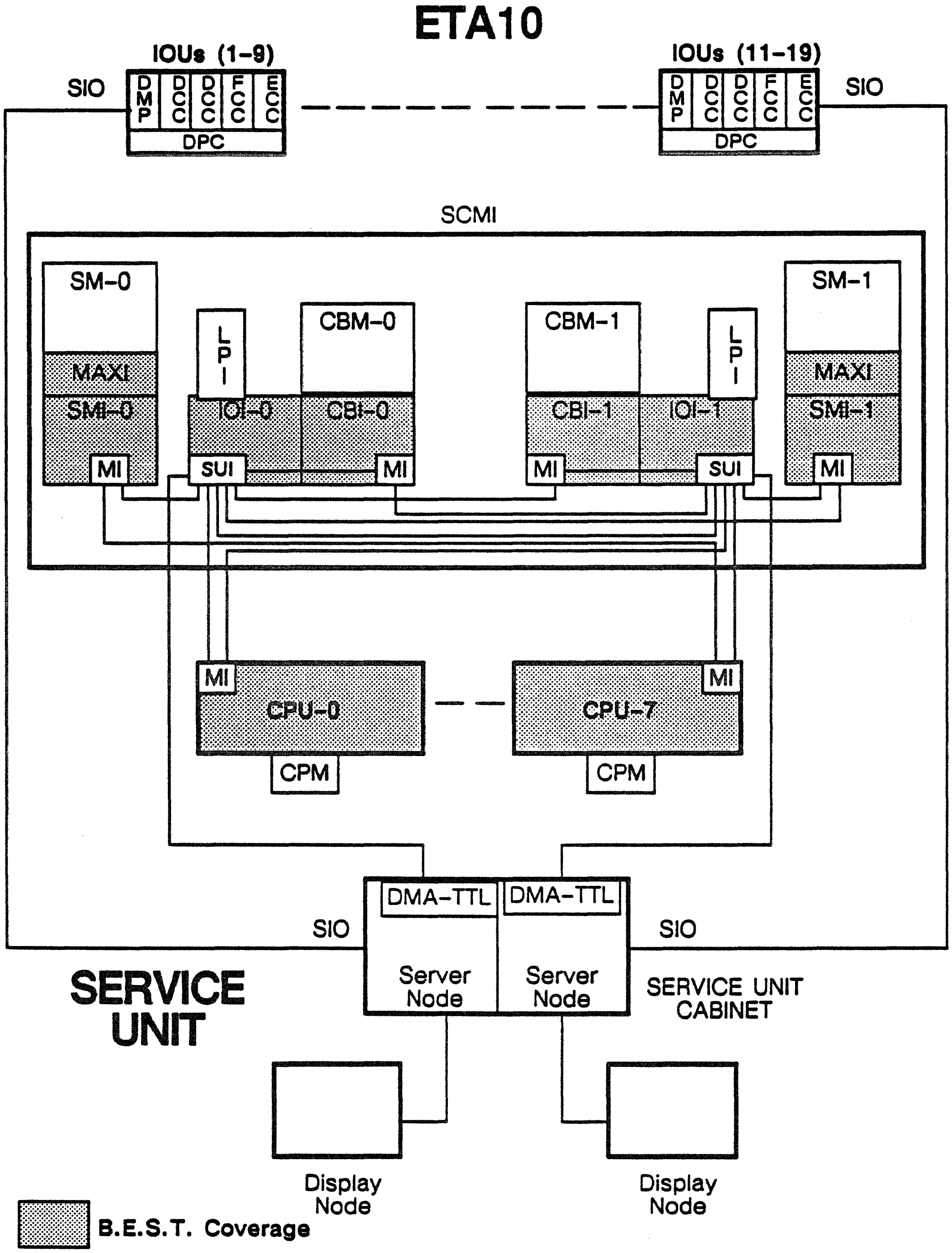


Figure 9-13. B.E.S.T. coverage of the ETA10 System.

Diagnostics Available Through the Service Unit

Diagnostics test the fundamental workings of the system by exercising the hardware to provide the information needed to resolve problems.

The SU maintenance operator can test all components of the ETA10 System using off-line diagnostics. Although all diagnostics are controlled through the service unit and the diagnostic applications, they may actually be loaded into and run on other system components under control of the service unit. Diagnostics are designated by the component on which they run. The three types of diagnostics are:

- CPU–based diagnostics
- IOU–based diagnostics
- SU–based diagnostics

Figure 9–14 shows the coverage of the diagnostics on the ETA10 System.

The ETA10 System is tested by component, generally a system cabinet or a large assembly that contains a set of elements. Each diagnostic contains a group of modules that test individual hardware functions, instructions, or situations consisting of sequence–dependent failures or sequence timing errors.

When diagnostics execute under control of the service unit, they own the hardware under test. That hardware must be demoted to the Maintenance state. If the diagnostic requires another unit to perform the testing, that unit must also be in the Maintenance state.

The paragraphs below present a complete list of the diagnostics currently available on the ETA10 System and a brief description of each. More detailed information about the diagnostics and how to run them can be found in *PUB–1119 Support Tools: Diagnostics.*

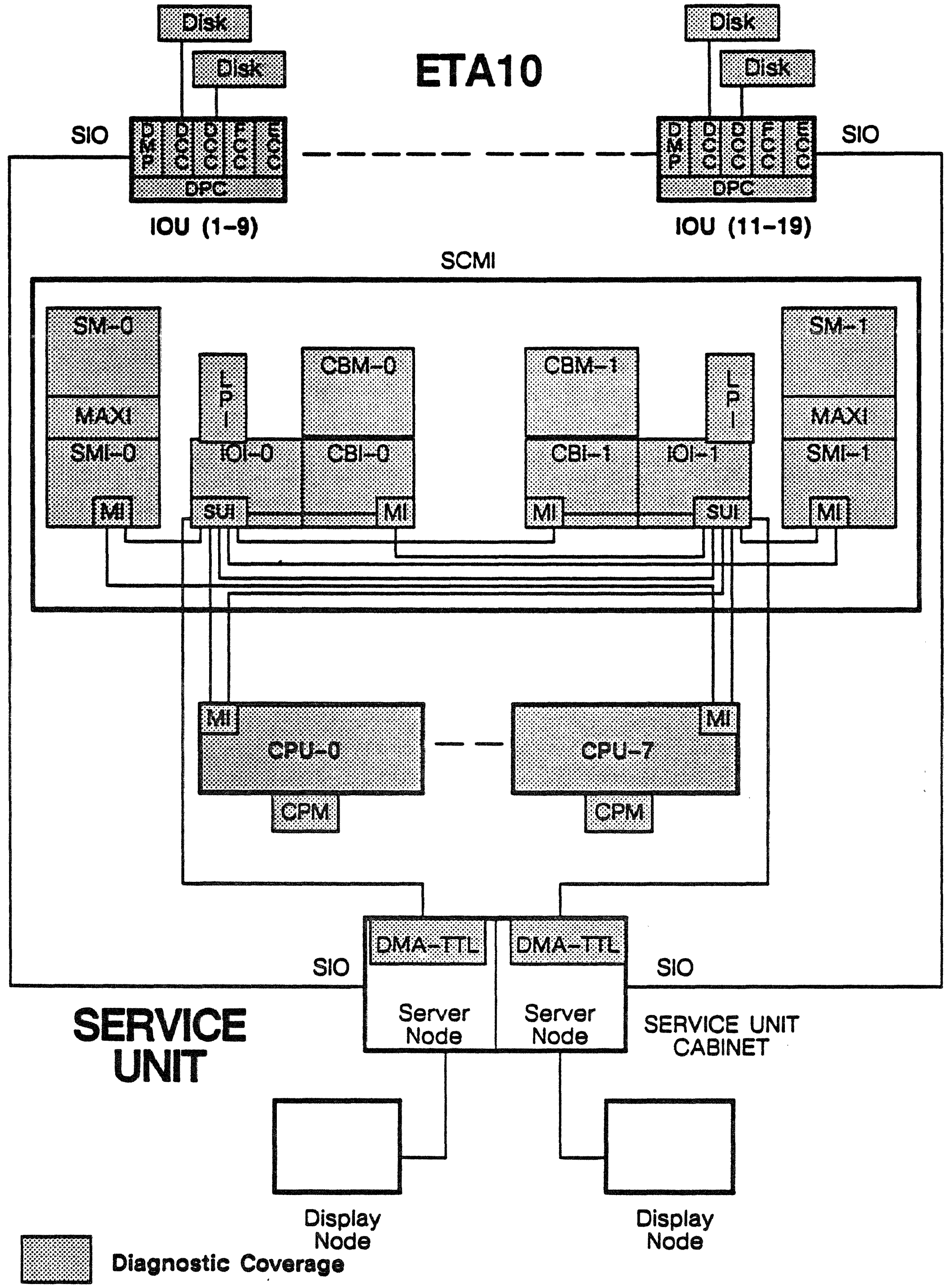


Figure 9-14. Diagnostic coverage of the ETA10 System.

CPU-Based Diagnostics

CPU-based diagnostics are loaded into the CPU from the service unit and run under control of the service unit. The following list comprises the CPU-based diagnostics.

Fixed Operand Command Tests, cpu_COMA through cpu_COMG, use fixed operands to verify the operation of the ETA10 instructions.

- cpu_COMA - checks all the options available to the CPU scalar fixed-point instructions.
- cpu_COMB - checks all the options available to a CPU scalar floating-point instruction.
- cpu_COMC - checks all the options available to a CPU normal vector instruction.
- cpu_COMD - checks all the options available to a CPU vector logical or scanning instruction.
- cpu_COME - checks all the options available to a CPU sparse vector instruction or vector macro instruction.
- cpu_COMF - checks all the options available to a CPU vector macro instruction or non-typical instructions not tested in previous tests.
- cpu_COMG - checks all the options available to the CPU select-link instructions.

cpu_MEMA checks CPU memory. Using load and store (5E/5F and 7E/7F) and other basic instructions, MEMA tests all writing and reading of central memory. MEMA writes test patterns, reads them back, and (with optional parameters) checks the data for validity. The diagnostic also contains a utility package to assist in troubleshooting memory problems.

cpu_VMD uses various instruction sequences to test CPU memory and its associated read and write paths.

cpu_RVEC verifies the operation of the floating-point hardware contained in Pipe 1 and Pipe 2 when operating in the streaming or vector mode. Both pipes execute the same instruction sequence using the same set of random operands and then compare the results.

Floating-point instructions are tested in order of ascending difficulty, primarily using random operands. Testing is done by duplicate hardware or by simulation of the instruction where duplicate hardware is not available.

cpu_REGD detects marginal errors and intra-chip read and write conflicts in the register file. REGD uses both fixed instruction sequences and randomly generated instruction sequences of 2C, 2D, 2E, or 30 instructions to test each selected group of registers.

cpu_RWO detects sequence time dependent errors in instruction execution. For each condition, RWO executes two sequences of the same instructions, each having random operands. The first sequence executes with no-ops interspersed among the instructions, or by emulating a switch, it executes using the hardware serial mode options. The second sequence executes with no interspersed no-ops. After executing both sequences, RWO compares the register file and the memory data flag register of the two sequences.

cpu_RAN detects sequence dependent instruction failures by simulating the instructions in each test string. RAN uses an ordered random sequence that is able to re-create any circumstances with a master random number.

cpu_CB_ZAP verifies the basic operation of the FE/FF (load/store) instructions. Then using these instructions, it tests addressing, control, and data retention of CB memory.

cpu_SM_ZAP verifies shared memory, the basic functions of the four SM instructions (18, 19, 1A, 1B), the SM registers (IQHA, IQTA, CQTA, and TRBSA), and the TRBs. The diagnostic verifies both monitor mode and job mode operations. Job mode operations are controlled by the CPU-based diagnostic interface.

cpu_MAINT_CNTRL_HRDW verifies the operation of a variety of features in the CPU. It checks the functionality of hardware features such as the Data Flag Branch Register (DFBR), the Breakpoint Register (including the comparison hardware), the Trace Register, and the Illegal Instruction Mask. It also tests interrupts generated by Type I and Type II illegal instructions.

cpu_BASIC_VIRTUAL tests the virtual addressing and job mode execution hardware in the CPU. The test detects failures in the Associative Registers (ARs), the load/store capability of the ARs, the load/store paths of the invisible package, space table search hardware, the job/monitor exchange, and the Job mode virtual address system.

cpu_CB_FUNCTION verifies all CB operations associated with a single CPU and a single CBI. The features that are verified with this diagnostic include:

- Post Semaphore instruction
- Wait Semaphore instruction
- Bit Branch and Swap instruction
- Bit Branch and Load/Store instruction
- CB Load and CB Store instructions
- CB instruction buffer
- Semaphore addressing/address incrementer
- Semaphore wait count incrementer/decrementer
- Semaphore Q-head/Q-tail addressing
- All G-bit subfunctions
- Monitor /Job mode execution
- CB access lockout codes
- CB base/limit pair

cpu_SM_FUNCTION verifies word/halfword transfers between CPU memory and shared memory. The diagnostic executes four SM instructions, 18, 19, 1A, and 1B, and verifies that the instructions, registers, and associated hardware features work correctly.

cpu_COUNTS tests the instrumentation counters, monitor interval timer, job interval timer, and the real time clock. The diagnostic also detects failures in the counter operations and failures in the instrumentation counter select networks.

cpu_DOMAIN tests the domain changes in the CPU hardware and detects failures in the domain packages and stacked domain packages (domain stack).

cpu_CB_MEMORY_STRESS tests data retention of a specified area of CB memory that is selected with parameters. The diagnostic can use either CBI 0 or CBI 1 and supports both 1 and 4 million word CB memories.

cpu_SM_MEMORY_STRESS verifies specified area of Communication Buffer memory from the CPU. Both random data and addresses are used.

SU-Based Diagnostics

SU-based diagnostics are loaded and executed from the service unit. The following list comprises the SU-based diagnostics.

su_CPM_PATH is designed to verify the proper operation of the path from the MI to the CPU memory stacks and to the individual stacks. Failures are detected in CPU memory due to unused address bits, memory wrap-around and Chip Size parameters in the system configuration file.

su_CPM_BASIC_MEMORY is a continuation of the su_CPM_PATH diagnostic. It checks for data retention of a specified area of memory that is selectable from parameter options. The test default settings check an area of memory large enough for diagnostics requiring a portion of CPM to be operational. Errors are detected and reported using data checking and SECDED error latching under operator control.

su_CBM_PATH is used to test the path from the IOI's SU port to the communication buffer (CB) memory stacks and then to the stacks. The CB Load command and CB Store command are also tested.

su_CB_BASIC_MEMORY is an extension of the su_CBM_PATH diagnostic and tests the data retention of a specified area of the CB that is selectable by test parameters. Errors are detected and reported using data checking and SECDED error latching.

su_CB_BASIC_FUNCTION is a continuation of the su_CBM_Path and su_CB_Basic_Memory diagnostics by completing the testing of the CB Load and CB Store commands. It also tests the Timestamp, Post Semaphore, Wait Semaphore, Bit Branch And Swap, and Bit Branch And Load/Store commands.

su_SM_PATH checks the path from the MI on the IOI/CBI to the SM stacks, and also tests basic operation of the stacks. The test supports both 256K and 1 Mbyte memory chips.

The test is capable of using either SMI 0 or SMI 1 in combination with either the IOI 0 or IOI 1 board in order to complete the defined tests.

su_SM_BASIC_MEMORY is a continuation of the su_SM_PATH diagnostic. It tests the data retention of a specified area of shared memory (selectable by parameters). The test supports both 256K and 1 MB memory chips. The default settings that are provided test an area of memory large enough for diagnostics requiring a portion of SM to be operational. Errors are detected and reported using data checking and SECDED error latching.

A feature is provided to ignore the single bit SECDED error status when SECDED correction is enabled, allowing memory to be checked to the level that it is usable by other diagnostics. When a more thorough test of SM is desired, SECDED correction can be disabled and any single SECDED error status is monitored along with data check.

su_MAINT_INTERFACE is designed to verify proper operation of the Maintenance Interface (MI) path from the SU to the Service Unit Interface (SUI) on any of the 20K-ALSI boards.

su_CP_SECDED tests the functionality of the CPM SECDED logic as well as the SECDED related MIDR and MODR bits. The diagnostic assumes that bit addresses 0 through 40000 of the selected CPM have been verified error free with SECDED correction off. su_CP_SECDED detects and reports errors using data checking and SECDED error latching, and supports both 64K and 256K memory chips.

su_CB_SECDED tests the functionality of the CB SECDED logic as well as the SECDED related MIDR and MODR bits. The diagnostic assumes that the selected CB has been verified error free with SECDED correction off. su_CB_SECDED detects and reports errors using data checking, status checking, and SECDED error latching, and supports both 64K and 256K memory chips.

su_SM_SECDED tests the functionality of the SM SECDED logic as well as the SECDED related MIDR and MODR bits. The diagnostic assumes that the selected SM has been verified error free with SECDED correction off. su_SM_SECDED detects and reports errors using data checking and SECDED error latching, and supports both 256K and 1 Mbit memory chips.

su_ACOM is used to confirm the operation of the scalar fixed-point instructions. The diagnostic provides the required testing before the CPU can diagnose itself. The ACOM test provides fixed-point instruction sequences, and tests each instruction using single instructions, multiple instructions, and conflict sequences.

su_IOP_SBC verifies that the communication link to the selected IOP is sound and the basic operation of the single board computer's (SBC's) memory, parity checkers, timers, processor, interrupts, common bus, and local bus.

IOU-Based Diagnostics

IOU-based diagnostics are loaded into the IOU from the service unit and run under control of the service. The following list comprises the IOU-based diagnostics.

iou_CB_BASIC_MEMORY is an extension of the iou_CBM_Path diagnostic. It tests data retention of a specified area of the CB that is selectable by test parameters. Errors are detected and reported using data checking and SECDED error latching.

iou_CB_BASIC_FUNCTION is a continuation of the iou_CBM_Path and iou_CB_Basic_Memory diagnostics. It completes the testing of the CB Load and CB Store commands, and also tests the Timestamp, Post Semaphore, Wait Semaphore, Bit Branch And Swap, and Bit Branch And Load/Store commands.

iou_SM_PATH checks the path from the IOI port on the IOU to the SM stacks, and tests basic operation of the stacks. The test supports both 256K and 1 MB memory chips.

iou_SM_BASIC_MEMORY is a continuation of the iou_SM_Path diagnostic. It tests data retention of a specified area of shared memory that is selectable by parameters. The test supports both 256K and 1 MB memory chips. The default settings that are provided test an area of memory large enough for diagnostics requiring a portion of shared memory to be operational. Errors are detected and reported using data checking and SECDED error latching.

su_SBC communicates with any single board computer (SBC) over the SIO interface to ensure that the path exists and to verify that the SBC is operational. The test also exercises the communication protocol and ROM-based vendor-supplied diagnostics by using calls to the ROM-based kernel/monitor.

iou_DPB exercises the data pipe buffer (DPB) hardware in an IOP and verifies that the DPB is operational. Most hardware failures in the DPB are detected by this diagnostic. The only functions not tested are data transfers to and from the data pipe controller (DPC) and disk controller interface (DCI). If an error is detected it is isolated to the DPC currently under test.

iou_DPC exercises the data pipe controller (DPC) logic and the various data paths associated with it. It verifies that the DPC is operational and ensures that the data pipe buffer (DPB) can deliver and receive data through the DPC as expected.

iou_DCI exercises the disk controller interface (DCI) hardware in a disk controller channel (DCC). It verifies that the DCI is operational and ensures that the DPB can deliver and receive data through the DPC as expected.

iou_IDSM exercises an intelligent disk storage module (IDSM). All drive commands and all transfer types are tested. A drive that passes all tests without failure is presumed operational.

iou_FCI exercises the FIPS channel interface (FCI) logic using vendor-supplied diagnostic instructions.

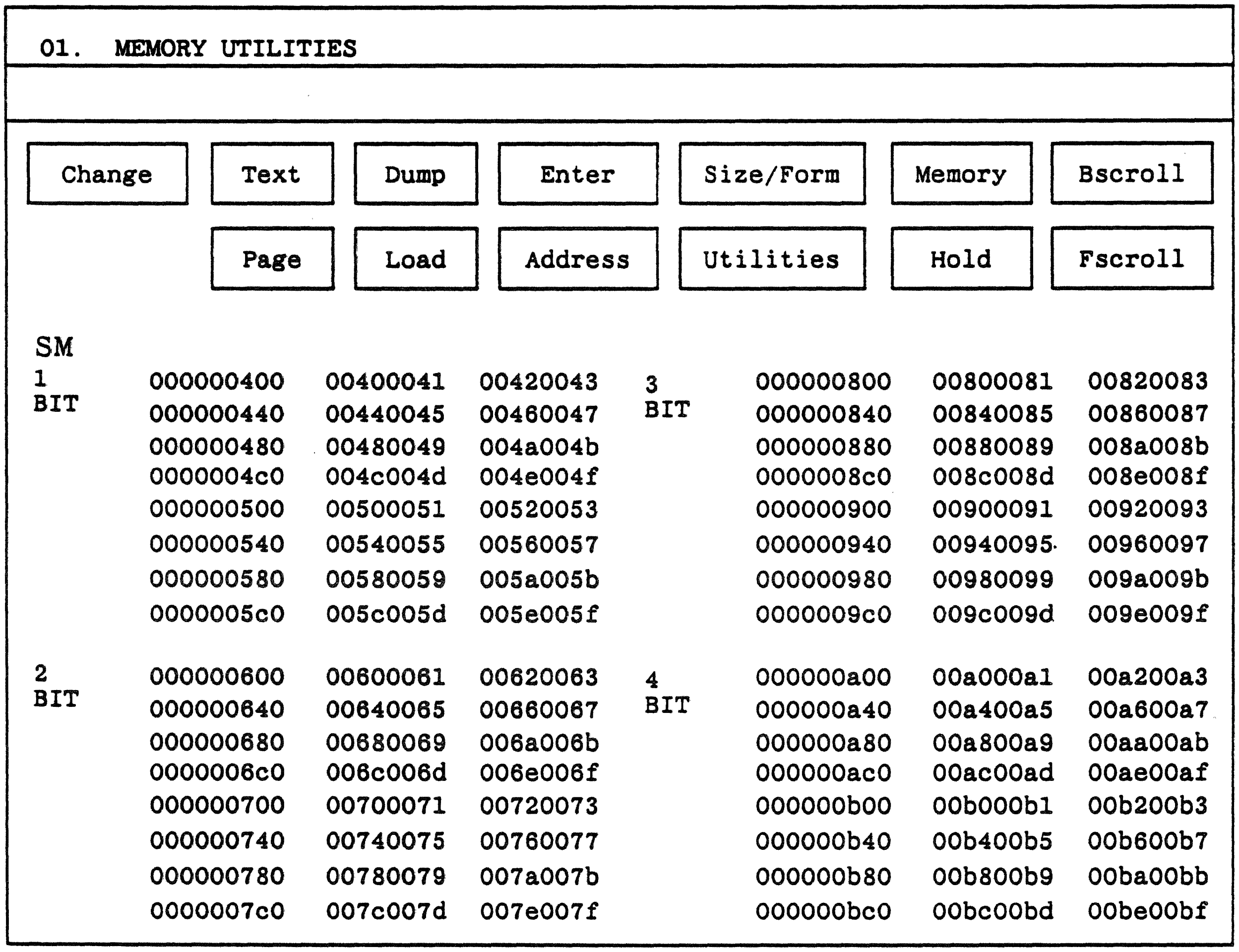
iou_NAD_PATH verifies the functionality of the lines between the FIPS-60 channel and the IBM NAD by passing data patterns to the selected area in the NAD memory.

su_EXP_MEM communicates over the RS-232 line with the single board computer that contains the expansion memory to insure correct operation of the path and verify that memory is operational. It exercises the ROM-based vendor-supplied diagnostics by calls to the ROM-based kernel/monitor.

Utilities Available on the Service Unit

Utilities available through the service unit allow operators to get a more detailed look at system components. Figure 9–15 shows an example of one of the utility displays.

Detailed descriptions of the utilities and how-to-use information is available in *PUB–1118, Support Tools: Utilities, Debugger, B.E.S.T.*



```
SM
1      000000400  00400041  00420043   3      000000800  00800081  00820083
BIT    000000440  00440045  00460047   BIT    000000840  00840085  00860087
       000000480  00480049  004a004b          000000880  00880089  008a008b
       0000004c0  004c004d  004e004f          0000008c0  008c008d  008e008f
       000000500  00500051  00520053          000000900  00900091  00920093
       000000540  00540055  00560057          000000940  00940095  00960097
       000000580  00580059  005a005b          000000980  00980099  009a009b
       0000005c0  005c005d  005e005f          0000009c0  009c009d  009e009f

2      000000600  00600061  00620063   4      000000a00  00a000a1  00a200a3
BIT    000000640  00640065  00660067   BIT    000000a40  00a400a5  00a600a7
       000000680  00680069  006a006b          000000a80  00a800a9  00aa00ab
       0000006c0  006c006d  006e006f          000000ac0  00ac00ad  00ae00af
       000000700  00700071  00720073          000000b00  00b000b1  00b200b3
       000000740  00740075  00760077          000000b40  00b400b5  00b600b7
       000000780  00780079  007a007b          000000b80  00b800b9  00ba00bb
       0000007c0  007c007d  007e007f          000000bc0  00bc00bd  00be00bf
```

Figure 9–15. A typical utility display (memory display in 4 blocks, hexadecimal without text).

File

The file utilities allow the SU operator to manipulate files and directories that have been created on the service unit by other programs, such as diagnostic programs. Using the file utilities, the operator can:

- Copy, rename, or delete files
- Compare two binary files
- Find information about files and directories
- Move about in the file hierarchy
- Display the contents of one or two directories
- Set the access permission for files and directories
- Create new directories

SU Debugger (SUDB)

The Service Unit-based debugger is known as SUDB. SUDB aids in start-up and debugging of the kernel, the Pager, and other basic components of the operating system. Through SUDB, the operator can view breakpoints and the Current Instruction Address Register (CIAR) and the Current Instruction Register (CIR) as external objects. Breakpoints are manipulated on the CPU.

IP/DP

The Invisible Package/Domain Package display utility provides a dynamic screen display of the contents of the invisible package, domain packages, or stacked domain packages. The information displayed contains:

- Current page sizes
- Stack index and limit
- Previous and current domain packages
- Data Flag register contents
- Space table keys
- Instrumentation counter contents
- Interrupt counter contents
- Load/Store register contents

The utility can be used only with CPU memory. It allows the user to scroll to the next or previous Domain or Stacked Domain package, or to enter the number of a specific package.

Page Table/Associative Registers

This utility provides an on-line screen display of the information contained in the Space table and associative registers. It displays the associative word as it exists in memory, the corresponding entry number, the usage code, lock code, virtual bit address, and physical bit address. Through the display the operator may:

- Change the base address from which to view the Space table.
- Select an entry number to be displayed.
- Select a different CPU.
- Move to the base address.
- Move to first 'END' usage code starting from the base address.
- Scroll up or down through the space table.

Memory

The Memory utility is used to display or modify the data stored in a selected memory type: shared memory (SM), communication buffer (CB) memory, and central processor memory (CPM). Through the display, the operator may:

- Display or enter memory using various format options and two different display window size options.
- Display data as hexadecimal only or hexadecimal with ASCII text representation.
- Display data in two, four, or eight separate blocks with different addresses.
- Enter or display addresses in bit, halfword, or word mode.
- Enter a CPM memory address as a page address and display the memory at the corresponding bit address.
- Scroll through the displayed memory.
- Copy a block of memory from one area to another.
- Pattern a memory area using fixed or incrementing patterns, and then verify the pattern.
- Search for a pattern in memory. Either an equal or unequal search can be made.
- Compare two areas of memory.

- Load a file to a specified area of memory.
- Dump the contents of a specified area of memory to a file.

Clock

Every array on each CPU, CPM stack, SMI, CBI/IOI, and CBM stack has a coarse and a fine clock value. The coarse value provides the clock delay in nanosecond units, the fine value in picosecond units. These clock values can be individually adjusted (tuned). Clock tuning values are permanently maintained in the clock tuning database of the SU. The Clock utility allows the operator (usually a site engineer) to modify individual clock tuning codes, load the clock codes for an entire unit, or verify the clock chain. Typically, the Clock utility must be run after running the su_MAINT_INTERFACE diagnostic.

CB

The Communication Buffer utility is a tool used by engineers on the manufacturing checkout floor and at the customer site. It is a diagnostic tool, and not typically used by operators or other customer staff.

The CB utility enables the operator to enter CB commands, send them to the CB, and view the results, including status and error count information. The operator can enter and edit a set of commands, load them from a file, and save them to a file.

DMAP

The DMAP utility is used to start and stop DMAP software, get status, and load DMAP software onto the DMAP board.

The direct memory access processor (DMAP) provides communications between the SU and the MIs of the ETA10. The DMAP is a single-board computer plugged into the SU server node. It runs under loadable software called controlware.

Master Clear

The Master Clear utility provides six different types of master clear states to the ALSI-20K arrays.

Maintenance arrays and refresh control arrays are affected differently by the various master clear types. The following table shows the six types of master clears that may be sent and the state of the arrays after each master clear completes.

Master Clear Type	Arrays not Cleared	Array State after Master Clear
Power up	None (all cleared)	Idle
Refresh SMI	Maintenance	Frozen
Normal SMI	Refresh Control	Idle
Startup	Maintenance	Frozen
	Refresh Control	
Channel	Not applicable	Not applicable
Loop Channel	Not applicable	Not applicable

Table 9–1. Master clear types and resulting array status.

Status and Control

The Status and Control utility allows the operator to monitor and control the maintenance interface (MI) lines on the CPUs, the CB, and the SM. (Lines specific to B.E.S.T. are controlled and monitored by B.E.S.T. software.)

Each of these devices contains two registers: a maintenance input data register (MIDR) and a maintenance output data register (MODR). There is one MIDR and one MODR for each CPU, each side of shared memory, and each side of the communication buffer.

The utility determines the status of a device by reading and decoding data from its MODR. It sends control signals by encoding and writing data to the MIDR.

Test Mode

The operator uses the test mode utility whenever it is necessary to monitor the execution of a program. For example when debugging a problem that causes a part of the operating system to hang. TMOD can clear the system and restart the test program at regular intervals. Program execution can be monitored on the SU display node. The Test Mode Utility functions are to:

- Run a test program.
- Select the configuration of the system to be tested: one or more CPUs, SMIs, CBIs, or any combination.
- Examine closely, during test execution, the state of a CPU and the contents of important registers.

Pin Map

The Pin Map utility allows you to locate particular pin locations for particular logic arrays or for particular board layouts.

Each array is mounted on a standard board or an adapter board. The pin matrix layout differs between the two board types. For each board type, the pin holes for each array pin matrix are ordered by an ordinal number called the real pin number. In addition, for each particular array, the pin holes are reordered by a second ordinal number called the virtual pin number.

PCSP

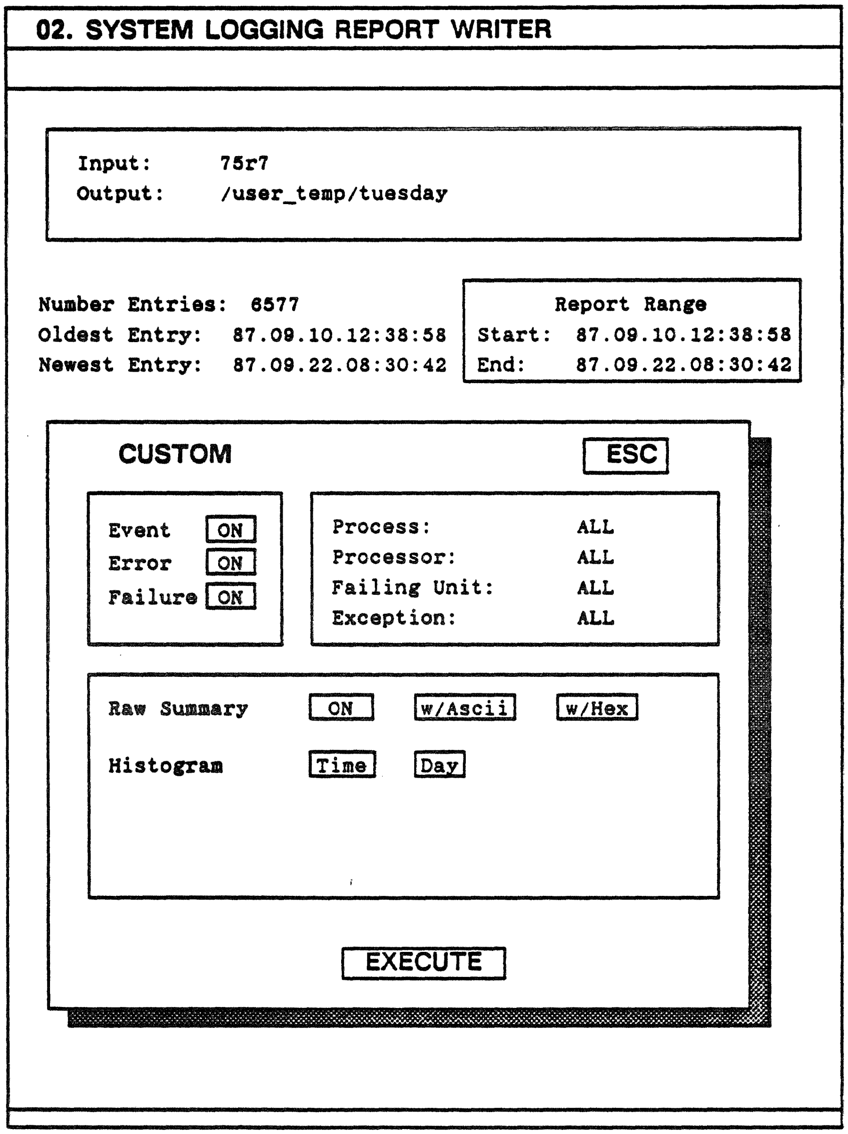
The PCSP (Power Cooling Supervisor Processor) utility allows you to load and start the controlware in the PCSP Single Board Computer, and to load database information for the data acquisition processors (DAPs).

The PCSP single-board computer communicates between the service unit and the main computer DAPs. It runs under a loadable software driver called controlware.

Use the PCSP utility whenever you want to stop or restart the PCSP single-board computer, load the PCSP single-board computer controlware, or load the DAP database. Invoke this utility only while the PCS system is in a stable state (powered up or down, with no point writes or sequences in progress). Only ETA personnel should use this utility since it can change the database and controlware.

System Logging

The System Logging display applications provide two functions. One is to provide SU operators with a window into the System Logging database. The other is to provide a capability for collecting this data for preparing reports and to aid in troubleshooting system problems (figure 9-16).



02. SYSTEM LOGGING REPORT WRITER

```
Input:      75r7
Output:     /user_temp/tuesday

Number Entries:  6577                          Report Range
Oldest Entry:   87.09.10.12:38:58    Start:  87.09.10.12:38:58
Newest Entry:   87.09.22.08:30:42    End:    87.09.22.08:30:42

CUSTOM

Event   ON        Process:           ALL
Error   ON        Processor:         ALL
Failure ON        Failing Unit:      ALL
                  Exception:         ALL

Raw Summary      ON      w/Ascii      w/Hex

Histogram        Time    Day
```

Figure 9-16. Report writer main display.

Display

The Display application provides a window into the System Logging database. This database is a record of all system events collected by SLA through interfaces to other SU applications and the ETA10 operating system features.

Each entry in the database contains the following information:

- Sequence number
- The date and time the event occurred
- The type of error that occurred
- The process and processor that was running when the event occurred
- The failing unit
- The module code

The operator can monitor system events as they occur, or query the database to find specific events based on various parameters.

The database collects entries to a specified level and is then automatically archived.

The type of events automatically recorded in the SLA database are:

- Software and hardware errors
- Environmental events monitored by PCS
- Operational events; operator control commands, system configuration status changes, system invoked recovery actions, and system tuning parameters
- Performance events

In addition to events that are automatically logged, the operator can manually enter events into the database.

Report Writer

The Report Writer application provides a method for generating reports from the System Logging and Analysis (SLA) database based on certain search criteria.

Reports may displayed in a raw summary report format that is similar to the display summary mode or they may be displayed as a histogram, and they may be viewed on screen or printed.

The criteria for the report may include collecting information by event, error, or failure and by a specific process, processor, failing unit, or exception. The operator may also select various date and time selections, such as: since today, since yesterday, since two days ago, all, or a specific range of date and time.

The SU Support Processes

The SU support processes are responsible for handling information within the service unit, providing communications between application programs, logging information and events, and providing the underlying control mechanisms for the various applications available to the SU operator.

As shown in figure 9–17, support processes include servers, drivers, execution modules, diagnostic interfaces and monitors, and libraries.

The most important of these processes are:

- System monitor activities (SMTR)
- System configuration control (SCC)
- System Logging and Analysis (SLA)
- Maintenance interface server/driver

Although many of these support processes (or portions of the support processes) run in other units of the computer system, the service unit is the focal point of control and descriptions of their operation are best described in relation to the operation of the SU.

The service unit's files are in a hierarchical, tree structured system based on the host workstation. Service unit operating system files reside on the server node disks, as do all other SU files; copies of the display files reside on the display node disks.

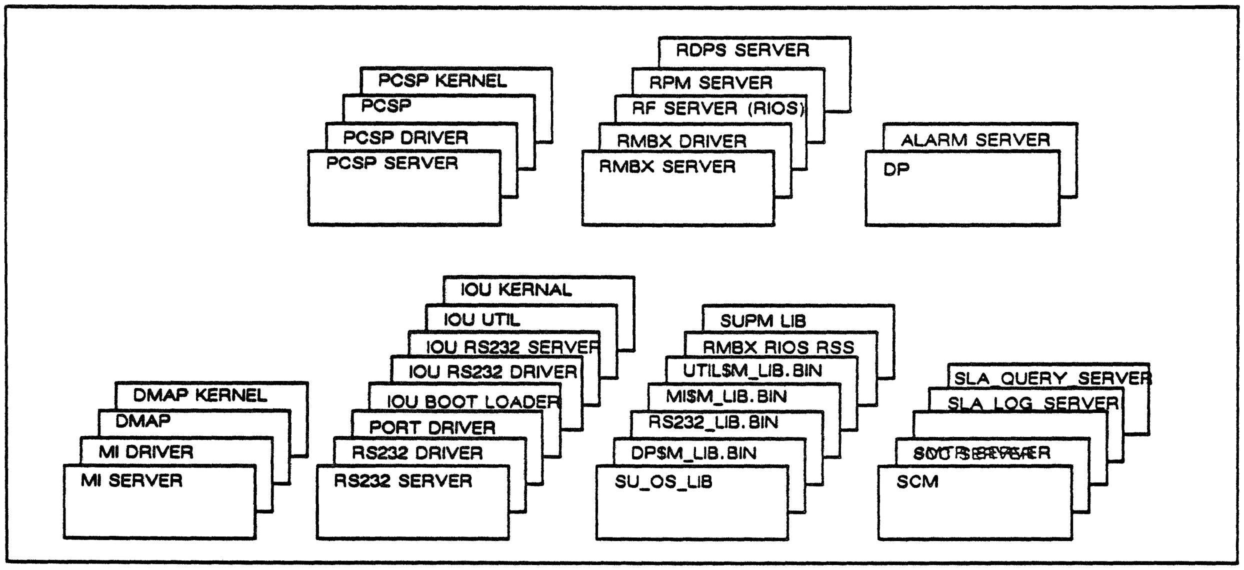


Figure 9–17. SU support processes, libraries, etc.

System Monitor (SMTR)

The system monitor (SMTR) receives fault notification from sources system wide, verifies the errors and forwards the faults to system logging and analysis. (Figure 9-18 shows the interfaces to SMTR). SMTR also provides periodic verification of hardware and software operation.

Parts of system monitor execute on all processing elements of the ETA10. The processes that make up SMTR are SMTR_SU_SERVER and SMTR_SU_PROBE.

System monitor runs in both server nodes of the SU. Error conditions detected elsewhere on the SU are communicated to the system monitor via a host system mailbox.

Faults detected by system monitor include:

- Central processor instruction stack parity error.
- Central processor multiple match error.
- Central processor illegal monitor mode instruction.
- Central processor memory faults.
- Central processor memory double SECDED error.
- Single bit SECDED error.
- Communication buffer or shared memory faults.
- I/O interface communication error.
- Power and cooling supervisor (PCS) faults.
- Service unit node faults.

Error control and error probe are the two main functions of SMTR. As described below, they execute as separate tasks to accomplish their monitoring functions.

Parts of error control execute on all the processing elements of the system including each SU node. Error probe executes concurrently on both SU server nodes and on each SU display node.

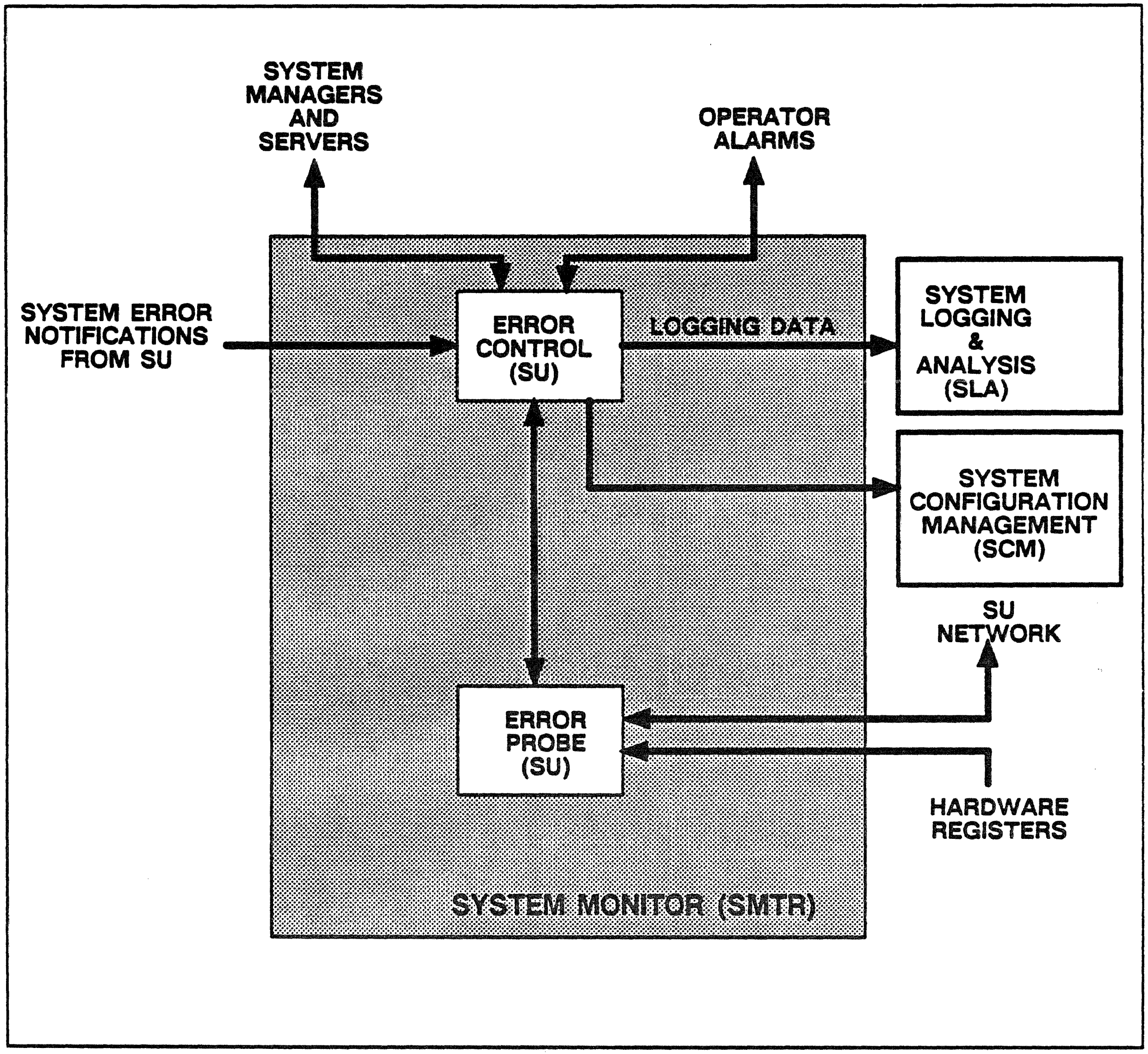

SYSTEM MANAGERS AND SERVERS
OPERATOR ALARMS
SYSTEM ERROR NOTIFICATIONS FROM SU
ERROR CONTROL (SU)
LOGGING DATA
SYSTEM LOGGING & ANALYSIS (SLA)
SYSTEM CONFIGURATION MANAGEMENT (SCM)
SU NETWORK
ERROR PROBE (SU)
HARDWARE REGISTERS
SYSTEM MONITOR (SMTR)

Figure 9–18. System monitor interfaces.

Error Control

Error control of SMTR provides a focal point for reporting system errors. Fault notifications are received from local processes, interprocess communications, and error probe. Configuration change requests are received from the operator. All system errors are validated and passed to System Logging and Analysis. Alarms are generated by SMTR based on events recorded.

Error Probe

Error probe of SMTR is responsible for verifying the operation of all hardware and software by periodically polling for hardware error information and PCS status.

Any abnormal conditions discovered by error probe are reported to error control for logging and recovery action, if possible.

System Configuration Control (SCC)

System configuration control (SCC) is the process responsible for maintaining an accurate record of all software and hardware units of the system configuration. The definition and status of these units, as well as their interrelationships, is contained in the system configuration table (SCT) that is managed solely by SCC.

Figure 9-19 illustrates how SCC interfaces with other features.

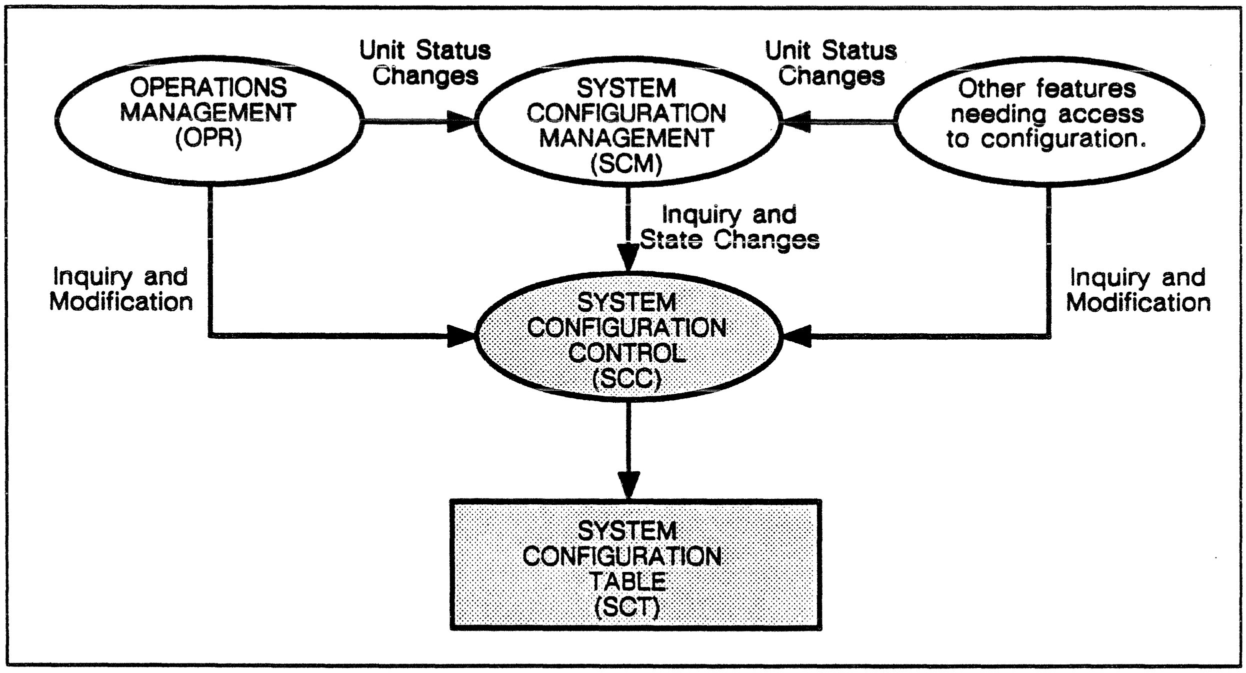

Unit Status Changes
Unit Status Changes
OPERATIONS MANAGEMENT (OPR)
SYSTEM CONFIGURATION MANAGEMENT (SCM)
Other features needing access to configuration.
Inquiry and State Changes
Inquiry and Modification
Inquiry and Modification
SYSTEM CONFIGURATION CONTROL (SCC)
SYSTEM CONFIGURATION TABLE (SCT)

Figure 9-23. System configuration control interfaces with other. features.

In a partitioned system, some hardware components, for example, CPUs and IOUs, must belong to a distinct partition. Other hardware components, for example, shared memory, may be shared among multiple partitions.

To support multiple partitions, multiple SCT files and multiple display nodes (in addition to the SU display nodes) are used. The SU has an SCT file that reflects the full ETA10 System hardware configuration. Each partition has a dedicated display node that has an SCT file on it that reflects the configuration of that particular partition. Figure 9-20 illustrates the SCT file setup to support partitioned systems.

The SCT files for the dedicated display nodes for each partition are set up with components owned by the operating system, which allows the operating system to run in a straightforward manner. The SU SCT file is set up with the components owned by diagnostics, which allows diagnostics and other hardware maintenance functions to run in a straightforward manner.

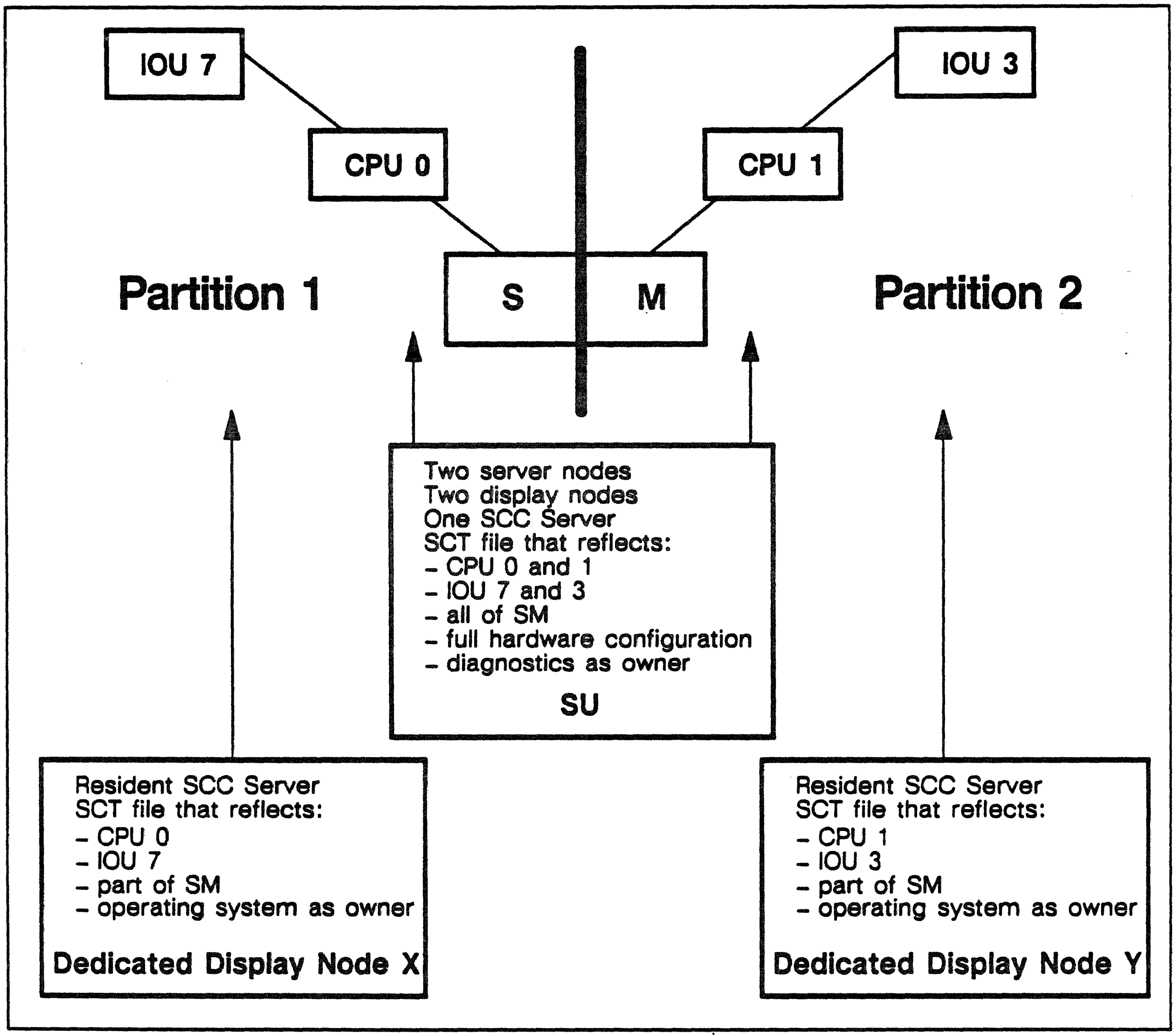


Figure 9-20. System configuration tables on a partitioned system.

System Configuration Table

A system configuration table contains a wide variety of configuration information including the following:

- Hardware unit definitions, interrelationships, status, and attributes
- Software server status and communication path identification (for example, mailboxes)
- Session categories
- Logical disk configuration and allocation information
- Network configuration parameters and statistics
- Site modifiable system parameters (defaults, limits, thresholds, tuning, and so on)
- System clock maintenance parameters

Each configurable unit (hardware and software) is identified by a unique name in the SCT along with the status, usage, and interrelationships of the configurable unit.

Changes in status are a result of either operator action or software installation, recovery, or resource management actions. They are coordinated by the system configuration management feature with all relevant operating system software components, and the resulting status changes are recorded in the SCT.

For a given software configurable unit, multiple copies (of different versions) may exist. In the SCT each version is identified by a version name. A default version is identified for use when a version name is not specified by the user. System configuration control provides for changing the default version for each configurable unit.

System configuration control provides for adding or removing different versions or backup file copies of a software unit, and the location and identification of these software configurable unit copies is maintained in the SCT.

Since a software configurable unit may not reside within a single file, a single SCT entry may not be sufficient to describe it. For that reason, software units are combined to form aggregate software units called *software families*. A software family is simply a grouping of software units, each of which may be of a different version, and is a software configurable unit in its own right.

Logical Disk Device Allocation

For the ETA10 file system, SCC provides the interlock for logical disk device allocation. SCC maintains allocation information associated with logical disk devices and locks logical devices while space is being allocated or deallocated.

SCC also defines and manages device classes. These device classes are defined by the site (except those predefined by SCC) and are associated with groups of logical disk devices.

Configurable Unit Classifications

Hardware elements have three classifications for maintenance purposes:

- Field replaceable unit (FRU) - hardware elements that can be quickly and easily removed and replaced.
- Field degradable unit (FDU) - the part of the system that must be removed from the active resource list during maintenance to protect the system from further interruptions due to a known failed FRU.
- Field serviceable unit (FSU) - the part of the system that the user cannot access during a maintenance action on a failed FRU.

The hardware concepts of FSU, FDU, and FRU do not relate directly to software configurable units. However, a software FRU can be considered as any collection of one or more software features' object modules (typically associated with corrective code).

- For CPU software, an FRU can be a domain or a set of one or more object modules, depending on the type of software. These FRUs are packaged into a process image, an object file, or a library to form an FSU.
- For IOU software, FRUs consist of the IOU supervisor components and software for each type of IOP. An IOU FSU is a collection of one or more IOP FRUs.
- For SU software, an FRU is a bound process or application library and an FSU is a collection of the same.

A software FSU can then be viewed as a collection of FRUs that define a product or a portion of a product (typically associated with a new system release). A software FDU can be considered equivalent to a software FSU.

Configurable units are, in general, the same as field degradable units, and these are the system elements that must be controlled and coordinated among the processing elements of the machine. Software features that provide service functions via a system defined mailbox are treated as hardware configurable units (for example, they are monitored, their status is controlled, and (in some cases) they are moved from one processor to another). This includes IOP device drivers, SU server/drivers and CPU-based system servers.

System Logging and Analysis (SLA)

System Logging and Analysis is the process responsible for logging and analyzing ETA10 System events that are significant from the standpoint of system reliability, availability, and serviceability (RAS). SLA records and analyzes events automatically and manually.

Automatically recorded events are those detected by various features of the operating system; they are logged without operator intervention. The classes of automatic events that are captured and analyzed by SLA are as follows:

- Error events: All hardware and software errors that are detected.
- Environmental events: RAS-related environmental conditions monitored by the power and cooling supervisor.
- Operational events: Control commands issued by the operator, status changes to system configuration, system invoked recovery actions, and system tuning adjustments.
- Performance events: RAS-related performance data such as an observed degradation of throughput or utilization. (Not benchmark events.)

Manual events are the hardware and software maintenance events logged by SU operator activity.

Each event that is recorded is identified by: the date and time of the event's occurrence, the feature that reported the event, the suspected failing unit, and data that is specific to the event (for example, configurable unit, temperature, humidity, addresses, registers, and so on).

Figure 9-21 illustrates the relationships between the parts of SLA.

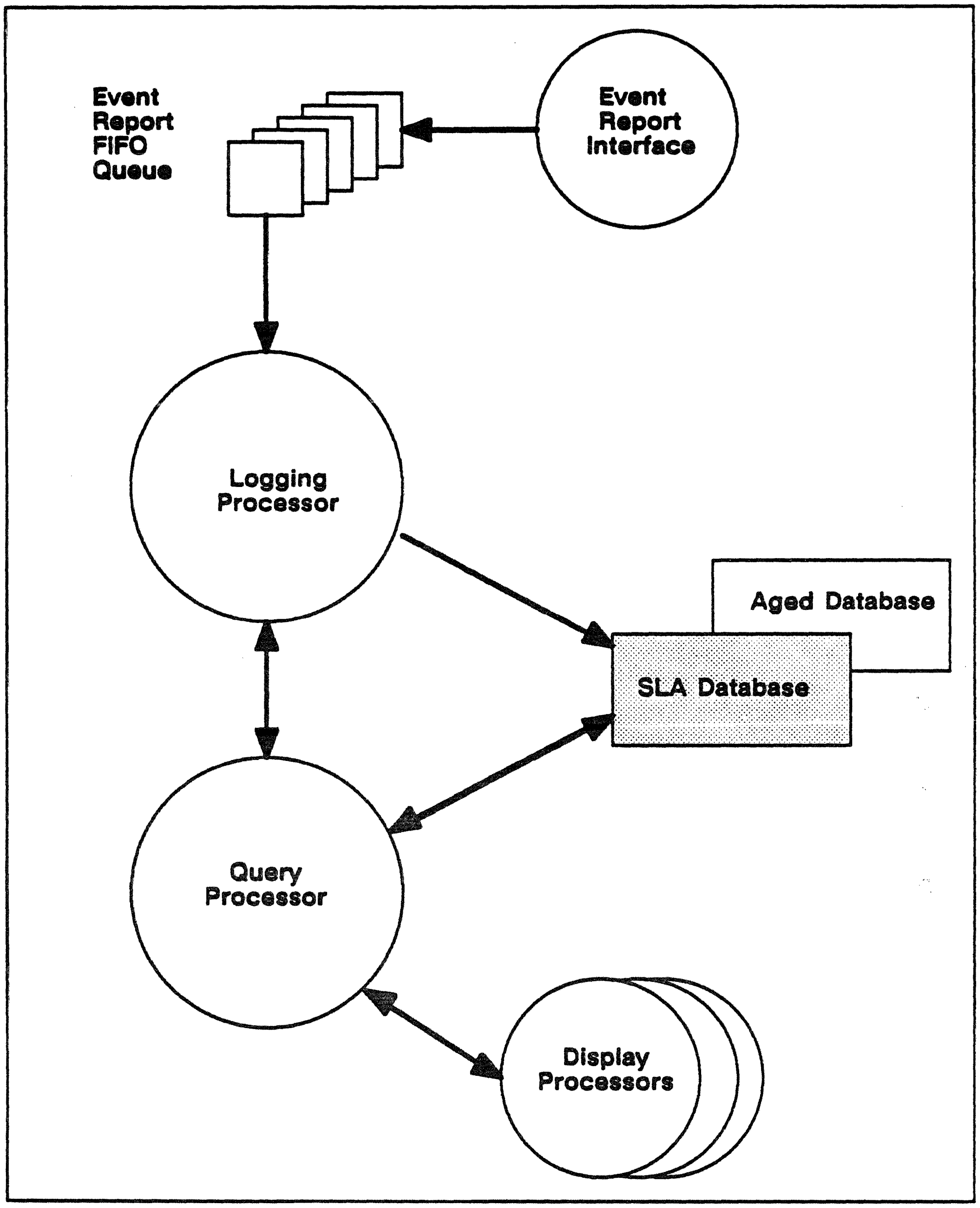


Figure 9–21. System Logging and Analysis relationships.

The process for logging events into the SLA database is as follows:

The event report interface is used by all SU-resident operating system features to report events from other features to SLA. The events are posted to the event report FIFO queue and are processed sequentially by the logging processor.

The logging processor enters event reports into the SLA database, and may also relay event reports to the query processor (if it is active). The logging processor limits the size of the database by aging the entire database when the configured record number limit is reached. This aging of the database and starting a new database file is coordinated with the query processor based on current query activity. Aging does not occur while a display processor is querying the database.

The query processor is charged with processing queries from the display processors and returning results to them. Each query from a display processor results in an event report being returned. This may be done automatically, in which case the events are reported to the display as they occur. Or the operator may call up the SLA display and scroll through the database. The query processor enables the display processors to view archived database files as well as the current database file.

Maintenance Interface Server/Driver

The maintenance interface (MI) server/driver provides SU application programs and diagnostic programs with a common method of communicating with the MIs, and provides the following operations:

- Read and write central processor memory, shared memory, and communication buffer data
- Read and write the maintenance registers (MIDR, MODR)
- Read and write the B.E.S.T. chains and function registers
- Read and write the clock chains and function registers
- Perform communication buffer operations
- Read status
- Perform low-level diagnostic operations

The interface to the server/driver consists of an open routine, a transfer routine, and a close routine. All calls are accompanied by a request and an acknowledge header block data structure that contains all of the specific information necessary to carry out the requested operation.

Two types of data transfer mechanisms are provided by the server/driver to the calling program. The calling program can send or receive data directly or it can request that the data be transferred to an SU file.

If the calling program is generating data or acting directly on the data, then it supplies a buffer. Data transfer occurs between the program's buffer and the interface hardware via the server/driver.

If the program supplies an SU file name, the data transfer occurs between the SU file and the interface hardware via the server/driver. For this type of transfer, the SU file must already exist, and the calling program is responsible for insuring that the server/driver process has the proper access permission to the file. This mechanism works only for SU files resident on the service unit and only non-record-structured files may be used.

Two types of communication paths to the server/driver are available to accommodate differing sets of requirements. They are an application path and an interrupt path. The path is selected by the calling program by calling the corresponding open routine.

The *application path* is used by most programs. It provides all of the operations necessary to carry out normal data transfer requests. Requests on this path are serviced in a round-robin fashion among the available connections.

The *interrupt path* provides a means of observing and controlling the server/driver. Normal operations are not provided by this path; it is used to enable/disable communication.

CHAPTER 10

Chapter 10

Power and Cooling Systems

Chapter Contents

List of Figures

Figures (continued)

Chapter 10

Power and Cooling Systems

In This Chapter . . .

This chapter describes the power and cooling systems of the ETA10 and the power and cooling supervisor (PCS) that is responsible for monitoring and controlling those systems.

The power distribution system supplies AC and DC power to all units of the ETA10 and to portions of the cooling system. Power for peripheral devices such as disk drives is not provided by the ETA10 power system. Refer to the vendor documentation provided with these devices for power information.

The ETA10 has two cooling systems. A conventional refrigeration unit provides cooling for circuit boards, power supplies, and other components of the SCMI cabinet as well as the power supplies and CP memory boards in the mainframe cabinets. A cryogenic system provides cooling for the central processors boards.

The cryogenic system's main function is to provide the extremely low temperature needed for the extremely fast processor speed. Cryogenic cooling of circuitry significantly increases the speed (and hence the performance) of the CPUs. Removing heat from the CPU boards is a secondary benefit of the cryogenic system. The central processor boards can be cooled adequately with conventional cooling technology.

In this chapter you will find information on:

- The Cryogenic System
- The Refrigeration System
- The Power Distribution System
- The Power and Cooling Supervisor (PCS)

The Cryogenic System

The cryogenic system illustrated in figure 10–1 circulates and regenerates the liquid nitrogen (LN_2) used to cool the central processor boards. Mainframe configurations with one or two CPU cabinets use one cryogenic system. Mainframe configurations with three or four CPU cabinets use two cryogenic systems.

Cryogenic cooling serves primarily to increase the processor speed of the system; it provides cooling to the CPU boards as a secondary benefit. The liquid nitrogen (LN_2)used in the ETA10 system is stable, non–toxic, commercially available and maintains the CP boards at a temperature of 77 degrees Kelvin (K) -- equivalent to -196 degrees Celsius.

Cryogenic System Overview

The cryogenic system is a closed–loop system. Commercially available liquid nitrogen is introduced into the system at installation time and the liquid is circulated through vacuum–jacketed (V–J) lines (double-wall, vacuum-insulated pipe sections) that carry the LN_2 to the units of the cryogenic system.

The system is composed of four major units:

- Storage and distribution skid, where a supply of liquid nitrogen is maintained for system use.
- Valve box, a device used to regulate and control the flow of liquid nitrogen from the storage and distribution skid to the cryostats and from the cryostats back to the storage and distribution skid.
- Cryostats, the vessels that contain the liquid nitrogen and hold the CPU boards.
- Cryogenerator, a device that removes heat from the gaseous nitrogen that returns from the cryostats and regenerates it into liquid nitrogen for reuse in the system.

The cryogenic system is manually controlled from the main control panel and the associated silicon controlled rectifier (SCR) control panel.

Meters on the main control panel display the cryostat pressure and temperature, the LN_2 return line temperature, the LN_2 flow rate, the pressure in the storage and distribution skid, the heater wattage (input), the centrifugal pump speed, and the amount of LN_2 in the system.

The SCRs in the control panel regulate the heaters in the storage and distribution skid that help control the pressure in the system. The panel also includes the AC input and power distribution circuits for the cryogenic system.

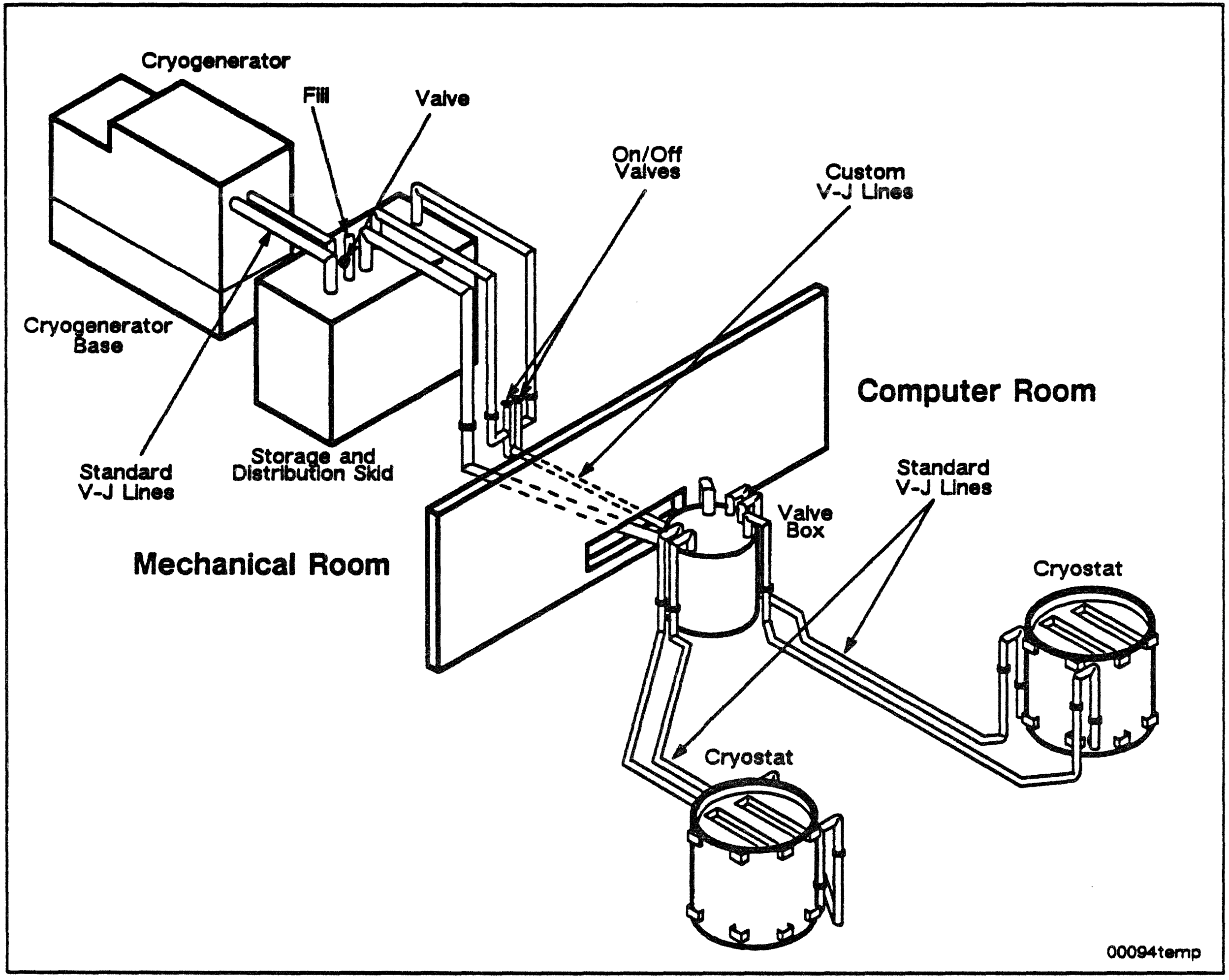


Fig. 10-1. Diagram of the cryogenic system.

Storage and Distribution Skid

The storage and distribution skid holds approximately 100 gallons of liquid nitrogen (LN_2) and is the focus for distribution and control of the cryogenic system. As shown in figure 10–2, the storage and distribution skid contains heaters that help control system pressure, pumps for distributing the LN_2 throughout the system, and feed and return lines from other system components.

During normal system operation, LN_2 is fed from the storage and distribution skid to the valve box by one of two pumps immersed in the tank. Either pump may be turned on at the control panel, however the on/off valve (see figure 10–1) must be manually adjusted for the appropriate pump.

The pump develops pressure above that of the storage and distribution skid, and this pressure is used in the valve box to balance the load between the two cryostats.

A two–phase liquid/gaseous nitrogen mixture is returned from the valve box to the storage and distribution skid after the CPU boards in the cryostats have been cooled. The liquid portion of the mixture returns to the pool of liquid nitrogen for recirculation in the system, the gaseous portion of the mixture remains on top of the pool and is pulled into the cryogenerator, where it is condensed into liquid nitrogen and returned to the storage and distribution skid.

Operating pressure for the storage and distribution skid is maintained slightly above atmospheric pressure by the heaters in the storage and distribution skid and the constant addition of LN2 from the cryogenerator.

Control of the cryogenic system is handled through the control panel. A control loop is set up based on maintaining the operating pressure in the storage and distribution skid at a predetermined set point. If the pressure falls below the set point, the heaters come on, heating the liquid nitrogen and creating gaseous nitrogen, which brings the pressure in the storage and distribution skid back up to the set point.

If the pressure rises above the set point, the current to the heaters is reduced. The cryogenerator load increases and it produces more liquid nitrogen, cooling down the temperature in the storage and distribution skid and reducing the pressure.

In addition to providing a ready supply of LN_2 for the system, the large volume of liquid nitrogen in the storage and distribution skid provides reserve operation in the event of power failure or during maintenance on the cryogenerator. Fittings at the top of the unit accept commercially available LN_2 if the reserve supply is depleted.

If the cryogenerator is out of service for any reason, an escape valve allows excess gaseous nitrogen to vent from the storage and distribution skid.

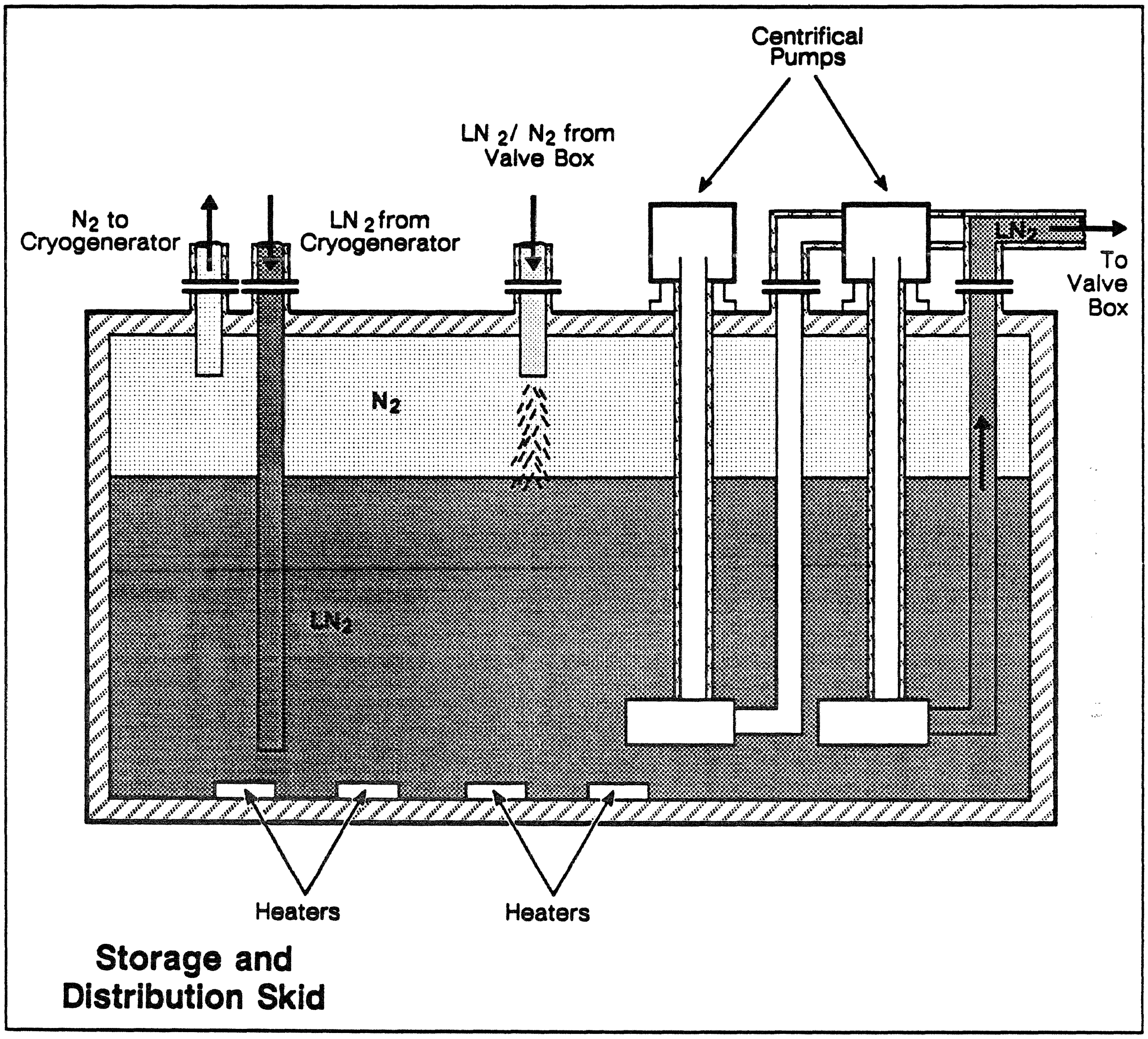

Centrifical Pumps
LN $_2$/ N$_2$ from Valve Box
N$_2$ to Cryogenerator
LN $_2$ from Cryogenerator
LN$_2$
To Valve Box
N$_2$
LN$_2$
Heaters
Heaters
Storage and Distribution Skid

Fig. 10–2. Functional diagram of the storage and distribution skid.

Valve Box

The valve box contains the on/off valves that control the flow of liquid nitrogen to the cryostats and from the cryostats back to the storage and distribution skid.

As illustrated in figure 10–3, the pump in the storage and distribution skid delivers LN_2 to the valve box through a vacuum–jacketed line. Inside the valve box, the LN_2 is channeled through two inlet valves, one for each cryostat. These valves are adjustable and are used to balance the flow to the cryostats so that both of them are supplied with an adequate flow of LN_2.

Two return valves in the valve box (these are not control valves, they are fully open or fully closed only) direct the flow of the two–phase liquid/gaseous nitrogen returning from the cryostats back to the storage and distribution skid.

A bypass valve in the valve box allows LN2 to be looped back from the valve box directly to the storage and distribution skid. This valve is used during system installation to bring the cryogenic system on line for testing before the rest of the computer system is installed.

Each valve box can support two cryostats.

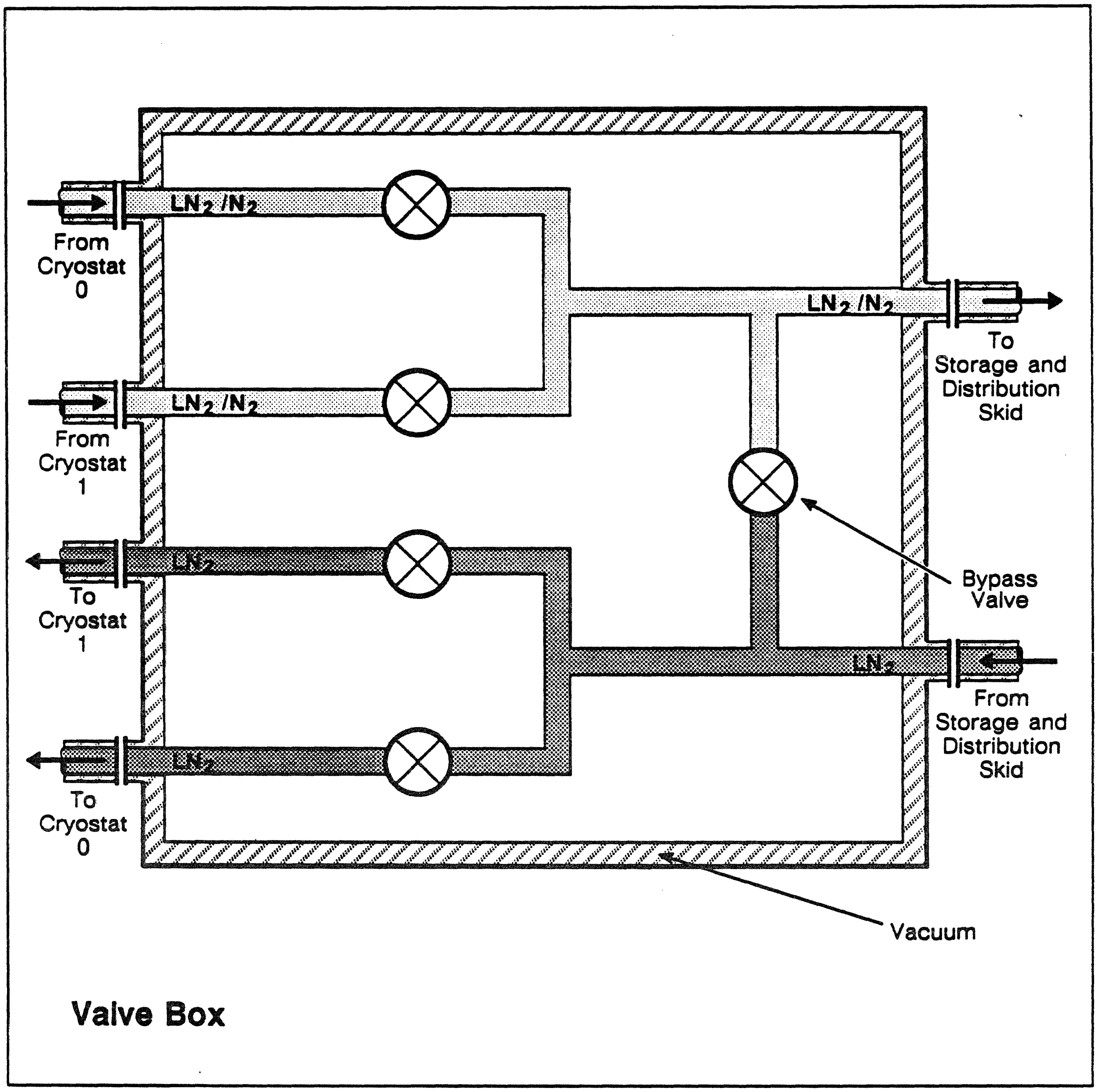


Fig. 10-3. Functional diagram of the valve box operation.

Cryostat

The cryostat is a stainless steel, vacuum-insulated tank that holds the central processor boards of the ETA10 System. During system operation the cryostat is filled with liquid nitrogen which cools the central processor boards and makes possible the very high system clock rate.

As shown in figure 10–4, LN_2 flows from the valve box into the bottom of the cryostat containing the CPU boards and flows up and over them, spilling over the sides of the inner container into a second (enclosing) container. The CPU boards are suspended in the liquid nitrogen from the top of the cryostat. As the liquid nitrogen flows over the cpu boards, it absorbs heat generated by the arrays on the boards and some of the nitrogen is boiled off, becoming gaseous nitrogen.

This liquid nitrogen/gaseous nitrogen mixture is forced out of the cryostat by the pressure of the incoming LN_2 and returns to the valve box where it is then channeled to the storage and distribution skid. A vapor space of several inches sits above the level of the liquid nitrogen that bathes the CPU boards.

Each cryostat can contain up to two central processor boards.

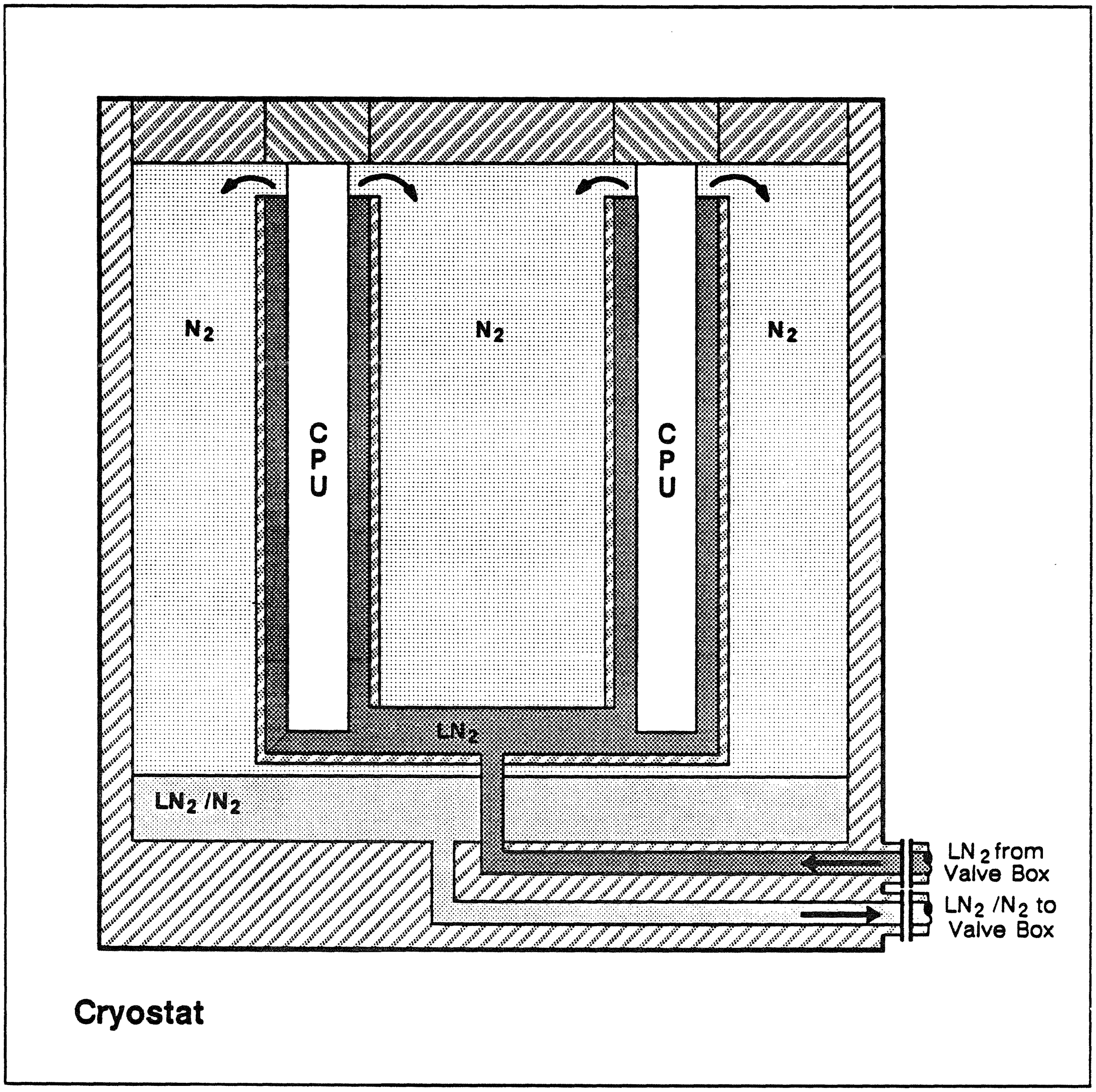


Fig. 10–4. Functional diagram of the cryostat operation.

Cryogenerator

The cryogenerator is a refrigerating machine designed to produce the extremely low temperatures necessary to liquify nitrogen.

As shown in figure 10-5, gaseous nitrogen (N_2) is pulled into the cryogenerator from the top of the storage and distribution skid by natural suction caused by the temperature differential inside the cryogenerator.

The cryogenerator cooling process takes place in a closed system using a technique of compressing and expanding helium (the working gas) within cylinders that act as condensing heads to precipitate the N_2 into LN_2. The precipitated LN_2 runs off the head and is collected into a common channel that returns the liquid nitrogen to the storage and distribution skid for recirculation. Helium is used as the working gas to produce the extreme cold needed to condense the nitrogen from a gaseous to a liquid state.

Heat generated during the operation of the cryogenerator is removed by a cooling water system.

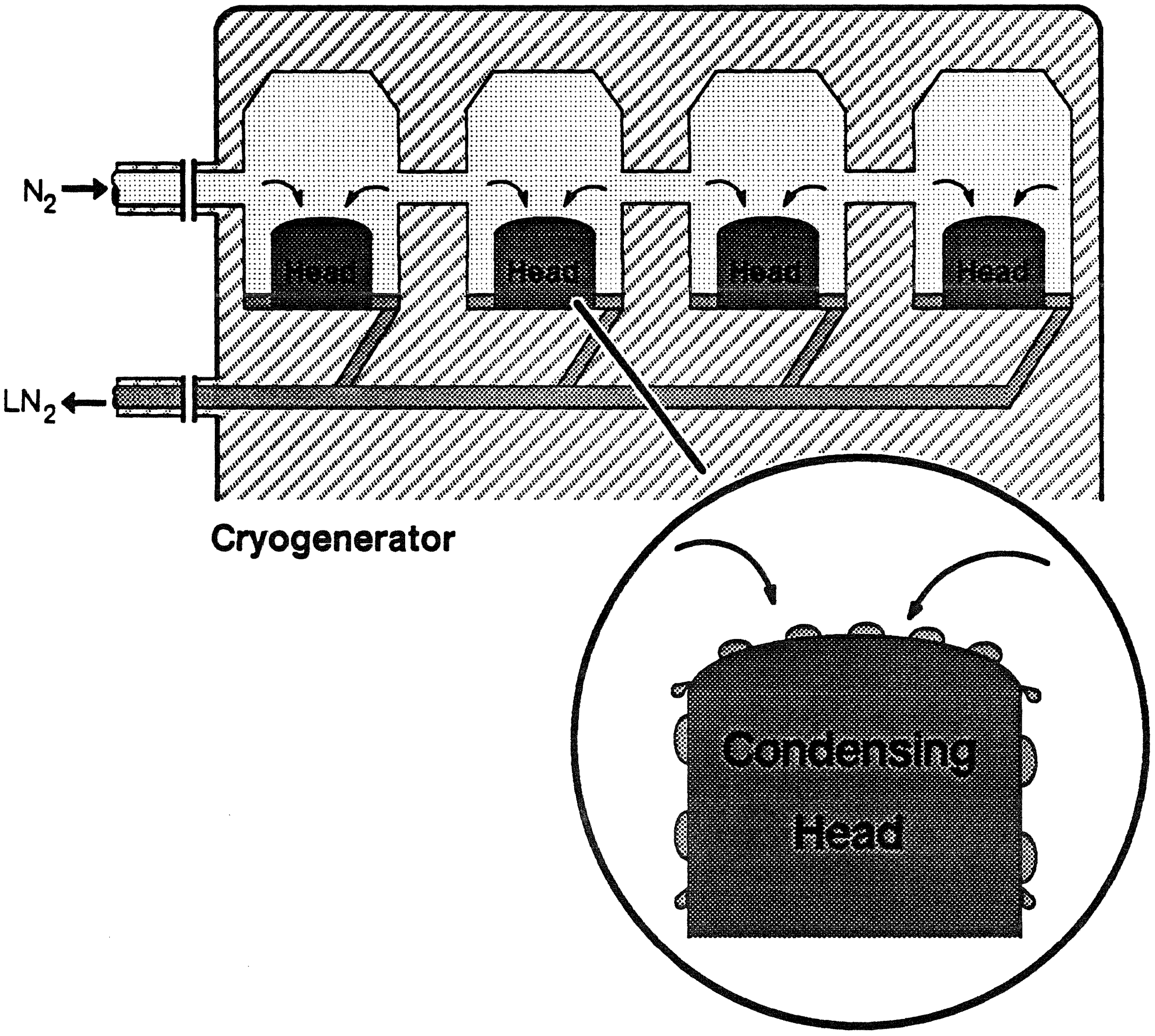


Fig. 10-5. Functional diagram of cryogenerator operation.

Refrigeration System

The refrigeration system (figure 10-6) consists of a condensing unit and the distribution lines and manifolds necessary to route refrigerant to the appropriate system components.

The condensing unit circulates refrigerant to the manifolds located beneath the computer room floor. There, liquid/suction lines connect to cold plates and cooling coils in the system components which do the actual cooling. Cold plates provide cooling for each cryostat power supply, and for the SCMI, SM interface, and SM power frames. CP memory is cooled by a blower that circulates air over a cooling coil, and a single, large cooling coil services all boards in the SCMI.

Heat is transferred from the devices and delivered back to the condensing unit where cold water removes the heat.

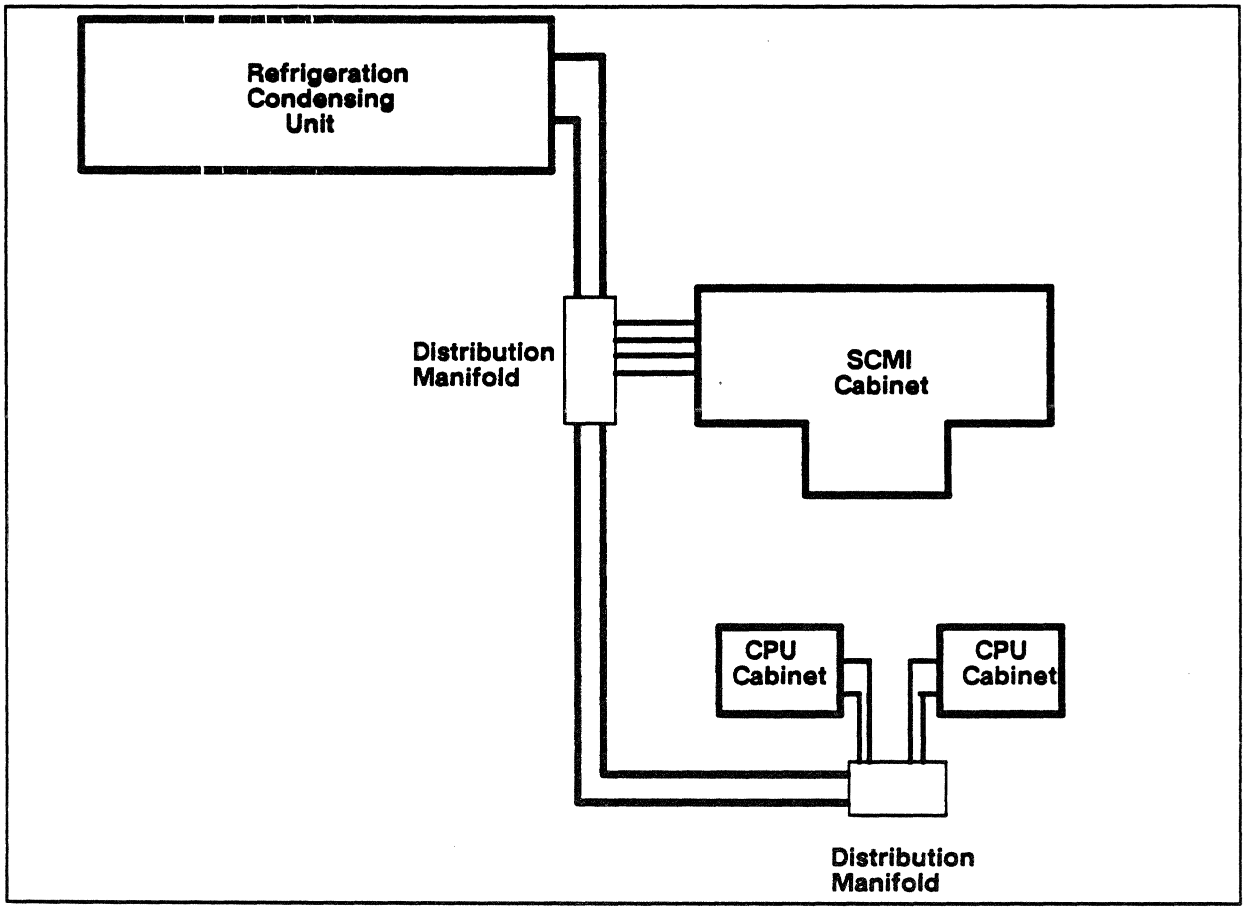


Fig. 10-6. Diagram of the refrigeration System.

Power Distribution System

The power distribution system for the ETA10 provides electrical power to the logic boards and auxiliary equipment (blowers, PCS boards, and so on) in the CPU cabinets, the SCMI cabinet, and the SU network, figure 10–7.

The system takes 208V AC (415 and 380V AC may also be used), 3–phase power, rectifies and boosts it to 320V DC power and then converts the high voltage DC current to the low voltages required by the logic boards. Redundancy for the power system is provided at several levels.

The major components of the power distribution system are the:

- Power distribution unit
- Energy storage circuit
- Boost converters
- Power supplies and control modules

All system components are grounded for safety (through earth grounding) and to provide a path for radio frequency interference (using a reference–plane grid).

System power requirements are:

Input voltage: 208V AC, 3–phase, 50/60 Hz.

Mainframe voltages: CPU: 5V DC
SMI: 4V DC and 5V DC
CPM: 5V DC
SM: 5V DC

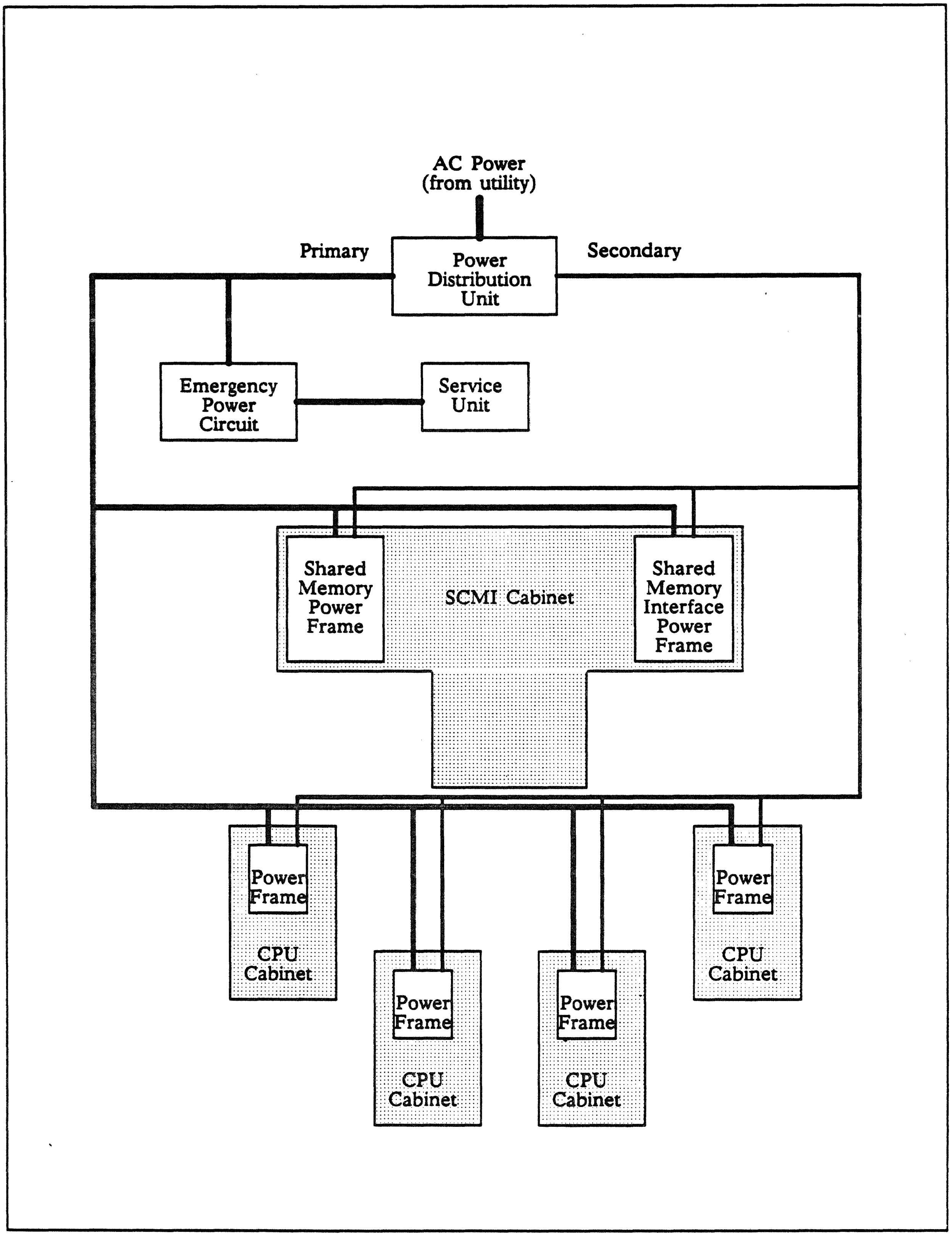


Fig. 10-7. Diagram of power distribution in the ETA10.

Power Distribution Unit

The power distribution unit (PDU) monitors incoming power (voltage, current, and other parameters) and distributes it to other components of the power system. It provides a tie-in for various alarms, limits inrush current, and isolates the system from transients on the power lines. The PDU's monitor panel displays readouts of power parameters including: phase readouts, output current, line-to-line output voltage, and so on.

The primary side of the PDU provides power to the energy storage circuit and auxiliary power to the power frames. The secondary side of the PDU provides main power to the power frames (figure 10-7) for the boost converters.

Boost Converters

The boost-converters in the power frames convert the AC current distributed by the PDU to filtered and regulated 320V DC power. This power is then distributed to the DC-to-DC power supplies where it is converted to the required DC voltages for the logic boards.

This 320V DC bus provides isolation from line voltage variations. Ride-through protection for the logic circuits is provided by capacitor storage, which can hold up power for one to two seconds at maximum load.

Power Supplies and Control Modules

Power supplies convert the DC voltage from the boost converters to the appropriate voltages required by the logic modules. All power supplies are wired so that if a power supply fails, other power supplies can carry its load. The power supplies are factory adjusted; margin and incremental stepping is allowed through the power and cooling supervisor (PCS) displays.

Redundant control modules in each power frame manage the power supplies. These units have internal redundancy built into each module.

Energy Storage Circuit

The power distribution system includes an uninterrupted power supply (UPS) that provides up to 10 minutes of emergency power to the system clock, PCS components, and the SU network.

The Power and Cooling Supervisor

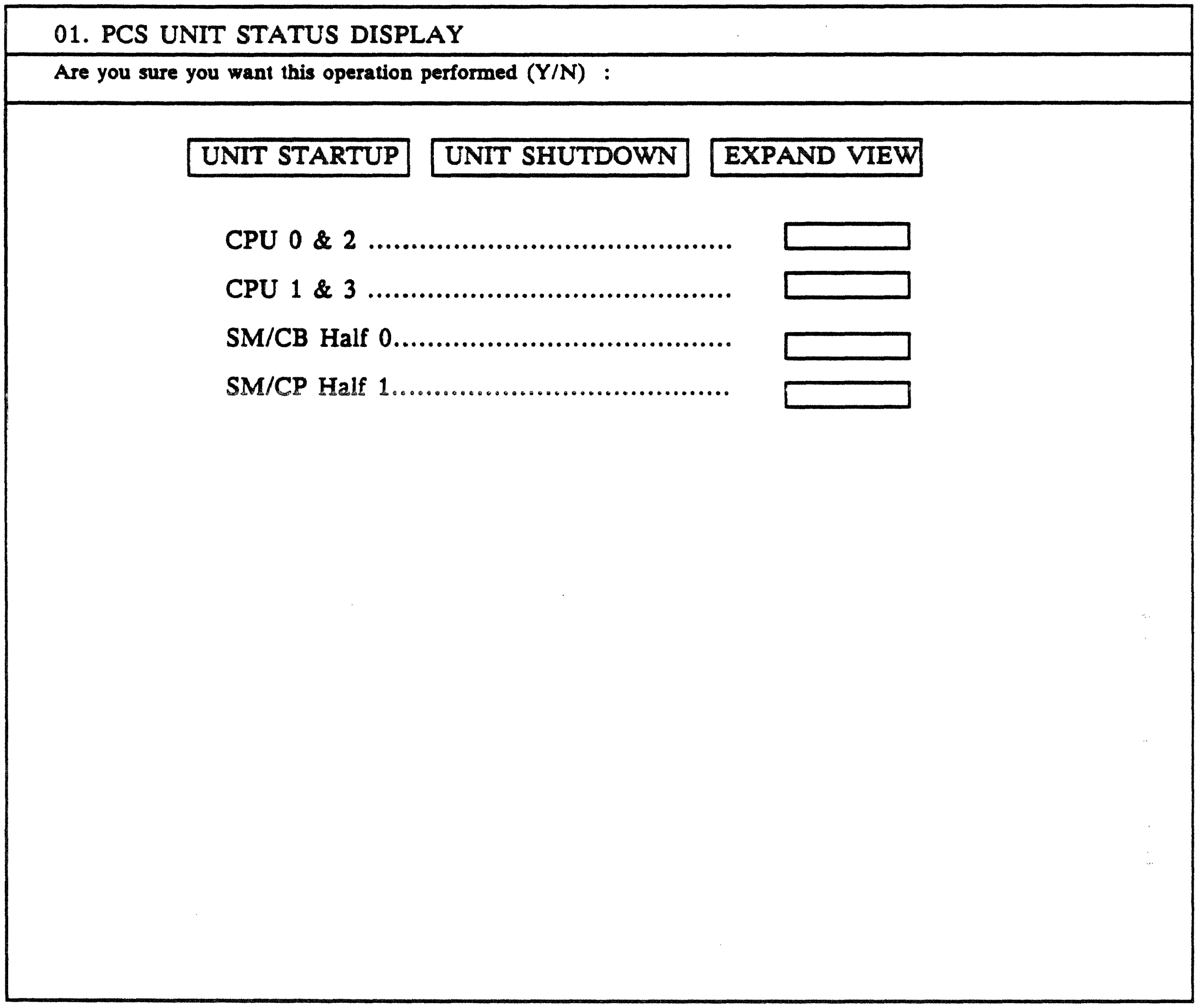
The power and cooling supervisor (PCS) provides service unit operators with the capability to monitor and control the system's power and cooling units.

PCS relies on a system-wide set of elements to ensure the safe and proper operation of the power, cooling, and maintenance interface systems in the ETA10. PCS collects, translates, and analyzes data using data acquisition processors (DAPs), data acquisition boards, and digital and analog interface boards. A large network of system sensors, more than 250 continuously collect and transfer data into the power and cooling supervisor system. Control points that are controlled through PCS provide control and adjustment of all power supplies and cooling functions.

As a system safeguard, DAPs and the power supplies needed to drive the sensors and actuators are powered by the uninterruptible power supply which provides a system ride-through when AC power is lost.

The PCS sensor network includes a number of redundant sensors; failed sensors or actuators are detected and reported.

Service unit operators interface with the PCS system primarily through the PCS System Status and Control displays. Figure 10-8 shows a typical main display for PCS. The actual display depends on the ETA10 system configurations.



01. PCS UNIT STATUS DISPLAY

Are you sure you want this operation performed (Y/N) :

CPU 0 & 2 .... □

CPU 1 & 3 .... □

SM/CB Half 0....................................... □

SM/CP Half 1....................................... □

Figure 10–8. PCS system status and control display from a service unit display node.

Power and Cooling Supervisor Components

Figure 10-9 shows the basic components of the power and cooling supervisor. The power and cooling processors are housed in the service unit's server nodes, and the rest of the components (data acquisition processors, interface boards, sensors and actuators) are located in components of the mainframe and refrigeration systems.

Power and Cooling Supervisor Processor (PCSP)

The PCS processors (PCSP) are microprocessors that communicate over serial channels (RS 232) with the digital and analog data acquisition processors (DAPs).

The PCSPs are the interface between the SU display node operator and the DAPs. They display points distributed throughout the ETA10.

As shown in figure 10-9, the data acquisition processors are installed in the SM power frame, and in the main control panel of the refrigeration system. The Interface cages are installed in the SM power frame, in each central processor power frame, and in the main control panel of the refrigeration system. The sensors and control points are distributed throughout the system and are not shown.

Data Acquisition Processors

The data acquisition processors (DAPs) in the cryogenic system and SCMI cabinet continuously monitor points associated with them and update the PCS database. The DAPs also actuate the control points under their supervision and perform analog to digital conversions. The DAPs interface to the control and monitor points through interface cards in the interface cages (IF Cages). Checkpoints are used by the DAPs to keep the environmental conditions within specified ranges. These ranges are not controlled from the service unit but are programmed into the DAPs. The DAPs maintain the safety of the system regardless of the operating condition of server nodes on the SU network.

Interface Cards

Interface cards provide the direct hardware link with the sensors and actuators that make up the PCS. They provide no processing of signals, but they do perform signal conversions.

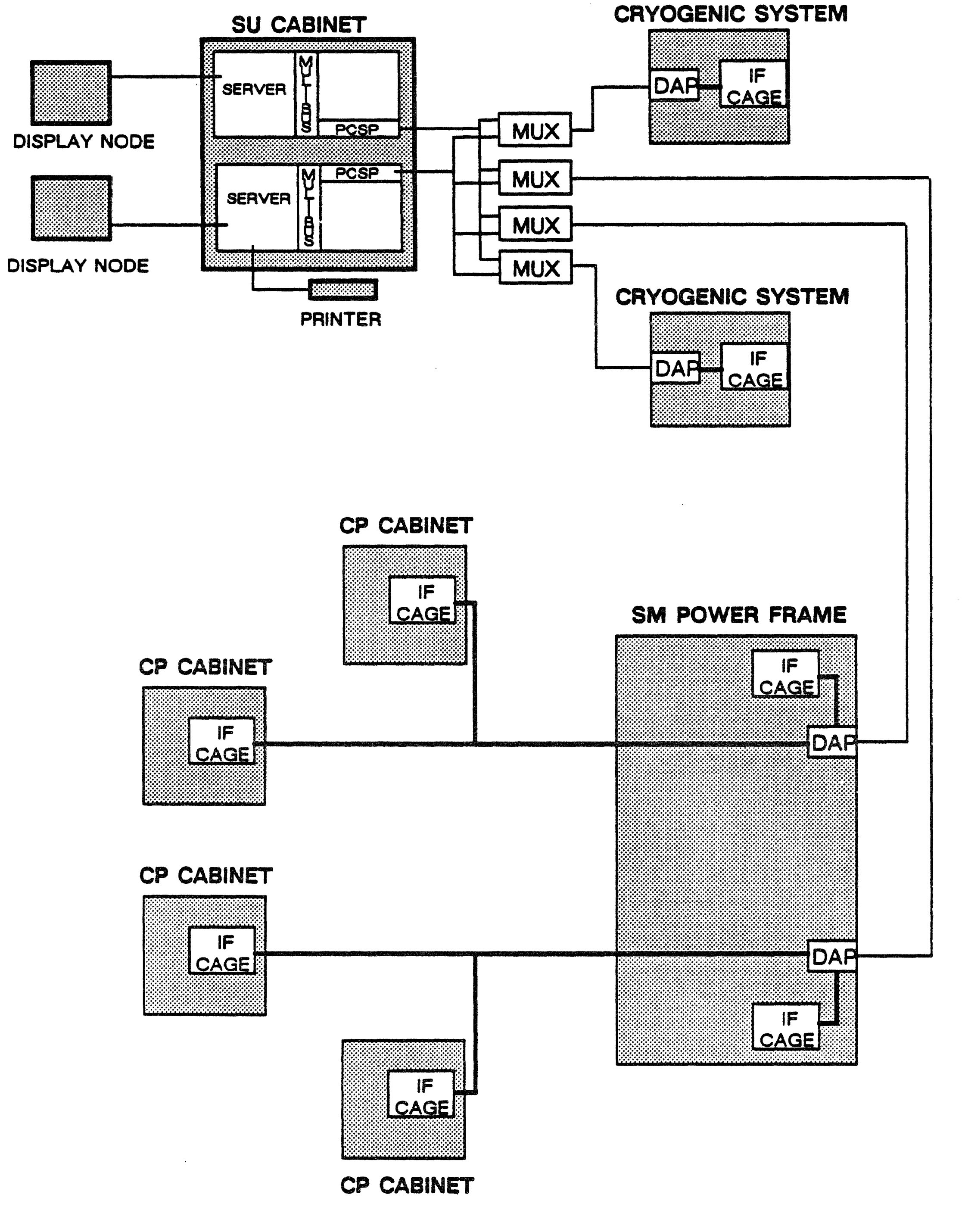


Figure 10-9. Component diagram of the power and cooling supervisor system.

Sensors

Throughout the computer system, analog and digital sensors report status to PCS. Analog sensors monitor temperature, pressure, power and current levels. Digital sensors monitor on/off states.

When a sensor detects a condition outside prescribed warning limits, the data acquisition processors takes corrective action in response to the error, and an error message is placed in the SLA file.

There is some redundancy of the sensors, but failing sensors are usually detected by conflicting reports from other sensors in the group. For example, physical inconsistencies in reports from temperature and pressure sensors are reliable indicators of a sensor malfunction.

Control Points

Control points are digital actuators that turn blowers off and on, open or close valves and so on. Power supply trimming is also accomplished using digital control points.

GLOSSARY

REFERENCED DOCUMENTS

INDEX

Acronym Guide and Glossary

AIP	Application Interface Processor
AVC	Access Violation Code
BEST	Built-in Evaluation and Self-Test
BLAP	Base/Limit Access Pair
BOF	Beginning-of-File
BOG	Beginning-of-Group
BOP	Beginning-of-Partition
BOR	Beginning-of-Record
CB	Communication Buffer
CBM	Communication Buffer Manager
CCS	Communication Control Subsystem
CDC	Control Data Corporation
CLP	Common Login Processor
CPMM	Central Processor Memory Manager
CPU	Central Processing Unit
CSIP	Cold Start System Initialization Process
CSMA/CD	Carrier Sensing Multiple Access with Collision Detection
DAP	Data Acquisition Processor
DIP	Dual In–line Package
DMA	Direct Memory Access
DMAP	Direct Memory Access Processor
DPC	Data Pipe Controller
DPFS	Disk Physical-File System
EOF	End-of-File
EOG	End-of-Group
EOI	End-of-Information
EOP	End-of-Partition
EOR	End-of-Record
FDCM	File Directory/Catalog Manager
FDU	Field Degradable Unit
FIPS	Federal Information Processing Standard
FRU	Field Replaceable Unit
FSM	File Support Module
FSU	Field Serviceable Unit
GFID	Global File ID

GFUAT	Global File Usage Attributes Table
GS	Global Scheduler
IBM	International Business Machines Corporation
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
IOC	I/O Connection
IOI	I/O Interface
IOP	I/O Processor
IOU	I/O Unit
IPC	Interprocess Communication
IQM	Input Queue Manager
ISI	Intelligent Standard Interface
LAN	Local Area Network
LCN	Loosely Coupled Network (CDC)
LFID	Logical-File ID
LFS	Logical-File System
LFUAT	Local File Usage Attributes Table
MCB	Message Control Block
MHN	Multi-Host Network
MI	Maintenance Interface
MIDR	Maintenance Input Data Register
MODR	Maintenance Output Data Register
NAD	Network Access Device
NOS	Network Operating System (CDC)
OIN	Open Interconnection Network
PCP	Power and Cooling Processor
PCS	Power and Cooling System
PCSP	Power and Cooling Supervisor Processor
PFID	Physical File ID
PID	Process ID
PM	Process Management
QMS	Queue Management Subsystem
RETA	Remote ETA10 login facility
RHF	Remote Host Facility
RM	Resource Management
RMS	Rotating Mass Storage

RPC	Remote Procedure Call
RSN	Request Serial Number
RTC	Real Time Clock
SBC	Single Board Computer
SCC	System Configuration Control
SCM	System Configuration Management
SCMI	Shared memory, Communication buffer Memory, and Interfaces
SCT	System Configuration Table
SECDED	Single Error Detection Double Error Correction
SLA	System Logging and Analysis
SM	Shared Memory
SMI	Shared Memory Interface
SMM	Shared Memory Manager
SMTR	System Monitor
SS	Session Scheduler
SU	Service Unit
SUDB	Service Unit Debugger
TCP	Transmission Control Protocol
TCU	Trunk Control Unit
TM	Transmission Manager
TMOD	Test Mode Utility
TRB	Transfer Request Block
TTL	Transistor-Transistor Logic
UM	User Management
VSOS	Virtual Storage Operating System (CDC)
WSA	Working Storage Area
WSL	Working Storage Length

AC Distribution Cabinet

The cabinet that houses equipment to regulate incoming, main service, and AC current, and distributes it to the mainframe cabinets and portions of the cryogenic system.

Access Permissions

Permissions used by the logical file system to control user access to files.

Access Protocol

1. The *carrier sensing multiple access with collision detection* IEEE 802.3 algorithm used by the ETA10 to access the Open Interconnection Network.
2. The trunk control time slot number algorithm used by the ETA10 to access the Loosely Coupled Network.

Access Violation Code

Access rights associated with domain keys. The three kinds of access rights are *read*, *read/write*, and *read/execute*. A key can be associated with any one of the rights. js

Account and Project Registry

The account and project registry contains relationships between users, projects, and accounts.

Accounting File

A file of session-level resource usage maintained by the accounting software.

Addressing

Bits, bytes, half words, words, and pages are always addressed from left to right; the first bit of the addressed entity is the left-most bit of that entity. Addresses are 48-bit quantities containing enough information to reference a specific bit.

Alias Node

An indirect identifier of either a directory or file node.

ALSI-20K Board

A 44-layer circuit board (also called "high-tech board") populated with CMOS based ALSI-20K gate arrays. Each board is a mainframe component and is one of the following:

Central Processor board
Shared Memory Interface board
Communication Buffer Interface—Input/Output Interface board

Apollo Domain Ring Network

The network that interconnects Apollo workstations. It connects to the ETA10 through an OIN gateway.

Apollo Gateway

An Apollo Computer Company Domain ring node equipped with an Ethernet TCP/IP board that connects Apollo workstations and other remote user entry and display stations to the ETA10. Also the host for remote ETA10 login RETA software.

Apollo Workstation

A user entry and display station that communicates with the ETA10 over the Domain ring network and the Open Interconnection Network.

Application

User-accessible software designed to perform a certain procedure or solve a particular type of problem when given appropriate commands and data by the user.

Application Interface Processor

The RHF access method component that receives Loosely Coupled Network requests from a service application.

Application-to-Application Protocol

The sequence of network blocks transferred within RHF access method frames and message frames required to transfer files over the Loosely Coupled Network.

Array

An integrated circuit that contains a matrix of simple logic gates interconnected to form unique, customized patterns.

Associative Registers

The set of 16 registers in the associative unit in which the space table is rippled through and read until a match for the requested virtual address is made. These registers perform the virtual-to-real translation of page addresses.

Associative Word

Contains the virtual and physical address of each page in central processor memory; these words are read by the associative registers to do virtual-to-real address translation.

Attached File

A VSOS environment permanent file made available to a user when he/she issues an ATTACH command.

Attenuator

The device that connects a NAD to the Loosely Coupled Network trunk.

Availability

The probability that at any given time the subject system, equipment, function, or process is able to perform its specified operation under stated conditions.

Base/Limit Access Pair (BLAP)

Two memory words that reside in the domain package and denote the lowest communication buffer address the domain can access (base), the highest address the domain can access (limit), and the access rights the domain has to those addresses (access).

B.E.S.T.

Built-in Evaluation and Self-Test software for the service unit used to detect problems in the ETA10 ALSI-20K gate array boards.

B.E.S.T. Array Register

The portion of array logic reserved for B.E.S.T. It consists primarily of three registers (control, input, and output) acting as a single-bit, serial-shift register.

B.E.S.T. Chain

The serial data path created by connecting the B.E.S.T. array register test data output (TDO) pin of one array to the test data input (TDI) pin of the next array, and so on, until all arrays on a unit are connected.

Binary Software Unit

A software unit in the system configuration table identifying the binary data necessary to construct an executing process.

Block

Memory blocks are sized areas used to allocate and manage shared memory. Shared memory blocks in initial releases of the operating system software are 2K words (2048 bytes).

Blocked State

The state of a process temporarily ineligible for execution. A process can be unblocked by another process, by process management, or by the monitor, at which time it is moved to a queue.

Boost Converter

A system component that converts AC current distributed by the power distribution unit to filtered and regulated DC power for distribution and use by the DC-to-DC power supplies.

Boundary Error

Describes a request that references two units of shared memory or communication buffer, or a request to an invalid area of either memory.

Bounds Error

Describes an illegal memory request to an area in communication buffer memory that is outside the area specified by base and limit addresses. (base/limit access pair)

Bridge

A device interconnecting two local area networks that conform to similar communication protocols.

Byte

The smallest unit of data managed by the system at the file level. rs
A byte is an 8-bit quantity; the address of the left-most bit is a multiple of 8_{10}. The lowest three bits of a byte address are set to zero.

Central Processing Unit (CPU)

The central processor, central processor memory, and associated cooling systems and power supplies.

Central Processor (CP)

The scalar processor, vector processor, register file, processor status registers, and interfaces to other system components. The ETA10 central processor is a single 44-layer printed circuit board populated with 240 CMOS ALSI chips that operates at room and super-cooled temperatures.

Central Processor Memory (CPM)

A memory of four million virtually-addressed words dedicated to its associated processor. Addressed as 32-bit half words even though memory addresses in instructions specify a bit address.

Central Processor Memory Interface

Provides a 512-bit bandwidth access for the scalar and vector processors as well as for shared memory ports; has three read and two write ports.

Central Processor Memory Manager

A memory manager software feature that allocates and deallocates central processor memory for CPU processes. An independent central processor memory manager resides on each CPU.

Central Processor-Shared Memory Port

The connection between the central processor and the shared memory interface.

Chaining

Allows instructions to be linked together to produce results more efficiently; chaining requires that intermediate results be shortstopped.

Channel Processors

Components in an I/O unit that control the connection between a particular network or peripheral device channel and a data pipe to shared memory. There are three types of channel processors: FIPS, VME, and disk.

Cluster

A collection of processes. The relationship between processes in a cluster is arbitrary, but it is specified by and known to the system.

Cold Start Initialization Process (CSIP)

When all CPUs of the ETA10 are to be initialized, a cold start sequence is used. In this case, the first CPU is cold started, and the remaining CPUs are warm started.

Command

A system or user-defined string associated with the initiation of a particular process or request to the operating system. Consists of a reserved word or filename specifying the task to be done, and optional parameters.

Command Language

The rules for specifying commands and controlling the order of their interpretation and/or execution.

Command Procedure

A list of commands which can be executed by invoking the name of the list.

Command Shell

The system program which interprets the command language and causes the appropriate processes to execute.

Common Login Processor

The operating system utility responsible for accepting and validating user login information.

Communication Buffer (CB)

A unique memory structure that enables communications between all system processors and enables central processors to share data during multiprocessing operations. Communication buffer is one-half million words that can be expanded to one million.

Communication Buffer Interface (CBI)

Provides access ports for central processors, I/O processors, and the service unit processors to the communication buffer.

Communication Buffer Manager

Memory manager software that allocates and coordinates use of the communication buffer.

Communication Buffer Port

Connects the communication buffer to the scalar processor's register file.

Communication Control Subsystem (CCS)

Central processor software that controls the ETA10 interface to the Open Interconnection Network.

Configurable Unit

Configurable units are hardware or software entities that can be added to or deleted from the ETA10 system configuration. They are the basic units managed by the system configuration control feature.

Connection

A network link that supports sequenced and verified message transfers between two host devices.

Connection Path

One route between remote Loosely Coupled Network hosts.

Connection Request Block

Thirty bytes of connection request information sent by the RHF access method on behalf of an application within an RHF access method frame and within a message frame over the Loosely Coupled Network.

Controllee

An executable file generated by the VSOS LOAD command. A process spawned by another.

Controller
A process which spawns another, or controllee, process.

Controlware
A processor program for a particular processing unit integral to a product that provides the product with a set of functional operating characteristics.

CP Memory Object
A block of memory allocated by the CP memory manager.

Cryogenerator
The refrigeration system that condenses nitrogen gas and furnishes liquid nitrogen to the system.

Cryogenic System
Consists of the refrigeration unit, liquid nitrogen distribution system, storage tank, cryostat, and controls.

Cryostat
A liquid nitrogen container that consists of the vacuum shell, containment vessel, sealing element, and insulation. ETA10 central processor boards are housed in cryostats.

Current File Position
The byte address of the first byte of the current partition plus either the current partition offset (if the file is structured), or the byte address of the current byte (if the file is unstructured).

Current Partition Address
The file address of the beginning of the current file partition.

Current Partition Offset
The number of bytes read from or written to the current file partition. The current partition offset is also the point at which a partial partition read or write operation is to begin.

CYBER 205
A supercomputer manufactured by Control Data Corporation. VSOS is the operating system used on the CYBER 205.

CYBIL
The CYBER implementation language, used for developing ETA Systems system software.

Data Acquisition Processor (DAP)
Part of the power and cooling system that collects information at various locations throughout the system and actuates control points. The DAPs are responsible for performing digital-to-analog conversions and updating the power and cooling system database.

Data Flag Branch Register

A 64-bit register containing data that enables programs to branch to special routines when certain conditions or results occur.

Data Pipe (DP)

A fiber optic cable that provides the physical connection between an I/O unit and the I/O interface to shared memory and communication buffer. A data pipe is shared by all I/O processors and I/O channels in an I/O unit.

Data Pipe Buffer

A 128 Kbyte memory used by a single board computer for temporary storage during data transfers between a data pipe and an I/O channel.

Data Pipe Controller

Manages the transfer of data between the mainframe I/O interface and I/O channel processors via a data pipe; major sub-components are the data pipe interface and the data pipe bus interface.

Data Set

The NAD component that performs bit–serial encoding and decoding of all message frames transferred over a given Loosely Coupled Network trunk.

Demand Buffering

The allocation of CP memory buffers by the logical-file system only as required to satisfy user requests for logical file data.

Device Class

Logical devices are assigned to device classes based on the performance attributes and allocation permissions or descriptors that belong to each class.

Direct Access

The logical file access mode in which data can be written to file positions other than end-of-file.

Direct Memory Access Processor

The single board computer in the service unit that provides communications to the ETA10 maintenance interfaces.

Directory Node

A logical-file system object used to group other directory and logical file nodes.

Directory Set

The complete set of file nodes and directories in the logical-file system.

Disk Channel

The physical control and data path between a set of disk devices and a disk channel processor. Up to eight disk devices can be daisy-chained to one disk channel. Controlled by the disk channel controller software.

Disk Channel Controller

Software that interfaces between disk physical-file system software and the disk channel interface.

Disk Channel Interface

The disk channel processor board that enables connection to the disks.

Disk Channel Processor

An I/O processor (single board computer and data pipe buffer), and a disk channel interface board.

Disk Control Module

The logic and control circuitry that communicates with the disk channel interface and directs the disk unit.

Disk Label

The data stored on a disk that declares the logical device and the device classes supported.

Disk Physical-File Map

The table stored on disk that specifies all the segments of all the physical files on the disk.

Disk Physical-File System

Disk channel processor software that enables the logical-file system to operate on physical files.

Display Node

A service unit node with a keyboard and display monitor.

Domain

A CPU hardware feature used to define the CP memory access keys, the CB base/limit access pairs, and domain change information for controlling a process.

Domain Distributor/Collector

A function that defines the true entry and exit points of a domain. The domain distributor/collector provides a single point of entry to (distributor) and a single point of exit from (collector) the domain.

Domain Management

A series of software feature that support protected routines and data structures from unauthorized access and modification.

Emergency Power Cabinet

Houses the equipment to provide the service unit, SCMI cabinet, and silicon controlled rectifier control panel with approximately five minutes of operation if main service power is lost.

End-of-Information

The point within a disk file that marks the end of all data and record manager control delimiters.

Energy Storage Circuit

Also known as the uninterrupted power supply (UPS). This component provides up to 10 minutes of emergency power to the system clock, the power and cooling system components, and the service unit network.

Ethernet Cable

The common physical medium for connecting Open Interconnection Network equipment.

Ethernet Protocol

The definition of the 18-byte Ethernet frame used to send TCP/IP packets over a local area network.

Exchange

A central processor switch between monitor mode and job mode; exchanges are cause by a hardware interrupt or by an EXIT FORCE instruction.

Exchange to Job Mode

Puts the central processor into job mode to start a new process or to resume execution of an interrupted process.

Exchange to Monitor Mode

Puts the central processor into monitor mode to do cleanup required by a completing process or to respond to an interrupt received by the processor.

Explicit I/O

Logical file read and write operations that occur through the I/O buffer and are specifically requested by a process or user.

F Type Record

The record in a logical file that permits a single record type of fixed format.

Failure

An unexpected or unintended condition in either hardware or software that results in a fault.

Fault

An observation made during normal system operation that either hardware or software is not performing as intended.

Field Degradable Unit (FDU)

The support classification for a part of the system that must be removed from the active resource list to protect the system from further interruptions due to a known failed field replaceable unit.

Field Replaceable Unit (FRU)

The support classification for a part of the system that can be easily and quickly removed and replaced.

Field Serviceable Unit (FSU)

The support classification for a part of the system that the user cannot access during a maintenance action on a failed field replaceable unit.

File

A data structure accessible by name to users and the system. A collection of records.

File Activation

The process by which the file system establishes a connection between the pathname, the global file identifier, the attributes of a file, and the file itself so that operations may be performed on that file. The inverse of this operation is known as file deactivation.

File Directory/Catalog Manager

That portion of the logical-file system that provides for the controlled creation of logical files and directories.

File Information Table

A VSOS environment table used to describe the characteristics of a file.

File Object

A region defined in shared memory that permits the caller to store and retrieve data from a device.

File Pathname

An ordered list of directory names, down to the name of the file.

File Support Module

A logical-file system software feature that opens and closes files. The file support module also manages the tables that maintain the in-use conditions of files for tasks.

FIPS Channel

Provides the connection between the FIPS 60-2 channel interface and the Multi-Host Network control devices; it is controlled by the FIPS channel processor.

FIPS Channel Interface

The physical connection between a FIPS channel processor and a NAD. Data flow thereon conforms to FIPS channel protocol.

FIPS Channel Processor

An I/O processor (single board computer and data pipe buffer) and a FIPS channel interface board.

FIPS Channel Protocol
The protocol for communication between a FIPS channel processor and a network access device (Multi-Host Network).

Fixed Length Record
See *F Type Record.*

Free-Space Map
The bit map stored on a disk that designates which portions of the disk are available for allocation to physical files.

Full Word
A 64-bit quantity in which the address of the left-most bit is a multiple of 64_{10}. The lowest six bits of a full-word address are set to zero.

Gateway
A device that interconnects two local area networks with different communication protocols.

Global File ID
The global identifier of a file or a directory that uniquely identifies the object without regard to its pathname or locality of action.

Global Memory
A 9 Mbyte memory accessible to all single board computers in an I/O unit.

Global Scheduler
A process manager software feature that is responsible for scheduling sessions and processes within the multiple CPUs of the computer system.

Group
A collection of records in the logical file structure.

Half Exchange
Occurs after the central processor is master cleared, puts the processor into execution in monitor mode.

Half Word
A 32-bit quantity in which the address of the left-most bit is a multiple of 32_{10}. The lowest five bits of a half-word address are set to zero.

High-tech
A term applied to the 44-layer boards manufactured by ETA Systems that are populated with ALSI 20K chips.

Host-to-Host Protocol
The Control Data Corporation communication protocol that defines the RHF access method frame and supports the loosely coupled network connection service used by applications to transfer network blocks.

I/O Processor

A sub-component of a channel processor that contains a single board computer (MC68020) and a data pipe buffer.

I/O Unit

A multi-processor computer unit connecting peripheral devices and networks to the ETA10.

Implicit I/O

Logical file read and write operations initiated by page faulting.

In-Place I/O

Logical file read and write operations conducted directly from the user's buffer without intermediate central processor memory buffering.

Inheritance Information

Information that is communicated (through process management) from a parent process to its descendants.

Initiating State

The first state of a process not yet eligible for execution.

Initiator

The network host and application software that initiates a connection to a remote host server.

Input/Output Interface (IOI)

The major unit that provides all I/O units and the service unit access to the shared memory interface and the communication buffer interface.

Input/Output Processor (IOP)

A processor that controls the transfer of information between shared memory and a peripheral device or network channel. An I/O processor contains a single board computer (68020) and a data pipe buffer.

Input/Output Unit (IOU)

A multi-processor computer that connects a data pipe to peripheral devices and networks. An I/O unit contains a set of independent, multi-functional channels and data pipe interfaces, and directly accesses both shared memory and communication buffer.

Input Queue Set

A queue object used by the global scheduler to structure session input and execution.

Instruction Segment

A portion of an instruction; the issue unit divides instructions into portions that can be processed in one minor clock cycle. (Scalar instructions are usually one segment in length.)

Instruction Set

The ETA10 set has 256 function codes, 40 of which are unused; it is model independent and vector-oriented. The set is compatible with Control Data Corporation CYBER 205 instructions.

Instrumentation Counters

A set of 48-bit counters that track events occurring within a central processor.

Intelligent Standard Interface

A Control Data Corporation communication protocol that directs disk devices.

Interface Cards

Part of the power and cooling system that provides the direct hardware link with the sensors and actuators.

Internet Protocol

A Department of Defense communication protocol used to route communications between hosts on local area networks.

Interrupt Network

Connects the 48-bit interrupt registers in each central processor, I/O unit, and service unit in order to handle interrupts passed between processors.

Invisible Package

An element of the process package (a locked-down process object) containing information required for process execution and restart. A process's invisible package is loaded into the CPU's *processor status registers* (see glossary entry) when a process becomes active.

IOU Cabinet

A cabinet that houses one I/O unit and connects to the SCMI cabinet via a data pipe. It contains the channel processors, peripheral channels, and data pipe interfaces that enable peripheral device and network transfers.

IOU Supervisor

The lowest level software that runs on I/O processors and supports operating system features.

Job

A collection of commands that is scheduled, executed, and thought of as a unit. May execute through batch or interactive sessions. Also defined as a session; the basic unit of work on the ETA10.

Job Interval Timer

A 32-bit timer used by application programs to time execution intervals.

Job Mode

The period in central processor operations during which the processor fetches, executes, and returns results from instructions contained in user programs. Same as user mode.

Kernel

The layer of the operating system distributed across the ETA10 system that governs system configuration, global scheduling, process management, memory management, and file management.

LANCOPY

The ETA10 user command and software utility that transfers files between the ETA10 and Apollo gateways over the Open Interconnection Network.

Local Area Network

A workstation-based network that can connect a large number of interactive users. Network operations use the TCP/IP communication protocol. ETA Systems' local area network is called the Open Interconnection Network.

Local File

A file which is immediately available for use by a VSOS environment user. Local files can be temporary or permanent.

Local File ID

The identifier for an active file that may be used in place of the pathname or global file ID to reference the file.

Locked-Down Process Objects

A set of memory objects created and locked down in CP physical memory by process management at the time a process is initialized. These objects include all the virtual page, general register, domain, and other restart information needed for the kernel and hardware to set a process running. The locked-down process objects are: process register table, process page table, process package (made up of invisible package, domain packages, and domain stack), process address maps, and process descriptor block.
js

Logical Device

A grouping of a set of physical disk devices that are treated as one addressable device. Portions of one disk may be assigned to different logical devices. In initial releases of the operating system software, each disk unit is a separate logical device.

Logical File

An object by which the ETA10 system and its users store, transmit, and manipulate data; often referred to simply as a file.

Logical File Node

A logical-file system object used to represent a logical file.

Logical File System

The portion of the operating system responsible for managing files, directories, and file I/O. Logical-file system components include the file directory/catalog manager, the file support module, and the record manager.

Logical State
A state associated with a hardware configurable unit that defines how the unit is currently being used by the operating or maintenance software.

Look-Ahead Buffering
The allocation of CP memory buffers by the logical-file system in advance of application request for file data.

Loosely Coupled Network
A system of interconnected processing entities, which may include computers and storage devices, operating in conformance to Control Data Corporation Loosely Coupled Network communication protocol.

Mailbox
The remote procedure call mechanism for passing messages between all system processors.

Maintenance Interface (MI)
The combination of hardware and software that provides all connectivity, buffering, and latching between the system components and the service units for monitoring and maintenance functions.

Maintenance Owner
A member of the set of designated owners of configurable units that specifically excludes the operating system.

Maintenance State
A state during which diagnostic activities or maintenance actions are performed on a configurable unit. The unit is either powered-off or has no system software loaded, and is isolated from other system elements.

Mapin
The central processor memory manager function that associates a logical file with virtual central processor memory addresses.

Maxiboard
The board upon which the shared memory stacks are installed; each shared memory unit has eight maxiboards.

Message Frame
The frame used by a trunk control unit to transfer maintenance commands, RHF access method frames, and message responses.

Minimum Allocation Unit
The operating system parameter that specifies the smallest number of sectors that can be allocated to a physical file on a disk.

Monitor Interval Timer
A 32-bit timer set during an exchange to job mode that sends an interrupt during a hang or endless loop (or similar condition) that allows monitor to regain control of the processor.

Monitor Mode

The period in central processor operations during which monitor software and other system processes perform system work; user processes do not execute in monitor mode.

Multiple Match

An error caused by duplicate address entries in the associative registers; the process referencing a duplicate entry aborts.

NAD (Network Access Device)

The device that is cabled to an I/O unit that provides the physical and electrical interface to up to four Loosely Coupled Network trunks.

NAD Driver

FIPS channel processor software that manages connections to remote RHF hosts over the Loosely Coupled Network.

Network Block

A network message, command, or back block is transmitted by an application within an RHF access method frame and message frame over the Loosely Coupled Network. Network back blocks are returned to confirm receipt of network message and command blocks.

Nodename

The 1- to 31-character name associated with each file and directory node entry in the directory set.

Non-Server Process Image

A process image which has no executable program linked to it. A non-server process image must have a program linked to it before it can be executed as a process.

NOS

(Network Operating System) A Control Data Corporation operating system. To maintain the ETA10 NADs in a loosely coupled network, there must be at least one host on the network running NOS.

Open File Table

A table created for and used by the shared memory manager to translate logical file requests to disk physical-file requests.

Open Interconnection Network

ETA System's local area network; see also *Local Area Network.*

Operations Management

The operating system kernel features that interfaces with the service unit utilities.

Ownership

A characteristic of configurable units that changes according to unit status. A unit is owned by the entity currently responsible for managing its resources. This may be the operating system, a customer engineer, a diagnostic subsystem, or a unit may be unowned (free).

Page

An allocation unit of CP memory. Page size in initial releases of the operating system software is 2K words.

Page Fault

A page fault occurs when a process requests a page not currently in central processor memory.

Page Table

A table that contains the page table entries. One of a process's locked-down process objects. The hardware uses a copy of the page table (in the space table) to associate a virtual address with a physical address.

Pager

A system process belonging to central processor memory management that responds to page faults.

Partition

A logical group of data such as a collection of records.

Partition Number

The number of partitions of the same type which precede the particular partition delimiter in a file, plus one.

Path ID

The identification within a Loosely Coupled Network host of a path to a connection.

Pathname

An ordered list of node name identifiers that, together with a point of origin within the directory set, uniquely identify a file system object.

Permanent File

A file that is stored until it is explicitly destroyed.

Permanent File Transfer Service

The RHF service that transfers files between the ETA10 and remote Loosely Coupled Network hosts.

Physical File

The elemental unit of disk data storage available to the logical-file system.

Physical-File ID
The unique name by which the logical-file system identifies a physical file.

Physical ID
The address of an RHF host on the Loosely Coupled Network.

Physical Sector Address
An ordinal number used to identify a sector on a disk.

Power and Cooling Supervisor
A combination of hardware and software that enables the ETA10's power and cooling system to be monitored and controlled through the service unit. The power and cooling supervisor relies on a system-wide network of sensors and actuators to provide information and to perform control operations.

Power and Cooling Supervisor Processor
The single board computer in the service unit that communicates with the data acquisition processors.

Power and Cooling System
The set of features that includes the power distribution system, the cryogenic system, the refrigeration system, and the power and cooling supervisor.

Power Distribution Unit
The system component that monitors incoming power and distributes it to other system components. It provides a tie-in for various alarms, limits inrush current, and isolates the system from transients on the power lines.

Precision
Reflects the number of fully accurate bits in a computed result, and depends upon the number of bits used in computation. Using 32-bit half words on the ETA10, numbers are precise to seven decimal digits. In a 64-bit full word, numbers are precise to 14 decimal digits.

Private File
A file owned by one VSOS environment user name.

Process
The execution of the smallest unit of work that can be executed on the ETA10. An executing program is a process.

Process Descriptor Block
A central processor memory object created by process management to maintain process attributes and scheduling information.

Process ID
A system-wide unique identifier assigned by process management when it creates a process.

Process Image

A linked set of one or more domains. The process image consists of those domains that are accessible to a process.

Process Management

The software feature responsible for the recording and modifying of process related variables. Process management is also involved in CPU initialization, building the process queues for the monitor, and setting up the process packages and register tables for all locked-down servers.

Process Package

A central processor memory object created by domain management to hold the invisible package, domain package, and domain stack.

Processor Status Registers

An assortment of registers in the CPU that contain information about program execution, including execution address, keys, access violation codes, intermediate vector results, instrumentation counts, and data flag branch status. These registers are loaded from a process's invisible package when the process begins running, are saved to the invisible package when the process is blocked. Not to be confused with the general purpose registers of the register file. js

Pseudo–hardware Unit

An entry in the system configuration table that is linked to an entry for a hardware processing element which represents an executing instance of a binary software unit.

Public File

A file which is owned by the VSOS environment, and is automatically attached to a session.

Record

A structured collection of data. A file can consist of a set of records.

Refrigeration System

Cooling system that has components for the cryostat power supply, the SCMI cabinet, the shared memory interface, the shared memory power supplies, and central processor memory. The refrigeration system is separate from the cryogenic system.

Register File

A set of 256 directly addressed, 64-bit general purpose registers in the central processor. The lower 128 64-bit registers can also be addressed as 256 32-bit registers. Scalar instructions reference its registers as locations of source and result operands. Vector instructions reference registers containing memory locations of source and result operands.

Register Table

One of the locked-down process objects created by process management to hold the CPU registers for a process while it is not running. Also called *register block*.

Remote Host Facility (RHF)

A set of system applications used to transfer data between Loosely Coupled Network host computers.

Remote Procedure Call

A feature that uses mailboxes to transfer messages and requests between central processors, I/O units, and the service unit.

Remote System Support (RSS)

A central facility that supplies hardware and software support to customers. In some cases RSS may be a repair parts depot.

Resident Set

The set of a process's pages currently resident in CP memory.

Resource Manager

Any software feature that manages a hardware resource (such as a memory) independently or with other software.

RETA

Remote ETA10 login software that operates on Apollo gateways and connects Open Interconnection Network users to the ETA10.

RHF Access Method

Central processing unit software that controls the ETA10 interface to the Loosely Coupled Network.

RHF Access Method Frame

The frame that holds a connection request block or a network block and is inserted into a message frame before transmission over the Loosely Coupled Network.

RHF Logical ID

A name associated with the physical identifier of an RHF host on the Loosely Coupled Network.

Scalar Processing Unit

A set of arithmetic units that produce one result per machine cycle; also contains the instruction pipe and the register file.

SCMI Cabinet

The cabinet that houses shared memory and its interface, communication buffer and its interface, as well as the I/O, service unit, and data pipe interfaces, and the power supplies for these components. (SCMI is an acronym for Shared and Communication Buffer Memories and Interface.)

SECDED

Stands for single error correction, double error detection, and is a system memory protection mechanism. Single errors are correctable, double errors are fatal.

Sector

A portion of a track on a disk. A sector is numbered and can hold a specified number of characters. It holds the smallest piece of data that can be physically transferred to and from a disk.

Segment

A number of consecutive minimum allocation units on a disk.

Serial I/O Line (SIO)

A communications channel that allows the service unit to communicate directly with an I/O unit to run diagnostics or transfer data.

Server

A process initiated and run by the system rather than by a user.

Server Node

The service unit node that performs most of the monitoring and control work of the service unit. This node is powered by a 32-bit processor with two Mbytes of internal memory, and is equipped with a 500 Mbyte disk drive.

Service Unit (SU)

Provides monitoring and control capabilities for the hardware and software components of the ETA10. The service unit is a set of server and display nodes connected by a 12 megabit network.

Service Unit Diagnostics

A set of diagnostic tests invoked from the service unit which evaluate the status and operability of ETA10 system components.

Service Unit Interface (SUI)

A system component that interfaces the service unit with the high–tech boards of the computer system for running diagnostics or transferring data.

Service Unit Utilities

Service unit software used to initialize and operate the ETA10.

Session

A series of commands executed by one user. Sessions can be batch or interactive. Same as a job.

Session Dayfile

A temporary file created for each session which is maintained by the command shell to record commands and statuses produced by the session.

Shared Access

The logical file access mode in which multiple users maintain a common position within a logical file.

Shared Memory (SM)

Ranges in size from eight to 256 million 64-bit words, and supports a virtual address space of up to two trillion words. Major system storage resource and staging memory accessed by all central and I/O processors. Addressed on half-word boundaries.

Shared Memory Interface (SMI)

Provides the communication link for shared memory and the eight central processor ports and one I/O port. Five access slots allow transfers to as many as five central processor ports at one time.

Shared Memory Manager

A memory manager software feature that allocates the large independent memory used by all system processors for general-purpose, in-system storage.

Shared Memory Object

A region defined in shared memory that can be used for data storage.

Shared Memory Port

Connects central processor memory to shared memory, transfers one 64-bit word per cycle between the memories.

Shortstopping

An option that enables chained instructions (both vector and scalar) to execute more quickly. Intermediate results are saved in special shortstop registers and re-inserted as source operands for successive instructions.

Silicon Controlled Rectifier Cabinet (SCR)

The cabinet that houses the control panel for heaters in the storage and distribution skid.

Simultaneous Access

The logical file access mode in which multiple users can independently access a single file.

Single Board Computer (SBC)

A component used in both I/O units and the service unit. They are 68020 32-bit processors programmed to serve a variety of functions. In the I/O units, they are controlled by I/O supervisor software; in the service units, they are controlled by service unit supervisor software.

Socket

An end point of an Open Interconnection Network connection.

Space Table

A copy of the memory's page table that is used by the virtual address mechanism for associative words pointing to virtual pages in central processor memory. The space table lists all pages in central processor memory and serves as the currently executing process's page table.

Sparse Vectors

Are vectors with a great many zero or near-zero elements. Special instructions reorder such vectors to minimize the storage and calculation of (near) zero elements, while maintaining their positional significance.

Sponsor

When configurable units are connected by two maintenance paths to the service unit, one to each server node, only one path may be active at a time. The sponsor is the server node supporting the active maintenance path. All maintenance activity is carried out through the sponsoring server node.

Sponsorable Unit

Any configurable unit that is connected by two maintenance paths to the service unit, one to each server node. Such units include all high tech boards, the I/O units, and the power and cooling system loop monitor.

State

The current status and availability of any given system component is referred to as its state. A processor may be either *on-line* or *off-line,* for example.

Storage and Distribution Skid

A tank that holds the liquid nitrogen for storage and distribution. It provides a reserve capability during maintenance operations or in the event of a power failure.

System Configuration Control

A system manager software feature that is responsible for maintaining an accurate record of all elements of the system configuration.

System Configuration Management (SCM) A feature responsible for initial system configuration and the promotion and demotion of configurable units. It handles any operator notification or participation with regard to automatic reconfiguration.

System Configuration Table (SCT) The operating system table, maintained exclusively by the system configuration control feature, that defines the configurable units of the ETA10, their attributes, and status.

System Interface Library

A collection of subroutines provided with the VSOS user environment to allow applications to communicate with operating system functions.

System Log and Query

A service unit utility that logs and displays ETA10 system events.

System Logging and Analysis (SLA)

A feature of system control that records errors and performance data in system logs, identifies trends in system operation based on sequences of faults, and notifies system monitor of required changes in system configuration.

System Monitor

A system manager software feature that receives fault notification from throughout the computer system. System monitor has pieces executing in each processing element of the computer system and provides the focal point for reporting system errors. System monitor is also responsible for running hardware and software verifications.

System Start Source

A hardware control that allows the service unit operator to change how the system synchronizes itself.

Task

A unit of work known to the VSOS user environment. A task in the VSOS environment maps to a process on the ETA10 system.

TCP (Transmission Control Protocol)

United States Department of Defense communication protocol that defines the way TCP/IP packets are sequenced to provide a reliable TCP/IP stream connection service.

TCP Packet

The 24-byte frame used to sequence and transfer up to 1452 bytes of data over a TCP/IP stream connection.

TCP/IP Protocol

A Department of Defense communication protocol. The IP protocol defines the way in which IP packets carry routing information between networks. The TCP protocol defines the way in which TCP packets are inserted into and carried by the IP packets within Ethernet frames.

TCP/IP Protocol Processor

A board installed in a VME channel interface board that provides the physical and electrical interface to an Ethernet local area network.

Temporary File

A file that is destroyed at the end of the VSOS environment user session in which it was created.

Thrashing

Thrashing occurs when the system spends more time handling page faults than doing productive work.

Transceiver

The hardware connector that taps directly to an Ethernet cable and interfaces a TCP/IP protocol processor to the Open Interconnection Network.

Transceiver Protocol

1. For the open interconnect network, the Manchester format bit-serial encoding and decoding of TCP/IP packets.
2. For the Loosely Coupled Network, the format for bit-serial encoding and decoding of message frames.

Transmission Manager

The VME channel processor software that manages connections between the ETA10 and remote TCP/IP hosts via the Open Interconnection Network.

Trunk Control Interface

The NAD element that coordinates Loosely Coupled Network activity for up to four trunk control units.

Trunk Control Unit

The element contained in the network access device that sends and receives message frames over a Loosely Coupled Network trunk.

Trunk Coupler

The device that connects a network access device to the Loosely Coupled Network by physical connection to the network trunk.

Twos Complement Arithmetic

Arithmetic used in the ETA10 central processor. In the twos complement system, the left-most bit holds the sign (0 for plus, 1 for minus), and the remaining bits hold the number itself. Positive numbers are represented by their binary equivalent. Negative numbers are formed by replacing the 1s in the positive number with 0s, the 0s with 1s, and adding 1 to the result; a carry from the left-most bit is discarded.

U Type File

An unstructured file which contains a continuous byte stream of data.

Unattached File

A file that exists outside the VSOS environment session, but can be made available to a user who issues an ATTACH command.

Usage Attributes

Attributes pertaining to a particular opening of a file. Usage attributes may be listed and altered without affecting either the file's saved attribute or the attributes of other activations of the file.

User

Any person who has the privilege to use the computer system.

User Environment

A layer of software built on top of the operating system that consists of a command shell, command language, commands, utilities, and operating system interface routines. Together these features provide software facilities for the ETA10 user, handling interaction between the user and the operating system kernel.

User ID

A system-wide unique identifier for anyone privileged to log in to the ETA10 system.

User Management

A system entry software feature that validates sessions and connections when they first enter the computer system. It also provides utilities that permit user environments to provide an interface for a batch submission utility.

User Profile

A list of system privileges and features available to a certain user; a subset of the user registry.

User Registry

A table that contains the system database of user identifications and privileges.

V Type Record

Logical file record positioned by control words within the file.

Vacuum-Jacketed (V-J) Lines

Double-walled and vacuum-insulated pipes that carry the liquid nitrogen between components of the cryogenic system.

Valve Box

The system component that contains the on/off valves to control the flow of liquid nitrogen to the cryostat.

Vector Processing Unit

Contains two pipelines that compute memory-to-memory results at the rate of two results per machine cycle in full-precision or 64-bit mode, and four results per machine cycle in half-precision or 32-bit mode.

VME Bus Protocol

The IEEE P1014 communication protocol for directing a TCP/IP protocol processor connected to the Open Interconnection Network.

VME Channel

Provides the physical connection through the protocol processor to the Open Interconnection Network; it is controlled by the VME channel processor.

VME Channel Interface

A VME channel processor board that holds a smaller TCP/IP protocol processor board that connects to the Open Interconnection Network.

VME Channel Processor

An I/O processor (single board computer and data pipe buffer), a VME channel interface board, and a TCP/IP protocol processor board.

VSOS

(Virtual Storage Operating System) The CYBER 205 operating system; EOS supports a VSOS user environment.

VSOS Environment

The user environment on the ETA10 supporting the VSOS command set and flat file structure (the user interface to the CYBER 205).

Warm Start Initialization Process (WSIP)

When at least one CPU is running, a warm start sequence is used to initialize the remaining CPU's. See also *Cold Start*.

Working Storage Area

A set of virtually contiguous bytes in virtual central processor memory aligned on a byte boundary. The working storage area is used as intermediate storage when storing or retrieving information from the file system.

Workstation

A service unit display node. Display nodes have 4 Mbytes of local memory, a 15-inch color display, a keyboard with a 3-button mouse, a tape cartridge drive, and an 86 Mbyte disk drive.

Referenced Documents List

FROM ETA SYSTEMS, INC.

Number	Title
000211	Mainframe Subsystem Instruction Specification for the ETA10
000215	Mainframe Subsystem Instruction Specification for the ETA10
PUB–1051	VSOS Environment Reference Manual: Concepts and Commands
PUB–1084	VSOS Environment Reference Manual: System Interface Library Calls
PUB–1118	Support Tools: Utilities, Debugger, B.E.S.T.
PUB–1119	Support Tools: Diagnostics

FROM APOLLO COMPUTER, INC.

Number	Title
002348	Getting Started With Your DOMAIN System

FROM BRIDGE COMMUNICATIONS, INC.

Number	Title
09–0016–02	Ethernet System Product Line Software Technical Reference Manual Volume One – Kernel and Support Software
09–0017–01	Ethernet System Product Line Software Technical Reference Manual Volume Two – Protocols
09–0024–01	Communications Server User's Guide
09–0025–01	Gateway Server User's Guide

09-0043-00	Communications Server Getting Started Guide
09-0047-00	Ethernet System Product Line Cable Guide
09-0049-00	Ethernet System Product Line Network Control Server Installation and Operation Guide
09-0076-00	Product Line Overview
no number	Connection Service User's Quick Reference Guide

FROM COMMUNICATION MACHINERY CORP. (makers of the TCP/IP Protocol Processor (ENP-10) and the TCP/IP Ethernet Board (ENP-44))

Number	**Title**
622200	UNIX Internet System Manager's Guide
622300	UNIX Internet Programming Guide
no number	ENP-10 User's Guide
no number	ENP-44 User's Guide
S602100	Internet User's Guide

FROM CONTROL DATA CORPORATION

Number	**Title**
60455480	Remote Host Facility Handbook For DEC/VAX Systems
60458500	CDC 380-170 Network Access Device Hardware Reference Manual
60458520	CDC FW200-B/C Network Access Device Hardware Maintenance Manual
60458550	CDC CYBER 170-Based Loosely Coupled Network Trouble Shooting Guide

60458570	CDC 380–200 Network Access Device Hardware Reference Manual
60458710	CDC FW201–B/C Network Access Device Hardware Maintenance Manual
60458720	CDC FW207–B/C Network Access Device Hardware Maintenance Manual
60458730	CDC FW205–B/C Network Access Device Hardware Maintenance Manual
60458870	CDC 380–370 Network Access Device Hardware Reference Manual
60459050	Remote Host Facility Handbook For IBM Systems
60459060	Remote Host Facility Handbook
60459460	Hardware Performance Analyzer User Reference Manual
60460620	Remote Host Facility Usage
SMD 800171	CDC CYBER 170 Series Loosely Coupled Network Maintenance Software Handbook

FROM STANFORD RESEARCH INSTITUTE, INTL.

Number	Title
RFC–764	Telnet Protocol
RFC–765	File Transfer Protocol
RFC–768	User Datagram Protocol
RFC–791	Internet Protocol
RFC–793	Transmission Control Protocol
RFC–854	Internet Telnet Protocol
RFC–861	Internet Telnet Options

FROM THE U. S. DEPARTMENT OF DEFENSE

Number	**Title**
MIL-STD-1777	Internet Protocol
MIL-STD-1778	Transport Control Protocol

Index

Reader Comment Sheet

Providing our readers with effective documentation is one of our most important goals. You can help us improve our documentation by taking a few moments to review this ETA Systems publication. If you fill out this comment sheet and include your name and address on the reverse side, we will send you an ETA Systems pen to thank you for your help.

□ Yes □ Somewhat □ No Is this publication easy to read and use?
Comments:

□ Yes □ Somewhat □ No Does it tell you what you need to know?
Comments:

□ Yes □ Somewhat □ No Is the organization of topics logical?
Comments:

□ Yes □ Somewhat □ No Are there enough examples?
Comments:

□ Yes □ Somewhat □ No Are the examples helpful?
Comments:

□ Yes □ Somewhat □ No Are the illustrations effective?
Comments:

- Are there any errors in this publication? (Please list errors in the format shown below if possible.)

Page number	*Severity*	*Description of Error*
4–17	typo	ETP should be CTP
4–32	critical	4 inch slot should be 4 mm slot

Please include your name and address so we can send you an ETA Systems pen. If you would like a reply to any questions about the document, check off the applicable box. Fold this sheet on the dotted lines, seal it with tape, and send it to the address below. Thank you for your time and input.

□ Yes, I would like a reply. □ No, I don't want a reply.

Name: **Date:**

Company: **Phone:**

Street Address:

City: **State:** **Zip/Country:**

fold 1

fold 2

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 2294 ST. PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

ETA SYSTEMS, INCORPORATED
TECHNICAL COMMUNICATION DEPT.
1450 ENERGY PARK DRIVE
ST. PAUL, MN 55108

ETA
SYSTEMS

ETA Systems, Incorporated
1450 Energy Park Drive
St. Paul, MN 55108