Operating System

DDB-B2.3

OPERATING SYSTEM

TASK FORCE REPORT

April 20, 1973

Task Force Members

David Jasper	Executive Advisor
Al Irvine	Chairman - NCR
John Jantzen	Chairman - CDC
Steve Clothier	CDC
Roy Kuntz	NCR
Jeff Landreth	CDC
Larry Leskinen	CDC
Dave Morris	NCR
Dwight Olson	CDC
Jim Whelan	NCR

OPERATING SYSTEM

TASK FORCE REPORT

I. EXECUTIVE SUMMARY

- The mission and assumptions granted in the charter did not require significant modification.
- A 'common operating system' is possible.
- A 'common operating system' is necessary as the glue to sustaining an integrated product line in the eyes of the user.
- A common software set is necessary to achieve a total savings on a joint effort, i.e. development costs will not be reduced, only focused; support costs are reduced only if the product line is integrated with common software.
- A single (joint) development is feasible and desirable. The product, however, would not be a single or common operating system but rather a cohesive 'construction kit' for several versions of the operating system.
- The kit would allow us to produce versions which were optimized for space, or performance in different operational environments, such as batch, interactive, and transaction. But below a cutoff point and above a defined size the probability of successfully using a single set of modules was judged to be slight.
- The Operating System, in order to span the range with one set of modules must have a consistent functional architecture for all models of the line. The architecture was defined.
- The Operating System was sub-divided into twelve functional components for detailed analysis:

- Task Management
- Job Management
- Memory Management
- Job Control Language Processor
- Device Drivers
- Record I/O
- Block I/O
- Scheduling and Allocation
- Binding
- Device Allocation
- Operator Control
- Data Management

- Each component investigated appeared to have the same functional and philosophical characteristics from both companies' points of view.
- Most components appeared to require parameterization or different versions in order to serve in the target range of configurations or for different operational environments. Device drivers were an exception; they appeared to be independent of the preceding variables.
- The cost effective use of the 6150 for the "PPU" changes the O.S. approach from that used in the CYBER 70 to an approach requiring more sophisticated memory management and multi-tasking.
- A concept of locally public memory was described.
- Scheduling and allocation of resources is the primary component affected by changes of scale and operational environments.

- Appendices showing present SCOPE 3.4 distribution of components was attached at the request of the JAC. The O.S. Task Force also voted on a probable distribution of components in the proposed Operating System.

II. RECOMMENDATIONS

A. Success Criteria

There are four major issues which the Operating System Task Force recognized as critical in being able to state that a common operating system is a recommended venture.

1. Agreement had to be reached that there was a commonality in functional requirements for operating over the range.
2. We had to recognize the possibility of at least one possible structure of an operating system to cover the range.
3. There had to be general agreement that a peripheral processor based operating system was feasible and desirable for the entire range.
4. We had to be convinced that no technology break throughs would be required to produce a common operating system to cover the joint product line.

The Operating System Task Force believes that these criteria have been met as evidenced by the following conclusions and recommendations.

B. Summary Conclusions and Strategy for a Common Operating System

The Operating Systems Task Group believes that the most important concept which must be understood by any reader of this report is that we do not believe that a "Common Operating System" or a "Single Operating System spanning the full range" is attainable. We do however believe that a joint development effort is feasible and desirable which would produce what we shall call an Operating System Construction Kit.

The group applied the majority of its energy to comparing their experience and preconceived ideas regarding functional features and design concepts of an operating system for the range of configurations under study. This was done with the goal of arriving at a measure of the feasibility of an "Operating System Construction Kit" with significantly greater confidence than could be attached to the feasibility of a "Common Operating System" as suggested by the JAC in their report to the Steering Committee. An "Operating System Construction Kit" was understood to be a set of modules defined and implemented such that, although redundant versions of some modules may be required to span the range, the interfaces to those modules are processor independent and consistent to the extent that to replace one module with another version of that module would necessitate no redesign or reprogramming of the remaining modules. Further it is recognized that not all modules need reside in the peripheral processors. Some modules may, in fact, be CPU modules. We arrived at the conclusion that such a "Construction Kit" is feasible from a 256K bytes, single processor (6150) system through a system containing a single P2 central processor, four 6150 peripheral processors, and 8M bytes of memory to a system composed of multiple P3 central processor, twelve 6150 peripheral processors with 32M bytes of shared memory.

This conclusion is based on our assumption that a satisfactory and consistent virtual storage strategy can be attained (See Virtual Storage Task Group Report).

It is recognized that the above range of configurations does not cover the full range of configurations defined in the table of Integrated Product Line System Modules prepared by the Strategy Task Group (See Appendix C). The Operating System Task Group believes there is a significant risk associated with assuming that we can utilize common modules and interfaces as well as retaining a consistent operating system structure and architecture above and below the indicated range.

An "Operating System Construction Kit" as defined above, would, from our customer's point of view, appear to be a set of versions of an operating system for an integrated product line. This is due to the fact that we, the manufacturers, would configure several versions (three to six) from the "set of modules" which together would cover the range of configurations and operational environments under consideration. However not only could a customer move an application which he had developed from one version to another with no difficulty, but it would indeed be true that many modules appearing in one version would appear in one or more other versions, and further than that the inter-module interfaces would be consistent across all of the versions.

It is in fact the tightly disciplined control of inter-module interfaces which is the heart of our understanding of what needs to be accomplished in order to say an "Operating System Construction Kit" has been realized across the range of products.

For example, suppose that when the development is complete and the dust has cleared, every module in the set appears in two versions and furthermore that when we configure the various operating system versions we discover that some of them contain only the "A" versions of modules and the others contain only "B" versions of modules. We would still argue that we had attained our goal of producing an "Operating System Construction Kit" if the inter-module interfacing was such that a version of a given module could be replaced with the other version of that module.

The interchangeability of modules is the pivotal issue with regard to the attainment of an "Operating System Construction Kit." It is of the utmost importance, not only to attain the necessary degree of configurability, but more importantly it is necessary in order that the unit testing of modules can minimize the system integration, system generation (build), system test, field configuration and field installation costs and schedules.

A clear understanding of this issue focusses our attention on the single most critical element of risk. The degree to which we realize this goal of interchangeability of modules will determine the degree to which we succeed in our endeavor. Furthermore, the realization of an adequate level of success will require a well conceived and implemented set of tools, practices and procedures, and administrative controls . . . in short a comprehensive "software engineering discipline" which is understood, accepted, and enthusiastically supported by management and development personnel.

The Operating System Task Group believes that an adequate level of success in these areas can be attained, they do not believe that it will be easy, but they agree that it is necessary. Furthermore, we believe that quite independent of any joint development activity, each company must pursue vigorously the area of technology necessary to produce an "Operating System Construction Kit".

The technology of 'Operating System Construction Kits' has evolved since the middle 1960's with the introduction of OS/360. Today both NCR and CDC use System Build or System Generation techniques which allow parameterization of well defined functional modules. Additionally, recent developments in the industry include table driven techniques and syntax driven mechanisms. However, the use of replaceable modules in the Operating System Construction Kit creates interdependencies that compound the interface and testing problems. The additional development complexity and cost, however, are expected to remain within reasonable limits. In other words, the requirement to span the full range increases the magnitude of the problem appreciably but does not require a technology break through. Further, parallel advancement in software writer's languages and software engineering techniques aid in the advancement of configuration independent Operating System Construction Kit technology.

C. Recommended Actions

The Operating System Task Group recommends five specific activities which should be initiated as quickly as possible:

1. Obtain full corporate commitment by both companies to the joint development of the Operating System Construction Kit as defined. This assumes that in one way or another a single organization is established and given the responsibility for that development.

2. Since the software engineering tools and procedures are critical to ensure success, the specification of a total software engineering system must be initiated.

3. A clearly delineated statement of requirements and goals for the Joint Product Line must be prepared. However, work on Items 1 and 2 above can proceed in parallel for perhaps as much as six months preceeding such a statement.

4. An integrated product line has been expressed as a multi-dimensional opportunity. The basic dimensions are 1) direct cost saving through shared development, production and support, and 2) a broad product offering in the eyes of the user. The first point does not require an "integrated product line" but rather only the sharing of some components. Further, it is the belief of the Operating System Task Force that a certain critical mass of commonality must be achieved in order to successfully achieve and maintain a truely integrated product line. A common set of operating system versions is the least amount of commonality to sustain an integrated product line and is in fact fundamental to any other "external" or software commonality.

5. We further recommend that any joint venture must go beyond an agreement to implement operating systems to common specifications. Any such agreement would be untenable due to the many conflicting pressures in each company. As a result, during the varying stages and at various levels of the development process commonality would be lost to the extent that its value would be neglible.

III. MISSION AND ASSUMPTIONS

A. Mission

The main mission was consideration of the full opportunity of cooperation. There may well be other levels of cooperation in the area of operating systems, however, the operating system task group did not give explicit consideration to them but rather assumed this was to be a consideration of the Strategy Task Group. The mission was described then as:

> Propose and define method(s) for achieving the following goal:
>
> A single set of modules which can be used to construct general purpose operating systems for configurations spanning from the top of the standard product line down to a minimum.

The proposal should quantify the degree of anticipated success in several dimensions:

1) Range of configurations;
2) Range of features (subsettability);
3) Range of O.S. type emphasis (e.g. time sharing, remote batch, transaction processing.

B. Assumptions

- The logical or virtual architecture is consistent for all models of the joint product line.

- The minimum member of the joint product line for this O.S. will consist of: single processing element, 256K bytes, 100M bytes RAS, keyboard and CRT operators console, plus peripherals.

- A large system is assumed to be
  - > 2 CPU's with option for their own private executable memory
  - > 10 PPU's
  - > 100M bytes main memory
  - > 3 block access storage devices plus a large variety of peripherals

- Multiple systems are to be considered and consist of some combination of large systems which are coupled via dynamically shared peripherals and block access storage devices. Each system executes its own copy of the operating system.

- Network systems are considered to be multiple systems which are linked but do not share peripherals and block access storage devices.

- The O.S. will reside primarily in the PPU's.

- Same data formats across all processing elements of the P.L.

- All modules of the O.S. can be written in one software writers language which will be defined by that task force.

- Memory will be managed by the O.S. with a consistent virtual memory scheme.

- Permanently available systems disk.

- V.M. and co-resident virtual machine proposal will be developed by other task forces.

- There is no particular requirement to support old devices or file formats in the new O.S. except as required for co-resident virtual machines.

- Hardware modifications generated by this task force can be included in the system design.

C. Critical Issues

- General strategy for a specific tasking structure for CPU - PPU - memory resource allocation including multiprocessing, scheduling, dispatching, etc.

- General strategy for O.S. levels of features and services for a range of configurations.

- General strategy for O.S. levels of capability to perform timesharing, remote batch, transaction services, etc.

- General strategy for O.S. construction to satisfy modularity, flexibility, configuration independence, optimization of performance, linkage mechanisms, and control methodology.

Other Issues:

- Reliability, availability, serviceability consideration.

- Human factors

- Performance

  PPU logical structure

  P-language

  Identification of critical performance issues.

- Effect of virtual memory on O.S. and PPU's.

- Estimate the number of PPU's required in the minimum configuration (virtual).

- Any special considerations for project control.

- Recommended service functions (e.g. measurements, modeling, system integration, regression tests).

IV. CONSIDERATIONS OF ENVIRONMENT AND SCALE

A. Operational Environments

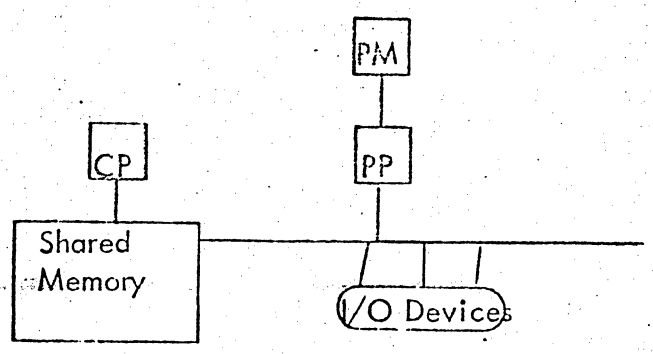
Six modes of access (operational environments) to the system (batch, remote batch, interactive, transaction, real-time and time-critical) have been considered. Those six have been reduced to three primary distinct modes of access. Remote batch and batch are considered to be the same. Some features required for time-critical applications will be included in the operating system but general support for these applications were not included in our discussions because these applications generally require specialization of the operation system for each application. Real-time and interactive requirements are similar and will be provided as capabilities in the operating system (real-time with small intervals will create problems). Therefore, three modes of access to the system were considered in this report. They are batch, interactive and transaction.

Scheduling and allocation appear to be the major differences in the operating system due to the three modes. The intent of the allocation and scheduling algorithms differ in different operating environments. Response time is the important objective in interactive systems, total throughput for batch systems and in transaction systems, the objective is to maximize throughput yet provide "reasonable" response time. Each of these objectives can be achieved primarily through the scheduling and allocation algorithms. Combinations of operational environments are also dependent on scheduling and allocation in meeting objectives. Co-existance of the different modes of access is an issue of volume and scale. In the larger systems it is clear that all three modes of access may co-exist and scheduling and allocation appear to be the major problem for the operating system in that co-existance.

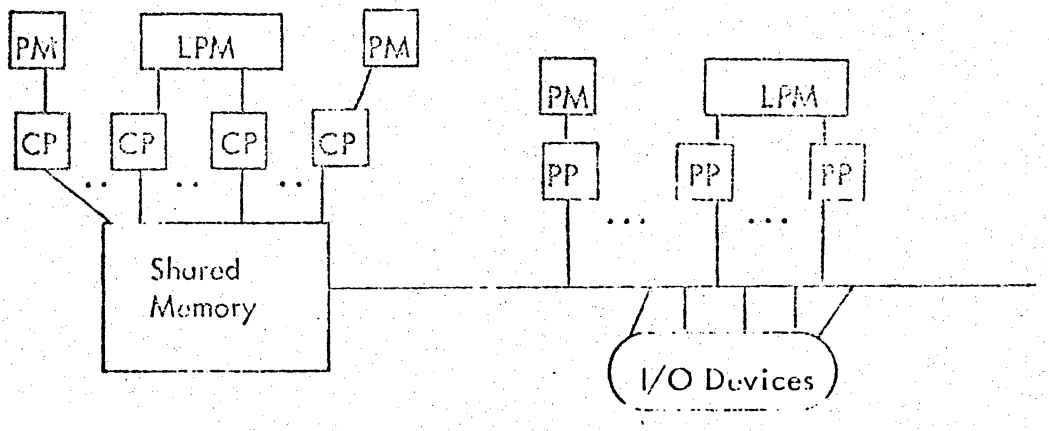
Different scheduling and allocation techniques and algorithms will be required for each of operational environments and for combinations of these. Additional techniques and algorithms will be required for various systems in the range (at least different algorithms will be required for the upper end and the lower end). The effort required for different algorithms can be reduced by using table driven techniques.

B. Issues of Scale

From the view point of the operating system, the Integrated Product Line must have a consistent functionally logical architecture. That architecture is described by the logical structure of the computing system which the operating system must support. From the assumptions described in Section III, this logical structure is shown as follows:



Minimum Logical Configuration



General Logical Configuration

One of the key questions then is what is required to provide an operating system which supports such a broad configuration range? It has been assumed that it is feasible to develop an operating system for the large general configuration. In considering the remainder of the range, memory has been identified as a key factor in the issue of scale. With shared memory sizes ranging from relatively small to extremely large, differences center around various strategies for optimum use of that memory. On the low end memory is extremely critical and optimization centers around effective utilization of that memory. On the high end the optimization centers around effective CPU utilization.

The task group has identified three potential techniques for minimizing resident memory requirements.

Compaction - By the use of interpretive execution techniques code compaction could be derived for OS modules thus deriving a savings in shared memory requirements. The exact extent of the savings is somewhat controversial but could be as much as 4 to 1 compaction.

Performance Degradation - Certain modules would be more finely overlayed or moved to non-resident storage. This would reduce overall shared memory requirements at the expense of decreased performance.

Reduction of Features - Certain features of the operating system could be eliminated by removing certain modules of the operating system or replacing such modules with modules providing a reduction of capability.

It was agreed that the above techniques could be incorporated to yield an overall reduction of shared memory requirements to provide a significant lowering of the minimum shared memory requirement to 1/4M Bytes. However, by following the above implementation approach, additional complexities will be introduced in this area. The significance of this is related to the quality of the techniques developed during implementation to cope with the added complexity.

V. ANALYSIS OF O.S. FUNCTIONAL CAPABILITIES

A. Definition of Key O.S. Modules

In order to assess the feasibility of a single development of a set of modules which can be configured into various installation-specific operating systems to cover the range encompassed by the Joint Product Line, the set of modules was partitioned into twelve categories. The set of modules falling into a given category is called a 'component'. The twelve components which were selected are briefly described below and a section for each component which discusses our deliberations and observations follows.

Task Management

Task Management is that collection of modules which provide services to tasks as entities. These services include task creation, invocation and inter-task communication primitives.

Job Management

Since "jobs" are arbitrary collections of tasks, Job Management is that collection of modules which provide operating system services which deal with collections of tasks which are designated by the user as a "job". These include job initiation, pre-allocation of resources to jobs, and job termination.

Memory Management

That collection of modules which provide the operating system services relating to the allocation, de-allocation, garbage collection (etc.) of real memory.

Job Control Language Processor

That collection of modules which effects those actions specified by the user in a "JCL Task". Where a JCL Task is a task whose source form is Job Control Language.

Device Drivers

Those device dependent modules which perform the functions associated with transferring a physical data records to/from an external device from/to memory.

Record I/O

Those modules which provide access to user defined logical records.

Block I/O

Those modules which provide the device dependent interface between Record I/O and the Device Drivers. These include buffer management and device scheduling.

Scheduling and Allocation

Those modules, which contain the decision-making strategies with regard to the assignment of resources to a collection of competing requestors of such resources.

Binding

That collection of modules which provide the Operating System services associated with resolving references among separately compiled program units. These services fall into two major categories "static" and "dynamic" binding services.

Device Allocation

Those modules which deal with the specific problems of allocating I/O devices as resources (see Scheduling and Allocation).

Operator Control

Those modules which provide the means by which an operator and the Operating System communicate with each other (see Job Control Language and the Job Control Language Processor).

Data Management

Those modules which provide the I/O and data structuring services above and beyond Record I/O, Block I/O and Device Drivers.

B. Analysis of Capabilities

Task Management

A task is the smallest unit of work recognized by the system. Thus, it is also the ultimate consumer of resources. It has two parts: the procedure which is never modified and the dynamic context which differs for different instances of execution and which may be modified. Task Management handles the creation of, termination of, control of, and communication among the tasks in the system.

Primitive Task Management Functions

- Task initiation

  ESTABLISH - creates an inactive task

  CALL - creates a process referred to as the callee of the process that issued the CALL (the caller).

- Task termination

  DISESTABLISH - destroys an inactive task

  CALLEND - terminates the issuing process.

- Events

  WAIT - puts a process into wait state (flushing all of its dynamic context to shared memory) until the occurrence of an event or set of events. Four types of events have been idenfified:

1. Alarm - a specified elapsed time or time-of-day.
2. Process termination - the destruction of a process after it has issued a CALLEND.
3. Signal - an event caused by another process issuing a SIGNAL function. The process waiting for a signal has the option to "extinguish" the signal when he is awakened by it (he will execute mutually exclusive of other processes who specified "extinguish") or to leave the signal set (he will execute along with any other process waiting for that signal).
4. Process directed wakeup - The process has been specifically named by another process to be awakened.

SIGNAL - sets a named signal. If this is an event another process has selected (been waiting for), it will be made ready (awakened). If the signal has not been selected, it will be saved until a selection is made. If the signal is already set, the signaler has the option of having this setting disregarded or having it stacked behind the previous setting.

- Inter-process parameter passing

Along with the ESTABLISH, CALL, and SIGNAL functions, a process may pass some parameters. This may be either by name (shared segments) or by value (copied data from one process's address space to the other's).

EXEC Tables

- Task list - is composed of task definition blocks and process control blocks. It appears that an abbreviated process control block of up to 200 bytes would be permanently resident in shared memory, showing such things as linkages to other processes and current state. Other elements of the process control block such as the address space description (segment and page tables) could possibly be moved out to secondary storage if necessary.

- Event lists - contain signals that have been set or selected and times-of-day that processes are waiting for.

- Ready list - shows all processes that are ready to execute on a processor. This is the list searched by each processor when it needs something to do.

Tasks are the basic mechanism for load leveling across processors. This takes two forms:

1. Load leveling across different types of processors is to be done by re-coding and/or re-compilation of a task (not procedure) for the desired type of processor. The interface between tasks is strictly processor-independent; a task does not know what kind of processor is used by the tasks that is communicates with.
2. Load leveling across processors of the same type is to be done by maintaining all dynamic context of a process in shared memory whenever the process is not executing on a processor.

Switching the peripheral processor among processes can be done by firmware or software. Firmware switching may be faster but results in a significant loss of control over scheduling. Control over scheduling is important because of the wide variation with operational environment (see section on Scheduling and Allocation). Software switching is, therefore, preferred unless performance analyses show significant degradation in total system performance.

It is necessary for tasks to communicate. This is generally done through two kinds of functions which are sometimes combined:

1. Event handling functions provide facilities to synchronize operations.
2. Parameter passing functions provide facilities for passing data.

Events can be handled with any of three methods:

1. Events trigger immediate scheduling cycle (Interrupt Driven).
2. Events set a flag in memory which must be tested later (Memory Driven).
3. Events transmit data to a task, the data is queued until transmission can occur (Message Driven).

Any hardware configuration and most software systems provide some elements of each method. Appropriate software techniques will usually allow one method to be converted to another, if necessary.

Processes are either ready or waiting. The state is usually represented by the process being in either a ready list or a waiting list. Event posting functions cause processes to be placed into the ready list. Event waiting functions cause processes to be placed into the waiting list.

Parameters can be passed between processes either of two ways:

1. Data is placed into and removed from a segment shared by the communicating processes. In this method the processes control the communication themselves.
2. Data is moved from the address space of one process to that of the other. This facility must be provided by the system.

Processes must initiate the execution of other processes, control their execution and recognize their termination. This allows a system structure where a system task spawns (creates under its control) job processors which spawn user processors, etc. (see the section on Job Management).

Task structure is important in system structure. Because a task can run only on some real and virtual machines, tasks must be structured so that functions can be executed by the processors appropriate to it. It is desirable to be able to use different processors for the same task without changing the structure.

Task dynamic contexts and working sets can be easily identified if tasks correspond to source language (PASCAL) procedures. This can greatly simplify operating system implementation.

Both peripheral processors and central processors will execute task management. All task management dynamic data resides in shared memory where it can be accessed by all processors.

Effect of Varying Size and Operating Environment

We believe that the basic architecture described above will be unaffected by large variations in the numbers of tasks. We expect as many as 1000 processes in the 1-4M byte range of systems.

This also does not seem to affect the architecture. There will be a problem in real-time environments with small time intervals since we expect a reasonably large task switch time (up to one millisecond).

JOB MANAGEMENT

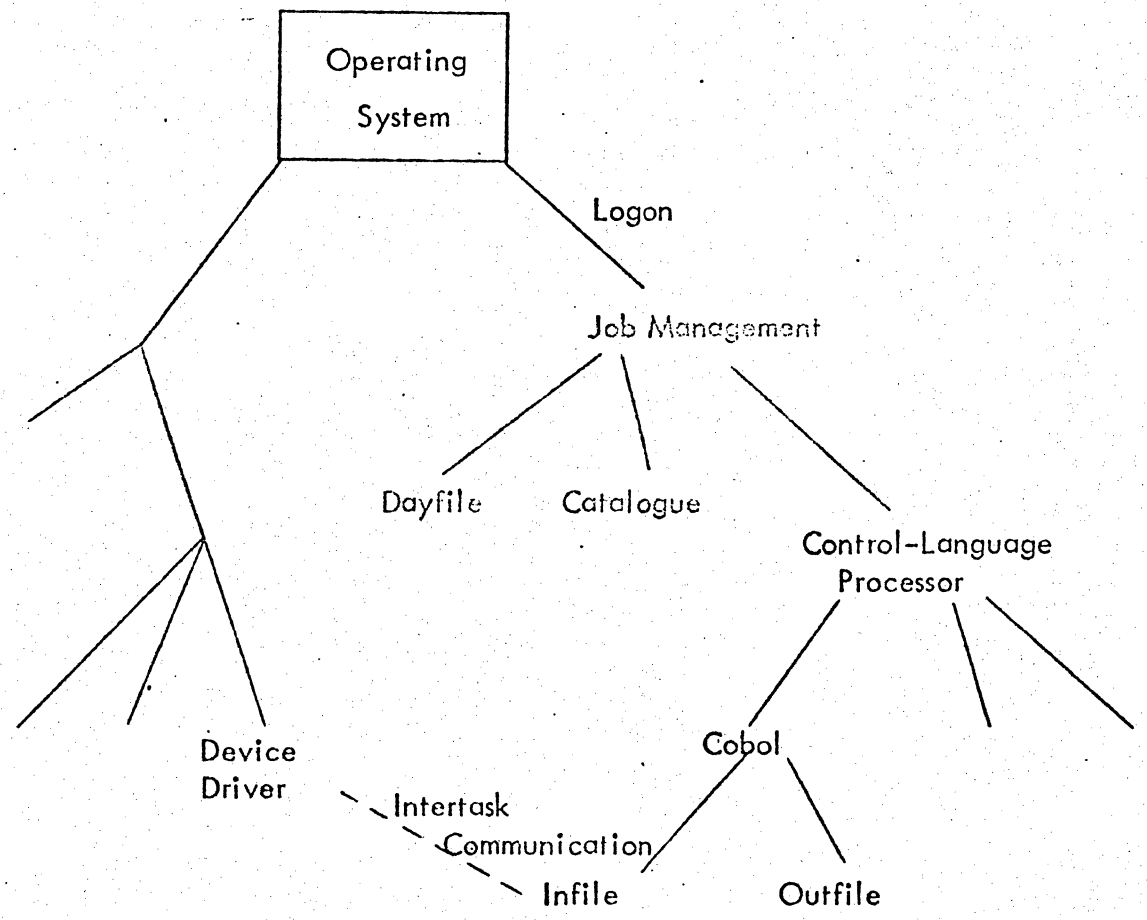
Job management here is defined mostly by exclusion. It is those modules which are not JCL, I/O and data management, product set, applications, or task management. It includes spooling of input and output files, job scheduling, and operating environment definition.

We have identified three modes of access to the system: interactive, batch, and transaction. Transaction differs from interactive in that it usually has shorter, more well-defined periods of activity. It appears to us that all three modes can co-exist in the same system by allowing suitably flexible scheduling algorithms, such as table-driven. Even though the implementation is probably non-trivial, it appears to be less a function of the wide range of scale we are talking about than the variation in application of any one size system.

A "job" is the external user of the system's idea of a piece of work. The operating system in general deals with tasks,with a job being an artifically related collection of tasks for the convenience of the user.

For example, a job might be considered to be all the tasks created as a result of the items between "Logon" and "Logoff" control items. There will probably be at least one "Operating System Job." There must be accounting information provided at a job level.

The following diagram depicts the relationship among system, jobs, and tasks. All tasks which become active as a result of the Logon are part of a single job.



Note: For the purposes of this diagram a file is an asynchronous process which provides data to its parent process via the intertask communications mechanism.

Several facilities directly related to job management must be provided by the operating system.

Interjob communication must be provided for. The use of global variables through the job control language seems the proper mechanism. That is, there will be a class of variables which are global to the entire system or to a set of jobs in addition to variables which are local to all tasks of a job.

Jobs must be able to initiate other jobs. In effect, this means that a job must be able to present a Logon and a Job Control Language file to the operating system.

There must be facilities for "deadline" scheduling of jobs. That is, the system must be able to initiate a job by a certain specified time and/or guarantee completion of a job by a certain specified time.

Preallocation of resources will be allowed. An installation may require that all resources be preallocated to all jobs.

Job sequencing, both basic ordering and conditional execution must be supported. Sequencing can be accomplished with the "jobs initiate jobs" mechanism. This can also help with sequential resource allocation in a required preallocation environment.

The facility to terminate jobs as jobs (that is, all tasks relating to a job) must be provided.

One area affected by operational environment and installation is the accounting. The accounting implementation will have to be parameterized in such a way that selected portions can be turned on/off.

Considerations of Scale - Differing requirements on job management over the possible configurations.

The requirements of job management are essentially constant over both dimensions of scale and operating environment (small, overlap, large | batch, interactive, transaction).

Conclusion

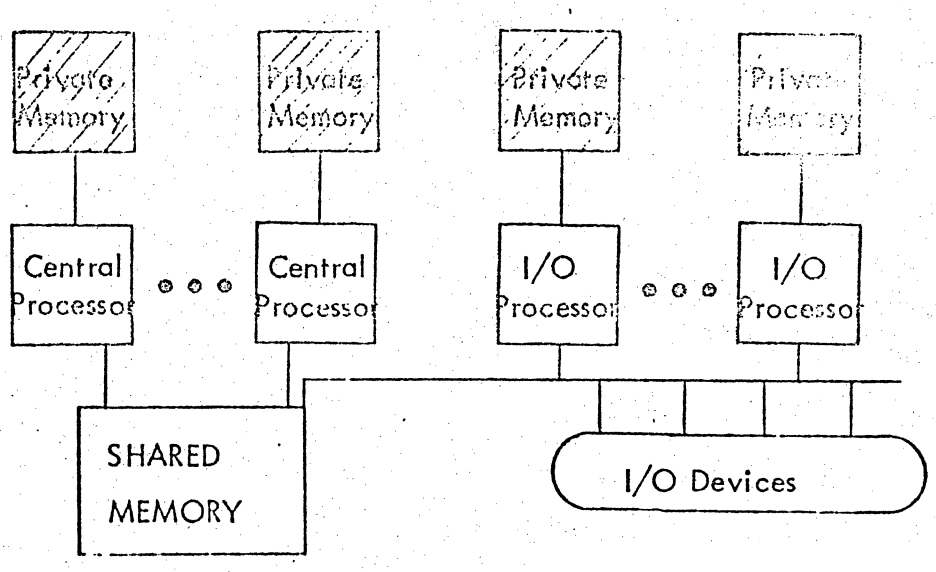
The job management facilities previously planned and required by CDC and NCR are basically identical. We see no reason why a common set of facilities could not span the entire range of the joint product line.

MEMORY MANAGEMENT

In order to discuss memory management, it is necessary to make some assumptions about the type and configurations of memory possible on a reasonable system. Given the four elements, Cyber 80 processor, 6150 processing element, Cyber 80 memory, and 6150 MSU, a large number of configurations, some with highly different characteristics, can be postulated. Unless hardware can make memory transparent, the operating system will have to management it. Thus, memory is a resource and will be managed by kind and access attribute.

A logical configuration has been specified into which most "reasonable" configurations can be mapped. Only those actual configurations which can be mapped to the logical configuration are considered further. Other possible configurations are not considered part of the common product line by the operating system.

The logical configuration was evolved through two iterations. The earlier configurations may be of some interest as rationale for the ultimate logical configuration. This first configuration is shown in the diagram below:



The major constructs in this configuration are that there is one memory which is directly accessible by all processors, both central and peripheral. Also, each processor may optionally have memory private to it. Memory management then has two or possibly three aspects, shared* memory management and private memory management, which can be further divided into C.P. private memory management and peripheral private mamory management.

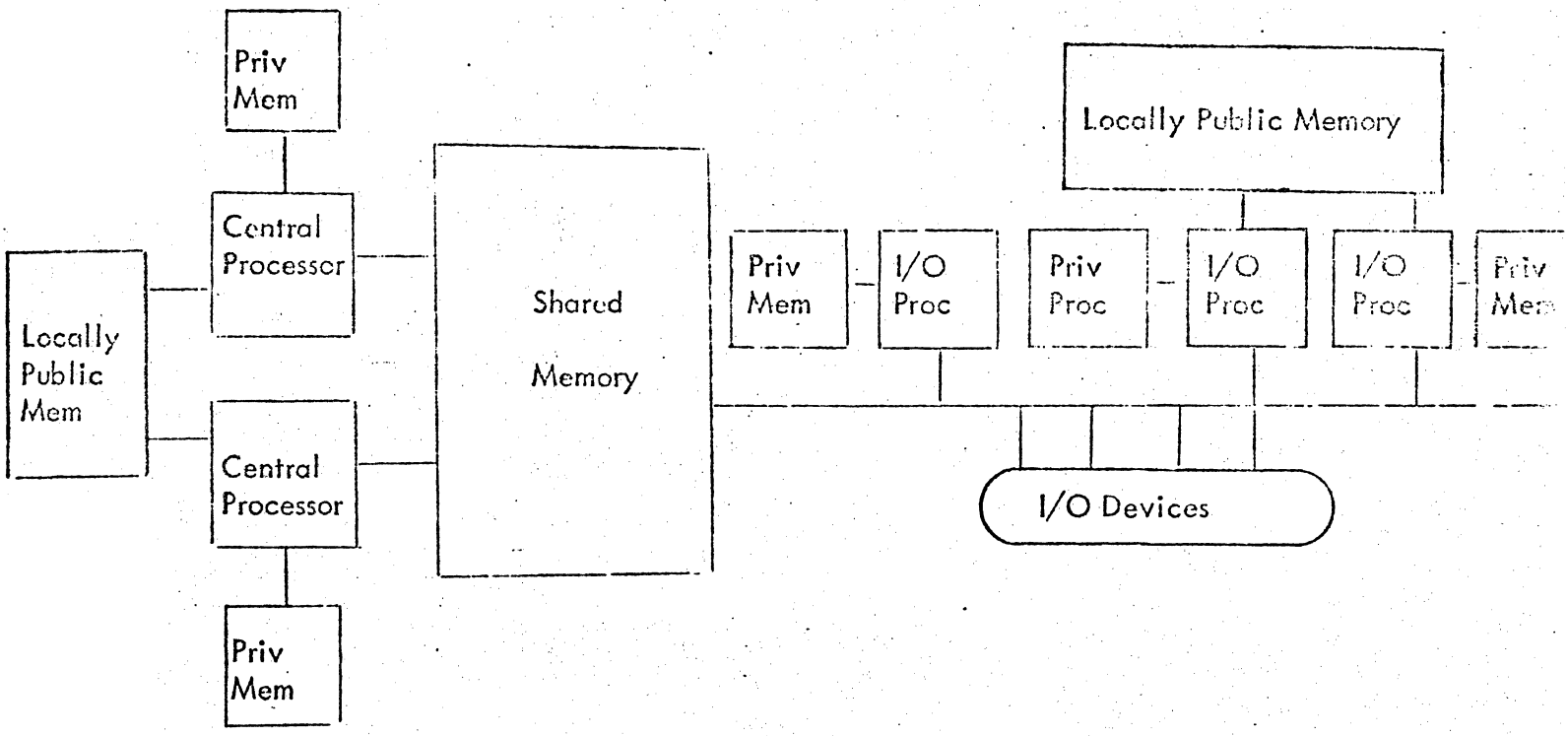
The advantages of private memory seem to center around reducing conflicts in shared memory. The advantages of shared memory are to reduce data redundancy, lower total memory requirements, etc. The task group has only considered private memory as it relates to the peripheral processors. Private memory usage on a central processor would be very dependent upon the type of memory used; cost, speed, size, etc.

There are three distinct alternatives to the amount of private memory a processor could have:

1. None - at least logically which leaves open the option for a hardware cache.
2. Small - which implies mono-tasking in private memory with a single operating system task at a time with roll out or partial roll out between tasks.
3. Larger - which implies that the operating system multi-tasks the private memory.

*Shared memory is often called main memory in other contexts by the task force.

Since there are some definite cost/performance tradeoffs concerning private versus shared memory, it is desirable* to expand the previous logical configuration to include the one shown below:



* 1. Locally Public Memory and shared memory maybe a logical partition of physical shared memory.
2. The operating system task group believes that physical configurations analogous to this logical configuration are desirable but questions of cost/performance relative to the physical configurations must be answered by the System Design Group.
3. See "Memory Management and Task Structures" on next page.

This configuration introduces the concept of "locally public memory", that is, memory that is shared by some, but not necessarily all processors. This construct gives the greatest degree of flexibility in configuring for cost effectiveness, since only those processors which need to access data or execute code in the Locally Public Memory access it and yet they can share a single copy of that code or data.

Memory Management and Task Structure

It is highly desirable for a given task to be able to migrate between processors in a multiprocessor configuration. This allows desirable flexibility in load balancing. Large private memories tend to complicate this while optimizing accesses to main (shared) memory. To permit this task migration the following rule is offered:

Rule When a processor reaches the point at which it can no longer perform useful work on a task, it must place the dynamic context of that task in the shared memory where "another" processor may pick up that task when it is ready for servicing.

Private and Locally Public Memory Management

For private and locally public memory management strategy there are three points in a spectrum which roughly correspond to the amount of private memory; none, small, large:

1. Tasks are unaware of the location (re: shared or private memory) of dynamic context.

   This may utilize a cache memory approach.

2. Tasks believe that all dynamic context is in shared memory.

   This corresponds to the 6000 system where an entire task is swapped into private memory.

3. Tasks decide which parts of dynamic context are in shared and which in private memory.

   This approach would probably be used to multiprogram a peripheral processor.

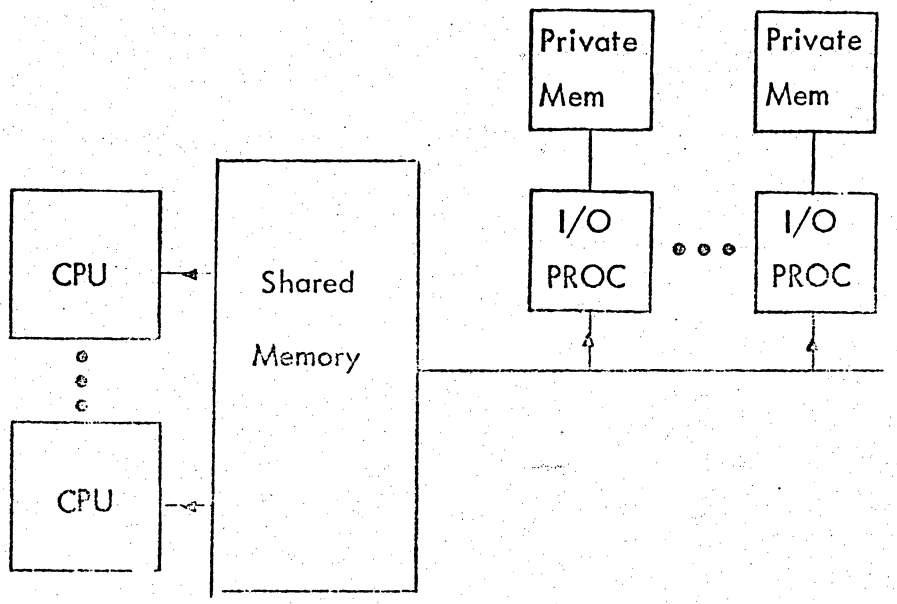
Shared Memory Management

It was agreed that the deallocation of a page and the allocation of a page when pages are available in shared memory would best be done by the processor requiring the function. This means that that part of shared memory management would have to be duplicated to run in any processor. Also, this, like other facilities, implies some interlock mechanism which can be used between processors of different types.

Virtual Memory

Along with the hierarchies of shared, locally public, and private memory, virtual memory has a major effect on memory management. One of the assumptions of the operating system task force was that "user memory can be managed by the operating system with a consistant virtual memory scheme for all CPU's." The system design task force has provided the further assumption that virtual memory capabilities are not optional, but will be present, at least to the Cyber 80 memory.

Since the operating system will reside primarily in the peripheral processors, the peripheral processors must be able to access virtual shared memory in a way consistant with the CPU's.



In the preceding diagram, the points marked with a " ▲ " are where consistent virtual memory accesses to shared memory are required. There are several alternatives to providing this access from the peripheral processor:

a) peripheral processor software

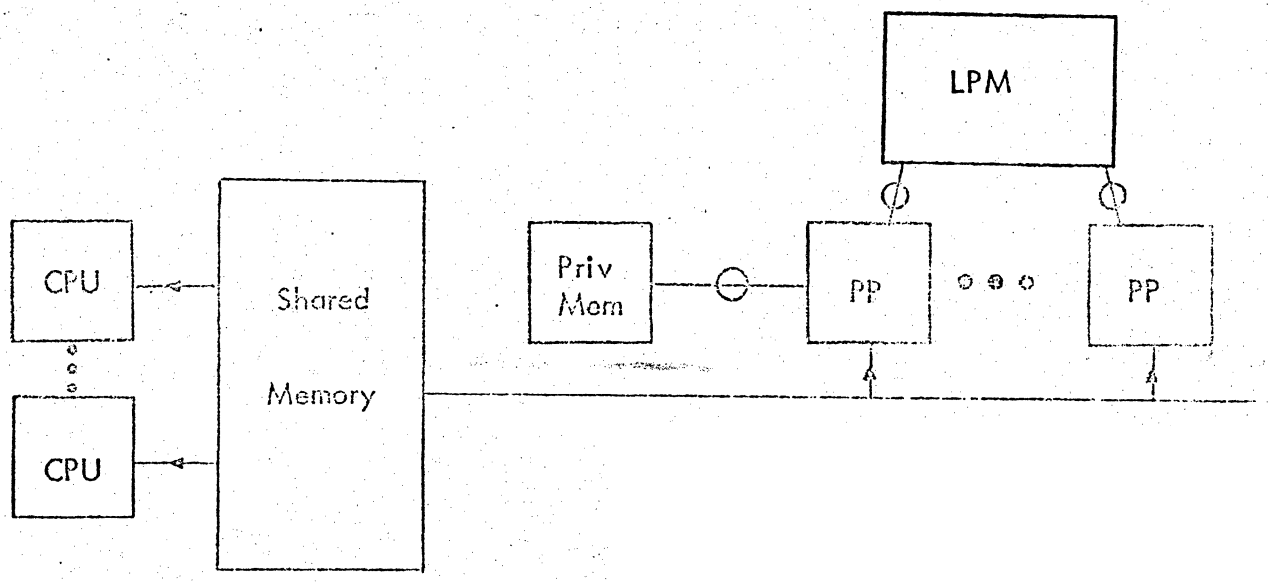
b) peripheral processor firmware

c) peripheral processor hardware

d) the central processor upon demand

or some combination thereof.

A large part of the operating system should, and for a system of the expected magnitude probably must, be basically user tasks. In fact, only that part of the operating system which CDC has called EXEC or NCR has called KERNEL or NUCLEUS will not "look" like a user tasks. It is desirable to allow operating system tasks to have all of the advantages and receive all the protection of user tasks. Virtual memory is certainly one of these advantages. Since the operating system will be executing primarily in the peripheral processor and addressing to a large degree peripheral processor memory, some virtual memory scheme is necessary in addressing peripheral processor memory. The diagram below shows the virtual memory accesses required:



▲ Accesses to shared (main) memory

○ Accesses to private (MSU) memory

It was generally agreed by the operating system task force that any code written (independent of the language in which it is written) for a virtual memory system should and will be different from the same program written for a non-virtual memory system. This implies that if virtual memory is not available on the peripheral processor, then programs will have to be rewritten (not just recompiled) if the decision that it functions on the PP or CP changes.

To allow reasonable migration of tasks between CP and PP, the PP virtual memory mechanism must be, if not identical, a proper subset of the CP mechanism. The paging mechanism will run in both CP and PP units. The acquiring and releasing of pages when there are pages available will run from the processor requiring the function while page swapping will probably best be serviced by the peripheral processor. The key to a good paging mechanism is a good method of working set management. This could be accomplished by:

1. User aids (Pascal compiler, etc.)
2. Segment usage (entering/exiting)
3. Hardware aids (page modification)

Structure of the Operating System

- All dynamic context of the task will reside in shared memory.
- All task control tables will reside in shared memory.
- Shared memory will be addressed virtually by a process.
- Multiple address spaces in shared memory will be on a segment basis.
- Segments in shared memory will be paged.
- Protection in shared memory of a process will be on a segment basis.

- The smallest unit of sharing by a process will be on a segment basis.
- Large portions of the operating system will operate under virtual memory.
- Processes must be able to dynamically allocate/deallocate segments.
- While a process is in execution, a method of locking pages/segments must be available.

Scale and Operating Environment

The major impact of scale on memory management comes at the point of inclusion or non-inclusion of the Cyber 80 memory in the system. Some part of memory management is going to be concerned only with that memory. It will have to be designed in such a way that it can be conveniently included or not in the system.

The addition of private memory on central processors will also have some impact on memory management. This impact will be dependent upon the type of memories added.

Differing operating environments will necessitate differing scheduling and allocation mechanisms as described in that section of this paper.

BLOCK I/O

Block I/O will provide the device dependent interface between record I/O and device I/O. Block I/O functions include managing I/O buffer space, scheduling I/O requests to the appropriate device driver waiting for I/O request completion and informing the requestor of the outcome of the request. Logical error recovery will be performed by block I/O.

Block I/O must be structured such that they can be adapted to the operational enrivonment:

1. Over the spectrum of memory configurations, one memory management technique most probably will not lend itself to optimizing memory for all increments of the spectrum. Hence, the structure of block I/O must be adaptable enough to include the appropriate memory management techniques.
2. Likewise, the required device optimization varies with operational environment and the structure must accommodate these variances.

RECORD I/O

Record I/O will provide the user with access to files at the logical record level. Record I/O groups records into blocks and interfaces with block I/O to read or write the data. Record I/O is device independent except for a set of device dependent rules which determine the conditions under which block I/O should be interfaced. Record compression/ decompression and collecting is provided by record I/O and is transparent to users accessing a file at the record I/O level. Automatic buffering of data blocks may be provided where applicable (sequential files). The virtual memory system may impact buffering of data blocks such that buffering may be impractical (excessive binding of pages or actually re-reading a buffered block).

DEVICE DRIVERS

A device driver will perform the function of device I/O. A device driver is a task with no differences from any other task other than some defined privileges. These privileges would include access to a device and use of I/O instructions. Further, a device driver is aware of only one request at a time; it performs the physical I/O transfer, and is specific to a device type or class. The device driver will inform its requestor or caller the outcome (hardware status or formatted status) of the I/O request. A device driver will perform hardware error recovery.

A device driver as an entity will exist as one procedure for each device type or class. There is to be one unique context for each I/O request. There is to be at most one active I/O task (device driver) active for each data path.

A device driver interface with its caller will be via shared address space. The shared address space will include the data area, request information (address, count, etc.) and outcome information. All pages of the shared address space to/from which I/O is being done must be bound for the duration of the I/O.

A device driver should be structured in such a way that only the portion necessary to perform I/O is resident in a limited memory system. Those parts which are exception conditions or requests (error recovery procedures, etc.) would be non-resident. This structure must be adaptable to the degree that portions can be included/excluded across the spectrum of memory configurations. The adaptability aspect implies a design and programming discipline which must be followed at implementation time. A device driver, so structured, should be able to span the range of a common operating system. The above does not contradict our conclusion that only one version of a specific device driver is necessary for all of our logical configurations and variations of operating environments. Device drivers may be the only module with this characteristic.

SCHEDULING AND ALLOCATION

Scheduling and Allocation are related topics having to do with the assignment of resource to a task. In particular, we consider scheduling to be making guesses about the order in which allocation is to be performed. Allocation involves actually attaching resource to a task.

There must be some interconnection among the various allocators in order to best use the resources of the system. In order to accomplish this a single scheduling list is proposed. This is a list of tasks in the system in the order in which they are expected to be serviced by a processor. Processor servicing is used because a processor must be allocated to a task before any use of a resource or deallocation can be performed. All resource allocators should use this list to help them decide how to allocate resources among competing tasks. This permits coordinated allocation of resources since the status of tasks with regard to resource allocations is summarized in the list. For example, if memory is a critical resource in the system then allocation should in general be to the task using the most memory.

A sample list might consist of three parts:

1. Expedited tasks - these tasks will be allocated a processor in their current order - no re-ordering will be done.
2. Assigned (Ready) tasks - these tasks can use a processor but their order may be changed.
3. Unassigned (Not Ready) tasks - these tasks cannot presently use a processor.

Beyond these three parts the list is ordered by other criteria including, but not limited to, possession of critical resources, time in system (aging) and externally supplied priorities. The nature and effect of these criteria is certain to change from installation to installation. It is therefore necessary for both NCR and CDC to provide for variation of scheduling algorithms. A single scheduling facility to manipulate the above list under control of an externally provided algorithm must be provided by both companies.

Such a scheduling facility could be table driven where the scheduling algorithm is established as entries in a table and the scheduler is a procedure or set of procedures which manipulate tasks according to these entries. The possible task states are represented as rows of a matrix. The columns of the matrix contain parameters used for ordering the tasks in each state and information about moving tasks from that state to another. Tasks fit one of the states at creation and are moved from state to state according to the matrix as events occur.

The intent of the allocation and scheduling algorithms differs in the different operational environments. In interactive systems, the important objective is usually response time. In batch system, total throughput is most important. Transaction systems usually attempt to maximize throughput without causing any response time to exceed some maximum. These objectives are achieved almost totally through the scheduling and allocation algorithms used. The combination of one or more operational environments is also heavily dependent upon a good scheduling and allocation algorithm.

Since the algorithms are based on operational environment, both NCR and CDC need to provide standard algorithms or tables for use or modification by their customers. In many cases, a single algorithm (table) will apply to both NCR and CDC customers.

More sophisticated scheduling algorithms require more space since the scheduling matrix requires more rows (states) and more parameters per row. The use of such a matrix is likely to consume more processor power. On smaller systems, it may not be cost-effective to use very sophisticated scheduling algorithms. This means that scheduling algorithms provided for the top of the line may not be useful at the bottom. Also, the scheduling algorithms provided for the bottom of the line may not be able to get optimum performance out of the top of the line.

The only major difference of scale in resource allocation is in the number of resources. It is possible that the additional resources in a large system would complicate the scheduling and allocation algorithms (in particular the processor scheduling algorithm since it must handle all interdependencies). Additional complexity is more likely due to the number of types of resources, not the number of resources alone. In the range of scales being considered the number of types of resources is likely to be fairly constant.

Frequent allocation and de-allocation of resources occurs in interactive systems. This contrasts to batch systems where resources are usually allocated on a job-by-job basis. Transaction systems are usually built to run with pre-allocated resources (except for processors). This allocation is changed only in case of error or other extraordinary conditions. The less frequently allocation is performed the less sensitive the system is to improper design of the algorithms. This means that sophisticated allocation algorithms, sensitive to details of the environment are required in interactive systems but simpler algorithms are adequate for batch. Transaction systems require a sophisticated processor scheduler only.

JOB CONTROL LANGUAGE PROCESSOR

The Job Control Language (JCL) is the language used to communicate with the system and direct it to do work. The JCL processor is the system routine that interprets and processes the JCL statements.

The system or job control language has many faces, and may, by some definitions, be a number of languages. It is the language(s) used by operators and users to communicate with the system and direct its operation. The language(s) must support: remote/main console operators, remote terminal operators, remote/local batch users, and interactive users.

The confusion as to how many languages there are arises from recognition of the overlapping requirements among users. For example, all users will want to manipulate job and file parameters, both the console operator and interactive users will want to function in an interactive mode, etc.

The capabilities and general approach developed by CDC's Canadian Development Division, (CDD), and described in their Command Language Description document satisfies the needs of NCR and CDC across the range of the product line and in the three major operating environments; i.e., batch, interactive and transaction. The following are some of the characteristics and features:

--The JCL is terminal oriented which subsets effectively for all operational environments.

--The JCL should be natural and not require programming experience to use it effectively.

--The user is not restricted to rigid formats.

--The JCL must provide the capability to be redefined at the user level and allow the user to define his own commands; i.e., it must be extensible.

--Inter-job communication will be provided through the JCL using global variables.

--Jobs may initiate other jobs. This allows job sequencing.

--Special features and facilities may be provided to privileged users; e.g., operators and operations managers.

--The JCL should provide 'help' to the user.

--The JCL Processor must provide two modes of operation. One provides for immediate interpretation and processing of JCL statements. The other allows for collecting JCL statements without interpretation.

The risks in a common JCL processor are minimal. Much of the work has already been done by CDD in defining the language and the processor.

DEVICE ALLOCATION

Device Allocation can take place at any time when a task decides it requires the use of a device. In some cases (at installation option) all allocation will take place at the start of a Job.

Usually devices are allocated to system tasks which will operate the devices as requested by several users. It is also possible (at installation option) for devices to be allocated directly to users. This option will be used mainly in batch systems. Such allocation is usually discouraged in interactive systems and not provided in transaction systems. Smaller systems also discourage allocation of devices to users because of the smaller number of total devices available.

Allocation of a removable volume device (e.g., disc, magnetic tape) implies the mounting of the proper volume on that device. In some cases, allocation may depend upon where the operator mounts the volume. Allocation of devices to users implies the existence of private volumes for mounting on those devices.

Communications devices are treated exactly as are other devices in that they are normally allocated to system tasks (e.g., Transaction Distributor) but can be allocated to users, if necessary. Dial-up terminals are treated as removable volumes with the currently dialed in terminal being "mounted."

One algorithm for I/O device allocation is not sufficient for all systems over the range and all operational environments. Multiple routines may have to be provided to satisfy various users.

BINDING

Binding is done at times defined by operations performed on the program such as compiling, linking (entirely a binding function), loading, and execution (dynamic binding). The later binding times provide more flexibility in operating environment at the expense of redundant and wasteful binding operations. Very dynamic systems such as most interactive systems require later binding times than constant systems such as commercial batch systems. Transaction systems usually involve frequent but well-controlled bindings. Scientific batch systems do not usually require late binding times but their changing nature usually means that the overhead involved in later binding is not harmful. In order to reduce overhead costs, operating systems code should be dynamically bound only when necessary. Virtual Storage systems usually provide facilities to aid dynamic binding but improper use of these facilities can cause severe performance degradation. There is no reason why systems cannot be built to both avoid the severe degradation associated with dynamic binding and provide facilities for earlier binding when desired.

Static Binding

How you can tell

- It's normally a one time affair between the operating system and the process.
- The operating system normally links all external references of a process prior to execution.
- Depending on the versatility of the operating system pre-allocation and and binding of real memory may be somewhat dynamic.
- Both CPU and PPU could partake in the affair.
- Linear space requirements are normally known prior to execution.

Dynamic Binding

How you can tell

- Its an affair with the operating system that can go on for the life of the process.
- Operating system links and loads only that procedure that was externally referenced.
- The external references of the procedure called will not be resolved.
- There is normally no real memory pre-allocated to the procedure of a process.
- Both the CPU and PPU could partake in the affair.

Control Data views dynamic linking and loading as an integral part of the Virtual Memory mechanism (at least as of 4/10/73). It is assumed that the operating system will use dynamic loading and linking in a large portion of its processors. NCR and CDC's use of dynamic loading and linking is questionable for first release for the operating system.

OPERATOR CONTROL

Operator Control is that part of the system that provides the computer operator with information about the status of the system and the jobs being processed, and the ability to influence or control the behavior of the system within certain limits.

Operator Control is required by all systems over the range. The systems cannot operate without operator action. Operator action should be directed primarily by the system. Operator communication and control occurs through use of the JCL. The operator may have special facilities and features provided for privileged users. A common Operator Control facility can be provided by including parameters that can be set by installation option.

Both CDC and NCR believe that the operator communicates with the system through the Job Control Language.

A common Operator Control facility can be provided with minimal risk. Since the facilities provided in this area are through JCL, there are few, if any, problems and issues that need resolution outside of the JCL area.

DATA MANAGEMENT

The "operating system" will to the best of our knowledge provide to the data management system all interfaces, task/process interfaces and procedure interfaces, which are available in the system thus fulfilling the requirements (requests) of data management.

OPERATING SYSTEM TASK FORCE

APPENDICES

APPENDIX A - DEFINITIONS

ALLOCATION - The act of picking a request for a resource and honoring that request.

DEVICE DRIVER - An operating system task which performs a physical I/O transfer from a specific device.

DEVICE HANDLER - The portion of a device driver which is not sensitive to scheduling criteria.

DEVICE TYPE - A class of devices which are similar enough in operation that they can be handled by a single device driver procedure.

DYNAMIC CONTEXT - The data for a process which is changed during execution.

EVENT - Anything which causes a list of requests for service to be altered.

EXEC - The non-task procedures necessary for managing the interaction between processes, the state changes of a process, the creation of processes and inactive tasks, and the selection of processes for a processor.

JOB - The immediate user of a computing system's view of a piece of work. From the operating system it is a rather arbitrary set of tasks that are performed as a result of information between a users LOGON and LOGOFF statements.

JOB CONTROL LANGUAGE - The language used by an operator or other immediate user of a computing system to request services and direct the operation of the system.

MEMORY, LOCALLY PUBLIC - Memory accessible by multiple, but not necessarily all, physical processors in a computing system.

MEMORY, PRIVATE - Memory accessible by only one physical processor.

MEMORY, SHARED - Memory accessible by all physical processors in a computing system.

PROCESS - An active instance of a task. A process is the only entity applied to a processor by EXEC to get work done.

A process normally consists of:

1. A dynamic context unique to this process, and
2. A set of procedures shared with other processes that have been created from the same inactive task.

A process has its own address space which may have certain segments shared with other processes.

READY STATE - The state of a process which is able to use a processor as soon as it can be allocated.

SCHEDULING - The act of ordering lists of requests for service at the time requests are made.

TASK - 1. The smallest unit of work recognized by the system.

2. The object in the system to which resources are allocated.

These two definitions are not contradictory. Either one can be used as the primary definition and the others will follow.

TRANSACTION DISTRIBUTOR - A service function for transaction systems. It inputs transactions from files and distributes them to the appropriate processing tasks. The entire operation is controlled through a transaction distribution language.

WAIT STATE - The state of a process which cannot use a processor until some event occurs.

The following pages show the object size of SCOPE 3.4 modules grouped somewhat by function. All numbers are octal 60-bit words. The "Proc" column identifies the processor (P or C) used by the module.

Size	Proc	Name	Description
Miscellaneous			
1346	P	MTR	System monitor
700	C	CPMTR	System monitor
4000	C		ECS Driver
175	C	RECOVR	
1314	P	TDS	Terminate deadstart
7757	subtotal		
Display Routines			
1300	P	DSD	Dynamic system display
350	P	8X9	6000 resident overlay
1420	P	8XA-8XR	18 DSD console command overlays
2237	P	8DA-8DZ	18 DSD display driver overlays
345	P	8Y9	7000 station routine
1156	P	8EA-8EZ	17 7000 station routines
52	P	IDL	
52	P	LDL	Overlay loader and dayfile message processor for DSD
52	P	LDM	Device queue manager
77	P	LGM	Display good morning
571	P	LMH	DSD Mother Hairpiece
1275	P	DIS	Routine to process control cards from console
11633	subtotal		
Permanent File Manager			
1211	C	PFCCP	Permanent file control card processor
551	P	LPC	Drop permanent file mass storage
1313	P	PFC	Permanent file catalog
677	P	LPF	Permanent file queue manager
1264	P	LFC	Catalog function part II
1312	P	PFA	Permanent file attach
17	P	2FA	Utility I/L processor PFA segment
1302	P	LPF	Load permanent files
1005	P	PFP	Permanent file purge
1270	P	PFE	Permanent file extend
1306	P	PFR	Permanent file rename
471	P	LPD	PFA Delay Overlay
422	P	EPF	Send permanent file audit information
732	P	PFS	Position function storage allocation
35	P	PRM	Permission code processor for perk

Size	Proc	Name	Description
Permanent File Utilities			
24336	C	AUDIT	Audit permanent files
17237	C	PFDUMP	Dump PFD and RBTC
11566	C	DUMPF	Dump permanent files to tape
21042	C	LOADPF	Load permanent files
5003	C	TRANSPF	Transfer permanent files
1066	P	DPF	Dump permanent files
1051	P	TPF	Transfer permanent files
223	P	LDU	DUMPF initialization
1001	P	TPT	Transfer permanent file tables
465	P	PFD	Permanent file dump
107500 subtotal			
		MASS STORAGE I/O	
Allocatable Devices			
350	C	SPM	Stack processor monitor
460	P	3D0	Allocatable device file open
6	P	1S5	Interface between stack processor mgr. and 1SP/3D0
327	P	1RN	Release record block chain
370	P	4ES	Enter stack requests
617	P	1SP	Stack processor main program
76	P	3SP	Driver overlay for 6603-I
141	P	3SQ	Driver overlay for 6638
124	P	3ST	Driver overlay for 6603-II
124	P	3SR	Driver overlay for 865
122	P	3SU	Driver overlay for 814
117	P	3SV	Driver overlay for 821
126	P	3SW	Driver overlay for 841
121	P	3SS	Driver overlay for 854
1127 subtotal			
1075	P	1EP	Stack processor main program-ECS I/O buffering
102	P	3EP	ECS Driver overlay for 6603-I
145	P	3EQ	ECS Driver overlay for 6638
127	P	3ET	ECS Driver overlay for 6603-II
133	P	3ER	ECS Driver overlay for 865
130	P	3EU	ECS Driver overlay for 814
126	P	3EV	ECS Driver overlay for 821
131	P	3EW	ECS Driver overlay for 841
126	P	3ES	ECS Driver overlay for 854
1266 subtotal			
262	P	1SX	Process stack processor errors
337	P	CEM	Central error manager
102	P	7EC	Generate ECS buffers
3340 subtotal			

Size	Proc	Name	Description
Non-allocatable Devices			
167	P	5DA	Family disk pack label processor
402	P	1PK	Sequential pack close
502	P	3PK	Sequential pack initialization
1015	P	1DA	Disk pack label routine
264	P	EKG	Family pack end of job processor
Tape Labels			
1260	C	LABEL	
3631	P		14 Overlays

UNIT RECORD I/O DRIVERS

Size	Proc.	Name	Description
1067	P	1IR	JANUS Main Program
1160	P	1IS	JANUS Main Program
1061	P	2IS	JANUS Routine
521	P	1IQ	Initiate JANUS Control Point
121	P	1IU	JANUS Backspace Print Name
343	P	2LP	On-Line Printer Driver
257	P	2FC	On-Line Card Punch Driver
363	P	2RC	On-Line Card Reader Driver
11	P	1PL	Plotter Program (Dummy)

TAPE SCHEDULING

223	P	VSN	Visual Serial Number
302	P	1TS	Tape Sampler
172	P	2TA	Tape Assignment Overlay

TAPE DRIVERS

427	P	1MF	Multifile Position Routine for ANSI Labelled Tapes
1275	P	1MT	Driver for Long Record Stranger (L) Tapes for 7-Track Tapes
542	P	1RT	SCOPE Tape Read Driver
540	P	1RS	Stranger (S) Tape Read Driver for 7-Track Tapes
565	P	1NR	9-Track S-Format Tape Read Driver
500	P	1h9	9-Track SCOPE Format Tape Write Driver
606	P	1WI	SCOPE Tape Write Driver for 7-Track Tapes
411	P	1TF	Forward Skip Routine for Tape
522	P	1WS	Stranger (S) Tape Write Driver
406	P	1NW	9-Track S-Format Tape Write Driver
632	P	2TB	Backward Skip Routine for Tape
507	P	1R9	9-Track SCOPE Format Tape Read Driver
257	P	6WM	Dayfile Messages for I/O Requests
155	P	7W1	Overlay to SWK
34	P	7W2	Overlay to SWM
10354	P		19 Recovery overlays
52	P	6LM	Load Field Name Messages
223	P	6LC	Load Conversion Table into MMTC
101	P	7T1	ANSI/DISPLAY code conversion Table for MMTC Memory
101	P	7T2	EBCOIC/Display Code Conversion Table for MMTC Memory
310	P	8T3	Segment for Loading of MMTC Conversion Memory

JOB PROCESSING

Size	Proc.	Name	Description
217	P	ACE	Control Card Reader
1160	P	1AJ	Advance Jobs
1076	P	1EJ	End of Job Processor
176	P	2TJ	Translate Job Card
510	P	1IB	Initiate a Batch Job
652	P	1SI	Swapin or Rollin a Job
131	P	6SI	Overlay to Process Parity Error for 1SI
621	P	1SO	Swapout or Rollout a Job
1200	C		Integrated Scheduler
I/O Interface			
304	P	CIO	Circular Input/Output
113	P	MSG	Add Message to Dayfile
306	C	CFC	
262	C	IORANDM	
250	C	IO	
60	C	SYS.RM	
1537	C	FILE	
6000	C		Sequential records
600	C		Word addressable
13000	C		Indexed sequential
3000	C		SAC
6000	C		8-bit

Size	Proc.	Name	Description
CHECKPOINT/RESTART			
62	C	CHEKPT	
1512	C	RESTART	
1105	P	CKP	Tape Checkpoint
43	P	CYL	Reset FNT for Restart
316	P	RST	Restore Control Point Area for Restart
456	P	LRC	Reload Core for Restart
DUMP/RESTORE I/O QUEUES			
500	C	XXXRESQ	
500	C	XXXDMPQ	
125	P	XRQ	Restore Queue
256	P	XDQ	Dump Queue
MISCELLANEOUS			
1266	C	TRAP	
3613	C	TRAPPER	
107	C	SETCORE	
137	P	L00	Load Octal Corrections
226	P	MEM	Process Memory Function
314	P	LLT	Load Jobs from Tape
317	P	LDF	Dump Dayfile
140	P	LBT	Tape/Disk Blank Labels
162	P	LTD	Dump Output File to Tape
13320	C	DMFECS	
220	P	RPV	Process Reprieve Function
203	P	STS	Status Routine
1066	P	DMP	Dump CM

UTILITIES

LOADER

Size	Proc.	Name	Description
10255	C	SEGBILD	
612	C	SEGRES	
6400	C	LOAD	
457	C	LOADC	
1634	C	LOADM	
6566	C	LOADU	
520	C	LOADUC	
2070	C	LOADUM	
257	C	UOLOAD	
65	C	LIBRARY	
2551	C	LOAD0	
1405	C	LOAD01	
522	C	LOAD02	
1237	C	LOAD03	
306	P	LDL	Loader Utility
345	P	LDV	Absolute Overlay Loader
1031	P	LDW	Absolute Overlay Loader

EDITLIB

4074	C	EDITLIB	
11430	C	EDITSYS	
5723	C	EDITUSR	
615	P	MDI	Move System Directory (EDITLIB use
2	P	6MD	Dummy EDITLIB Overlay
61	P	SRB	EDITLIB Routine to Complete Disk Address of Record

UPDATE

13362	C	UPDATE	

JOB DEPENDENCY

53		TRANSR	
17		TRANSF	
115	P	JOB	Process Job Dependency

FILE UTILITIES

Size	Proc.	Name	Description
111	C	BKSP	
6325	C	COMPARE	
156	C	COPY	
603	C	COPYBF	
604	C	COPYCF	
603	C	COPYCR	
602	C	COPYBR	
5320	C	COPYN	
215	C	COPYSBF	
3337	C	COPYL	
2444	C	COPYECD	
2172	C	COPYXS	
65	C	DISPOSE	
310	P	DSP	Process Dispose Function
125	C	SKIPB	
125	C	SKIPF	
37	C	REWIND	
37	C	UNLOAD	
37	C	RETURN	
240	P	1CL	File Close Routine
345	P	2TC	Close Tape File
621	P	1T0	Tape Open Routine
72	P	6T0	Overlay to 1T0
165	P	1OF	Open File Routine {All Files}
4	P	OPE	Open Routine {Dummy}
4	P	CLO	File Close Routine {Dummy}
546	C	REQUEST	
1163	P	REQ	Request Card Processor
204	P	3R0	Req Overlay Containing [illegible]
300	P	1RQ	Req Overlay
261	C	LISTMP	

Size	Proc	Name	Description
INTERCOM			
20036	C	LOGIN	
51	C	CONNECT	
15741	C	SITUATE	
22	C	DISCCNT	
14365	C	BATCH	
11316	C	@	
31412	C	PAGE	
17711	C	SEND	
11517	C	LOGOUT	
21560	C	CONVERT	
21430	C	BRESS@	
30647	C	TEACH	
7546	C	FILES	
13501	C	STORE	
10526	C	DISCARD	
13266	C	FETCH	
14604	C	XEG	
46140	C	PASSWRD	
7364	C	ASSETS	
6522	C	READ	
20614	C	ERRORS	
453	C	TESTLP	
36141	C	EDITOR	
7	P	RNE	Check if INTERCOM control point
162	P	CON	Connect file name to TTY
1302	P	1ZF	Multiplexor driver
1256	P	8ZF	Multiplexor driver
125	P	9ZF	Multiplexor driver
1027	P	1XP	High speed EXPORT processor
506	P	2XP	Process special directives
253	P	3XP	Process output data stream
134	P	5XP	Output banner to terminal
75	P	6XP	Output laced card
235	P	4XP	Process input data stream
2	P	1XG	Graphics input/output processor
24	P	7XP	End processing
771	P	1LX	LCC EXPORT processor
325	P	3LX	Overlay to 1LX
542	P	4LX	Overlay to 1LX
1140	P	1WE	INTERCOM V4 wide band driver
751	P	0ZZ	INTERCOM LCC driver initiator
1312	P	1ZZ	INTERCOM LCC driver
45	P	9ZZ	INTERCOM LCC driver
344	P	1CI	Common communications interface
304	P	3CU	Assign new user table
404	P	3C1	User table processor
537	P	3CT	User table processor

Size	Proc	Name	Description
INTERCOM (cont.)			
766	P	3CX	Command processor for 1CI
173	P	3CF	Clean up phase of 1CI
1145	P	1I1	Starts INTERCOM at control point
365	P	3TT	READ/WRITE for remote terminal
134	P	3T1	READ segment of 3TT
371	P	3T2	WRITE segment of 3TT
1135	P	1BR	INTERCOM VR4 buffer manager
751	P	1QP	MUJ processor
11	P	1QM	1QP overlay
412	P	1PT	INTERCOM VR4 low speed EXPORT processor
572	P	8PT	Input file transmission
506	P	9PT	Output file transmission
15	P	1PJ	Process job card
67	P	1I0	Send dayfile message to INTERCOM terminal
426	P	1IM	Sends messages to terminals from PP routines
371	P	1DS	H Display generator for INTERCOM VR4
232	P	T76	INTERCOM 4.1 7000 display generator
772	P	TBL	INTERCOM VR4 table transmitter
306	P	FNT	Modifies VR4 FNT entry for batch and DROPO
214	P	IUP	Initiate user program
45	P	IAP	Initiate another program
340	P	MES	Message transfer routine
125	P	2ME	Overlay to MES
150	P	3ME	Overoay to MES
32	P	MUJ	Multi-user job initialization
37	P	MAC	MUJ accounting
1071	P	FAD	File attach/detach for MUJ
6	P	GBJ	Begins graphics mode
6	P	GEJ	Ends graphics mode
2	P	1GJ	Update IGS queue when IGS [illegible] stream defined
2	P	2GJ	Format SCOPE error messages for 274 IGS
2	P	1GR	274 IGS recovery

Table 1. Integrated Product Line System Models

Type of System	Monthly Rental	Scientific System Perform.	BDP System Perform.	Compe-tition	No. of CPUs	Type of CPU	Amount of Real CPU Memory	Amount of Virtual CPU Mem.	No. of 6150 I/O PPE	Amt. of I/O PME (KB)
SUPERSCALE	350	25x8600	3x198	3x198	3	8600(P4)	1-16 MB	8000 MB	5-10	100-350
(P4)	275	20x8600	1.2x198MP	198MP	2	8600(P4)	1-16 MB	8000 MB	4-8	80-280
	*160	11x8600	1.1x198	198	1	8600(P4)	1-16 MB	8000 MB	3-5	30-180
VERY LARGE	200	8x8600	1.3x198	198	3	P3	1-16 MB	8000 MB	4-6	80-200
(P3)	150	5.5x8600	1.2x168MP	168 MP	2	P3	1-8 MB	8000 MB	3-5	80-160
	* 80	3x8600	1.1x168	168	1	P3	1-8 MB	8000 MB	2-4	20-100
LARGE	100	2.5x8600	1.3x168	168	3	P2	1-8 MB	8000 MB	3-5	50-180
(P2)	75	1.8x8600	1.2x158MP	158 MP	2	P2	1-8 MB	8000 MB	2-4	30-125
	* 40	1x8600	1.1x158	158	1	P2	1-8 MB	8000 MB	1-3	15-75
MEDIUM	50	1x8600	1.2x158	158	4	6150X**	0.5-4 MB	16 MB	1-3	10-40
SCIENTIFIC	35	.75x8600	3.5x145	145 MP	3	6150X	0.25-2 MB	16 MB	1-2	10-30
& BUSINESS	* 25	.5x8600	2.5x145	145	2	6150X	0.25-2 MB	16 MB	0-1	0-15
(P1)	15	.3x8600	1.5x145	small 145	1	6150X	0.25-2 MB	16 MB	0-1	0-15
MEDIUM	45	--	1x158	158	3	6150 50ns	0.5-4 MB	16 MB	1-3	10-40
BUSINESS	30	--	2.3x145	145 MP	2	6150 50ns	0.25-2 MB	16 MB	1-2	10-30
(P0)	* 15	--	1.5x145	145	1	6150 50ns	0.25-2 MB	16 MB	0-1	0-15
SMALL	15	--	1.5x135	135	2	6150 90ns	0.25-1 MB	16 MB	0	--
BUSINESS	* 10	--	1.5x125	125	1	6150 90ns	0.25-.5 MB	16 MB	0	--
(P)	5	--	1.5x115	115	1	6150 90ns	0.064-.25 MB	16 MB	0	--

*Target Configurations

** 6150X denotes a 6150 that is yet to be defined and performs 0.3 x 8600 in Scientific CPU performance.

APPENDIX D - DISTRIBUTION OF O.S. MODULES

The O.S. Task Group generated the list of functional elements in the following table as characterizing the bulk of an operating system. The members of the group were then polled as to where in their best judgement the functions should be performed in a system consisting of both PPUs and CPUs using the following definitions.

PP - The function should only be done in the PP

CP - The function should only be done in the CP

Both - The function would be performed in both the CP and PP

Either - Determination as to where the function is to be performed is uncertain

ASSUMPTIONS

1. Hardware can support any distribution of components required.
2. The size and number of processors in a configuration will be determined by the distribution of O.S. components rather than vice versa.

Votes of the Task Group Members

	PP	CP	BOTH	EITHER
Device Drivers	9			
Block I/O	8			1
Record I/O	2	2	5	
Data Management	7	1	1	
Loaders - Static	8			1
- Dynamic	1	3	3	2
Task Management			9	
JCL Processor	7			2
Operator Control	8		1	
Utilities	3		4	2
Shared Memory Mgmt. - Swap	4		1	2
- No Swap			9	
Job Management	5		2	2
Hardware Diagnostics			--- 9 ---	
I/O Device Allocation	7		2	