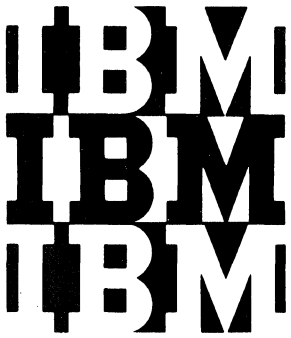


File No. S360-36
Form Y28-6658-0

Program Logic

IBM System/360 Operating System

MVT Control Program Logic Summary

Program Numbers 360S-CI-535

360S-DM-508

This publication introduces the internal logic of the MVT control program of System/360 Operating System. It contains general descriptions of the operating environment of the control program, the initial program loading procedure, and the job management, task management, and data management functions. Detailed descriptions of the implementation of these functions are in the program logic manuals listed in Appendix B.

The MVT configuration of the control program is designed for use with System/360 Models 40, 50, 65, and 75 having 262,144 (256K) bytes or more main storage.

Program Logic Manuals are intended for use by IBM customer engineers involved in program maintenance, and by system programmers involved in altering the program design. Program logic information is not necessary for program operation and use; therefore, distribution of this manual is limited to persons with program maintenance or modification responsibilities.

PREFACE

This publication contains a general description of the internal logic of System/360 Operating System control program (MVT configuration). It is an introduction to the more detailed program logic manuals of the control program.

Since the control program is dependent on the hardware characteristics of System/360, the first section of this publication discusses the operating environment of the control program. The second section describes how the control program is brought into storage and initialized. The last three sections describe the three functional areas of the control program: job management, task management, and data management. A glossary provides definitions of many of the terms used in discussing the control program.

Before using this publication, the reader should be familiar with the contents of:

IBM System/360: Principles of Operation, Form A22-6821

IBM System/360 Operating System: Concepts and Facilities, Form C28-6536

IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646

First Edition (July 1967)

This publication corresponds to Release 12.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for reader's comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

ILLUSTRATIONS

FIGURES

The multiprogramming with a variable number of tasks (MVT) configuration of System/360 Operating System allows the multiprogramming of system functions and up to 15 jobs. The control program in the operating system performs supervisory and service functions that increase the efficiency of job step execution.

OPERATING ENVIRONMENT

At any given time in a multiprogramming environment, main storage can contain programs and data from several independent sources. These programs and data relate to both the tasks currently in the system, and the control program that supervises the whole operation. The characteristics of the control program routines and the storage that they occupy are such that programs of one job step cannot affect the control program, its data, or the programs and data of other job steps.

ROUTINE CHARACTERISTICS

To maintain exclusive control over, and to ensure the integrity of all programs, the control program takes advantage of hardware characteristics of the System/360. One such characteristic -- the two operating states of the CPU -- permits the control program to restrict the use of certain control and I/O instructions (i.e., the privileged instructions). In the operating system, only certain control program routines operate in supervisor state.

ORGANIZATION OF MAIN STORAGE

The organization of main storage for the operating system is based on another hardware characteristic - the protection feature. The 16 protection keys in System/360 allow the control program to protect its own main storage, and the main storage assigned to up to 15 job steps. Main storage in the MVT configuration of the operating system is divided into four areas:

- Fixed area, which contains the supervisory portion of the control program (the nucleus).
- System queue area, which contains data (queues and control blocks) required by the control program.
- Link pack area, which contains programs whose usage can be shared during concurrently-executing job steps.
- Dynamic area, which contains programs and data used during the job steps.

These areas are established when the control program is loaded and initialized.

LOADING AND INITIALIZING THE NUCLEUS

Before the operating system can be used, the nucleus must be loaded and initialized. The Initial Program Loading (IPL) program reads the nucleus into main storage from the system residence volume. The Nucleus Initialization Program (NIP) initializes the nucleus by setting up tables and calculating the address of various routines and the areas of main storage. After IPL and NIP are complete, the control program is ready to supervise processing of user specified jobs.

FUNCTIONS OF THE CONTROL PROGRAM

The control program has three functions: job management, task management, and data management.

JOB MANAGEMENT

Job management is the processing of communications from the programmer and operator to the control program. There are two types of communications: operator commands, which start, modify, and stop the processing of jobs in the system, and job control statements, which define the work being entered into the system. Processing of these commands and statements is referred to as command processing and job processing respectively.

Most of the job management routines operate in the problem state.

TASK MANAGEMENT

Task management is primarily a supervisory function. The routines that perform

task management operate in supervisor state, and are often collectively referred to as the supervisor. The supervisor controls all the tasks in the system. This control includes allocating some system resources to tasks, passing CPU control to the proper routine when a control program service is requested for a task, determining the priority of performance among the tasks, and passing control to routines of the tasks.

DATA MANAGEMENT

Data management routines perform operations associated with input/output devices. This includes allocating of space on direct-access volumes, storing, naming, and cataloging of data sets, and scheduling of I/O operations. The data management routines are primarily service routines. All, except the access method routines, run in supervisor state.

The control program has characteristics independent of the functions (i.e., job management, task management, and data management) that it performs. These characteristics affect both the control program routines, and the way these routines use main storage.

The control program has routines that operate in each of the two operating states, supervisor state and problem state. Therefore the use of certain control and I/O instructions is restricted to certain routines.

Organization and assignment of main storage in the operating system is based on the protection feature of System/360. The protection feature allows sections of main storage to be reserved for use only by certain routines. The operating states and the protection feature are described in the publication IBM System/360 Principles of Operation.

ROUTINES IN SUPERVISOR STATE

Only certain routines of the control program operate in supervisor state. These routines can execute a special group of instructions called privileged instructions, which perform functions such as starting I/O operations, enabling and disabling interruptions, and changing storage protection keys. Thus since the control program has exclusive control over the privileged functions, it can insure the integrity of all the programs and data in a multiprogramming environment.

Control program routines in the supervisor state perform both supervisory and service functions. Some are resident, some are nonresident.

RESIDENT ROUTINES

The resident routines of the control program (the nucleus) are loaded into main storage during the initial program loading (IPL) procedure, and are never overlaid by another part of the operating system. The nucleus contains all the task management routines (except for some nonresident SVC routines), one job management routine, and the I/O supervisor and BLDL routine of data management. The routines in the nucleus operate under program status words (PSWs) with protection keys set to zero. The nucleus is one load module that is a member of the NUCLEUS partitioned data set (SYS1.NUCLEUS).

The routines in the nucleus perform primarily supervisory functions. The service routines of the nucleus are the resident SVC routines.

SVC routines are entered as a result of SVC interruptions, and perform control program services. There are four types of SVC routines:

- Type 1 SVC routines, which are part of the nucleus and are disabled (masked) for all interruptions except machine-check interruptions.
- Type 2 SVC routines which are part of the nucleus but may be enabled (interruptable) for part of their operation.
- Type 3 SVC routines, which are nonresident, may be enabled, and are not larger than 1024 bytes.
- Type 4 SVC routines, which are nonresident, may be enabled, and are larger than 1024 bytes. They are brought into main storage in segments of 1024 bytes or less.

NONRESIDENT ROUTINES

The nonresident control program routines that operate in supervisor state are types 3 and 4 SVC routines, I/O error-handling routines, and part of a system environment recording (SER) routine.

The nonresident SVC routines reside in the SVCLIB partitioned data set (SYS1.SVCLIB), and they operate either from areas defined in the nucleus called SVC transient areas or from the link pack area. Like the resident routines, nonresident SVC routines operate under PSWs with protection keys of zero.

As long as an SVC routine (or a module of an SVC routine) is in a transient area, that copy is used as many times as it is requested.

The I/O error-handling routines reside on SYS1.SVCLIB and operate from the I/O supervisor transient area. They are called

by the I/O supervisor, and either correct an error that occurred during an I/O operation, or post a code for an access method routine.

The nonresident portion of the SER routine resides in the LINKLIB partitioned data set (SYS1.LINKLIB). The SER routine saves critical information about the system when a machine-check interruption occurs.

ROUTINES IN THE PROBLEM STATE

The control program routines that operate in the problem state are job management routines and the access method routines. These routines operate from the dynamic and link pack areas. Job management routines operate under PSWs having protection keys of zero because these routines store data in the nucleus and system queue area. Access method routines operate under the same PSWs as their callers. The job management routines reside on SYS1.LINKLIB, the access method routines on SYS1.SVCLIB.

ORGANIZATION OF MAIN STORAGE

The relative positions of the four areas of main storage, and the program or data that occupies these areas is shown in Figure 1.

FIXED AREA

The fixed area is that part of main storage into which the nucleus is loaded at IPL time. The storage protection keys of the fixed area are zero so that its contents can be modified by the control program only. The fixed area also contains small areas called transient areas into which certain nonresident routines are loaded when needed.

Transient areas are defined in the nucleus, and embedded in the fixed area. There are two types of transient areas: SVC transient areas and the I/O Supervisor transient area; these areas are used by nonresident SVC routines and nonresident I/O error-handling routines, respectively. Like the rest of the routines in the fixed

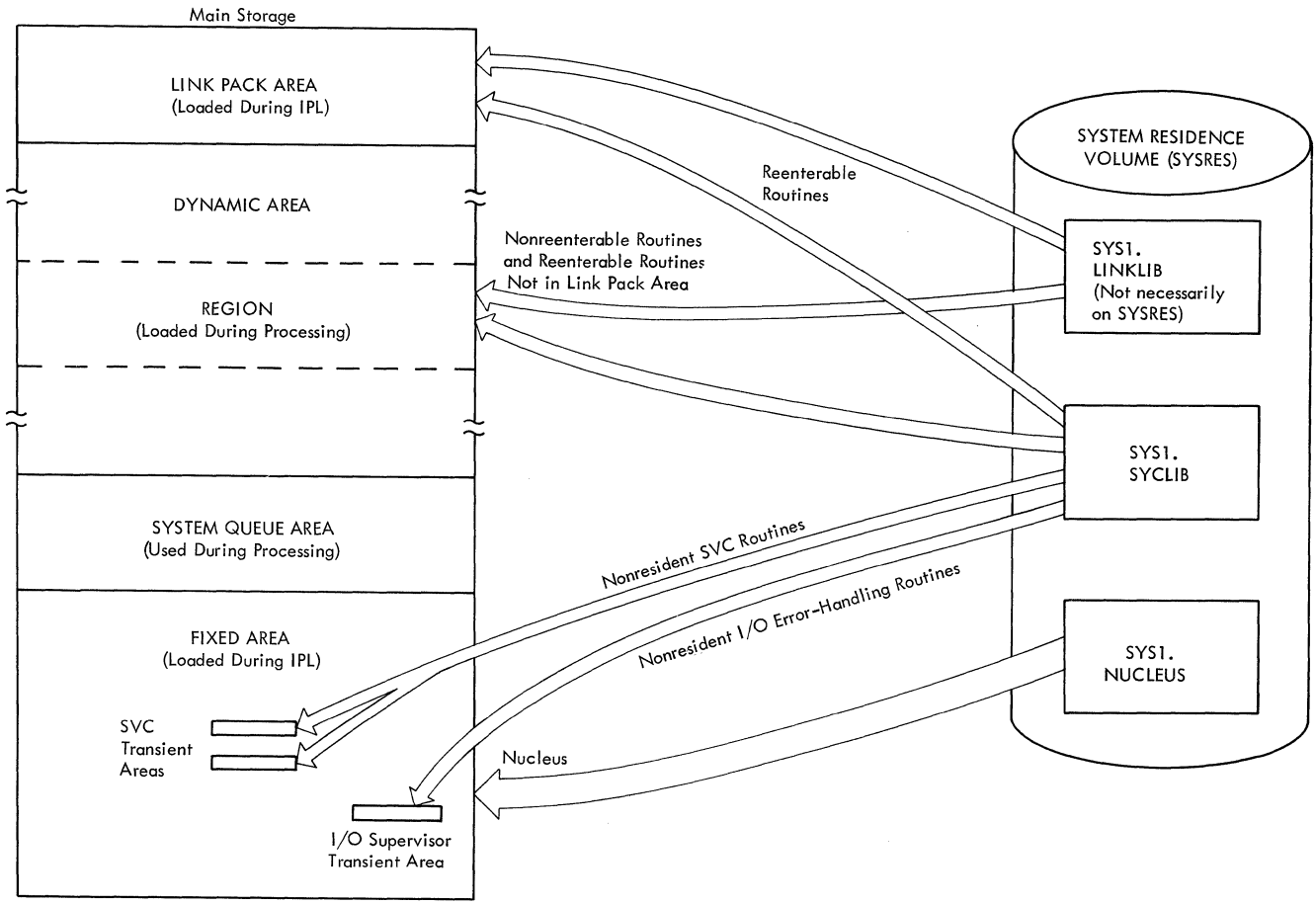


Figure 1. Areas and Contents of Main Storage

area, the transient area routines operate with protection keys of zero. All routines that operate from transient areas reside on SYS1.SVCLIB.

SVC Transient Areas

An SVC transient area is 1024 bytes in length and is reserved for nonresident SVC routines. In the MVT configuration, the number of SVC transient areas is specified at system generation. The minimum number is two. When a nonresident SVC routine (or a module of nonresident SVC routine) is required and is not already in the link pack area or one of the SVC transient areas, the routine or module is read into an available transient area. If no SVC transient area is available, and if none can be made available, the task for which the SVC routine was called is put in the wait state until an area becomes available. A transient area is available when it is empty or when the SVC routine that occupies it has completed its operation.

If no SVC transient area is available for an SVC routine, a transient area currently being used can be appropriated. An area is appropriated when the task requiring the area has a higher priority than all the tasks that are currently using the area. When several transient areas fall into this category, the area appropriated is the one having the lowest highest-priority user. The appropriated transient area is loaded with the SVC routine for the higher priority task. The SVC routine that is overlaid because of this higher-priority requirement is later reloaded so that it can complete its operation.

I/O Supervisor Transient Area

There is one I/O supervisor transient area. It is approximately 400 bytes in length and is reserved for nonresident I/O error-handling routines that are brought into main storage for the I/O supervisor.

SYSTEM QUEUE AREA

The system queue area is adjacent to the fixed area and provides the main storage space required for tables and queues built by the control program. The Nucleus Initialization Program (NIP) sets up the system queue area. Its storage-protection key is zero so that it can be modified by control program routines only. The data in the system queue area indicates the status of all the tasks and many of the resources in the system.

LINK PACK AREA

The link pack area contains reenterable routines that reside on SYS1.LINKLIB and SYS1.SVCLIB, and track addresses of other routines on SYS1.LINKLIB. The routines in the link pack area are used for all the tasks that require them, and need not be loaded into the various regions of main storage. The list of track addresses (the BLDL list) reduces the time required to find the listed routines on SYS1.LINKLIB. These routines are loaded into the region of the task that requires them. Types 3 and 4 SVC routines in the link pack area operate in supervisor state. The others generally operate in the same state as the routines that called them. The organization of the list and routines in the link pack area is shown in Figure 2.

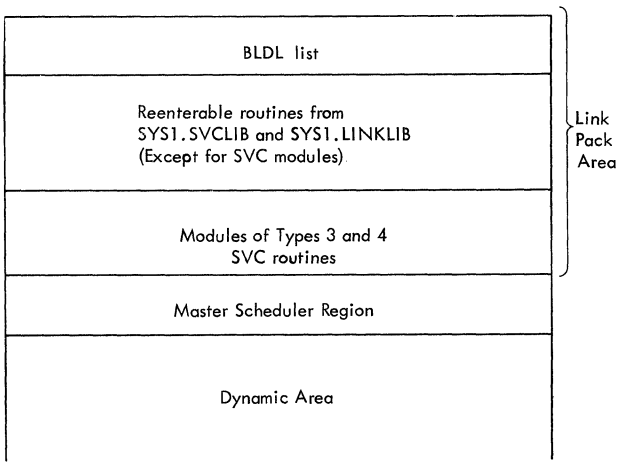


Figure 2. Upper Main Storage After IPL

The user selects the routines that he wants in the link pack area by creating lists of the desired routine names. At IPL time, the NIP program loads the indicated routines starting at the highest part of main storage and working downward. After nucleus initialization, the contents of the link pack area cannot be modified unless the IPL procedure is repeated.

In addition to user-specified routines, several system-specified job management and access method routines also reside in this area.

DYNAMIC AREA

The dynamic area fills the main storage requirements of job steps and system tasks. This area is all the main storage between the link pack area and the system queue area. As jobs steps and system tasks are initiated, storage from the dynamic area is allocated to them in blocks called regions. All storage requested by programs of a

given step or task is assigned from its region.

Regions are assigned from the highest available block of dynamic area storage that is large enough to fill the request. A region is assigned when the step or system task is initiated, and, remains assigned for the life span of the step or task. Any released region becomes part of the free storage in the dynamic area and is again available for allocation to a job step or system task.

The region for the master scheduler task is assigned at IPL time by the Nucleus Initialization Program. This region is adjacent to the link pack area (see Figure 2), and remains assigned for as long as the control program is in storage.

A region assigned to a job step has a protection key value from 1 to 15 associated with it. As 2K blocks of storage within this region are assigned during the step, the protection key of each block is set to the associated value. (Storage assigned to subpool 252 is an exception; its key remains set to zero.) Unassigned blocks of storage within a job step region have their protection keys set to zero.

Regions assigned to system tasks have protection keys of 0. Protection keys of dynamic area storage that is not part of any region are also set to 0.

Before the operating system can be used, the nucleus must be loaded from the system residence volume into main storage, and initialized. These functions are performed by the Initial Program Loading (IPL) program, and the Nucleus Initialization Program (NIP) respectively. These programs are described in the publication IBM System/360 Operating System: Initial Program Loader and Nucleus Initialization Program, Program Logic Manual, Form Y28-6661.

LOADING THE NUCLEUS

Part of the initialization procedure for the system residence volume was the placing of two IPL records at track 0, cylinder 0 preceding the standard volume labels.

To load the nucleus, the user specifies the system residence volume and presses the LOAD button on the console. This action causes the first of the two IPL records to be read into location 0 of main storage and to be given CPU control. This record reads the second IPL record which, in turn, reads the IPL program into main storage.

The IPL program clears and determines the size of main storage, and sets all the storage protection keys to zero. The IPL program then relocates itself into the upper portion of main storage, clears its old location, and loads the nucleus into the lower portion of main storage. After completing the operation, the IPL program passes control to the Nucleus Initialization Program.

INITIALIZING THE NUCLEUS

The Nucleus Initialization Program (NIP) is a control section assembled into the nucleus when the system is generated. NIP initializes tables in the nucleus, determines addresses and storage boundaries of routines and tables in the nucleus, checks and sets the interval timer, defines the boundaries of the system queue area, loads the link pack area, and assigns the region of the master scheduler task. NIP then passes control to a routine of the master scheduler task.

The initializing routines of the master scheduler task initialize the input and output work queues, open the SYS1.LOGREC data set, and execute any automatic commands specified during IPL. After the master scheduler has completed its initializing functions, it is placed in the wait state. If the automatic commands did not result in any tasks that can now be performed, the system is placed in the wait state.

SYSTEM RESTART

If during later processing, a system failure requires that the IPL procedure be repeated, the operator can preserve, at least, part of the contents of the input and output work queues. This preservation of queues allows the system to restart after the IPL procedure is complete without rereading all the jobs that were in the system when the failure occurred. The system restart routines purge the queues of unprocessable entries so that the remaining entries can be properly processed. The queue entries that are preserved are:

- Those representing jobs read into the system but not yet started.
- Those representing system messages and SYSOUT data sets.

The queue entries for jobs being processed when the failure occurred are purged from the queues, and a message is issued to the operator indicating that these jobs must be resubmitted. Any temporary data sets of these jobs are purged, but system messages and SYSOUT data sets that are complete are written.

The system restart routines that modify these queues during IPL are described in the publication IBM System/360 Operating System: MVT Job Management, Program Logic Manual, Form Y28-6660.

Job management routines process communications from the programmer and the operator to the control program. This processing falls into two categories: command processing and job processing.

Command processing is the reading, scheduling, and executing of operator commands issued via either a console device or an input job stream. Job processing is the reading and interpreting of control statements, the initiating of job steps defined in these statements, and the writing of system messages and system output (SYSOUT) data sets from the intermediate volumes on which they were originally placed.

Job management routines perform several tasks to accomplish command processing and job processing. The job management tasks are referred to as system tasks to distinguish them from the user tasks that are performed by processing programs. The job management tasks and the routines that perform them are described in IBM System/360 Operating System: MVT Job Management, Program Logic Manual.

When a system task is created, it is assigned a region of main storage. The routines for this task that are not in the link pack area are loaded into and operate from the region. Regions of system tasks have storage protection keys of zero. Routines that perform system tasks (unlike those that perform user tasks) operate under a PSW with a protection key of zero so that they can write not only in their regions but also in the system queue area.

COMMAND PROCESSING

Processing of commands has three phases:

- Reading the command
- Scheduling the command
- Executing the command

These three phases are performed under various system tasks. Some commands have all three phases performed as part of one task. However, processing of most commands requires several system tasks. Figure 3 shows the relationship between the system tasks and phases of command processing.

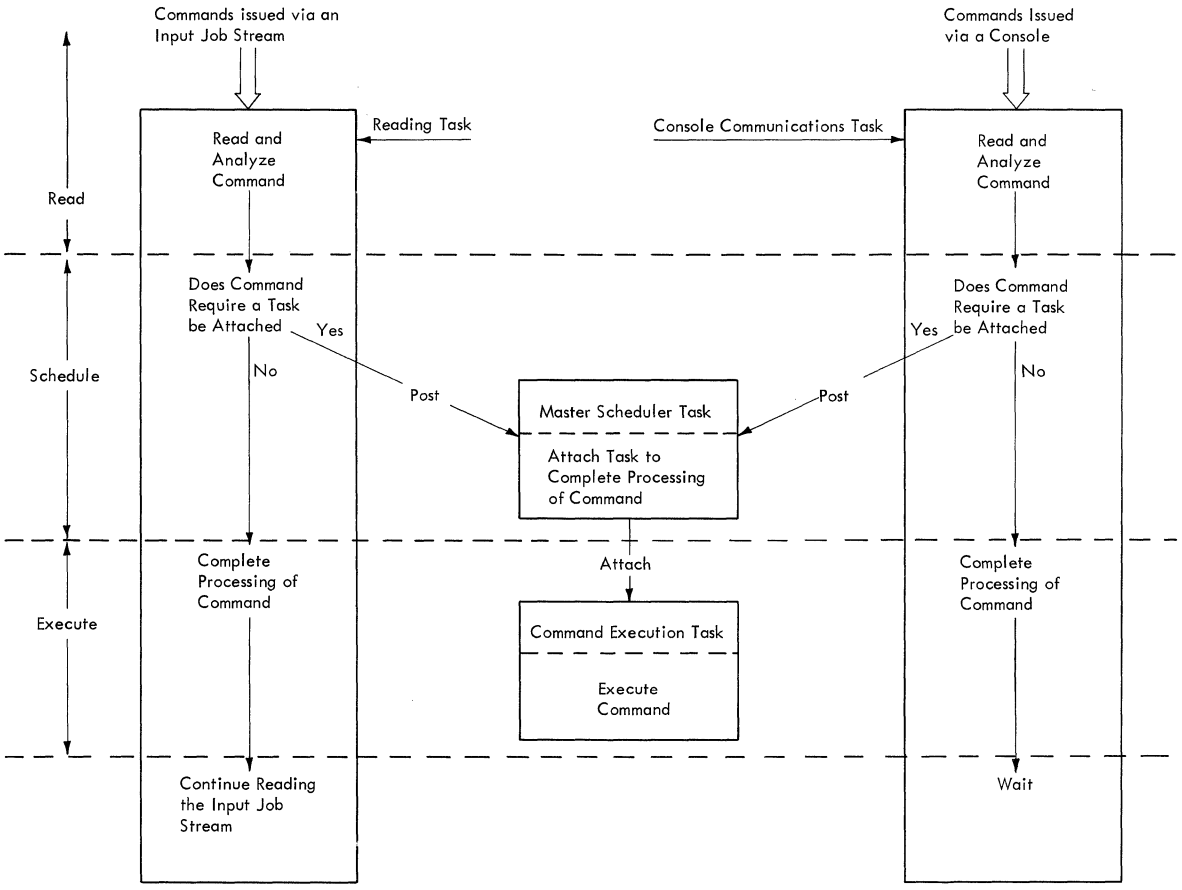


Figure 3. Relationship Between Tasks and Phases of Command Processing

READING THE COMMAND

Operator commands are entered into the system through either a console device or an input job stream. Reading of commands entered via a console device is performed by routines operating under the console communications task; reading of commands entered via an input job stream is performed by routines operating under the reading task associated with that input job stream.

Console Communications Task

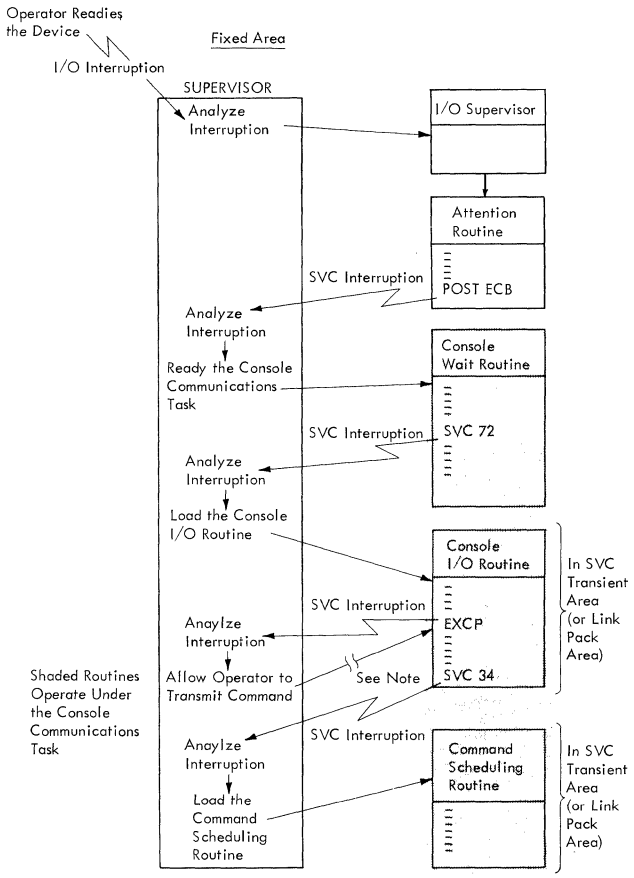
The console communications task is created at system generation (SYSGEN) time when its task control block (TCB) is assembled. The console communications task does not have a region of its own, its storage requirements are filled from the region of another command processing task - the master scheduler task.

Figure 4 shows the flow of control when a command is issued via a console device. The operator pushes the "Attention" button or readies that device, causing an I/O interruption to occur; CPU control is passed first to the supervisor and then to the I/O supervisor. After the I/O supervisor determines the cause of this interruption, it passes control to an attention routine which issues a POST macro instruction for an event control block (ECB) being awaited by the console communications task. The supervisor's processing of this "posting" includes readying the console communications task, and passing CPU control to a routine of this task.

The reading of commands for the console communications task is performed by the Console Wait routine and the Console I/O routine.

The Console Wait routine resides in the nucleus, and is the controlling routine in the console communications task. It is the first routine that receives control for this task, and is the routine that issues the WAIT macro instruction to return the task to the wait state when it is complete. The Console Wait routine receives CPU control from the supervisor, and issues an SVC 72 instruction causing control to be passed to the Console I/O routine.

The Console I/O routine is a type 4 SVC routine that reads a command from a console device into main storage. This routine requests main storage to receive the command and issues an EXCP macro instruction that enables the operator to transmit the command. When the command has been transmitted, the Command Scheduling routine schedules the command by determining whether or not execution of the command requires another system task.



NOTE: While Command is being transmitted, other processing is performed. Control returns to the Console I/O routine after the command is transmitted.

Figure 4. Flow of Control When a Command is Issued via a Console Device

Reading Tasks

When a command is issued via an input job stream, the initial processing and an analysis of this command is part of the reading task associated with that input job stream (see Figure 3). The Interpreter Control routine of the reading task detects the command and invokes the Command Scheduling routine. This routine analyses and schedules the command for the reading task in the same way as it does for the console communications task. After completing its operation for a reading task, the Command Scheduling routine returns CPU control to the Interpreter Control routine which continues reading the input job stream.

SCHEDULING THE COMMAND

Scheduling a command is the storing of the command, and the readying of another system task to continue the command's processing. The Command Scheduling routine (a

type 4 SVC routine) operates under either the console communications task (when the command was issued via a console device), or a reading task (when the command was issued via an input job stream).

If execution of the command does not require additional tasks, the Command Scheduling routine completely processes the command, and returns control to the routine that called it. In the console communications task, the Console I/O routine receives control and passes it to the Console Wait routine; the Console Wait routine places the task in the wait state. In the reading task, control is returned to the Interpreter Control routine which continues reading the input job stream.

However, processing of a command usually requires additional system tasks for execution. If the task for executing the command does not yet exist, the Command Scheduling routine schedules execution of a command by creating a command scheduling control block (CSCB), placing the CSCB in the CSCB queue, and posting an ECB being awaited by the master scheduler task. The routines of the master scheduler task attach the task needed for executing the command.

If the system task for executing the command is already in the system, an ECB in the CSCB for that task is posted.

The CSCB queue contains a CSCB for each command that has a task created for its execution. Command processing routines use this queue to associate commands with the tasks or functions that the commands affect. When creation of a new task is required, a CSCB is created, and an indicator is set in it (i.e., the CSCB is made pending.)

The routines of the master scheduler task check the CSCB queue and attach a task for each CSCB that is pending. The Attach routine of this task scans the CSCB queue until it finds a pending CSCB. When one is found, the Attach routine removes it from the pending status, and attaches the appropriate task. The Attach routine then continues scanning the CSCB queue, attaching a task for each pending CSCB. When no more pending CSCBs are found, the Wait routine of the master scheduler task causes the task to be placed in the wait state.

EXECUTING THE COMMAND

A given command is executed during a system task that is in one of three categories:

- The task is the same task during which the command was read (i.e., the console communications task or a reading task).
- The task is already in the system and has functions other than execution of the command.
- The task is created especially for execution of this command.

A discussion of which commands are in each of the three categories, and the tasks required for processing of these commands is given in IBM System/360 Operating System: MVT Job Management, Program Logic Manual. Relatively few commands are executed under the task that read them. When the Command Scheduling routine detects such a command, it performs all the required processing and returns control to the routine that called it.

Some tasks can include the processing of more than one type of command. For example, the routines that process a MODIFY command can operate under control of a writing task created in response to an earlier START WTR command.

The job processing tasks (i.e., reading, writing and initiating tasks) are created especially for execution of a given command. Each time a START command is issued, a new job processing task is created for the requested function.

JOB PROCESSING

Job processing is made up of three types of tasks:

- Reading tasks, which control the reading of input job stream and the interpreting of the control statements in these input streams.
- Initiating tasks, which control the initiating of job steps whose control statements have been read and interpreted. Termination procedures are also part of initiating tasks.
- Writing tasks, which control the transferring of system messages and user data sets from direct-access volumes on which they were initially written to some other external storage medium (usually a tape, printer, or punch).

These tasks are created in response to START RDR, START INIT, and START WTR commands respectively. Whenever such a command is issued, the resulting command processing includes the attaching of a reading, initiating, or writing task by a routine of the master scheduler task.

There may be more than one of each of the job processing tasks. The user may have input job streams read from several input devices by issuing a START RDR command for each input stream. Each command results in a reading task being attached for the associated device. The user may have system messages or data sets being written on several output devices by issuing a START WTR command for each device; a writing task results from each command. Up to 15 initiating tasks can exist concurrently. Each such task is created in response to a START INIT command.

The command processing routine that attaches any job processing task does not test to see whether a reader, writer, or initiator is to be started. The first routine to receive control in any job-processing task is in the link pack area. This routine requests a region of main storage for the task, and invokes the System Task Control routine. The System Task Control routine is loaded into the region, identifies which type of job processing task it is currently performing, and then invokes the appropriate reading, writing, or initiating routine. The routines that perform the three job processing functions are sometimes called the job scheduler.

READING TASKS

When the current task is a reading task, the System Task Control routine passes control to the Interpreter Control routine. This routine reads the input job stream from the device associated with the task, and builds control blocks and tables from the control statements in the input job stream. The Interpreter Control routine resides on SYS1.LINKLIB; it operates from the region assigned to the particular reading task unless it is already in the link pack area.

The primary purpose of reading tasks is to place entries into the input work queue for use during initiating tasks. The input work queue is made up of control blocks and tables built during all the reading tasks in the system. When construction of the control blocks and tables for a given job are complete, the Interpreter Control routine invokes a queue manager routine which enqueues these blocks and tables in the input work queue. The blocks and tables of a given job are referred to as a work queue entry.

Work queue entries are enqueued according to the priorities of the jobs that they represent, and remain in the queue until their respective jobs are terminated. The major control blocks and tables in a work queue entry are:

- Job control table (JCT), which is built for each job from information in the JOB statement. This table contains job and job step attributes.
- Step control table (SCT), which is built for each job step from information in the EXEC statement. This table contains job step attributes.
- Step input/output table (SIOT), which is built for each DD statement and contains information needed to assign devices to the data set defined in the statement.
- Job file control block (JFCB), which is built for each DD statement and contains data set attributes. Construction of a JFCB is completed when its associated data control block is opened.

Figure 5 shows the flow of information between the control statements and these blocks and tables.

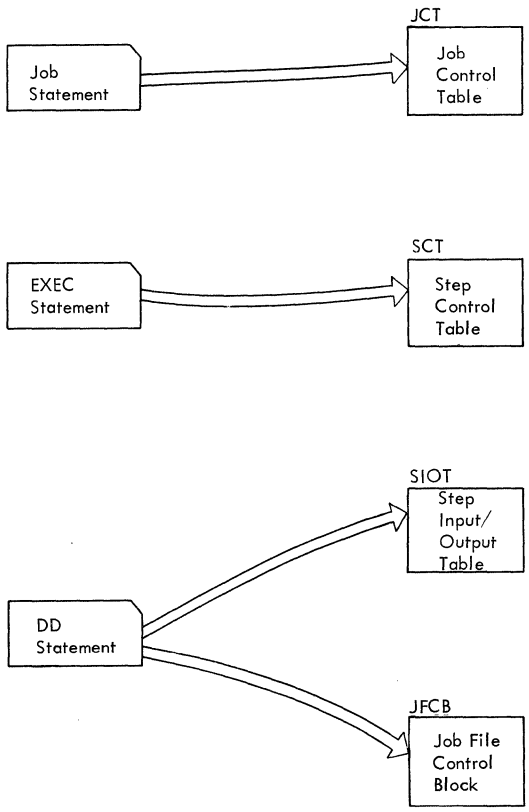


Figure 5. Information Flow Between Control Statements and Blocks of a Reading Task

Commands and Data Sets

When the Interpreter Control routine encounters a command in an input job stream, it invokes the Command Scheduling routine, and processing of this command becomes part of the reading task associated with the input device (see Figure 3). When a data set is encountered in an input job stream, it is written on a direct-access volume.

Termination of a Reading Task

A reading task is terminated either by a STOP RDR command, or when an end-of-data condition is detected on the associated reader device.

INITIATING TASKS

An initiating task has two parts, the preparing of job steps for execution, and the performing of termination processing when job steps are complete.

Preparing of Job Step for Execution

Preparing a job step for execution consists of:

- Acquiring a region of main storage.
- Locating data sets that are input to the job step.
- Assigning I/O devices required for the job step.
- Reserving auxiliary storage for data sets created during the job step.
- Attaching a task for the job step.

When the System Task Control routine determines that it is performing an initiating task, the Initiator routine is given CPU control. The Initiator routine invokes a queue manager routine to dequeue the first entry on the input work queue. (This is the entry of the highest priority job that is not already being initiated under some other initiating task.)

Once a work queue entry (i.e., a job) has been assigned to an initiating task, initiation, execution, and termination of the steps of that job are performed sequentially under control of that initiating task. Multijobbing is obtained when several initiating tasks are concurrently controlling the initiation, execution, and termination of the steps of different jobs.

In a multijobbing environment, one data set can be required for two or more jobs that are operating concurrently. Before starting to initiate the first step of the job, the Initiator routine invokes the ENQ/DEQ routine of the supervisor to determine if any of the data sets required during this job are currently being used in another job, and cannot be shared. If a data set required by any step in the job is not currently available, initiation of the first job step is suspended until the data set becomes available. Multiple use of a data set is allowed only when all users have specified that the data set can be shared.

Acquiring a Region: When the System Task Control routine determined that it was performing an initiating task, this routine requested a region large enough to meet its initial requirements. Later, after the Initiator routine reads the SCT, control is given to a routine in the link pack area. This routine determines the main storage requirements of the step, releases the region currently assigned to the initiating task, and requests a new region. This new region meets the storage requirement of either the job step or the initiating task - whichever is larger. The Device Allocation routine is then brought into the region to continue initiating the step. If there is not enough contiguous storage in the dynamic area to fill the request for the region, the initiating task is put in the wait state until enough storage is available.

Locating Input Data Sets: The Device Allocation routine determines which volumes contain input data sets from either the DD statement, or a search of the catalog. (A catalog management SVC routine is invoked to perform this search.) Once the volumes that contain the data sets are determined, the Device Allocation routine determines if any I/O devices are available for these volumes.

Assigning Input/Output Devices: A job step cannot be initiated unless there are enough I/O devices to fill its needs. The Device Allocation routine determines whether the required devices are available. If there are sufficient devices available, they are assigned to the step; if there are not, a message is issued to the operator. If the operator cannot make the required number of devices available, he may have the initiating task put in the wait state until sufficient devices are available, or he may cancel the job.

After devices are allocated, the Device Allocation routine builds a task input/output table (TIOT) from information in the JCT, SCT, and SIOTs, of the job step. The TIOT has one entry for each data set used during the job step; this entry contains pointers to other control blocks

needed in the processing of the data set. The TIOT is in the system queue area, and exists for the life of the job step. Figure 6 shows the relationship of the TIOT to other job processing control blocks. The shaded portion of the figure shows the major blocks and tables built during a reading task (see Figure 5). The nonshaded portion of Figure 6 shows the major blocks and tables created during initiating tasks.

Reserving Auxiliary Storage Space: Any direct-access volume space required by the output data sets of a job step is acquired at the request of the Device Allocation routine by a direct-access device space management (DADSM) routine of data management. If the volumes specified by the Device Allocation routine do not have the required space, the DADSM routine returns control to the Device Allocation routine which issues an operator message. The operator must then either make a new volume available, cancel the job, or indicate that the initiating task be put into the wait state until space becomes available.

When direct-access volume space is being assigned, a DADSM routine partially builds a data set control block (DSCB) for the output data set that will occupy the space. The DSCB is the label for that data set and contains data set characteristics obtained from the JFCB, and the track addresses assigned to the data set. Construction of DSCBs for output data sets is completed when the associated data control block (DCB) is opened.

Attaching a Task for the Job Step: The final operation in the initiation of a job step is the creating (attaching) of a task

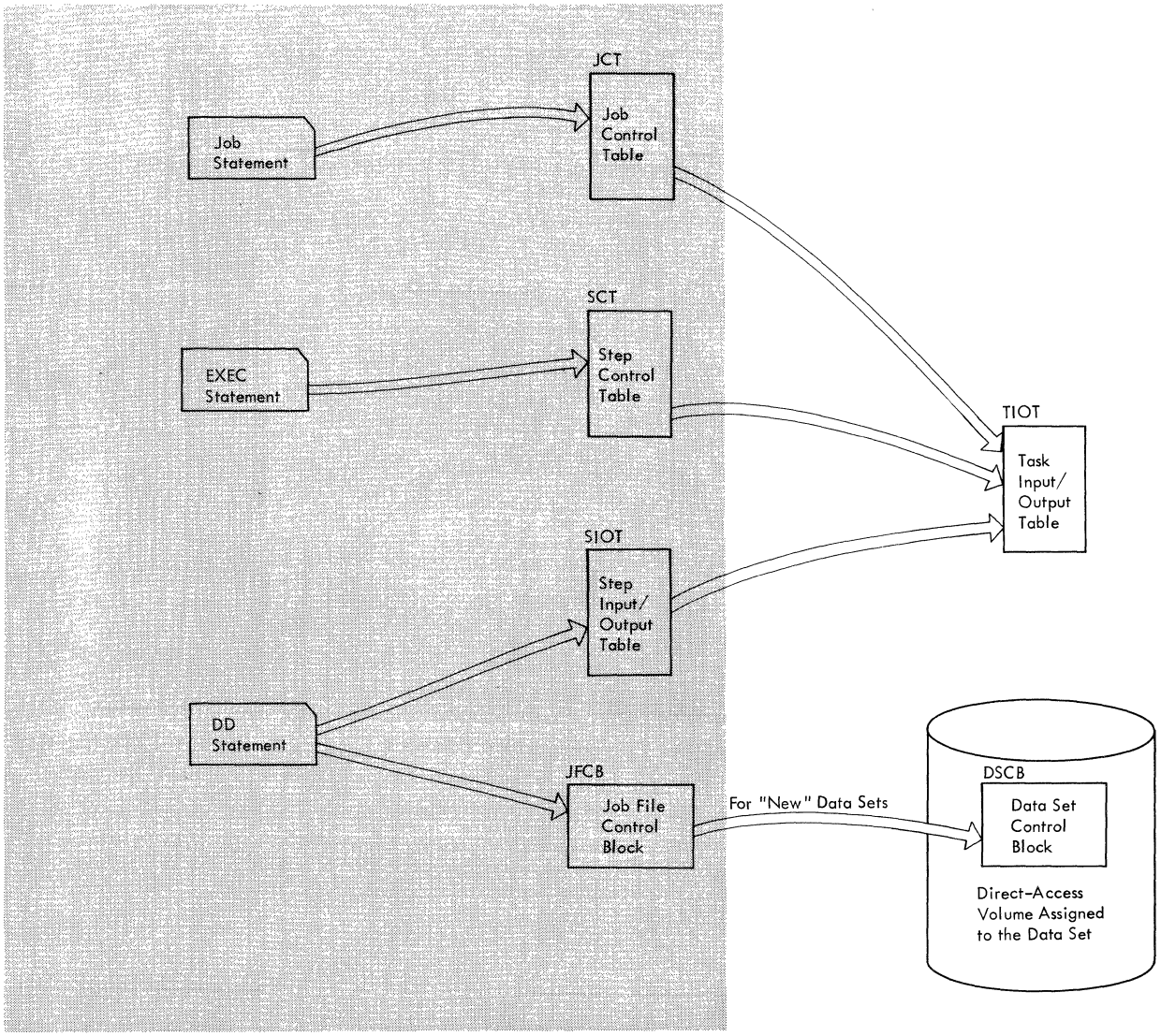


Figure 6. Relationship of Blocks of Initiating Task to Blocks of Reading Task

for that job step. The Step Attach routine issues an ATTACH macro instruction causing CPU control to be given to the Attach SVC routine. This routine builds a task control block (TCB) which the control program uses to control the job step processing; this TCB is the job step TCB. It is placed in the TCB queue according to the priority of the related job.

Since initiation of the step is complete, the initiating task is placed in the wait state until the step is to be terminated.

Terminating a Job Step

A job step is terminated either when it is complete, when a specified time interval expires, when an error prevents any more processing, or when the job is canceled by the operator. Any of these conditions cause the supervisor to make ready the initiating task that controlled initiation of that step. Termination processing for the step is performed under control of this initiating task.

The Terminator routine of job management is brought into the region assigned to the step. This routine disposes of data sets created or used during the job step, and releases the I/O devices assigned to the job step. The Initiator routine issues a DETACH macro instruction to remove the job step TCB from the system.

After terminating the last step of a job, the Initiator routine performs some additional processing. It deletes the work queue entry for the job from the input work queue, and makes entries in output work queues for system messages and SYSOUT data sets.

Output work queues are used during writing tasks to indicate the SYSOUT data sets and system messages that are to be written. There are 36 output work queues, one for each SYSOUT class. A SYSOUT class is an alphabetic or numeric designation by which the user can group his messages and SYSOUT data sets.

An output work queue is made up of entries containing data set blocks (DSBs) and system message blocks (SMBs) for data sets and system messages in a given class. During step termination, the Initiator routine builds a DSB for each SYSOUT data set created during that step. SMBs are created for system messages as they are generated. Messages are blocked so one SMB can relate to several messages.

All the DSBs and SMBs in a given class for a given job are referred to as an output work queue entry for that class. After the last step of a job is terminated, the Initiator routine invokes a queue manager routine to enqueue the output work queue entries the DSBs and SMBs from that job into the appropriate output work queues.

WRITING TASKS

A writing task controls the writing of all system messages and SYSOUT data sets in a specified class (or classes) from the direct-access volume on which they were initially placed to a specified SYSOUT device. A writing task is created as a result of a START WTR command that specifies a class or classes of messages and data sets to be processed. When all outstanding SYSOUT data sets and messages in that class are written, the writing task is placed in the wait state, until more data of that class becomes available or until a STOP WTR command is issued.

When a writing task is started, the System Task Control routine identifies it as such, and passes CPU control to the first writer routine. The writer routines indicate to the queue manager routines which message class is to be written. The queue manager routine passes the first output work queue entry from the appropriate output work queue to the writer routines which write the associated messages and data sets. After all messages and data sets from this queue entry are written, the writer routine requests the next entry from the queue, and upon receiving it, writes the messages and data sets. When all the messages and data sets in all the classes associated with this writing task are written, the writing task is placed in the wait state. A writing task is again made ready when an output work queue entry is placed in one of the queues associated with this task.

Task management routines control the allocation and use of CPU, main storage, and programming resources. The task management functions are:

- Interruption supervision. The analysis of interruptions to determine what supervisor processing is required, and the determination of which task is to be performed after the supervisor processing is complete.
- Task supervision. The recording of what tasks are currently in the system, their statuses, priorities, and the programs they require.
- Main storage supervision. The allocating and freeing of main storage, and recording of what use is being made of any portion of main storage.
- Contents supervision. The loading of programs into storage, the recording of what programs are currently in main storage, and what characteristics these programs possess.
- Timer supervision. The setting and maintaining of the interval timer from information provided in timer macro instructions.
- System environment recording. The collecting and recording both hardware and software data when a machine-check interruption occurs.

Except for some nonresident SVC routines and part of the SER0 routine the task management routines are part of the nucleus. All task management routines operate in the supervisor state under a PSW with a protection key of zero.

The task management routines are described in the publication IBM System/360 Operating System: MVT Supervisor, Program Logic Manual, Form Y28-6659.

INTERRUPTION SUPERVISION

CPU control is passed to the supervisor via an interruption. An interruption can occur either because the interrupted program has requested some control program service, or because some event requires supervisory processing. There are five types of interruptions:

- SVC interruption, which occurs when an SVC instruction is executed.
- Input/output interruption, which occurs when an I/O operation terminates, or an I/O device is readied.
- Timer/external interruption, which occurs when an event (e.g., a timer or external signal) indicates the need for control program processing.
- Program Interruption, which occurs when either a program attempts an invalid operation (e.g., execution of a privileged instruction by a program in the problem state), or a data error (e.g., overflow) is detected.
- Machine-check interruption, which occurs when a recognizable machine error has occurred.

Interruption supervision has two parts, the analyzing of the interruption to determine what control program routine is to process the interruption, and the passing of CPU control to a program of a task after the interruption processing is complete.

ANALYZING THE INTERRUPTION

Any interruption except a machine-check interruption causes CPU control to be taken from the interrupted program and given to an interruption handling routine of the supervisor. There are four interruption handling routines, one for each type of interruption except machine-check. A machine-check interruption causes control to be passed directly to a system environment recording (SER) routine, if SER routines are in the operating system. The SER routine attempts to diagnose the error. If no SER routines are provided, the system is placed in the wait state.

The interruption handler does not process the interruption, but analyzes it and passes control to the proper interruption processing routine. The SVC Interruption Handler loads (if necessary) the SVC routine indicated in the SVC instruction and passes control to it. The I/O Interruption Handler passes control to the I/O supervisor. The Timer/external Interruption Handler determines what caused the particular

interruption and passes control to the appropriate routine. The Program Interruption Handler determines whether the user has provided a routine to process this particular type of program interruption. If such a routine is provided, it is given control; if not, the task being performed by the interrupted program is terminated. If the interrupted program is in supervisor state (e.g., an SVC routine), the associated task is always terminated.

The interruption handlers are disabled for all interruptions except machine-check so that they are not interrupted before they can save critical information about the interrupted program. This critical information comprises the registers' contents and PSW information necessary to return control to the interrupted program after the interruption is processed.

PASSING CONTROL TO A PROGRAM OF A TASK

Once the supervisor has completed its interruption processing it passes CPU control to a program operating under control of a TCB. If the program to receive control is the program last interrupted, CPU control is passed directly to it. (This occurs when certain type 1 SVC routines processed the interruption.) If, however, it is possible that another program (other than the one last interrupted) is to receive control, control is passed to the Dispatcher routine.

The Dispatcher routine checks the TCB queue for the highest priority, ready task, and passes CPU control to the program currently indicated to perform that task. This passing of control is referred to as dispatching a task.

The task that is dispatched, therefore is not necessarily the same task that was last interrupted. The interruption processing could have created or made ready several tasks of higher priority. Any such tasks are dispatched before the task that was last interrupted.

TASK SUPERVISION

Task supervision routines enter tasks into the system, supervise their performance and perform task termination processing. Tasks are represented to the control program, by task control blocks (TCBs); task supervision consists primarily of modifying the TCB queue.

When the initiation of a job step is complete, an initiating routine issues an ATTACH macro instruction causing the supervisor to create a task for the step. The supervisor uses this job step task, its TCB, and other associated control blocks and tables to control the processing originally specified as the job step.

Expansion of an ATTACH macro instruction includes an SVC instruction which causes an SVC interruption. CPU control is passed through the SVC Interruption Handler to the Attach SVC routine. This routine builds a TCB for the task, and a request block (RB) for the first program of that task. These blocks are placed in the TCB and RB queues respectively.

TASK CONTROL BLOCK QUEUE

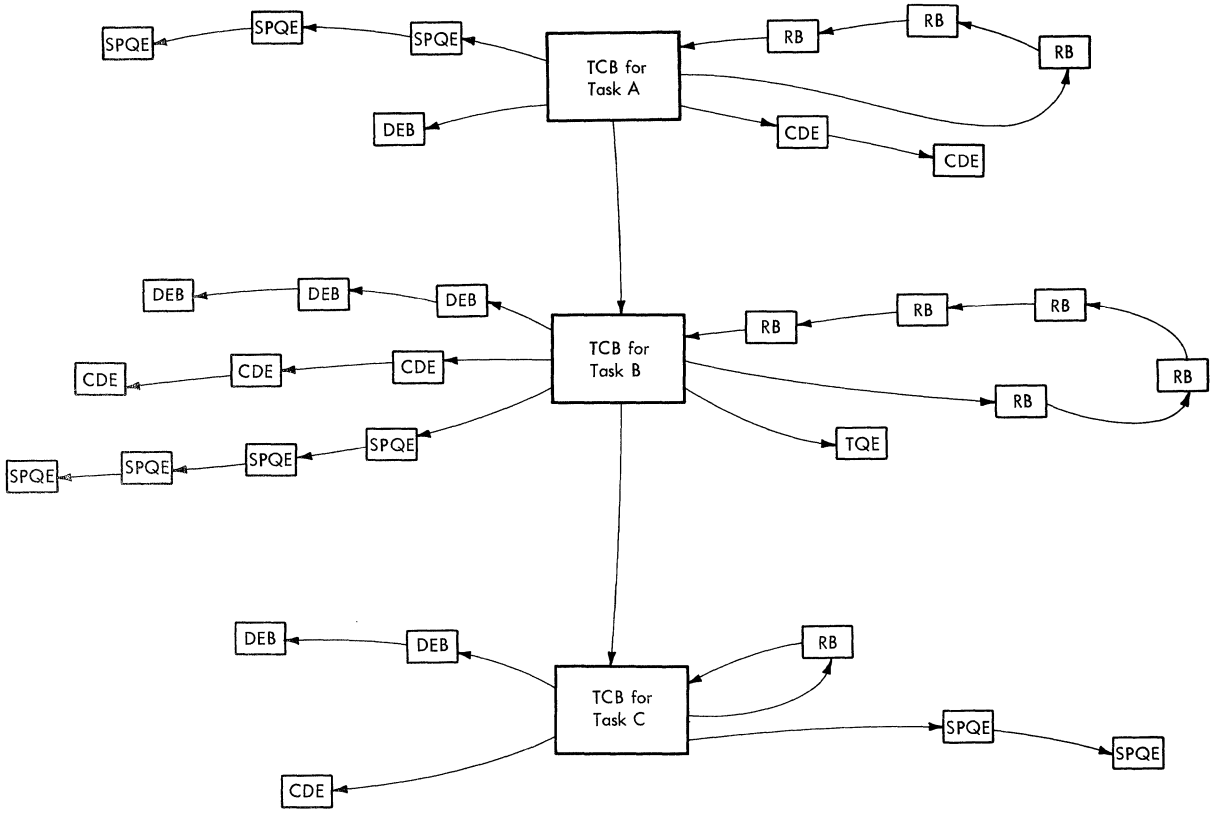
Whenever the control program needs any information about tasks, its starting point is the TCB queue. There is one TCB for each task currently scheduled is the system. A TCB indicates the status and characteristics of the task it represents. A major part of this status information is pointers to other control blocks and queues, and control information about resources needed during the task. Figure 7 shows the concept of the TCB queue and queues that originate from a TCB.

A TCB is placed in the queue according to the priority of its task, and is pointed to by the next higher TCB in the queue. The first TCB is pointed to by the communications vector table (CVT), which is part of the nucleus.

REQUEST BLOCK QUEUE

The supervisor builds a request block (RB) for each program of a task that is entered via a supervisor-assisted linkage (LINK, XCTL, or ATTACH), and for types 2, 3, and 4 SVC routines invoked by programs of a task. The RB is placed in the RB queue which originates at the associated TCB. The RB queue indicates, to the supervisor, the programs that perform a given task.

During a task, the addition of RBs to the queue as new programs are invoked, and the deletion of RBs as these programs are completed, allow the supervisor to determine which program (for a particular task) is to receive control at any given time. When a program terminates, the RB queue indicates whether the associated task has been completed, or whether execution of another program is required. If the RB for the terminated program is the only one on the RB queue, the task is complete. If other RBs are on the queue, the programs represented by these RBs must be performed before the task is complete.



SPQE - Element of a queue used in main storage supervision
RB - Element of a queue used in task supervision
CDE - Element of a queue used in contents supervision
DEB - Element of a queue used in data management
TQE - Element of a queue used in timer supervision

Figure 7. Concept of the TCB Queue

Figure 8 shows how the RB queue is modified during a task that is performed by three programs. The first (program A) was specified in the ATTACH macro instruction. The second (program B) was invoked by program A via a LINK macro instruction. The third (program C) was given control by program B via an XCTL macro instruction. When program A is executing, the RB queue for the associated task is shown in Figure 8.1. After program B is invoked, the RB queue is shown in Figure 8.2. After program C has received control via the XCTL macro instruction, the RB queue is as shown in Figure 8.3. When program C completes and issues a RETURN macro instruction, its RB is deleted, and the RB queue is again as shown in Figure 8.1.

When the last program of a task has completed, the task supervision routines usually delete the TCB from the TCB queue. If, however, the completed task had an ECB or ETXR parameter specified when it was attached, the TCB is not deleted from the TCB queue. The ECB parameter indicates that the attaching task needs information in the TCB of the completed task. The ETXR parameter indicates that an End-of-task Exit routine (ETXR) is to be executed. The ETXR routine has an RB enqueued to the TCB of the completed task.

If the completed task is a job step task, its associated initiating task is made ready. When this initiating task is dispatched, its routines complete termination processing for the job step.

MAIN STORAGE SUPERVISION

Main storage supervision routines control and allocate main storage in the dynamic and system queue areas. Storage in the dynamic area is assigned to job steps and system tasks. Storage in the system queue area is assigned to contain control blocks that must have a supervisor storage protection key.

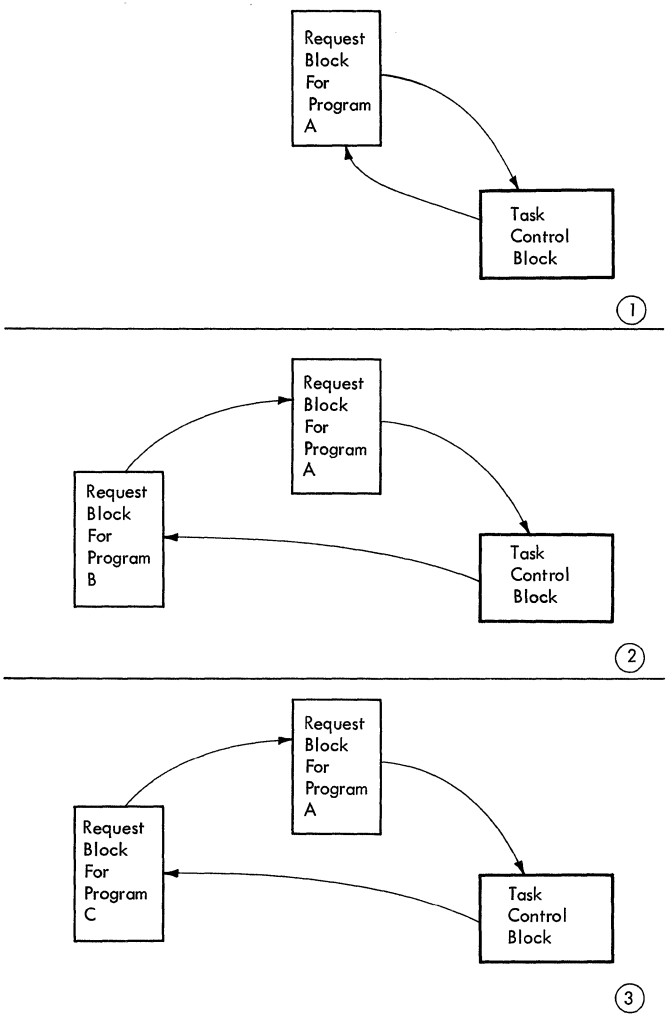


Figure 8. Example of the Modification of the RB Queue During a Task

STORAGE ALLOCATION IN THE DYNAMIC AREA

Initially, the dynamic area is a large number of unassigned, contiguous blocks of main storage, each 2048 (2K) bytes in length. As a system task or job step is initiated, an initiating routine requests the main storage required for the task or step. A main storage supervision routine (the GETMAIN routine) assigns enough contiguous 2K blocks to this step or task from the higher end of the dynamic area. This group of 2K blocks is a region. If there is insufficient free, contiguous storage for the step or task, the initiating task is put in the wait state until sufficient storage becomes available.

Storage Allocation in a Region

Storage within a region is assigned in response to requests from programs performing the step or system task associated with that region. Storage from a region is required not only for work areas, but also for programs not already in the link pack area.

Main storage supervision routines determine what storage in a region is available via control blocks called the partition queue element (PQE) and free block queue elements (FBQEs). The PQE of a given region is created when the region is assigned, resides in the system queue area, and is pointed to by the TCBs of all the tasks associated with that region. The PQE is, in turn, the origin of a queue made up of FBQEs. Each FBQE points to a group of contiguous, available 2K blocks of storage within the region, and resides in the lower end of the lowest 2K block of the group.

When all the storage in a region is available, or when all the available storage is contiguous, only one FBQE exists. As storage is assigned, used, and subsequently released, unassigned sections of the region become interspersed with the assigned sections. As this fragmentation occurs, an FBQE is created and enqueued for each unassigned section. When the releasing of storage in a region causes a larger available section to be formed from several smaller sections, FBQEs are deleted from the queue so that this new section has only one FBQE.

When storage is requested, the FBQEs are scanned for an available section large enough to fill the request. If none is available, the task is terminated. If the request can be filled, a sufficient number of 2K blocks from the unassigned storage is made a part of a subpool. The subpool to which this storage is assigned depends on how the storage is to be used.

Subpools

A subpool is, generally, all the 2K blocks of main storage allocated for a particular task under one label called the subpool number. (The exceptions to this definition are shared subpools and subpools in the system queue area.) Initially all storage in a region is unassigned, is not part of any subpool, and has a storage protection key of zero (see Figure 9). When storage within a region is required, unassigned storage in that region is made part of a subpool. The request for this storage specifies (either explicitly or by default) the subpool number of the subpool to which the storage is to be assigned.

When storage is requested in a subpool that doesn't exist (i.e., this is the first request specifying that subpool for a particular task), the subpool is created by allocating a sufficient number of contiguous 2K blocks from the unassigned storage

in the region. When the specified subpool already exists (i.e., there were previous requests), the storage currently part of that subpool is checked to determine if a contiguous area, large enough to fill the request, is available. If such an area is available, it is used to fill the request; if not, a sufficient number of contiguous 2K blocks are assigned to the subpool from the remaining free storage in the region. Insufficient free storage in the region results in termination of the task.

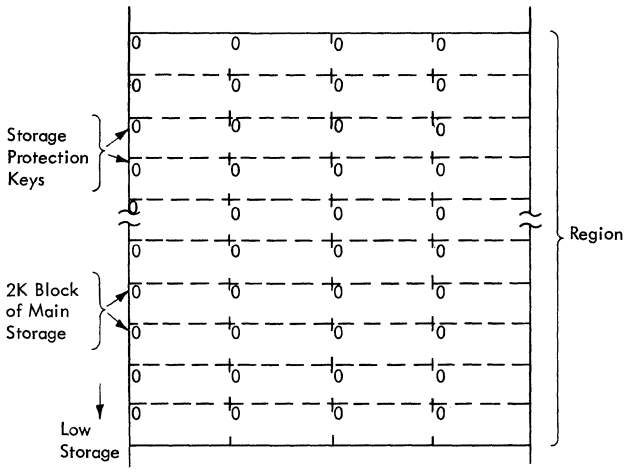


Figure 9. Initial Format of a Region

Storage made part of a subpool for any one request must be contiguous. The storage that makes up a complete subpool (i.e., for all requests specifying that subpool) can be noncontiguous.

Requests for storage in subpools of a region are made either by programs performing the task (for working storage), or by the control program (for storage to load a program or for working storage). The subpools specified in a given request depends on what the storage is to be used for, and on what type of routine made the request.

Assigning Storage to Subpools: Working storage for programs in the problem state is requested in any subpool between 0 and 127. Storage is assigned to these subpools from the highest available storage in the region.

Storage requested from a region by control program routines in supervisor state is assigned in subpool 251 or 252. Subpool 252 is specified either for storage needed to contain reenterable routines from SYS1.SVCLIB or SYS1.LINKLIB, or for storage to contain control program data that must be protected. Storage assigned to subpool 252 is given a storage protection key of zero to prevent programs of the job step from writing in the subpool. Storage is assigned to subpool 252 from the highest available storage in the region.

Subpool 251 is specified for storage needed to contain all serially reusable and nonreusable programs, and reenterable programs from private libraries. Storage in subpool 251 is assigned from the lowest available storage in the region. It has the same storage protection key as the programs using the region, and therefore its contents can be modified by these programs.

Subpools are usually assigned to a specific task. When one region is being used for several tasks (i.e., a job step task has one or more subtasks), each task can have separate subpools between 1 and 127, or the tasks can share subpools between 1-127. In any one region, subpool 0 is always shared, and there is only one subpool 251 and one subpool 252.

Subpool Queue: Each time a new subpool is created for a task, a subpool queue element (SPQE) for that subpool is placed in the subpool queue of that task. The subpool queue originates at the TCB of the task, and is made up of a series of SPQEs - one for each subpool of the task. The main storage supervision routines use the subpool queue to determine what subpools are being used in the task, and what storage is assigned to each of the subpools.

Each SPQE has, in turn, a queue of elements that indicates what storage in the region is assigned to that subpool. These queues are called descriptor queues. Each element in a descriptor queue represents one group of contiguous 2K blocks assigned to the subpool.

Figure 10 is an example showing how the main storage supervision routines modify the region, the subpool queue, and the descriptor queues of a task to obtain storage for the three programs previously discussed in the "Task Supervision" section and illustrated in Figure 8. The shaded portions on the left side of Figure 10 show the request block queue as the three programs are used (see Figure 8). The unshaded portions show how the region, and the subpool queue is modified as storage is assigned for the programs. This example assumes that the programs are not initially in storage; programs A and B are reenterable and reside on SYS1.LINKLIB, and program C is not reusable.

Figure 10.1 shows how the region and the subpool queue are set up for program A. When the task was first dispatched, the Link routine of the supervisor, not program A (which is not yet is storage), receives control. The Link routine requests storage for program A from subpool 252. If we assume that program A requires 3.5K bytes of storage, the main storage supervision

routines assign three 2K blocks of storage from the highest available part of the region to subpool 252, builds an SPQE, and enqueues it to the TCB. The first 2K block is a work area for Program Fetch (only one per region is required). The other two 2K blocks are for program A. A descriptor queue element (DQE) is built and enqueued to the SPQE. The single DQE indicates that, at the moment, subpool 252 is made up of only one contiguous area of storage.

If program A issues a GETMAIN macro instruction for 2K bytes of working storage in subpool 3, the region and subpool queue are modified as shown in Figure 10.2. Since there is no subpool 3, the highest available 2K block of storage in the region is assigned to subpool 3, and the SPQE and DQE are placed in the subpool queue (assume that the storage protection key for this job step is 5).

After the LINK macro instruction for program B is issued, the Link routine requests storage from subpool 252 for program B. If subpool 252 currently has a free section large enough for program B, this program is brought into that area and the subpool queue remains as shown in Figure 10.2. If however, subpool 252 must be enlarged for program B (assume that program B is 4000 bytes long), the main storage supervision routines assign two 2K blocks from the highest available part of the region. Since this addition to subpool 252 is not contiguous to the already-existing part of subpool 252, a new DQE is added to the queue. After program B receives control, the queues are as shown in Figure 10.3.

Figure 10.4 shows the region, and the subpool and RB queues after program C receives control. When the XCTL macro instruction for program C is issued, the XCTL routine requests storage (assume 6000 bytes) for program C from subpool 251. Since there is no subpool 251, three 2K blocks of storage from the lowest available part of the region is assigned to subpool 251 and the SPQE and DQE are placed in the subpool queue.

STORAGE ALLOCATION IN THE SYSTEM QUEUE AREA

Storage in the system queue area is used to build control blocks that can be modified by only the control program. The system queue area has storage protection keys of zero, and only programs that operate under a protection key of zero can write into it. Space in the system queue area is allocated to three subpools, subpools 253, 254, and 255.

Unlike the subpools in the regions, subpools 253, 254, and 255 can share a 2K block of storage. A request for storage for any of these three subpools will be filled from the highest available area of sufficient size even if this area is in a 2K block that is partially assigned to another subpool.

If the request for system queue space cannot be filled, the system queue area can be expanded by assigning to it 2K blocks of adjacent free dynamic area storage. Before such an expansion is attempted however, subpools 251 and 252 of all regions are purged. This purge results in the deletion of contents supervision queues from the system queue area. If this deletion of queues releases enough system queue space to fill the pending request, the system queue area is not expanded.

Once expanded, the system queue area will not be reduced to its original size until the IPL procedure is repeated. If the portion of the dynamic area, adjacent to the system queue area, is already assigned as a region, any attempt to expand the system queue area results in the system's being placed in the wait state.

CONTENTS SUPERVISION

Contents supervision routines bring non-resident routines into main storage, and record what routines are in the dynamic and link pack areas. Routines are brought in as a result of a LINK, ATTACH, XCTL, or LOAD macro instruction, if a usable copy of the desired routine is not already in main storage.

The characteristics of a called routine, and its location in main storage determine whether that routine is usable to the calling routine. Routines in the link pack area (all of which are reenterable) can be used by any routine that calls them, and, in fact, the one copy in that area can be used concurrently by several calling routines. Reenterable and serially-reusable routines in subpools 252 and 251 of a given region can be used only by other routines performing the task or tasks associated with that region.

Contents supervision routines determine whether a routine is in main storage by checking the contents directory. Each time a routine is brought into main storage, an entry for it is made into the contents directory. If the routine is brought in via a LOAD macro instruction, an entry is also made to a load list.

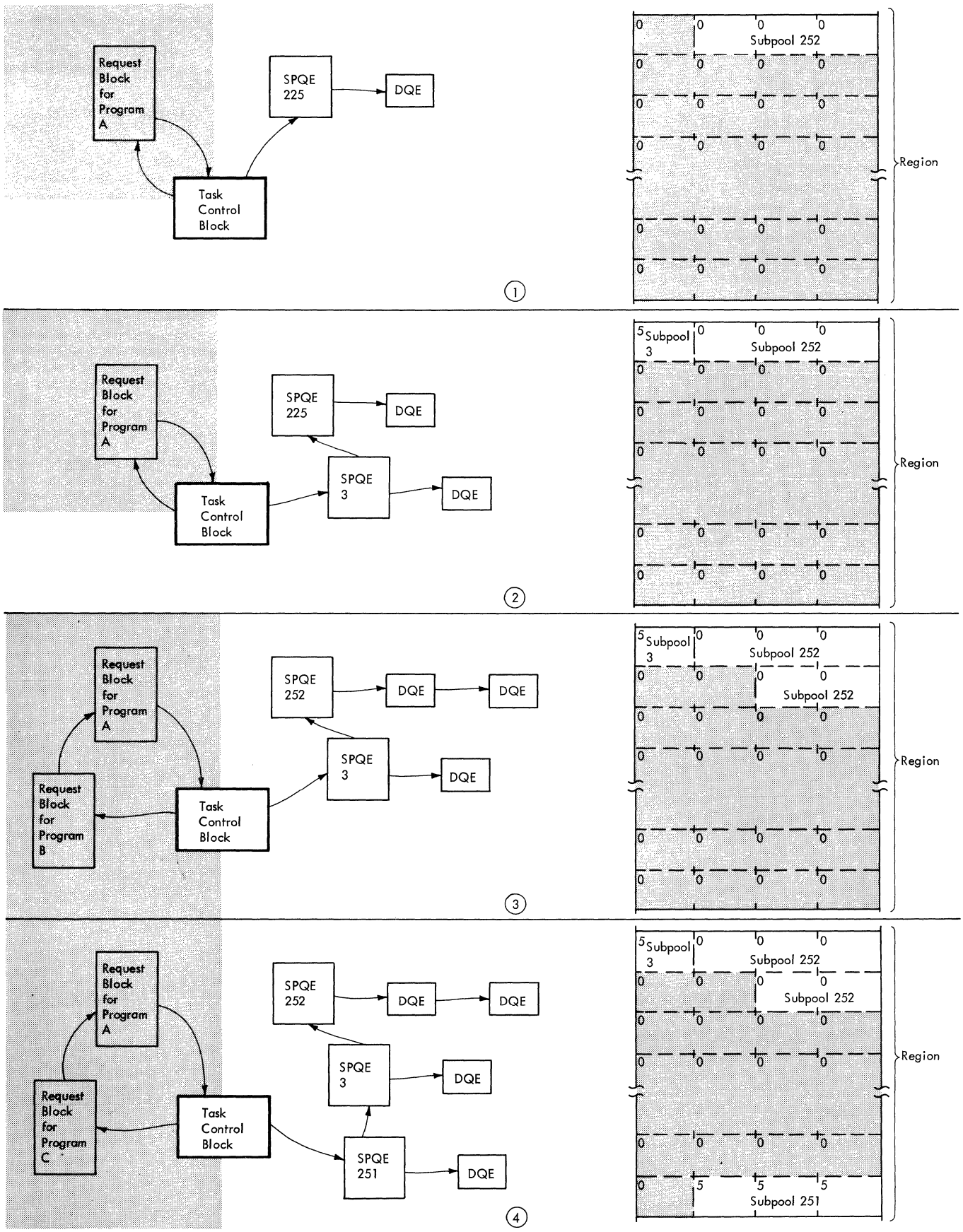


Figure 10. Example of Main Storage Allocation

CONTENTS DIRECTORY

The contents directory is a group of queues indicating the routines in the link pack and dynamic areas.

There are two contents directory queues for the link pack area, and one for subpools 251 and 252 of each region. The contents directory resides in the system queue area. The pointer to the contents directory queues for the link pack area is in the communications vector table. The pointer to the contents directory queue for a region is in the first TCB created for that region (system task TCB or job step TCB).

A contents directory queue contains a contents directory entry (CDE) for each program in the region to which it applies (or to the link pack area). When a reenterable or serially reusable program has completed its operation, the copy of that routine remains in the region, and its CDE remains on the queue. When a nonreusable program has completed its operation, its CDE is deleted from the queue. (The storage in subpool 251 occupied by the program is made available; sometimes this storage remains as part of subpool 251, sometimes it is released from subpool 251 for assignment to any subpool of the region.)

Figure 11 shows how a contents directory queue is affected for the programs discussed in the sections "Task Supervision" and "Main Storage Supervision" and shown in Figures 8 and 10. We assumed that none of the programs were already in main storage. During the processing of the ATTACH macro instruction that specified program A (which was previously defined as reenterable), the contents supervision routines determine that a copy is neither in the associated region, nor (since it is reenterable) in the link pack area. After main storage is assigned, program A is brought into subpool 252 and the contents directory queue for that region is as shown in Figure 11.1. If program A was already in the region a new copy is not brought in.

During the processing of the LINK macro instruction that invokes program B, the contents supervision routine determines that this program is neither in the region, nor in the link pack area. Main storage is requested and assigned in subpool 252, and program B is brought in. A contents directory entry is constructed and enqueued as shown in Figure 11.2.

The XCTL macro instruction involving program C results in a copy of that program being brought into subpool 251 of the region, and a contents directory entry is added as shown in Figure 11.3. When program C terminates and passes control to the supervisor, the contents directory entry for program C is deleted, and the contents directory is again as shown in Figure 11.2. (This deletion occurs because program C is not reusable.)

LOAD LIST

A load list is a queue of elements for routines in either the link pack area or a given region, that were invoked via a LOAD macro instruction. Each load list element corresponds to a loaded routine, and points to the contents directory entry for that routine. Each load list element contains a count of the number of times a LOAD macro instruction is issued for the associated routine during a given task. This count is decremented each time a DELETE macro instruction is issued for the routine to reflect the number of current users of the routine.

TIMER SUPERVISION

Timer supervision routines process both timer interruptions and requests for timing services. For a System/360 that has the interval timer feature, the operating system provides the capability of obtaining the date and time of day, measuring periods of time, and scheduling certain processing for a specific time.

The programmer specifies these functions by the timer macro instructions, TIME, TTIMER, and STIMER. The expansion of each of these macro instructions includes an SVC instruction which causes CPU control to be passed through the SVC Interruption Handler to the appropriate timer supervision SVC routine.

When the value in the interval timer goes from positive to negative (indicating the expiration of some interval), a timer/external interruption occurs. After determining that this interruption is a timer interruption, the Timer/external Interruption Handler passes control to a timer supervision routine (the timer second-level interruption handler). This routine performs any processing specified for the completion of this particular interval, and places a new interval in the timer.

Even if the programmer does not specify any timer intervals of his own, the timer supervision routines set the internal timer so that a timer/external interruption occurs every 6 hours; the first such interruption occurs 6 hours after initial program loading (IPL). The timer supervision routines also cause a timer interruption to

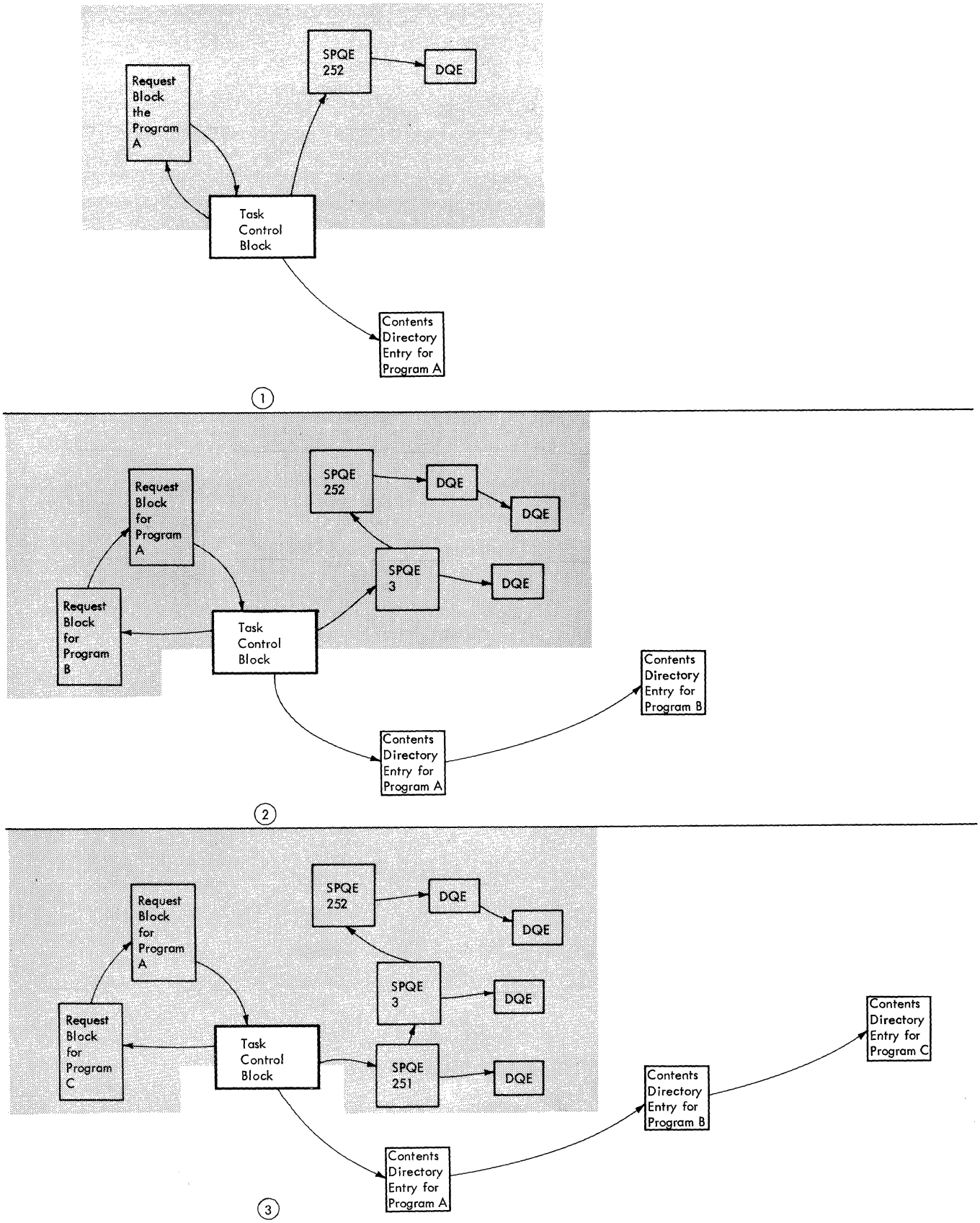


Figure 11. Example of the Modification of Content Directory During a Task

occur every midnight. The midnight interruption allows timer supervision routines to increment the date in the CVT. The 6-hour interruptions allow timer supervision routines to update two of three main storage locations called pseudo-clocks.

PSEUDO-CLOCKS

There are three pseudo-clocks, the local time pseudo-clock (LTPC), the 24-hour pseudo-clock (T4PC), and the 6-hour pseudo-clock (SHPC). The LTPC contains the time of day specified in the SET command when IPL was performed. The T4PC is incremented by 6 each time a 6-hour timer interruption occurs, unless its value is already 18. The 6-hour interruption that occurs when the T4PC contains 18 causes the T4PC to be set to zero. The 6-hour pseudo-clock (SHPC) contains the value of the next interval that will expire; this value is never greater than 6 hours.

In addition to the periodic timer interruption, other timer interruptions are required for program-requested timing of intervals. The timer supervision routines use the timer queue to record the lengths of intervals and the order in which these intervals expire.

TIMER QUEUE

The timer queue is a series of elements in the system queue area. Each timer queue element refers to a particular timer interval, and indicates both the length of the interval and the processing to be performed when the interval ends. Whenever a request to set the timer is issued, a timer queue element is placed in the queue. The elements are queued in the order in which the intervals expire. Thus when a timer interruption occurs, the first element in the timer queue is the one associated with the expired interval. After the interruption is processed, this element is removed from the queue, and the next interval to be timed is obtained from the element that is now first in the queue. Timer queue elements of intervals associated with particular tasks (as opposed to intervals being timed regardless of what task is being performed) have pointers to and from their respective TCBs. A task can have only one interval being timed at any given time.

When a timed interval is associated with a task, the timer queue element of this interval is removed from the timer queue every time this task loses CPU control. The element is returned to the queue before the task is again dispatched so that the timing can continue.

SYSTEM ENVIRONMENT RECORDING

The system environment recording (SER) routines collect and record hardware and software data whenever a hardware error is detected in the CPU, channels, control units, or I/O devices. SER is an optional feature in the operating system. If the SER function is not included, a machine-check interruption causes the system to be placed in the wait state.

There are two mutually exclusive SER routines, SER0 and SER1, both of which provide data about the error. SER0 always places the system in the wait state. SER1 either abnormally terminates the job step (if the error's effects are isolated within a job step), or places the system in the wait state (if the error affects the control program or a channel).

Both SER0 and SER1 are designed for a particular model of System/360. There are SER routines for the models 40, 50, 65, and 75. A SER routine that is run on a model for which it was not designed will either generate erroneous data, or (because the 'diagnose' instruction is model-dependent) cause all CPU and I/O operations to stop.

The SER routines are entered via the machine-check new PSW which, for a CPU error, is loaded by the hardware via a machine-check interruption, and for an I/O error, is loaded by the I/O supervisor.

SER0 ROUTINE

The SER0 routine has two parts, a resident portion in the nucleus and a nonresident portion that resides on SYS1.LINKLIB. Control is first given to the resident portion which saves some data, halts all I/O operations, and loads the nonresident portion.

The nonresident portion of SER0 saves the rest of the hardware and software data, places all the data into a record, writes this record in the SYS1.LOGREC data set, and issues a message to the operator. SER0 then loads the PSW that places the system in the wait state.

SER1 ROUTINE

The SER1 routine is entirely resident. Like the SER0 routine, SER1 collects and writes error data on the SYS1.LOGREC data set. However, SER1 analyzes the error to determine if the operating system can continue operation. If the error affects only one job step, that step is abnormally terminated, but processing of other job steps continues. If the error is a channel failure or affects the control program, SER1 loads the wait state PSW.

Data management routines perform the control program's services of:

1. Assigning and releasing space on direct-access volumes.
2. Maintaining the catalog.
3. Performing the I/O support (open, close, end-of-volume) processing.
4. Processing I/O operations.

The first three of these functions are performed by type 3 and 4 data management SVC routines; they reside on SYS1.SVCLIB and operate from either the link pack area or the SVC transient areas. These routines are invoked via SVC instructions that are either coded directly or generated as part of macro instruction expansions in the calling program. When the SVC instruction is executed, the resulting SVC interruption causes control to pass to the SVC interruption handler. The desired SVC routine is brought into an SVC transient area (unless a copy is already in one of the areas or the link pack area), a request block for the SVC routine is built and enqueued to the appropriate TCB, and the SVC routine is given control. Upon completion, the SVC routine returns control to the supervisor.

The fourth function, processing of I/O operations, is performed by access method routines and the I/O supervisor. The access method routines reside on SYS1.SVCLIB and operate from the link pack area, or the region of their associated task. These routines are loaded by the open SVC routine, and are entered via a branch instruction that is part of the expansion of the macro instruction for that access method.

The I/O supervisor is part of the nucleus. The portion of the I/O supervisor that processes I/O operations is a type 1 SVC routine invoked by an EXCP macro instruction normally issued by an access method routine.

ASSIGNING SPACE ON VOLUMES

Assigning of tracks and cylinders on direct-access volumes is performed by the direct-access device space management (DADSM) SVC routines of data management. These routines are used primarily by job management routines during the initiating of a job step to get space for output data sets. The DADSM routines are also used by other data management routines to increase the space already assigned to a data set, and to release space no longer needed. The DADSM routines are described in the publication IBM System/360 Operating System: Direct-Access Device Space Management, Program Logic Manual.

The DADSM routine controls allocation of space on direct-access volumes through the volume table of contents (VTOC) of that volume. The VTOC is built when the volume is initialized by the direct-access storage device initialization (DASDI) utility program. The VTOC indicates the current usage of the space on the volume.

The VTOC is a collection of data set control blocks (DSCBs). Each DSCB corresponds either to a data set currently residing on the volume, or to contiguous, unassigned tracks on the volume. DSCBs for data sets are the data set labels, which contain characteristics of the data set and the tracks on which it resides. DSCBs for unassigned tracks indicate the locations of unassigned, contiguous tracks.

When space is needed on a volume, the DADSM routines check the VTOC for enough contiguous, available tracks to satisfy the request. If there are not enough contiguous tracks, the request is filled using up to five noncontiguous groups of free tracks. The appropriate DSCBs are modified to reflect the assignment of the tracks.

When space is released, the DADSM routines delete the DSCB of the deleted data set from the VTOC. A DSCB is built or modified to indicate that the tracks containing the deleted data set can be reallocated.

MAINTAINING THE CATALOG

The catalog is a collection of data sets that indicates the volumes on which cataloged data sets reside. The catalog management routines of data management maintain the catalog, and locate cataloged data sets.

To maintain the catalog, catalog management routines create and delete indexes, and catalog and uncatalog data sets. To locate a data set, catalog management routines search through the indexes, specified in the qualified name of the data set, for the index entry containing the last part of

the qualified name. This index entry contains the serial number (or numbers) and device type of the volume (or volumes) on which the data set resides. The catalog management routines are described in the publication, IBM System/360 Operating System: Catalog Management, Program Logic Manual.

The catalog management routines are used primarily by job management routines and IEHPROGM utility program. Job management routines may invoke the catalog management routines during the initiation and termination of a job step. During initiation, a catalog management routine locates cataloged data sets. During termination, a catalog management routine may catalog or uncatalog data sets referred to during the job step and specified for the catalog. The IEHPROGM utility program invokes catalog management routines to perform any of their functions except locating a data set. Processing programs can also invoke the catalog management routines via the CATALOG, INDEX, and LOCATE macro instructions.

SUPPORT PROCESSING FOR I/O OPERATIONS

Support processing for I/O operations has three subdivisions:

- Open processing which is required before I/O operations can be performed.
- Close processing which is required after I/O operations have been completed.
- End of volume (EOV) processing which is required when space for a sequential data set on either a direct or sequential access volume is exhausted during an I/O operation.

The routines that perform these functions are the I/O support routines, (open, close, and EOV). Their operation is discussed in the publication IBM System/360 Operating System: Input/Output Support (OPEN/CLOSE/EOV), Program Logic Manual.

OPEN PROCESSING

Before any information can be read from or written into a data set, initialization must be performed. This initialization is referred to as "opening" the data control block of the data set, and basically consists of:

- Ensuring that the volumes required for reading or writing the data set are mounted.
- Constructing control blocks required by the I/O supervisor to initiate the I/O operations.
- Loading the access method routines that are to process the I/O operations on the data set.

Insuring Proper Volume Mounting

The Open routine determines whether the volumes required for the data set are mounted on devices assigned to the job step. If the volumes are not mounted, the Open routine issues a mounting message to the operator and, after mounting has been performed, checks the volume labels to verify that the proper volumes have been mounted.

The Open routine then locates the data set (or the space to receive the data set) on the volume. For tape, the volume is positioned; for direct-access volumes, the DSCB is read into main storage.

Constructing Control Blocks

The Open routine constructs (or completes construction of) control blocks that are used when the data set is to be read or written. These are the data control block (DCB), job file control block (JFCB), header labels or data set control block (DSCB), and the data extent block (DEB).

Completing the DCB, JFCB, and DSCB: The DCB, JFCB, and DSCB are in various stages of completion before the DCB is opened. The Open routine completes them by merging information from one block to another. There are two distinct merge operations, the forward merge and the reverse merge.

The forward merge is the passing information first from the DSCB (or standard tape label) to the JFCB, and then from the JFCB to the DCB. Information is passed only when the field of the block receiving the information is empty. The forward merge does not change any fields that already contain information.

The reverse merge is the passing of information from the DCB back to the JFCB, and then to the DSCB (or header label). This merge occurs after the forward merge, and after the Open routine has given control to any user-written DCB-exit routines. When the associated data set is for output, the reverse merge not only fills empty fields in the JFCB and DSCB, but also overrides existing fields except the DSORG field. When the data set is for input, the DCB to JFCB merge fills only empty fields; no JFCB to DSCB merge is performed. Figure 12, which is an expansion of Figures 5 and 6, shows the flow of information in the

forward and reverse merges. The numbers indicate the sequence in which the flow occurs.

Constructing the DEB: A data extent block (DEB) is built for each DCB being opened. The DEB contains the volume location (or locations) of its associated data set, and the names of the access method routines that are to be used on this data set. The DEB is used by the I/O supervisor in starting an I/O operation. The relationship of the DEB to other control blocks is shown in Figure 12.

The DEB is built from information in the DCB, JFCB, DSCB, and UCBs (unit control block) of the devices associated with the data set. A UCB exists for each device in the system. UCBs are built when the system is generated, and contain characteristics of the devices that they represent.

DEBs are built in the system queue area so that their contents cannot be changed by processing programs. All DEBs for a given task are placed in a queue originating at the TCB of that task.

The DEB is built by an access method executor module. These modules operate as part of the Open routine but perform processing unique to the access method to which they apply. The operation of the access method executors is described in the various access method program logic manuals.

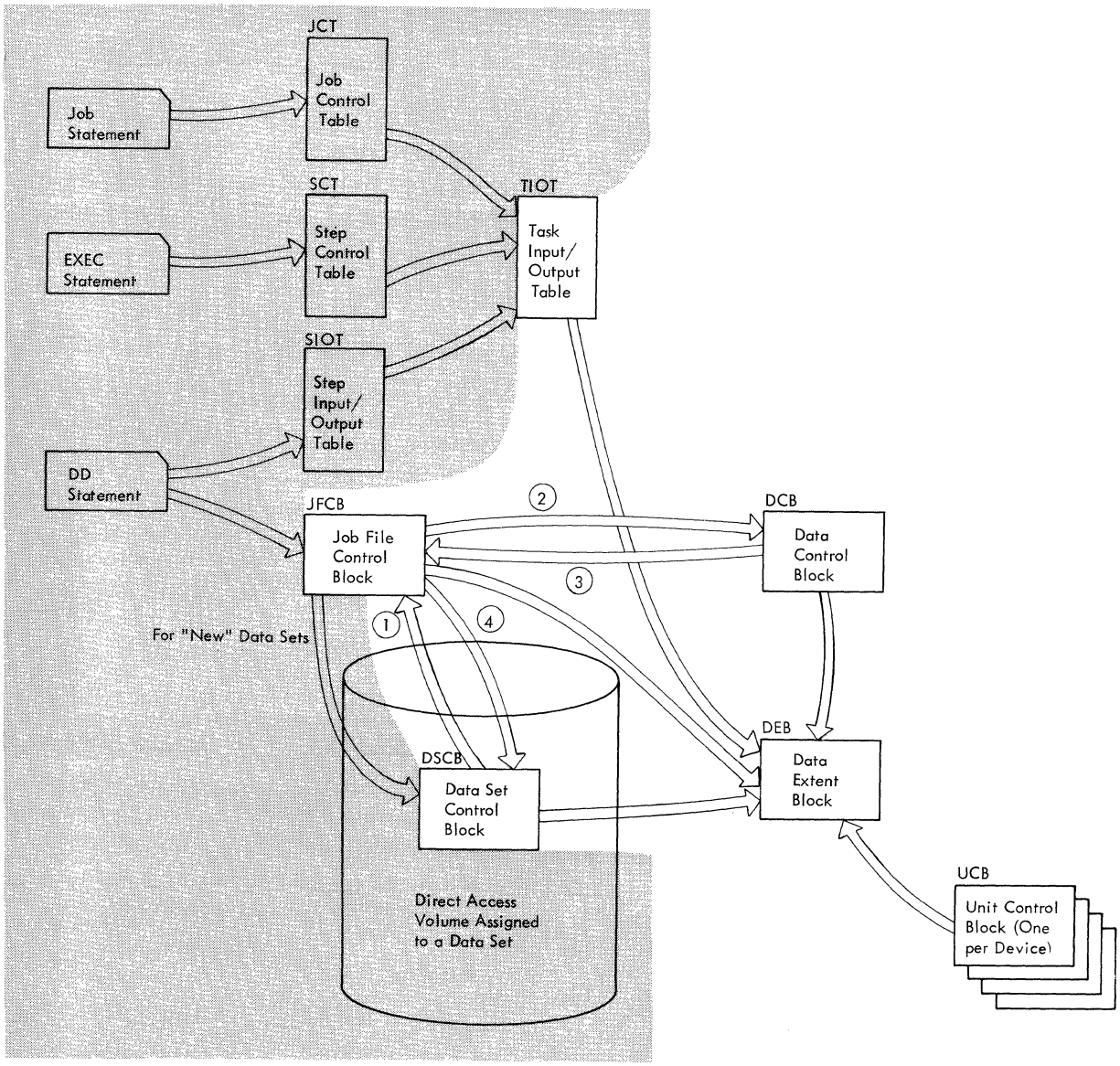


Figure 12. Flow of Information During the Merges of the Open Routine

Loading Access Method Routines

The Open routine uses the DCB to determine which access method is to be used on the associated data set. The executor of that access method then determines which routines of the access method are required to operate on the data set. These routines are then loaded into the appropriate region unless they are already in the link pack area.

CLOSE PROCESSING

After I/O operations on a data set are complete, the DCB of that data set should be closed. The Close routine restores the DCB fields that were filled by the forward merge during open, processes labels, determines volume disposition, and deletes the unneeded access method routines.

Label processing includes the building of trailer labels for output data sets on tape, and the updating of DSCBs of data sets with OUTPUT, OUTIN, or INOUT disposition.

Volume disposition includes not only dismounting instructions to the operator, but also the writing of tape marks and the positioning of tape reels.

If the access method routines associated with this close operation are not in the link pack area and are not required for any more I/O operations in the region, the storage that these routines occupy is released.

END-OF-VOLUME PROCESSING

End-of-volume (EOV) processing is performed when end-of-data set or end-of-volume conditions occur during an I/O operation on a sequentially organized data set. When a routine of a sequential access method encounters a tape or file mark or an end-of-volume condition, the routine issues an SVC instruction to pass control to the EOV routine.

EOV processing consists primarily of verifying and constructing labels. If the data set for which the condition occurred is continued on another volume, the EOV routine issues mounting instructions for the next volume and checks the mounting.

If the EOV condition occurred because direct-access volume space assigned to an output data set is used, the EOV routine invokes a DADSM routine to obtain more space for the data set.

PROCESSING I/O OPERATIONS

The processing of I/O operations is performed in two distinct parts: processing required to start the operation, and processing required when the operation is terminated.

STARTING AN I/O OPERATION

In the operating system, two portions of the control program are normally involved in starting an I/O operation requested by a processing program. These are:

- Access method routines, which organize the information required to initiate the I/O operation.
- The EXCP routine of the I/O supervisor, which initiates and supervises the I/O operation.

Figure 13 shows the relationship that exists between a processing program, an access method, and the I/O supervisor.

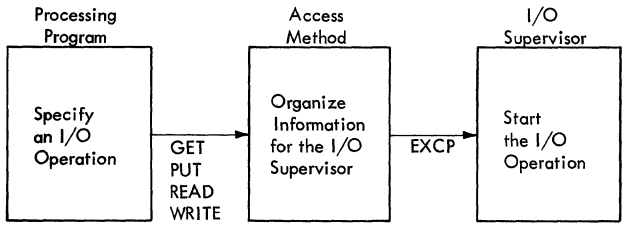


Figure 13. Relationship Between a Processing Program, an Access Method and the I/O Supervisor

The expansion of an I/O macro instruction specified in the processing program results in a branch to the access method routine. This routine gathers information used to initiate the I/O operation and places this information in control blocks. The routine then issues an EXCP macro instruction causing an SVC interruption. The SVC Interruption Handler gives CPU control to the I/O supervisor, which either starts or queues the I/O operation.

After the EXCP routine has completed its operation, it passes control to the Type 1 SVC Exit routine which returns control to the access method routine. This routine finishes its processing before passing control to the processing program that issued the I/O macro instruction. Figure 14 shows the flow of control for an I/O operation.

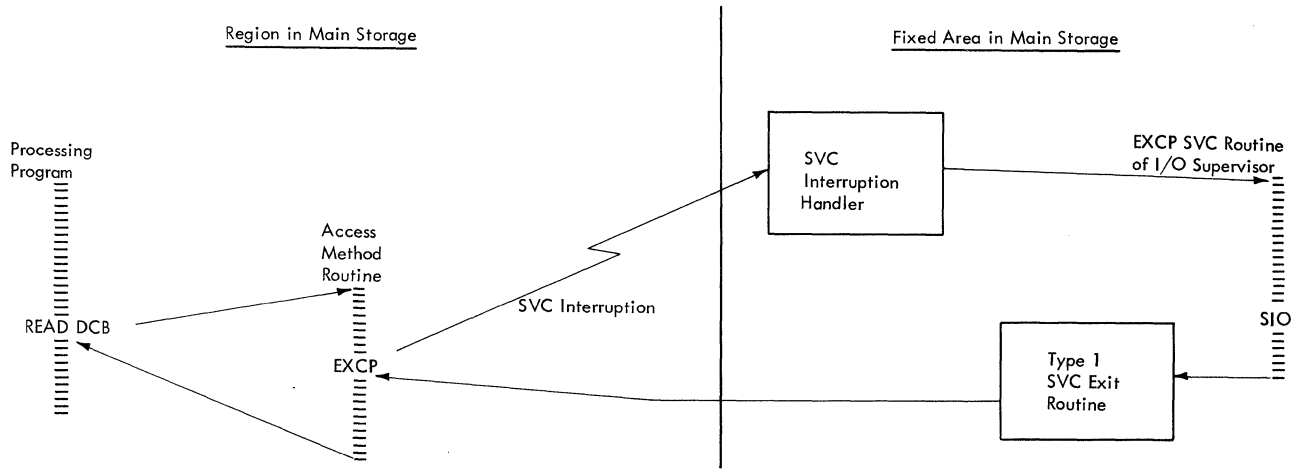


Figure 14. Flow of Control for an I/O Operation

Access Methods

Access method routines prepare information required by the I/O supervisor to start an I/O operation. Routines of certain access methods also perform services that are not directly associated with the actual I/O operation. These services include allocating and controlling buffer areas, moving data to and from the buffer areas, and blocking and deblocking records. When the user assembles his program, he indicates an access method in the DCB of the data set. When the DCB is opened, the access method routines to be used on the data set are brought into the appropriate region, unless these routines are already there or in the link pack area.

Routines of every access method construct a number of input/output blocks (IOBs) and channel programs for the I/O supervisor. The number of IOBs and channel programs built for a given data set depends on the number of main storage areas used by the data set, and the number of I/O operations to be performed on the data set.

An IOB contains information required by the I/O supervisor to start an I/O operation. Each IOB points to a channel program that is to be executed. When the operations for that channel program terminate, the channel status word (CSW) is stored in the IOB associated with the channel program.

A channel program is a group of one or more channel command words (CCWs) that specify I/O operations, and indicate the main storage areas for the data involved in these operations. The CCW and CSW formats are described in the publication IBM System/360 Principles of Operation.

For some access methods, the IOBs and CCWs are partially built by the Open Executor routine of that access method when the DCB is opened; these IOBs and CCWs are completed by the access method routine, when the I/O operation is being processed. For other access methods, the IOBs and CCWs are completely built during the processing of the I/O operation.

EXCP Routine

The EXCP SVC routine is the portion of the I/O supervisor that initiates I/O operations. This routine receives control from the SVC Interruption Handler and builds a request element for the requested I/O operation.

Whenever an I/O operation is in process, the UCB for the device points to the request element for that operation. When an operation cannot be started, the request element for that operation is placed in a queue for the device. This queue is the request element table.

The EXCP routine determines whether the I/O device associated with the operation is free, and if so, whether any channel associated with the device is free.

When both the device and an associated channel are free, the EXCP routine issues a START I/O (SIO) instruction to initiate the operation. If the device or all associated channels are busy, the request element is placed in the queue for that device. The elements that make up this queue are contained in the request element table.

The EXCP routine is described in the publication IBM System/360 Operating System: Input/Output Supervisor, Program Logic Manual.

After the I/O operation is started, an indicator is set in the UCB to show that the device is now busy, and a pointer to the request element is placed in the UCB. When an I/O interruption occurs, the I/O supervisor uses the request element in the UCB to determine the request that has been executed.

TERMINATING AN I/O OPERATION

I/O operations terminate either normally because the operation is completed, or abnormally because an error is detected. When an I/O operation terminates, an I/O interruption occurs, causing CPU control to be passed first to the I/O Interruption Handler and then to the I/O interruption supervisor portion of the I/O supervisor to process the interruption.

The I/O interruption supervisor posts the completion of the I/O operation, schedules error routines (i.e., places a request block for the routine in the request block queue) when the operation terminated abnormally, and, if possible, starts another I/O operation on the channel. The I/O interruption supervisor returns CPU control to the interruption handler. The I/O interruption supervisor is described in the publication IBM System/360 Operating System: Input/Output Supervisor, Program Logic Manual.

access method executor: A routine that is entered during the performance of the open, close, or end-of-volume function, and performs processing unique to the access method to which it applies. These routines operate in supervisor state from SVC transient areas, or the link pack area.

automatic command: A command specified during system generation, and executed after NIP in response to the AUTO parameter of the SET command.

catalog: One or more data sets that specify the volume or volumes upon which cataloged data sets reside.

command processing: The reading, analyzing and performing of commands issued via a console device or an input job stream.

console-communications task: The reading and analyzing of operator commands issued via a console device. Some commands are completely performed during this task; others require additional tasks.

contents directory: A series of queues that indicate the routines in the regions of the dynamic area and in the link pack area.

control volume: A direct-access volume that contains one of the data sets that make up the catalog.

disabled: (masked) A state of the CPU that prevents the occurrence of certain types of interruptions.

dynamic area: That portion of main storage that is subdivided into regions for use by the programs performing job steps and system tasks. The dynamic area is all the storage between the system queue area and the link pack area.

enabled: (interruptable) A state of the CPU that allows the occurrence of certain types of interruptions.

executor: See access method executor.

fixed area: That portion of main storage occupied by the resident portion of the control program (nucleus).

index: A record that is part of the catalog structure. Indexes are used to locate data sets.

initiating task: The job management task that controls the selecting of a job and preparing of the steps of that job for execution.

input work queue: The tables and control blocks built during reading tasks from the data in the JOB, EXEC, and DD statements. This queue provides input to initiating tasks.

interruptable: See enabled.

job processing: The reading of control statements from an input job stream, the initiating of job steps defined in these statements, and the writing of system messages and SYSOUT data sets.

job scheduler: The routines that perform the job processing functions (i.e., reading, initiating, and writing tasks) of job management.

job step task: The first task created for a job step. That task created in response to an ATTACH macro instruction issued by an initiator routine.

link pack area: The area of main storage that contains selected reenterable routines from SYS1.SVCLIB and SYS1.LINKLIB, and a list of track addresses of routines on SYS1.LINKLIB. The routines are loaded at IPL time, and can be used concurrently during all tasks in the system.

masked: See disabled.

master scheduler task: The command-processing task of searching a queue of pending commands, and of attaching a task for executing each of these commands.

multiprogramming: Using a computing system to fulfill two or more separate programming requirements concurrently. This is normally achieved via a supervisory program that makes decisions based on various conditions in the system, and then gives control to one of several separate programs.

nucleus: That portion of the control program that is loaded into the fixed area of main storage from SYS1.NUCLEUS at IPL time and is never overlaid by another part of the operating system.

pseudo-clock: A main storage location used by timer supervision routines to calculate timer intervals and time-of-day.

reading task: The job management task that controls the reading and interpreting of control statements, and the reading and analyzing of operator commands in an input stream.

region: A subdivision of the dynamic area that is allocated to a job step or a system task.

reenterable: The attribute of a program that allows a single copy of the program to be used concurrently in the performance of two or more tasks.

subpool: All the 2048 (2K) blocks of main storage allocated under one subpool number for a particular task.

subtask: Any task that is attached by a routine of some other task. Although the definition applies to almost every task (both system and user) in the system, the term subtask is most commonly used to refer to a task attached by a routine of a job step.

system queue area: The main storage area, adjacent to the fixed area, which is reserved for control blocks and tables built by the control program.

SVC routine: A control program routine that performs or initiates a control program service specified by a supervisor call (SVC).

system task: A control program function that is performed under control of a task control block.

SYS1.LINKLIB: The partitioned data set that contains the IBM-supplied processing programs and part of the nonresident portion of the control program. It may also contain user-written programs.

SYS1.LOGREC: The partitioned data set reserved for data gathered by the SER0 or SER1 routines. This data is hardware and software information required to diagnose machine-check interruptions and channel and I/O errors.

SYS1.NUCLEUS: The partitioned data set that contains the resident portion of the control program (i.e., the nucleus).

SYS1.SVCLIB: The partitioned data set that contains the nonresident SVC routines, nonresident error-handling routines, and the access-method routines.

transient areas: Main storage areas defined in the nucleus and reserved for either nonresident SVC routines or nonresident error-handling routines.

wait state (task): The condition of a task when it is unperformable because some event such as the completion of an I/O operation has not occurred.

wait state (system): The condition of the CPU when all operations are suspended. This condition is indicated by a bit setting in the current program status word.

work queue entry: The control blocks and tables created from one job in an input job stream and placed in the input work queue or one of the output work queues.

writing task: The job management task that controls the transferring of system messages and SYSOUT data sets from the direct-access volume on which they were initially written to a specified SYSOUT volume.

APPENDIX A: LIST OF ACRONYMS

The following list contains the full name associated with the acronyms used in this publication:

Acronym	Name
CCW	channel command word
CDE	contents directory entry
CPU	central processing unit
CSCB	command scheduling control block
CSW	channel status word
CVT	communications vector table
DADSM	direct-access device space management
DASDI	direct-access storage device initialization
DCB	data control block
DD	data definition
DEB	data extent block
DQE	descriptor queue element
DSB	data set block
DSCB	data set control block
ECB	event control block
EOV	end-of-volume
EXCP	execute channel program
EXTR	end-of-task exit routine
EXEC	execute
FBQE	free block queue element
INIT	initiator
IOB	input/output block
IPL	initial program loading
JCT	job control table
JFCB	job file control block
LTPC	local time pseudo-clock
MVT	multiprogramming with a variable number of tasks
NIP	nucleus initialization program
PQE	partition queue element
PSW	program status word
RB	request block
RDR	reader
SCT	step control table
SER	system environment recording
SHPC	six hour pseudo-clock
SIOT	step input/output table
SMB	system message block
SPQE	subpool queue element
SVC	supervisor call
SYSOUT	system output
SYSRES	system residence volume
TCB	task control block
TIOT	task input/output block
TQE	timer queue element
T4PC	twenty-four hour pseudo-clock
UCB	unit control block
VTOC	volume table of contents
WTR	writer
XCTL	transfer control

APPENDIX B: MVT CONTROL PROGRAM LOGIC MANUALS

The following list contains the names, form numbers, and abstracts of all the control program logic manuals that discuss the MVT configuration.

MVT Supervisor — Form Y28-6659

This publication describes the internal logic of the MVT supervisor. The MVT supervisor is one part of the control program of the IBM System/360 Operating System. The supervisor controls the basic computing system and programming resources needed to perform several data processing tasks concurrently. Specifically, it was designed to:

- Handle interruptions
- Supervise tasks
- Control programs in main storage
- Control main storage itself
- Supervise the timer
- Supervise console communications and the system log
- Supervise exiting procedures
- Supervise termination procedures

MVT Job Management — Form Y28-6660

This publication describes the internal logic of the job management routines for the MVT control program of the IBM System/360 Operating System. Included are discussions of input stream processing, work queue management, job initiation and termination, I/O device allocation, system output processing, and the scheduling and execution of operator commands.

Initial Program Loading/Nucleus Initialization Program — Form Y28-6661

This publication describes the internal logic of the Initial Program Loader (IPL) program and the Nucleus Initialization Program (NIP). The Initial Program Loader prepares main storage to receive the nucleus and then loads the nucleus. The Nucleus Initialization Program initializes the resident part of the control program and prepares main storage for control program operation.

Input/Output Supervisor — Form Y28-6616

This publication describes the operation of the I/O supervisor within the IBM System/360 Operating System control program. The I/O supervisor's components, the EXCP supervisor and the I/O interruption supervisor, are discussed in detail to show the internal structure and logic involved in the control of I/O devices and channels.

Catalog Management — Form Y28-6606

This manual provides detailed information on catalog management routines. These routines record identification of volumes used by data sets by maintaining information in logical records called indexes. The functions and structures of the routines are described, as are their relationships to other portions of IBM System/360 Operating System. This manual also describes the structure of catalog data sets that contain the indexes processed by catalog management routines.

Direct-Access Device Space Management — Form Y28-6607

This manual provides detailed information on direct-access device space management (DADSM) routines. These routines control the use of external direct-access storage by maintaining the information in data set control blocks. The functions and structures of the routines are described, as are their relationships to other portions of IBM System/360 Operating System. This manual also describes the structure of volume tables of contents which are processed by DADSM routines.

Input/Output Support (OPEN/CLOSE/EOV) — Form Y28-6609

This publication describes the internal logic of IBM System/360 Operating System input/output support. The discussion includes the relation of I/O program. Detailed descriptions of the open, close, and EOV routines provide the basis for the discussions of the other I/O support routines openJ, RDJFCB, Tclose, and FEOV.

Sequential Access Methods — Form Y28-6604

This publication describes the internal logic of the routines of the queued sequential access method, the basic sequential access method, and the basic partitioned access method of IBM System/360 Operating System.

Indexed Sequential Access Methods — Form Y28-6618

This publication describes the program logic of the two indexed sequential access methods: the queued indexed sequential access method (QISAM) and the basic indexed sequential access method (BISAM). It also discusses the relationship of indexed sequential access method routines to other parts of the control program.

Basic Direct-Access Method — Form Y28-6617

This publication describes the internal logic of the IBM System/360 Operating System basic direct-access method.

IBM S/360 Printed in U.S.A. Y28-6658-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM System/360 Operating System
MVT Control Program Logic Summary

Form Y28-6658-0

- Is the material:

	Yes	No
Easy to read?	☐	☐
Well organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐

- How did you use this publication?
  - ☐ As an introduction to the subject
  - ☐ For additional knowledge

  Other .....

- Please check the items that describe your position:
  - ☐ Customer personnel
  - ☐ IBM personnel
  - ☐ Manager
  - ☐ Systems Analyst
  - ☐ Operator
  - ☐ Programmer
  - ☐ Customer Engineer
  - ☐ Instructor
  - ☐ Sales Representative
  - ☐ Systems Engineer
  - ☐ Trainee

  Other .....

- Please check specific criticism(s), give page number(s), and explain below:
  - ☐ Clarification on page(s) .....
  - ☐ Addition on page(s) .....
  - ☐ Deletion on page(s) .....
  - ☐ Error on page(s) .....

Explanation:

- Thank your for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This publication is one of a series which serves as reference for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

IBM S/360 Printed in U.S.A. Y28-6658-0

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]