File No. S360-20
GC28-6535-7

OS

IBM Systems Reference Library

IBM System/360 Operating System: Concepts and Facilities

This publication describes the basic concepts of the IBM System/360 Operating System (the operating system) and guides the programmer in the use of its facilities.

The operating system is a comprehensive set of language translators and service programs operating under the supervision and coordination of an integrated control program. It assists the programmer by extending the performance and application of the computing system.

Information concerning Model 195 support is for planning purposes only.

Preface

This publication introduces the facilities of the IBM System/360 Operating System, and explains how they work together.

The first section is a general discussion designed to familiarize you with operating system concepts and terminology. A section on designing programs is followed by detailed discussions of the three main operating system functions. Language translator facilities are compared, and examples illustrating the efficient use of the system are given. The two final sections discuss recovery management and multiprocessing. A summary of the publication and a glossary of new terms are included at the end of the publication.

The emphasis in this book is on what elements are contained in the operating system and how these elements work together.

PREREQUISITE PUBLICATION

IBM System/360 Operating System: Introduction, GC28-6534

This publication describes the general organization, function, and application of the operating system. It also introduces other related publications and describes their contents.

Seventh Edition (June, 1970)

This is a major revision of, and obsoletes, GC28-6535-5. Information added for this new edition includes: extensions to System Management Facilities; Alternate Path Retry and Dynamic Device Reconfiguration; Operating System Volume Statistics; Remote Job Entry; 7094/Model 85 Integrated Emulator; MFT with subtasking; a new system utility program, IEHATLAS; in-stream procedures; direct system output writers; handling of machine malfunctions on the Model 195 by SER routines (Model 195 information is for planning purposes only); Channel-Check Handler Dynamic Loading; SER1 Wait State support; new sections on the Error Recovery Procedures and the OLTEP programs, and the STAE and SPIE macro instructions; corrections to FORTRAN H data management capabilities. The document should be reviewed in its entirety. Other changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

This edition applies to release 19, of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

Contents

Illustrations

Figures

Tables

Summary of Major Changes--Release 19

Item	Description	Areas Affected
System Management Facilities	This facility has been expanded and is now available with MFT. Job step CPU timing now functions in MFT as it did in MVT. You may limit the number of logical records processed by SYSOUT. SMF routines can now be used to determine data set activity for each problem program and also to acquire volume usage information for direct access devices. Output supplied by the SMF routines can be used to create and maintain inventories on direct access and tape devices.	50,52,56
I/O RMS (Alternate Path Retry and Dynamic Device Reconfiguration)	Alternate Path Retry ensures that a different channel will be tried on retry of channel-detected errors. This is done by marking failing paths offline and indicating the offline paths to the I/O Supervisor. Dynamic Device Reconfiguration can be requested by the operator any time during execution, or by the system after a permanent error for all demountable volumes. Dynamic Device Reconfiguration monitors the validity of the requests and responses of the operator, and attempts to force the operator to complete system-requested swaps before initiating his own.	58,59,61, 64,67,69
Operating System Volume Statistics	This facility monitors read and write errors. Through its two options -- Error Statistics by Volume and Error Volume Analysis -- statistics such as volume serial number and the number of temporary and/or permanent read or write errors, may be collected and a limit placed on the number of temporary read or write errors. Currently, this facility supports only tape volumes.	40,69
Remote Job Entry (RJE)	This facility provides job entry capability for users at remote keyboard terminals connected to a System/360 by communications lines. Once a job is entered into the job stream by RJE, execution of the job proceeds under supervision of the operating system.	34,74
7094/Model 85 Integrated Emulator	Integrated emulator programs allow object programs written for one system to be executed on another system using a compatibility feature that consists of added hardware and microprogrammed routines that aid emulation. The integrated emulator program is executed under the control of the operating system in a multiprogramming environment.	34,74
MFT Subtasking	The ATTACH macro instruction may now be used in an MFT system to create subtasks.	16,17,21,48, 50,51,53,54, 55

(Continued)

(Continued)

Item	Description	Areas Affected
IEHATLAS	A new system utility program: it is used to recover usable data from a defective track, assign an alternate track, and merge replacement data with the recovered data onto an alternate track.	15,70
In-Stream Procedures	An in-stream procedure is a set of job control statements placed in the input stream. They can be used any number of times during the job by naming the procedure in an EXEC statement.	43,70
Direct SYSOUT Writers	Direct system output writers control the writing of output data sets directly to the system output device during execution of the job.	44,47,54,69
System/360 Model 195	Machine malfunctions on the Model 195 are recorded by the SER routines. Note: This information is for planning purposes only.	58
Channel-Check Handler Dynamic Loading	CCH routines may now be loaded dynamically at Initial Program Loading (IPL) time or at nucleus initialization (NIP) time.	61
SER1 Wait State	SER1 ability to recover from machine checks has been extended.	61
Online Test Executive Program (OLTEP)	OLTEP has been added to this publication, although it has nothing specifically to do with this Release. It was added to give operating system recovery management more perspective. OLTEP is a part of a set of programs that can be used to test I/O devices, control units, and channels concurrently during the execution of programs.	62,72
Error Recovery Procedures (ERPs)	ERPs have been added to this publication. Although they have nothing specifically to do with this Release, they were added to give operating system recovery procedures more perspective. Error Recovery Procedures are standard procedures designed to ensure that all the routines that test particular devices provide a uniform type and quality of information.	60,61,69
Set Program Interrupt Exit (SPIE) macro instruction	SPIE has been added to this publication. Although it has not been modified for this release, it was added to expand the discussion on task termination. SPIE allows the user to request program error exits.	50
Specify Task Abnormal Exit (STAE) macro instruction	STAE has been added to this publication. Although it has not been modified for this release, it was added to expand the discussion on task termination. STAE allows the user to exit to a routine before ABEND processing.	50
FORTRAN H	Corrections are made to the data management language comparison table for FORTRAN H which may be used for scheduling for I/O operations, processing a direct data set, record overflow, and write validity checks.	41
Service Aids	Seven new programs designed to aid diagnosis of problems.	25

Section 1: Introduction

A data processing installation exists for one main purpose: to do work. To fulfill this purpose economically and efficiently, all the installation's machine facilities should be kept as busy as possible. An operating system also exists for one purpose: to increase the productivity of a computer installation. It achieves its purpose by managing the allocation of all available resources, including the central processing unit, main storage, input/output devices, and any programs that are in, or are part of, the system. Therefore, getting work done is the common denominator of an installation and an operating system.

The large volume of work that an installation must do cannot all be done at the same time, simply because there are not enough resources. Therefore, the work load is divided into independent jobs. When a job is introduced to the IBM System/360 Operating System (the operating system), the required resources are allocated to the job. Now the operating system has a task to perform, which is your job. Thus, the same unit of work is viewed differently by the user and by the operating system, as depicted in Figure 1.

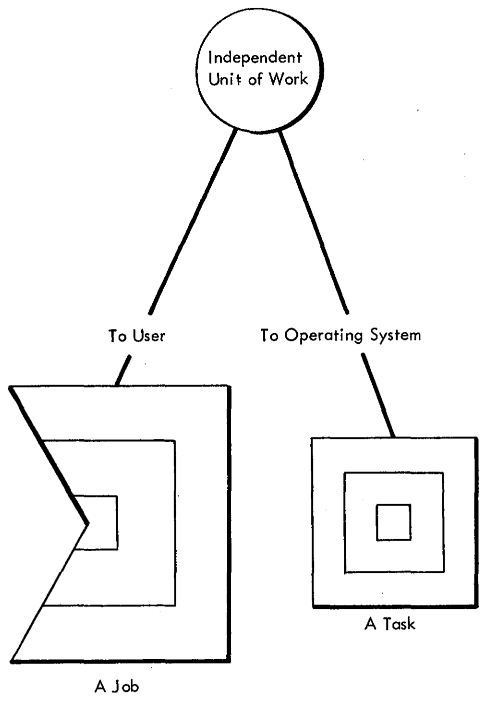


Figure 1. Work to the User and to the Operating System

A certain machine configuration and a certain operating system configuration are selected at an installation, depending on the type and volume of work that must be done there. The machine configuration must have certain components, such as a central processing unit and I/O devices; the operating system configuration must also have a certain component: a control program (the routines in the operating system that manage resources, implement data organization and communications conventions, and contain privileged operations). This control program may handle one task at a time, it may supervise execution of a fixed number of tasks at the same time, or it may supervise execution of a variable number of tasks at the same time. The type of control program selected depends on how much computing power is needed. However, regardless of the type of control program selected, its major functions are the same.

The Operating System and the Concept of Work

Getting work done is actually problem solving. The operating system enables you to concentrate on this goal by performing many routine data processing operations for you.

To be effective, the operating system must be general enough to accommodate a variety of applications on a wide range of hardware configurations. It is, therefore, made up of a general library of programs that can be tailored to the installation's requirements. Required system programs are selected, user-written programs added, and existing programs updated to meet the installation's needs.

The programs that compose the operating system are classified as a control program and processing programs. The three main functions of the control program are to:

1. Accept and schedule jobs in a continuous flow (job management);

2. Supervise, on a sequential or priority basis, each unit of work to be done (task management);

3. Simplify storage, retrieval, and maintenance of all data, regardless of the way it is organized and stored (data management).

The processing programs consist of language translators (such as the FORTRAN compiler), service programs (such as the Linkage Editor), and problem programs (such as your programs). You use the processing programs to define the work that the computing system is to do and to simplify program preparation.

The elements of the operating system are shown in Table 1. How these elements work together is described in general in the remainder of this section, and in more detail throughout the publication. Note that the description includes facilities that may optionally be omitted from the system.

•Table 1. IBM System/360 Operating System Elements

Control Program Elements		
Job Management	Data Management	Task Management
Processing Program Elements		
Languages	Service Programs	Application Programs
ALGOL	Data Set Utilities	User Written
Assembler	Independent Utilities	
COBOL	Linkage Editor	
FORTRAN	Sort/Merge	
PL/I	System Utilities	
RPG	TESTRAN	
	7094/M85 Integrated Emulator	

Operating System Concepts

To illustrate operating-system concepts, you will be introduced to the system by following the flow of one job through the system. The job involves compiling a program, loading it, and executing it. The emphasis is on what you must do and what the system will do in response.

Two illustrations are used to help you understand the flow of work through the system. The first is a table, which has already been shown in its completed form. How it reached this form will be shown as the example proceeds.

The second is a diagram to be developed along with the example. Figure 2 is first shown in its completed form. This diagram will be used and detailed throughout the remainder of the publication, both as a memory aid and to illustrate more clearly the unity of the system.

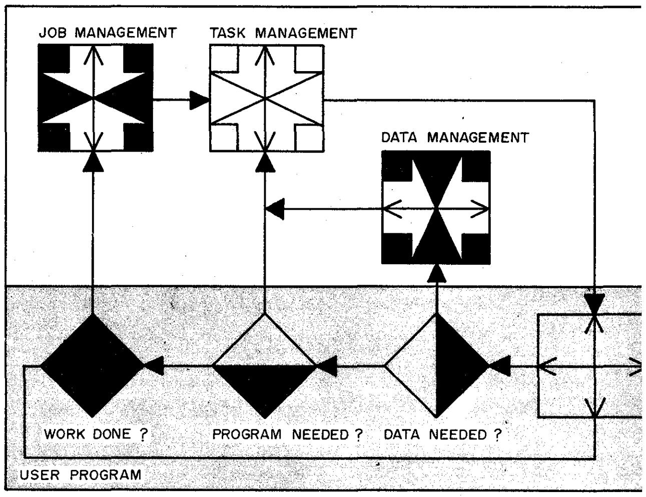


Figure 2. User Requests and System Response

What the User Must Do

Before any problems are solved and work is done, the necessary operating system components must be selected. This is done through system generation. The operating system produced is composed of the standard component included in every operating system (a control program), those optional components selected from available processing programs, and any additions you provide. The control program and most processors have a variety of optional features that may be selected for particular needs. The resulting system is tailored to your installation's work load and machine configuration. Furthermore, through the use of the nucleus initialization program (NIP), last-minute changes can be made to certain options specified during system generation. The operator makes these changes through the console.

Now that you have your system, you want to begin getting work done. You are given a problem and you must solve it; that is, design and code a program. To design a program that makes the most efficient use of the operating system's facilities, you must consider many factors. Some of the more important ones are the amount of main storage available, the programming and machine resources required, and whether this program is to be a resource for other programs.

In the simplest case, there may be enough main storage available so that your program can be self-contained: the program does not need to call any other programs during execution. This is called a simple structure; the program is loaded and executed as an entity.

However, your program may be too large to be in storage at one time. You can handle this situation by dividing the program into smaller, more easily managed subprograms. These subprograms could be designed so that one segment -- a root segment -- is always resident in storage, while the other segments overlay each other as required. This is called a planned overlay structure; the subprograms are loaded and executed serially.

Another possibility is to design your program so that the subprograms are executed one after the other, while programming resources are called when needed during execution. This is called a dynamic structure. It is advantageous where existing programs must be incorporated into a larger program.

One type of dynamic structure is one in which your program executes in parallel with other tasks in the system: the subprograms are loaded as required and programming resources are called dynamically, while the same process is going on at the same time for one or more other tasks.

A final point to consider when designing your program is whether it is to be a system program. There are three kinds of programs:

1. If the program modifies itself during execution, a fresh copy must be brought in every time the resource is needed. This is a non-reusable program.

2. If the program nullifies any changes during execution, a single copy may be used over and over again, as long as each request is completely honored before another is entertained. This is a serially reusable program.

3. Finally, you can design your program so that it does not modify itself during execution. In this case, a single copy may begin execution for request A, then completely honor a request from B, and then go back to finish executing for request A. This is a reenterable program.

During execution your program could be depicted as shown in Figure 3.

Once you have designed your program, you can code it. The source languages available are shown in Table 1A.

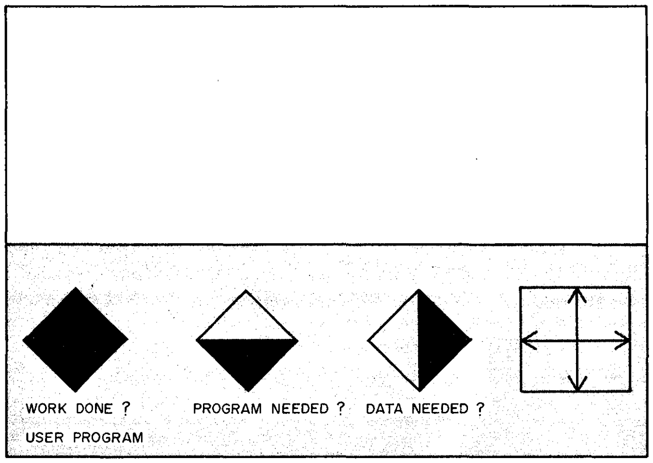


Figure 3. User Requests

Table 1A. Operating System Elements

Control Program Elements		
Processing Program Elements		
Languages		Application Programs
ALGOL Assembler COBOL FORTRAN PL/I RPG		

When your coding is complete, you have a source program or source module, which is transcribed onto cards or tape. It can then be processed by the system, as long as the system knows:

1. What resources the compiler needs to translate your source language program into machine language. (The output of a compilation or assembly is in machine language and is called an object module.)

2. What must be done to put the object module into a loadable format.

3. What resources your program needs and where your data is.

You provide this type of information through a job control language (JCL).

The primary purpose of the job control language is to communicate with the scheduling components of the operating system. You tell the system how you have divided your jobs into job steps (those units of work, each of which is associated with one processing program and related data), where your data is, and what resources are needed to execute each step.

The first step in the example is to process your source module. The processing is illustrated in Figure 4.

Since a language translator does not resolve all addresses, symbolic references, etc., an object module cannot be executed. It must be processed by a service program called the linkage editor that resolves these references and places the object module in a form suitable for loading. The output of the linkage editor is called a load module. Figure 5 depicts the production of a load module.

The linkage editor is one service program provided by the operating system, as shown in Table 1C. An alternate way of processing an object module is through the loader. The loader will load into main storage for execution object modules produced by a language processor and load modules produced by the linkage editor. Optionally, it can resolve external references. The loader does not produce load modules for program libraries. The loader is another service program provided by the operating system.

To obtain first an object module, then a load module, the system had to know what resources the language translator and the linkage editor required. To execute a program, you must tell the system, in your control statements, how you have divided your work into jobs and job steps, what resources they will need, and where your data is. This information is used before the system gives control to your program and during execution.

Table 1B. Operating System Elements

Control Program Elements		
Languages	Service Programs	Application Programs
ALGOL Assembler COBOL FORTRAN PL/I RPG	Linkage Editor Loader	User Written

Your input, either on cards or tape, may now be viewed in three parts:

1. Control statements that inform the system of required resources,
2. Your program, structured according to its purpose and coded in a source language,
3. Optionally, data, which your program will process.

Figure 6 now shows your input ready for entry into the system. This completes your preparation. It is now up to the system to honor your requests and supervise the execution of your work.

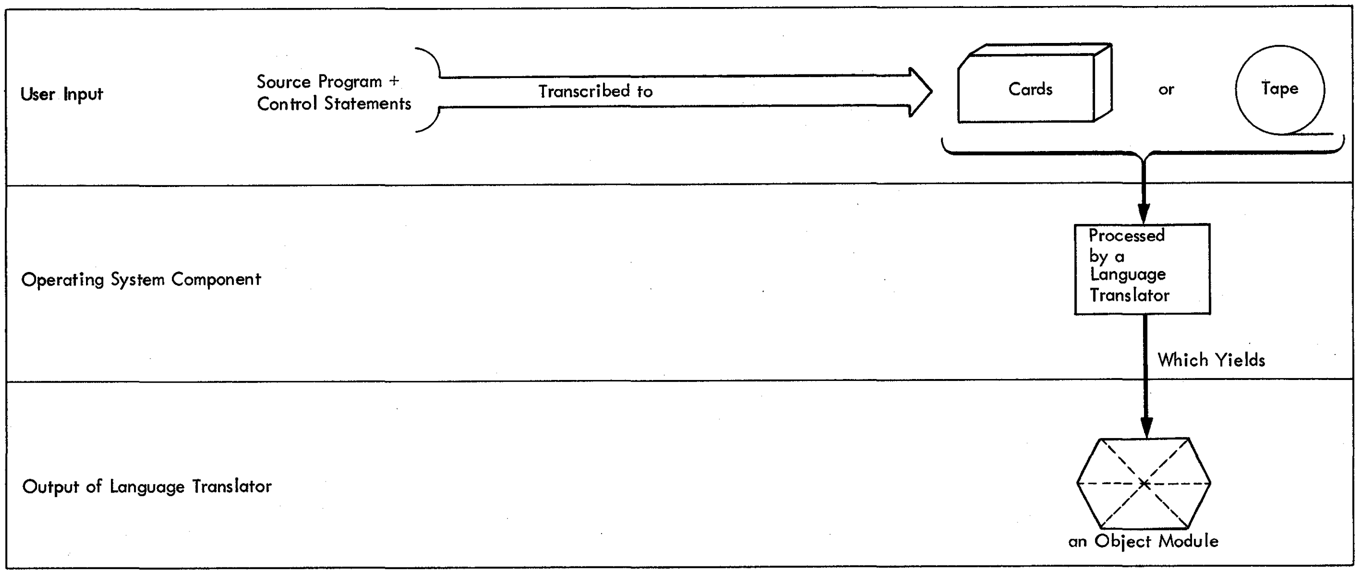

User Input
Source Program + Control Statements
Transcribed to
Cards
or
Tape
Operating System Component
Processed by a Language Translator
Which Yields
Output of Language Translator
an Object Module

Figure 4. Producing an Object Module

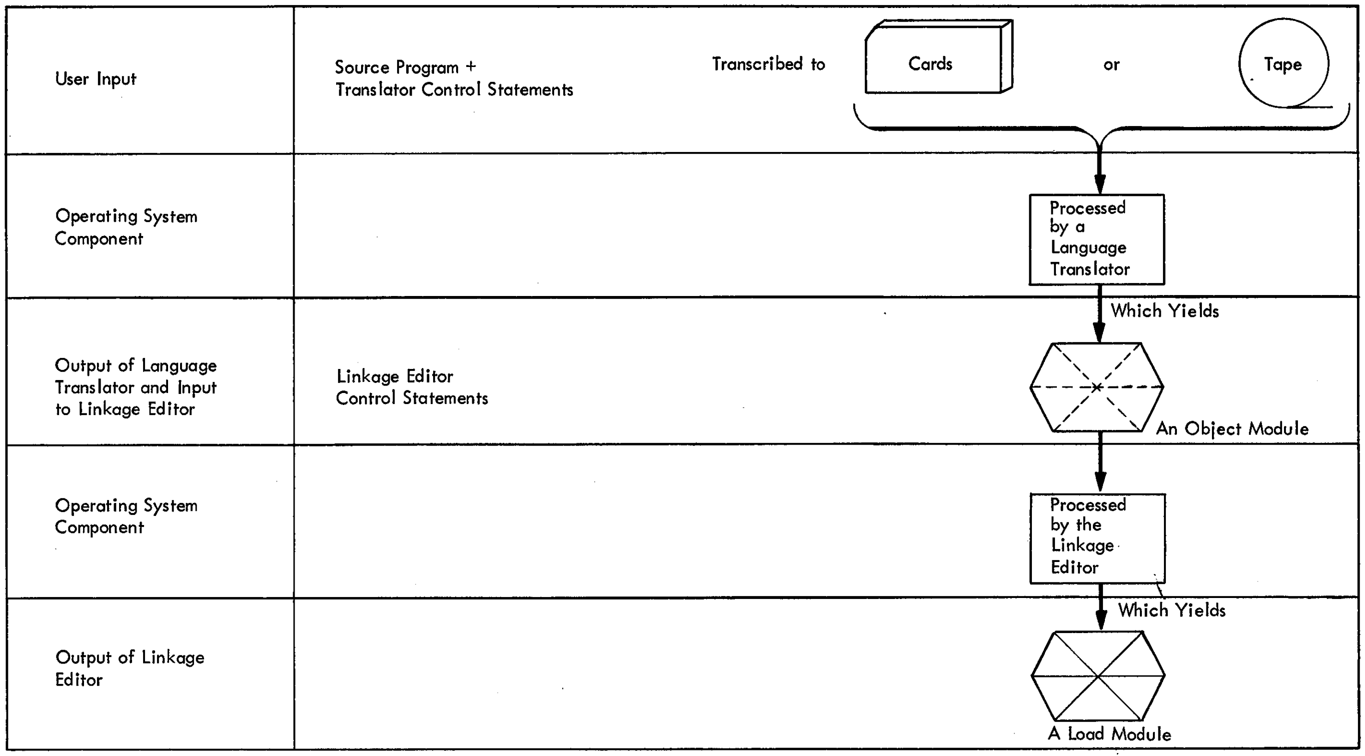

User Input
Source Program + Translator Control Statements
Transcribed to
Cards
or
Tape
Operating System Component
Processed by a Language Translator
Which Yields
Output of Language Translator and Input to Linkage Editor
Linkage Editor Control Statements
An Object Module
Operating System Component
Processed by the Linkage Editor
Which Yields
Output of Linkage Editor
A Load Module

Figure 5. Producing a Load Module

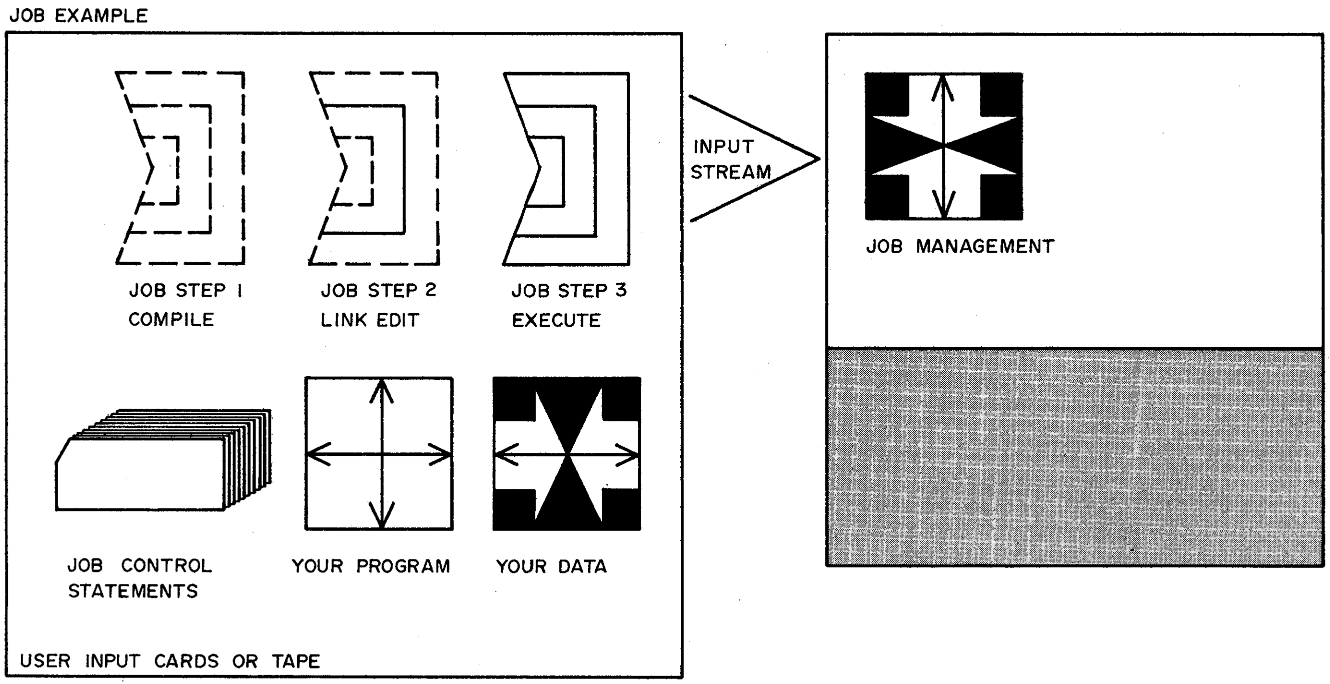


Figure 6. User's Input to the Operating System

What the System Does

The system must first read your source program into main storage and prepare it for execution. It must also make provision for allocation of resources.

The first functions performed by the control program are to read and analyze the control statements, and initiate the work. Note that the system also has tasks. Getting your job ready for execution is a task to the system. (A task is the smallest, independent unit of work that can compete for the resources of the system.) Therefore, until your program is actually given control, it is treated as data by the system.

The same functional area of the control program that reads and analyzes the control statements also:

1. Schedules the job step for execution, on either a priority or sequential basis.
2. Allocates input/output devices.
3. Initiates the work.
4. Handles the termination of the job step.
5. Handles communication with the operator.
6. Controls the output writers, write output on a system output device independently of the program that produced the output.

In our example, these procedures would apply for the compiling and linkage editing phases of the job, as well as for the actual execution of your program. This functional area of the control program is known as job management. It is the first and last portion of the control program encountered by your jobs.

One means of direct communication with the operating system is the job control language, and the flow of control statements and data entering the system is known as the input stream. When your program, in the form of object modules, is read into storage, it is routed first to a direct access device (disk or drum). (Object modules may appear in the input stream, but load modules come only from direct access devices, where the linkage editor puts them.)

Your program must be loaded into main storage, other resources allocated to it, and tables and records maintained about it so that processing may be resumed from the appropriate point in case of an interruption. These supervisor functions are handled by a group of programs known as task management.

The task management programs acquire all resources for your program other than I/O devices through assembler language macro instructions. These macro instruction names are mnemonic, descriptive, and always capitalized (for example, GETMAIN). Control is passed from job management to task management.

Finally, the operating system must store and retrieve data as requested. All named collections of data are called data sets and the general name for the operating system programs that control I/O and related operations (operations are requested by macro instructions) is data management.

Data management has two techniques for transmitting data between main and secondary storage -- the queued technique and the basic technique. These techniques offer you a wide variety of automatic services. With them you can retrieve and store logical records (or records defined in terms of the information they contain), hereafter simply called records. Blocking and deblocking may be left to the data management programs. Also, I/O operations can be synchronized with processing, and I/O overlap is automatically arranged. Furthermore, through the use of the Shared Direct Access Device option (Shared DASD) independently operating computing systems can share common data residing on shared direct access storage devices.

If the features of these access techniques do not meet your particular requirements, you can control I/O directly by writing your own channel programs. Scheduling and indication of completion are handled by the system.

Data transmitted to main storage may be placed in buffers, which are areas of main storage set aside to receive the data. Data management provides a wide variety of methods to handle these buffers. For example, you can process records while they stand in the buffers, or data management programs will move them to your work area. With the latter method, you need never be concerned with buffer location.

The buffering methods let you exchange your work area for a buffer so that you can continue processing while more records are being read or written. If you use exchange buffering, the data is not moved in main storage.

Thus, the final control program entry is made in Table 1C and the general diagram, Figure 7, is expanded to include the control program functions.

The example introduced all the control program functional areas, all the available source languages, a service program (the linkage editor), and the concept of user programs as system resources. However, there are other service programs available.

Table 1C. Operating System Elements

Control Program Elements		
Job Management	Data Management	Task Management
Languages	Service Programs	Application Programs
ALGOL	Linkage	User
Assembler	Editor	Written
COBOL	Loader	
FORTRAN		
PL/I		
RPG		

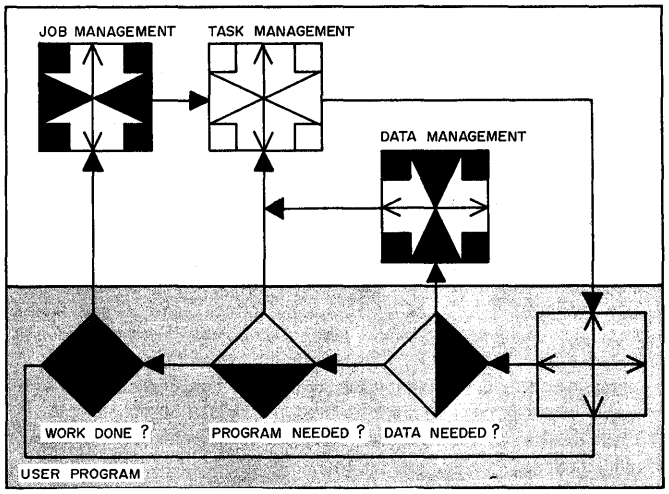


Figure 7. User Requests for System Services

Service Programs

Included also among the service programs are three types of utility programs: system utility programs, data set utility programs, and independent utility programs.

System utility programs, executed under control of the operating system:

- Modify system control data by building and maintaining catalogs and volume structures.
- Move and copy collections of data to rearrange data and create backup copies.
- List the system control data, such as a catalog, a directory of a partitioned data set, or a volume table of contents. (These terms are fully defined in Section 3.)
- Place a character set into a printer control area.
- Retrieve, edit, and write previously generated error environment records.
- Initialize and assign alternate tracks to a direct access volume; dump and restore the data contents of a direct access volume.
- Recover usable data from a defective track, assign an alternate track, and merge replacement data with the recovered data onto an alternate track. (This program is called IEHATLAS.)

Data set utility programs, also executed under control of the operating system:

- Copy and merge partitioned data set members.
- Copy records from sequential data sets (tape-like organization) and convert sequential data sets to partitioned organization.
- Compare records in sequential or partitioned data sets.
- Print or punch records in sequential or partitioned data sets.
- Update symbolic library modules at the source-language level.
- Create multiple data sets within one job for the sequential and partitioned access methods.
- Compress the members of a PDS within its original extent and provide a summary of remaining space.

Independent utility programs executed outside the operating system:

- Initialize and assign alternate tracks to direct access volumes.
- Dump and restore the data contents of a direct access volume.
- Recover usable data from a defective track, assign an alternate track, and merge replacement data with the recovered data onto the alternate track.

The sort/merge program is a generalized program that can arrange fixed- or variable-length records into ascending or descending order. The process can employ either magnetic-tape or direct access storage devices for input, output, and intermediate storage. The program is adaptable in the sense that it takes advantage of all input/output resources allocated to it by the control program. The sort/merge program can be used independently of other programs or it can be called directly by them; it can also be called through COBOL and PL/I.

The 7094/M85 Integrated Emulator allows object programs written for one system to be executed on another system using a compatibility feature. The compatibility feature consists of added hardware and microprogrammed routines that aid emulation. The integrated emulator program is executed under the control of the operating system in a multiprogramming environment.

Finally, to aid in testing programs, the operating system has a test translator facility: TESTRAN (available only to assembly language programs). TESTRAN helps uncover faulty logic by providing printed information about the actual working of a program. At the programmer's direction, TESTRAN describes the changing contents of storage areas, registers, and control blocks, and also describes control flow from one group of instructions to another.

Here is Table 1 in final form.

Table 1. Operating System Elements

Control Program Elements		
Job Management	Data Management	Task Management
Languages	Service Programs	Application Programs
ALGOL	Linkage Editor	User Written
Assembler	Loader	
COBOL	System Utilities	
FORTRAN	Data Set Utilities	
PL/I	Independent Utilities	
RPG	Sort/Merge	
	TESTRAN	
	7094/M85 Integrated Emulator	

Operating System Control Program Optio

There are three types of control programs: the primary control program (PCP), multiprogramming with a fixed number of tasks (MFT), and multiprogramming with a variable number of tasks (MVT).

The primary control program (PCP) schedules and executes job steps one at a time. This configuration is well suited for the installation that does not need a more powerful System/360 now, but may later.

MFT reduces the problem of CPU wait-time by supervising the execution of more than one job at a time. Each job is executed in its own area of main storage. The size of each of these areas, or partitions, is established when the system is generated, but may be changed by the operator if required. MFT is especially useful to users who must process a wide variety of jobs that require a corresponding variety of computing system resources. The system's capability of providing partitions as small as 8K bytes is a distinct advantage to the user with many small jobs.

MFT systems with subtasking (optional) allow the user to execute more than one task within a partition. Thus, MFT (with subtasking) extends the idea of priorities beyond between-job competition for resources to competition within jobs, that is, different priorities can be given to separate tasks of a job step.

MVT also supervises execution of more than one job step at a time. In addition, it allocates main storage dynamically to each job. MVT supports the large job customer and the customer who has many small jobs.

Before MVT can schedule a job, the programmer must request, through a control language, the amount of main storage required and the devices required. Since a single job will probably not require all of main storage nor all devices, the remaining resources can be given to other jobs. The programmer also has some control over the sequence of job scheduling. Instead of scheduling jobs in the order in which they are submitted, MVT schedules jobs according to specified priorities.

When two or more jobs are being executed concurrently, each job competes for the machine and program resources it needs, unless the jobs are being executed under

MVT with Model 65 multiprocessing. In that case, each job competes for use of one of the CPUs and the program resources it needs. The main factor in resolving the competition for machine resources is the scheduling priority of the job. When two jobs are being executed, the job with the higher priority uses the CPU when it needs it. When two jobs are being executed under MVT with Model 65 multiprocessing (i.e., one job in each CPU), each job uses one CPU. Both jobs can share reenterable program resources.

MVT also extends job-step competition for resources to competition within jobs, as discussed in the preceding description of MFT systems with subtasking.

Summary

The operating system is modular in concept to serve a wide variety of applications and to support a broad range of hardware configurations. Its control program functions of data management, job management, and task management allow closer supervision of all your jobs. Its processing programs allow you to add, modify, or delete functions, according to your changing needs. The operating system establishes a direct line of communication between the programmer and the computing system and keeps this line open with minimal clerical assistance from the programmer.

Section 2: Program Design

Program designing is of optimum importance because an efficient and economical installation is achieved only when all its facilities are as busy as possible.

Remember that your program may or may not need other programs to complete execution: the program may contain all the code necessary to complete execution or it may have to call another program. This is an illustration of the concept of modularity.

To obtain maximum efficiency, you should design your programs as easily implemented subprograms and structure them in one of three ways: simple, planned overlay, and/or dynamic. In any case, any load module on a library can be a programming resource for another load module. Thus, another way of looking at your programs is that they can be non-reusable, serially reusable, or reenterable.

Segmenting Programs

Designing large programs as smaller, more easily managed subprograms is a normal method of programming efficiency. It allows you to:

- Write programs that lend themselves to variations in parameters. Hence, you can defer specifications of particular I/O devices, control units, and channels.
- Write programs that may be somewhat inefficient if run separately, but provide maximum efficiency when run together. For example, if your programs are being executed in a multitasking environment, you might wish to process one task which makes heavy use of I/O with a task that uses little I/O, but that does a great deal of CPU processing.

The operating system expands this capability: your programs can be combined during compilation, linkage editing, job entry, or execution (see Figure 5). Thus, with little or no change, you can use any load module as a resource for any other load module. The net result is increased efficiency.

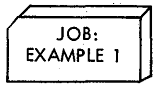
Combining Subprograms at Compilation Time

A program named EXAMPLE 1 is to be coded by two programmers.

Dubon codes subprogram A.
Wright codes subprogram B.

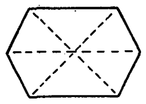
Subprograms A and B are keypunched and the decks combined for compilation:

This is a "batched compilation."

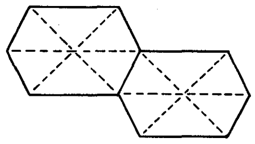


Combining Subprograms at Linkage Editing Time

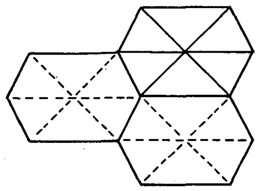
The FORTRAN compiler processes a source module and produces an object module.



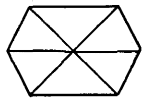
This object module is combined with one other object module



and one load module

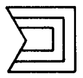


to produce a single load module ready for execution



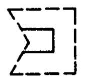
Combining Programs at Job Entry Time

You have a job



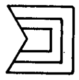
divided into three job steps

job step one



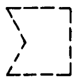
passes its results to job step 2

Job step 2 passes its results to job step three, which completes the job.

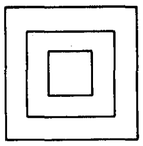


Combining Programs at Execution Time

Your job step



has been attached as task



Another program is needed to complete execution of this task. Your program issues the appropriate macro instruction and the program is brought into main storage and given control as a subtask of the job step task.

What the User Must Do

You are given a problem to solve. Because you must solve this problem in light of the programming and machine resources at your installation, you divide your program into more easily managed subprograms. Each subprogram is processed by a language translator, which produces an object module. In turn, the object module is processed by the linkage editor (and may be combined with other subprograms), which puts the object module into load-module form.

Job control statements describe the load module as a job or job step and indicate the location, organization, etc., of the data. The job step is read into storage and, when the system is ready, it is given control as a task in the system.

When a new task is created, the control program is given the name of the first load module that is to be executed. Depending on the structure you select, the named load module may be loaded into storage in one operation, or it may call upon other load modules during execution.

A Simple Structure: One load module contains all the user code necessary for task performance, that is the module doesn't pass control to any other modules during its execution. It is loaded into storage as an entity. Note that the requirement to use system code is not considered by this definition.

A Planned Overlay Structure: A planned overlay structure is a single load module created by the linkage editor program in response to overlay control statements. Unlike simple structures, however, it is not loaded into main storage all at once. Only one control section (called the root segment) remains in storage at all times.

A Dynamic Structure: In a dynamic structure, more than one load module is called upon during task performance. The control program acts as the intermediary, and program execution may be serial or parallel. Each of the load modules to which control is passed can be one of the three structure types. Table 2 summarizes the characteristics of these load module structures.

Table 2. Load Module Attributes

Structure Type	Loaded All At One Time	Passes Control to Other Load Modules
Simple	Yes	No
Planned Overlay	No	No or Yes[1]
Dynamic	Yes or No[1]	Yes

[1]A segment of a load module can dynamically call another load module.

For many applications, a simple structure and a planned overlay would be inadequate. For example, too many control sections (the smallest separately relocatable unit of a program) may be called upon during execution to have them all in storage at the same time. Furthermore, the effectively random order in which they are called may make the planned overlay structure difficult or impossible.

The operating system therefore permits load modules to be called dynamically (during the performance of a task). This is possible because:

- Program retrieval by name is a normal control program service.
- Main storage is allocated dynamically.
- Programs are treated as resources.
- Standard linkage conventions permit any load module to be executed by any other load module.

What the System Does in Response

For a simple structure: If a load module contains all the code necessary for task performance, it follows that a program designed as a simple structure does not dynamically employ another load module. That is, such programs do not issue a LINK, LOAD, XCTL (transfer control), or ATTACH macro instruction. These macro instructions require control program intervention, thus decreasing efficiency. (Note: Other load modules could be called upon indirectly, through control program intervention.) However, a simple structure program could itself be dynamically employed by some other (dynamic) load module.

You can associate the same load module with a number of different programs, each of which can correspond to a different entry point (any location to which control can be passed). During linkage editing, a primary program name and as many as 16 aliases (alternate names or entry points) may be specified. All names and the corresponding entry points are contained in the directory of the library in which the load module is stored. In addition, a load module can dynamically specify other names and entry points. However, these are not retained when the main storage space used by the load module is released.

Figure 8 illustrates what the system does for a simple structure called SIMPLE.

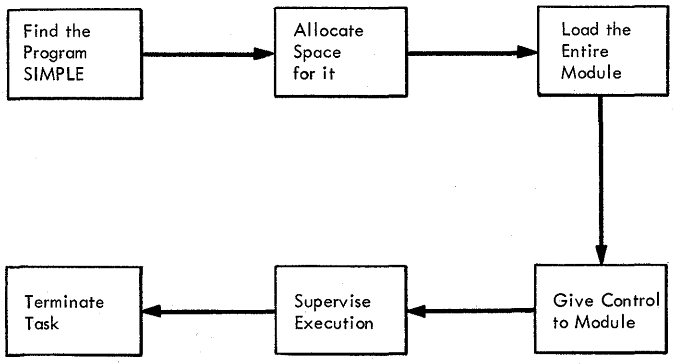
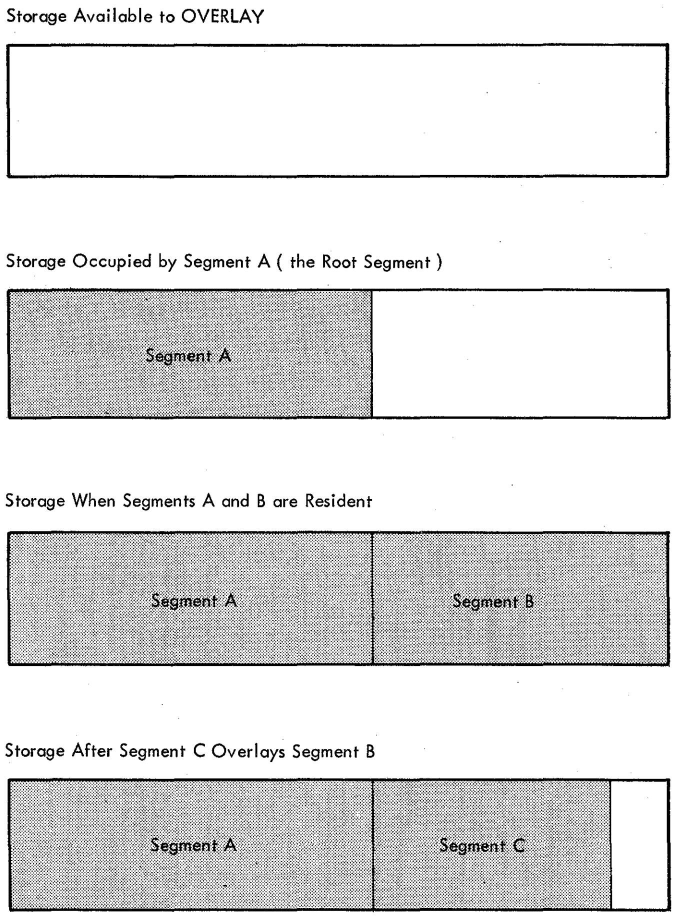


Figure 8. System Logic Flow for a Simple Structure

Note: SIMPLE could be a composite of several load modules that were combined into a single load module by the linkage editor. This does not change the fact that SIMPLE is a program with a simple structure: it is, at execution time, a single load module, all of which is in main storage.

For a planned overlay structure: The root segment is loaded, and the other logical control sections of a planned overlay program are loaded into main storage as required, each occupying an area of storage that may at some time be used by a different segment. Of course, since control sections are executed serially, you must plan the relationship of control sections in advance.

Figure 9 shows what storage might look like as the various segments of a program, OVERLAY, are loaded and overlaid. Note: The root segment -- A -- contains the entry point of the overlay program and tables (inserted by the linkage editor) needed to control the overlay execution. Note that storage must be large enough to accommodate the largest segment of an overlay, plus the root segment.



Storage Available to OVERLAY
Storage Occupied by Segment A (the Root Segment)
Segment A
Storage When Segments A and B are Resident
Segment A
Segment B
Storage After Segment C Overlays Segment B
Segment A
Segment C

Figure 9. Storage Allocation for a Planned Overlay Structure

During the execution of an overlay program, overlaid segments are destroyed rather than saved on secondary storage and then restored. When a previously overlaid segment is needed again, a fresh copy is loaded.

Figure 10 illustrates what the system does for a planned overlay structure called OVERLAY.

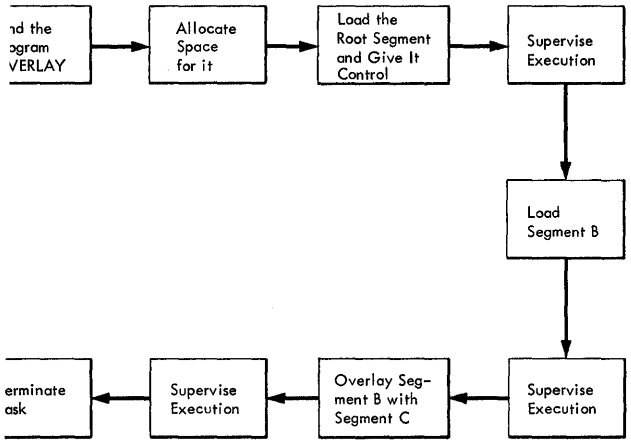


Figure 10. System Response for a Planned Overlay Structure

For a Dynamic Structure: A dynamic structure requires more than one load module during execution. Each load module required can operate as either a simple structure, a planned overlay structure, or another dynamic structure. The advantages of a dynamic structure over a planned overlay structure increase as the program becomes more complex, particularly when the logical path of the program depends on the data being processed. The load modules required in a dynamic structure are brought into main storage when required, and can be deleted from main storage when their use is completed. (Note in a dynamic structure with more than one task per job step, storage management is more complicated.)

LOAD MODULE EXECUTION

Depending on the configuration of the operating system and the macro instructions used to pass control, execution of the load modules is serial or in parallel. Execution of the load modules is always serial in an operating system with the primary control program or with MFT (without subtasking); there is only one task in the job step. Execution is also serial in an operating system with MVT, unless an ATTACH macro instruction is used to pass control. When an ATTACH macro instruction is used, a new task is created, and the load module to which control is passed is executed in parallel with the load module containing the ATTACH macro instruction. The execution of the load modules is serial within each task.

Figure 11 shows a dynamic execution when there is only one task per job step. This task involves three load modules: the program DYNAMIC, subprogram A, and subprogram B. Execution of DYNAMIC proceeds until the macro instruction LINK A is reached. In this example, subprogram B is used twice, once at the third level of control and once at the second level. If B is still available when called for the second time, the same copy of B is used, rather than a new copy (which would require additional loading).

Figure 12 shows a dynamic execution when there are two tasks per job step. During the execution of Task A, the system detects the need for execution of Task B. B continues processing until it must wait for the completion of an I/O event. Control is passed back to A. Note that A must wait for the completion of B, before it can complete processing.

Planned Overlay Versus Dynamic Structures: The control program facilities for planned overlay structures and for dynamic fetching of load modules are both designed to meet the need for executing programs larger than the storage areas available. Each method has its advantages.

Planned overlay structures can be more efficient in terms of execution speeds, because the linkage editing procedure permits direct references by one control section to values whose locations are identified by external symbols in another control section. In a dynamic structure, the values of their locations would have to be passed as parameters between separate load modules. Furthermore, when using a planned overlay, supervisory assistance is needed only to locate a single load module in the library. When using a dynamic structure, many load modules need to be located in order to execute an equivalent program. Also, with a planned overlay, supervisory assistance may be less frequent when going from control section to control section than if each section was a load module.

These advantages tend to diminish as problems get more and more complex, particularly when the logical selection of load modules depends on the data being processed. In this situation, the use of dynamically constructed programs is usually a better solution than planned overlay programs. Furthermore, load modules fetched dynamically may be reused in a job step; sections of load modules fetched using the planned overlay structure cannot be reused.

Although both approaches are solutions to the same problem, there is no prohibition against using combinations of the two. A load module, linked to dynamically, may itself operate in the overlay mode. The LINK macro instruction may also be used within a planned overlay program.

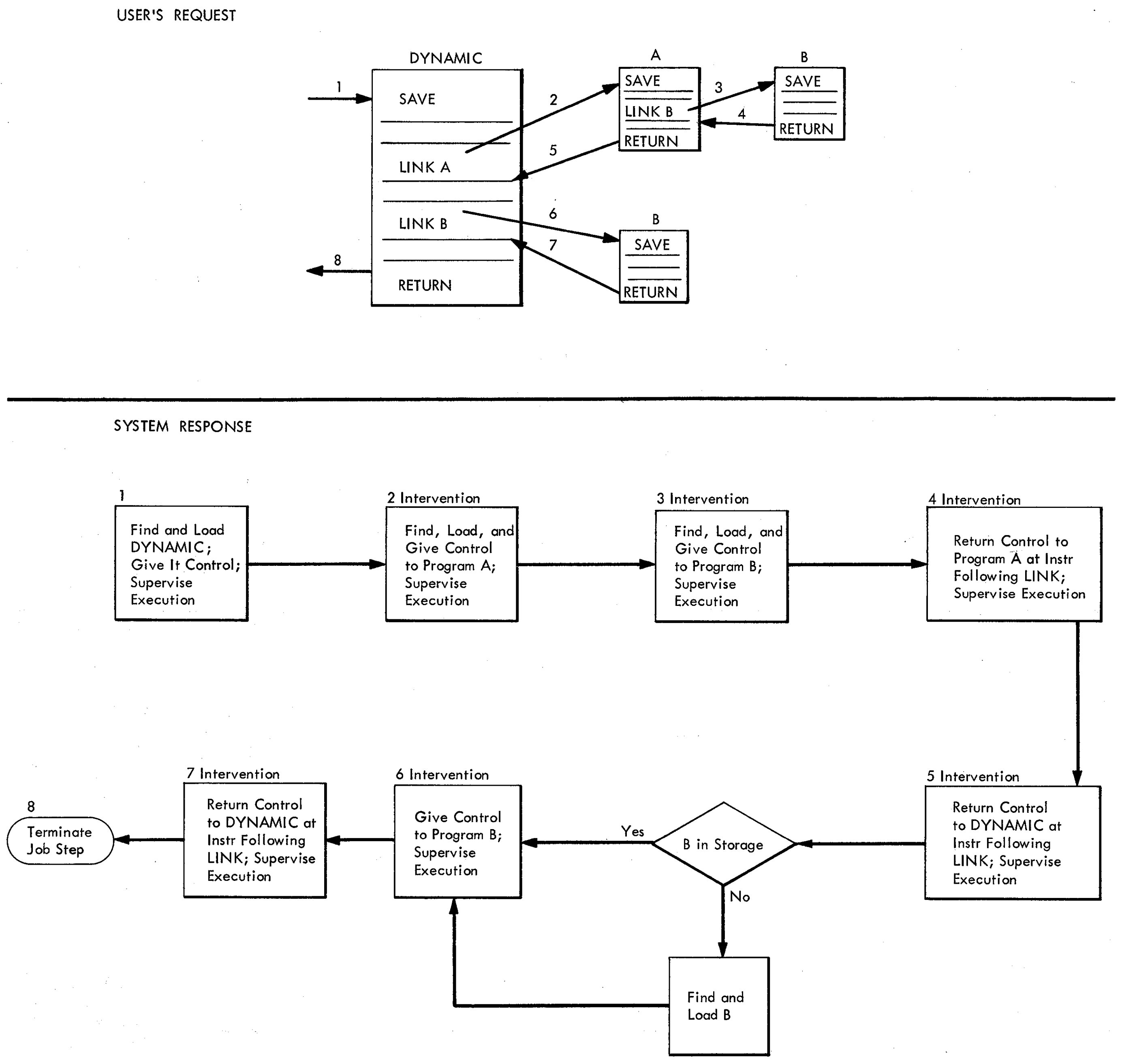
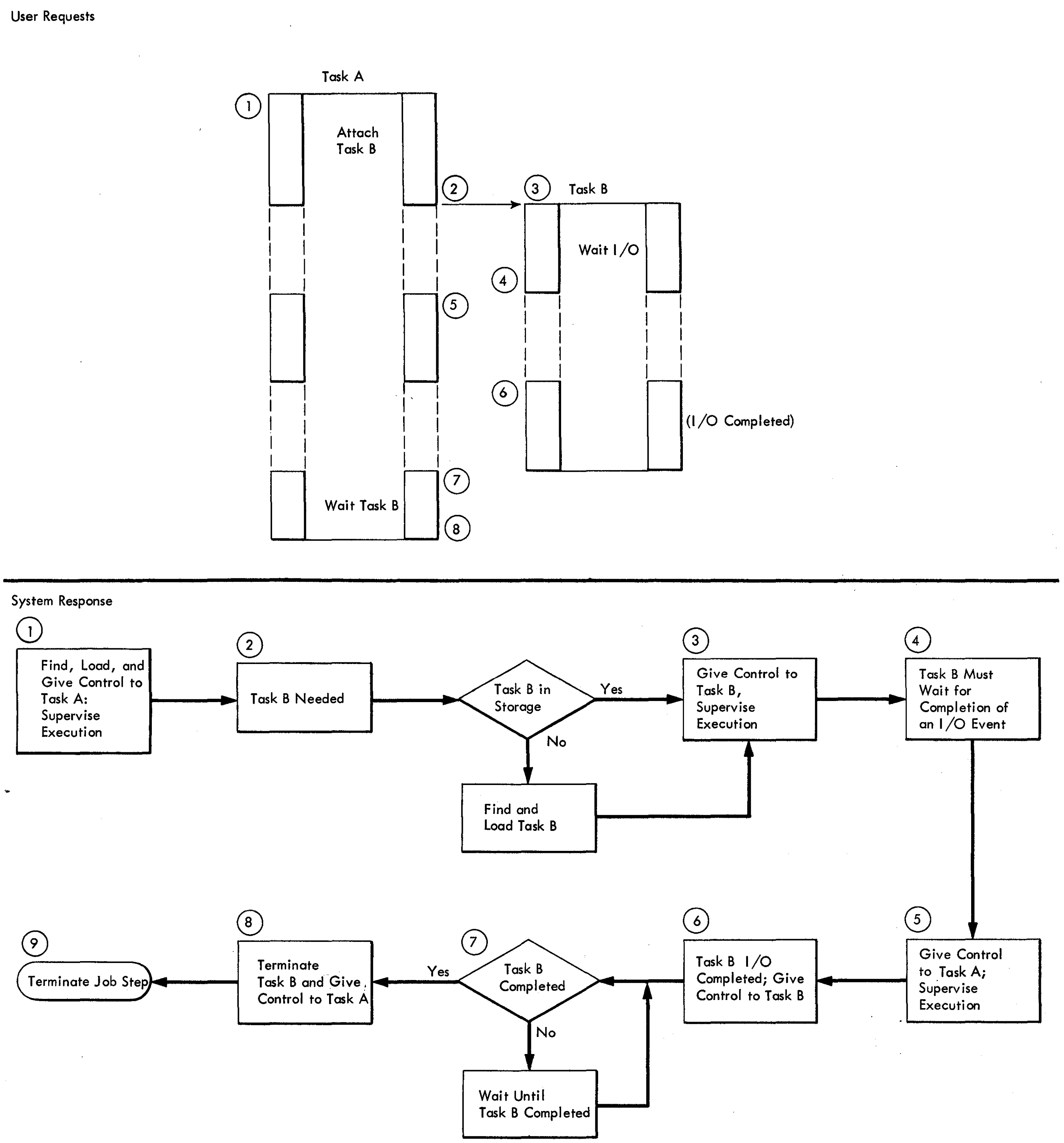

USER'S REQUEST
DYNAMIC
SAVE
LINK A
LINK B
RETURN
A
SAVE
LINK B
RETURN
B
SAVE
RETURN
B
SAVE
RETURN
1
2
3
4
5
6
7
8
SYSTEM RESPONSE
1
Find and Load DYNAMIC; Give It Control; Supervise Execution
2 Intervention
Find, Load, and Give Control to Program A; Supervise Execution
3 Intervention
Find, Load, and Give Control to Program B; Supervise Execution
4 Intervention
Return Control to Program A at Instr Following LINK; Supervise Execution
5 Intervention
Return Control to DYNAMIC at Instr Following LINK; Supervise Execution
B in Storage
Yes
No
Find and Load B
6 Intervention
Give Control to Program B; Supervise Execution
7 Intervention
Return Control to DYNAMIC at Instr Following LINK; Supervise Execution
8
Terminate Job Step

Figure 11. Dynamic Execution, One Task per Job Step



User Requests
Task A
1
Attach Task B
2
3
Task B
Wait I/O
4
5
6
(I/O Completed)
7
Wait Task B
8
System Response
1
Find, Load, and Give Control to Task A: Supervise Execution
2
Task B Needed
Task B in Storage
Yes
No
Find and Load Task B
3
Give Control to Task B, Supervise Execution
4
Task B Must Wait for Completion of an I/O Event
5
Give Control to Task A; Supervise Execution
6
Task B I/O Completed; Give Control to Task B
7
Task B Completed
No
Wait Until Task B Completed
Yes
8
Terminate Task B and Give Control to Task A
9
Terminate Job Step

Figure 12. Dynamic Execution, More Than One Task per Job Step

Program Design Facilities

One of the salient concepts pointed out in the first part of this section is that programs are resources. Once your program is in load module form and placed in a library, it can be made available to other tasks in the system. Hence, even though a program is executed by itself only once, it could be reused time and time again. Therefore, before coding any program, you should be familiar with the techniques involved in producing reusable programs and the special facilities you can request.

Libraries

Programs in load module form are placed in libraries. There are three types of libraries:

1. Temporary libraries are partitioned data sets created to temporarily contain a program until it is used in a later job step of the same job. This type of library is particularly useful for containing the program output of a linkage editor run until it is executed by a later job step.

2. Private libraries are partitioned data sets that house groups of programs not used frequently enough to warrant their inclusion in the system library. Private libraries are made available to a job or job step with special DD (data definition) job control statements.

3. The System library is a partitioned data set named SYS1.LINKLIB and SVCLIB that houses frequently-used programs, including compilers. This library is always open to users of the system.

Reusability

All load modules in any library are placed in one of three categories, as specified by the programmer at linkage editing time: not reusable, serially reusable, and reenterable. (This discussion is pointed toward program design concepts, not toward system use of reusable programs. You should note, however, that when a problem program calls another program, only MVT utilizes reusability attributes of the programs. The primary control program and the MFT control program treat all programs as non-reusable, unless they are brought into storage via a LOAD macro instruction. In that case, PCP and MFT assume that the programs are reentrant.)

Not reusable: Programs in this category are fetched directly from the library when requested. These programs alter themselves during execution, and will not execute correctly if entered again.

Serially reusable: A load module of this type is designed to be self-initializing, so that any portion modified during execution is restored before it is reused. The same copy of the load module may, therefore, be used repeatedly during performance of a task. In addition, a serially reusable load module may be shared between different tasks, provided that both tasks were created from the same job step. A further condition for use of the load module by more than one task is that it not be in use by one task at the time it is called for by another. If it is, the task requests are placed in a queue, waiting for the load module to become available.

Reenterable: Such a program is designed so that it does not in any way modify itself during execution, that is, it is "read-only." Reenterable load modules fetched from the system library are loaded in storage areas protected with the same storage key that is used for the supervisor program.

Since only the control program operates with a matching PSW protection key, reenterable programs are protected against accidental modification by any other user programs. Since a reenterable load module is never modified during its execution, it can be loaded once and used freely by any task in the system at any time. Specifically, it can be used concurrently by two or more tasks in multitask operations, that is, in MFT and MVT. One task may use it, and before the module execution is completed, an interruption may give control to a second task, which, in turn, may reenter the module. This in no way interferes with the first task resuming its execution of the module at a later time.

In MFT and MVT, simultaneous use of a load module is considered to be normal operation. Such use is an important factor in minimizing main storage space requirements and program reloading time. Many of the control program routines are written in reenterable form, so that they can be shared between tasks and reused within a single task. A load module of this category can be executed correctly even though the protection key in the program status word during task execution is different from the supervisor storage key. This is possible because the protection key comparison must be satisfied only when the contents of the addressed storage area are to be altered. The contents of storage areas containing reenterable programs are not altered in any way during execution.

Design of Reenterable Programs: A reenterable program is designed to use the general purpose and floating point registers for addressability and variables where practical, and to use temporary storage areas that "belong" to the task, and are protected with the task's storage protection key. Temporary or working storage areas of this sort can be provided to the reenterable program by the calling program, which uses a linkage parameter as a pointer to the area. Temporary storage areas can also be obtained dynamically by the reenterable program itself.

Note that the storage area obtained is assigned to the task, not to the program that requested the space. The space may subsequently be returned to the supervisor's control by a macro instruction or by task completion.

If a reenterable program is interrupted for any reason, the register contents and program status word (PSW) are saved by the supervisor in an area associated with the interrupted task, and restored later when program execution is to continue for that task. No matter what use is then made of the reenterable module, the interrupted task can resume its use of the module at a later time. The supervisor merely keeps the task's working storage area intact, and when required, restores the contents of the saved registers and the program status word. The reenterable load module is not affected and is unaware of which task is using it at any instant. Each task will have its own temporary storage area for use by the reenterable module.

Checkpoint/Restart

The checkpoint/restart facility is available with the PCP, MFT, and MVT control programs. By restarting from a checkpoint or from the beginning of a job step, this facility can be used to minimize time lost in reprocessing a job step that terminated abnormally due to a program or system failure. (A checkpoint restart, that is, one initiated by issuing a CHKPT macro instruction, may be either automatic or deferred. Checkpoint restart is described in the text below. A step restart, that is, one initiated through the use of special parameters on either the JOB or EXEC statements, may also be either automatic or deferred. Step restart is described in Section 5 of this publication.)

When the CHKPT macro instruction is executed, the control program saves all main storage areas, control information, and floating point and general registers needed to restart from the checkpoint. After execution of the CHKPT macro instruction, control is returned to the user's program and processing continues at the next sequential instruction after the CHKPT macro instruction.

The restart procedure is used to restore and run a previously checkpointed job step. This restart may be automatic (depending on an eligible ABEND code and the operator's consent) or deferred, where a deferred restart involves resubmitting the job.

At the checkpoint restart, the control program retrieves the checkpoint control information, ensures that volumes are correctly mounted, and repositions tapes. All programs and data are restored to the locations occupied at checkpoint time, and the job step continues. Control is returned at the instruction immediately following the CHKPT macro instruction.

Note: The checkpoint/restart facility does not copy any data sets. The programmer should locate checkpoints at those points where he can ensure data set integrity.

Service Aids

To aid in the diagnosis of problems, seven service aid programs are provided with the operating system: IMAPTFLE (used to create job control statements for applying a Product Temporary Fix -- PTF -- to a system library); IMAPTFLS (provides formatted lists of members of a library to which PTFs have been applied, or of all members of a library); IMASPZAP (allows the user to inspect and modify data in a load module); IMBMDMAP (produces formatted maps of load modules previously link edited into a partitioned data set, or link pack or resident reenterable load module areas, and of the nucleus); IMCJQDMP (provides formatted job queue dumps); IMDPRDMP (format and prints the output of the high speed version of IMDSADMP); IMDSADMP (can dump main storage to tape at high speed, and to either tape or a printer at low speed). These programs can be used to gather information about the cause of a failure; to format and print the information in a readily usable form; and to aid in developing and applying a fix for a problem. For a detailed discussion of these service aids see the publication, IBM System/360 Operating System: Service Aids, GC28-6719.

Language Comparison: Program Design Facilities

All program design facilities are available to users of the assembler language. Table 3 indicates which of the facilities are available to users of the higher-level languages.

•Table 3. Language Comparison: Program Design Facilities

Program Design Facilities	ALGOL	COBOL			FORTRAN			PL/I	RPG
		E	F	ANS[1]	E	G	H		
Simple Structure	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Planned Overlay Structure	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Dynamic Structure	Yes	No	No	Yes	No	No	No	No	No
Serially Reusable Programs	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Reenterable Programs	No	No	No	No	No	No	No	Yes	No
Checkpoint/Restart	No	Yes	Yes	Yes	No	No	No	Yes	No

[1]American National Standard COBOL.

Section 3: Data Management

The data management programs are primarily responsible for moving information between main storage and external storage and maintaining it in external storage. These programs are capable of locating data, preparing main storage areas for it, reading it, and writing it. As a user, you can write your own channel programs and buffer handling routines, or you can let the data management routines control the flow of data.

What the User Must Do

To manipulate your data, the data management routines must have a great deal of information about it. You supply some of this information when the data is first recorded, some when you enter your job control statements, and some at execution time. The data management programs must collect this information before they can work with your data. Giving the data a name enables the data management routines to identify it and find the information that describes it. When a logically related collection of data has been named and described to the system, we call that data a data set.

The term "data set" is used to describe all named, organized information presented to the system. The content of a data set may be a load module, data records to be processed, or a library of algorithms. The information in data sets is structured into records, which are logical groups of information, and blocks, which are physical groups of information separated by physical spaces.

Records may be in one of three formats: fixed-length, variable-length, or undefined length. The size of a fixed-length record is constant for all records in the data set. A variable-length record specifies its own length in its first four bytes. You provide this control information when creating the record; the system checks this information and uses it to block and unblock the records. Undefined-length records permit processing any records that do not conform to the fixed- and variable-length formats. Since the system does not perform length checking on undefined-length records, your program must deblock them.

Blocks contain one or more records. Data sets may be contained on any appropriate I/O device. When data sets are contained on direct access or magnetic tape devices, they are stored on volumes, which are standard physical units of secondary storage. Examples of volumes are:

- A reel of tape
- A disk pack
- A data cell
- A drum
- The part of an IBM 2302 Disk Storage Drive served by one access mechanism.

WHEN AND/OR WHERE TO SPECIFY INFORMATION

The data management programs can move data from secondary storage to main storage and place it back on secondary storage. However, before manipulating data, the routines must have a complete description of it. How then do you supply this information?

The Data Definition Statement

A major source of data set information is the data definition (DD) job control statement, which describes the data sets to be used in a job step. You must provide a DD statement for every data set you plan to use or create.

The information on a DD statement may be divided into two general categories -- parameters that describe the place where data resides or will reside, and parameters that describe attributes of the data. The parameter of the first category answers:

- What is the status of the data set being described? Is it a new one about to be created, or does it exist now? Do you wish to save the data set for future use, or do you need it only for this job or job step?

- Which I/O device or devices must be allocated for this data set? Do you need a particular device, or will any device (tape, disk, etc.) be satisfactory?

- Which particular volume or volumes contain an existing data set? Which are desired for a new data set?

- How much space is required to contain a new data set on a direct access device? What should the system do if you underestimate space requirements? What should be done with extra space if you overestimate? Must the space be contiguous? Must it begin at a specific address?

You can also provide information about the data itself, rather than its location:

- What is the name of the data set? The name may be given here or as a reference to another DD statement. If you omit the name, the system gives a unique name to the data set; the data set is known by this unique name only for the duration of one job, so you should include a name for all but temporary data sets.
- What is the format of your data? Are the records blocked or unblocked, fixed length, variable length, or undefined? If blocked, what is the block size? What is the record length? How are the records organized within the data set? (Information on data set organization appears later in this section.)

The data control block (DCB) parameter answers these questions. It also provides a means of specifying the way you want I/O buffers built and controlled, as well as device dependent information.

The advantage of specifying this information on a DD statement is that the parameters may be modified without reassembling or recompiling the program that uses the data set. However, most of the information on data set attributes may come from other sources. For example, information that remains constant may be stored with the data set itself in a data set label.

Data Set Labels

Labels are used by the operating system to identify volumes and the data sets they contain, as well as to store data set attributes. Data management routines automatically fill standard labels with data set characteristics when the labels are created. The information must originally come from the DD statement or your program. Once the label is written, however, you need not specify the information again. If the data set characteristics change, you simply supply the changes through another information source, and the system modifies your label.

Now we shall examine another important source of information -- the data control block (DCB).

The Data Control Block

The data control block is an area of main storage in your problem program that serves as the central collection point for the information about your data set. Every data set to be processed must have an associated data control block (DCB).

Programs written in a higher-level language also contain a data control block for each data set; however, the processor for each language is responsible for creating data control blocks. Since the higher level language programmer does not directly create data control blocks, he cannot directly fill them. Instead, he can use the data control block parameter of the DD statement to supply information. The assembler language programmer can use the data control block parameter and must use the data control block macro instruction.

Using the DCB macro instruction you can directly create a data control block and place information into it. You can also change the contents of some data control block fields dynamically, as your program is executing.

A data control block contains three categories of information: data set attributes, processing description, and exit information.

Data Set Attributes: Attribute fields contain information about record length, record format, block size, and data set organization (the organizations are discussed later in this section). For some organizations, information on a key portion of the records is also included.

Processing Description: This information describes buffer construction and handling methods, device dependent information (e.g., printer spacing or magnetic tape density), and special processing options available for some data organizations.

Exit Information: This category contains the address of routines you may want to use for special I/O purposes. For example, you may have a routine to do some processing after all of an input data set has been read, or a routine to do special label processing.

Separate input data sets may be logically connected (concatenated) for the duration of a job step. For example, five data sets, one produced on each day of a week, might be used as input to a single sort. To concatenate data sets, describe each by a separate DD statement, but give a name (ddname) to the first statement only; each data set in the group shares this common ddname. (The ddname is the name of the DD statement and it provides a logical relationship to the data control block that specifies the same ddname. The ddname should not be confused with the data set name -- DSNAME -- which specifies the name of a newly defined data set, or refers to a previously defined data set.) Whenever this ddname is used, each data set is automatically used, in the same sequence as the DD statements describing them.

ACCESS TECHNIQUES

The data management programs provide two general techniques for moving data; they are the queued technique and the basic technique. The queued technique offers you the maximum amount of automatic I/O facilities. The basic technique places some of the responsibility for data handling on the programmer, but gives him more direct control of I/O operations.

Queued Access Technique

The queued access technique deals with individual records. A record is retrieved by the GET macro instruction and written out by the PUT macro instruction. The first time you issue a GET, the data management programs move a block of one or more records into input buffers and either give you the address of the first logical record or place the record into your work area, whichever you specify. Each succeeding time you issue GET, you receive another record. When an input buffer is exhausted, it is automatically refilled with another block. Similarly, the PUT macro instruction places records into output buffers and, when a buffer is full, writes out the records.

When using the queued access technique, you can concentrate on data processing alone; the data management routines handle most I/O considerations. For example, I/O is automatically synchronized with your processing. When you issue a GET, the record you want is already in an input buffer, so processing may continue without delay. When a buffer is exhausted, the data management routines automatically refill it. The same principle applies for output records. They are collected in an output buffer and written when the buffer is full. If you are operating under a priority scheduling system, and if output is directed to a system output class (a class of system output units shared by all jobs), your data may be first written on a direct access device. When scheduling permits, the data is transferred to the proper device by the output writer, a job management program.

Since the queued access technique brings records into main storage before you actually request them, the data management programs need a method of anticipating your demands. Therefore, the queued access technique may be used only to retrieve records in a sequential order, as, for example, records on magnetic tape.

The Basic Access Technique

The basic access technique provides you with the READ and WRITE macro instructions for input and output. These instructions move blocks, not records. When you issue a READ, the data management programs fill an input buffer with a block. If the block contains more than one record, you must deblock it (isolate the individual record) yourself. Similarly, the WRITE macro instruction transmits a block to secondary storage. (As with the queued access technique, actual transmission to your specified device may be deferred and done by the system output writer if you are working under a priority system and output is going to a system output class.)

Unlike the queued access technique, the basic technique does not provide automatic synchronization of program processing and I/O. When you issue a READ, you cannot assume that the record is in main storage as you can with GET. You have the responsibility for determining that the I/O operation has been completed before you attempt to use the desired record. Data management provides macro instruction facilities to check for successful completion of I/O operations and, if necessary, to wait for completion of them.

DEVICE INDEPENDENCE

By selecting those functions of the access techniques that deal with the data transmission rather than device control, and by organizing data sequentially, you can write programs that are independent of the I/O devices they use. Such a device-independent program could, for example, accept an input data set from any magnetic tape or direct access device, or from any card reader; output could be recorded on any appropriate I/O device. All device-dependent information, such as device type and identification, may be supplied through job control statements.

Therefore, you are not tied to any I/O configuration after assembling or compiling your program. A program may be designed and tested with the programmer having no knowledge of the I/O devices to be used for execution. In fact, new devices may be added to an installation and used by the program without reprogramming or recompilation. Finally, you may experiment with different combinations and types of I/O devices to achieve optimum performance with your device-independent programs.

Programs that do not use sequentially organized data, or that must use device control facilities, still have some degree of device independence. Such a program is not tied to any specific device, although it may require a specific type, such as a disk drive or a magnetic tape drive.

THE SHARED DIRECT ACCESS DEVICE (SHARED DASD) OPTION

The Shared DASD option is selected at system generation time. It enables independently operating computing systems to share common data residing on shared direct access storage devices. This option is available with PCP, MFT and MVT (excluding MVT with Model 65 multiprocessing), and it provides control program functions needed to control device reservation and release.

System performance in a Shared DASD environment is highly application dependent, but, for most applications, you can expect:

- A reduction in data set creation and maintenance time and costs, and a reduction in space requirements, whenever a single data set can service multiple users.
- Flexibility in scheduling jobs on any CPU that has access to the common data base.
- Improved system availability through the continuation of job scheduling in the event of a malfunction in any CPU.
- An increase in the amount of work processed against a specific collection of data.

These general advantages are gained at the cost of a somewhat increased overhead time.

Use of the Shared DASD option requires a substantial planning effort on the part of the system programmer and may increase the operator's responsibility. For further information on the Shared DASD option and specific details concerning its use see the System Programmer's Guide publication.

What the System Does

Labels are used by the operating system to identify volumes and the data sets they contain, as well as to store data set attributes. Magnetic tape volumes can have standard or nonstandard labels, or they can be unlabeled. Direct access volumes used by the operating system must have standard labels. (Specific information on the contents and formats of standard labels is contained in the following publications IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646 and IBM System/360 Operating System: Tape Labels, GC28-6680.)

Direct Access Labels

Each direct access volume is identified by a volume label, which is stored in a standard location. This label contains a volume serial number and the address of a volume table of contents (VTOC). The table of contents, in turn, contains a data set label for each data set stored on the volume. Data set labels on direct access devices are called data set control blocks (DSCB). The system builds a DSCB for each output data set on a direct access volume.

Each direct access volume must be initialized by an IBM-supplied utility program before being used on the system. This utility program generates a volume label and prepares the table of contents. (See Figure 13.)

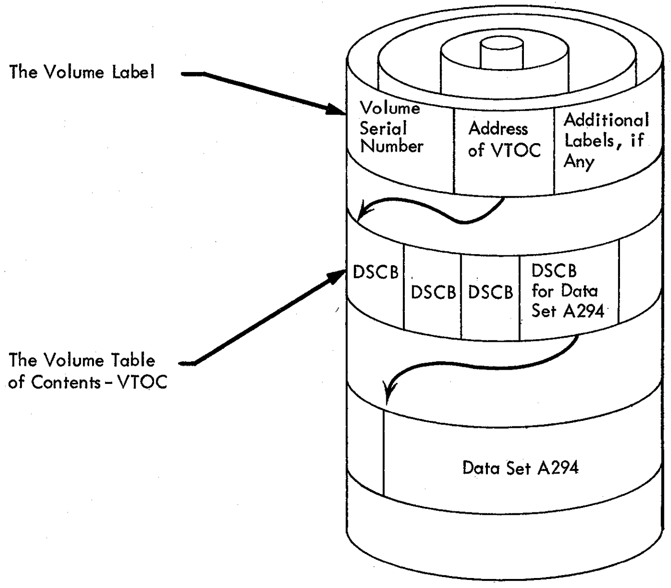


Figure 13. Direct Access Label

Magnetic Tape Labels

Standard magnetic tape labels consist of two parts: the volume label group, and header and trailer labels. The volume label group consists of a standard initial volume label and up to seven additional volume labels. These additional labels are optional and, if used, must be processed by your own routines. The initial volume label identifies a volume and its owner.

The header and trailer labels precede and follow each data set on the volume. Header labels contain system information, device-dependent information and data set characteristics. Trailer labels are almost identical to header labels and are used for the read backward feature on tape devices. (See Figure 14.)

Data Set Retrieval Services

Until now, we have been concerned with the data management programs that help your program manipulate data. But, as a programmer, you must also be concerned with your data when it is not being used. You must know which volume contains the data, and where that data is placed on the volume. Because these chores can quickly become tiresome and time-consuming, data management has facilities for automatically recording the serial number of a volume containing your data set, and the type of device on which the data set should be mounted. If more than one data set is recorded on a tape volume, each is assigned a sequence number to identify its relative position on the volume.

This information is associated with your data set's name and recorded on a direct access device in a logical structure called the catalog. When you need the data set, you supply its name through your job control language statements, and the system retrieves it. You must remember only the name. The system searches the catalog for the information associated with the name, then, acting on the information it finds, asks the operator to mount your volume on a device.

Because duplication of data set names in the catalog is not allowed, data management provides a scheme to help you establish unique names. The scheme works this way. Instead of supplying a one-word name, you can build a composite name of many words, each separated by a period. For example, the first name of a data set might be INPUT. Then you might want to qualify it with a more specific name -- perhaps PAYROLL -- followed by an even more specific name like APRIL. Its full name would then be INPUT.PAYROLL.APRIL (see Figure 15).

Generation Data Groups

A special facility is provided to name data sets that are updated periodically or are logically part of a group of data sets, each of which is created at a different time. Each update of the data set is called a generation; the number associated with it is called a generation number; a generation data group is the entire collection of chronologically related data sets that can be referred to by the same data set name. A particular generation can be referred to by either the absolute generation name or a relative generation number.

For example, the external name for all data sets within a generation data group for a payroll might be named A.YTDPAY. Each data set is also automatically

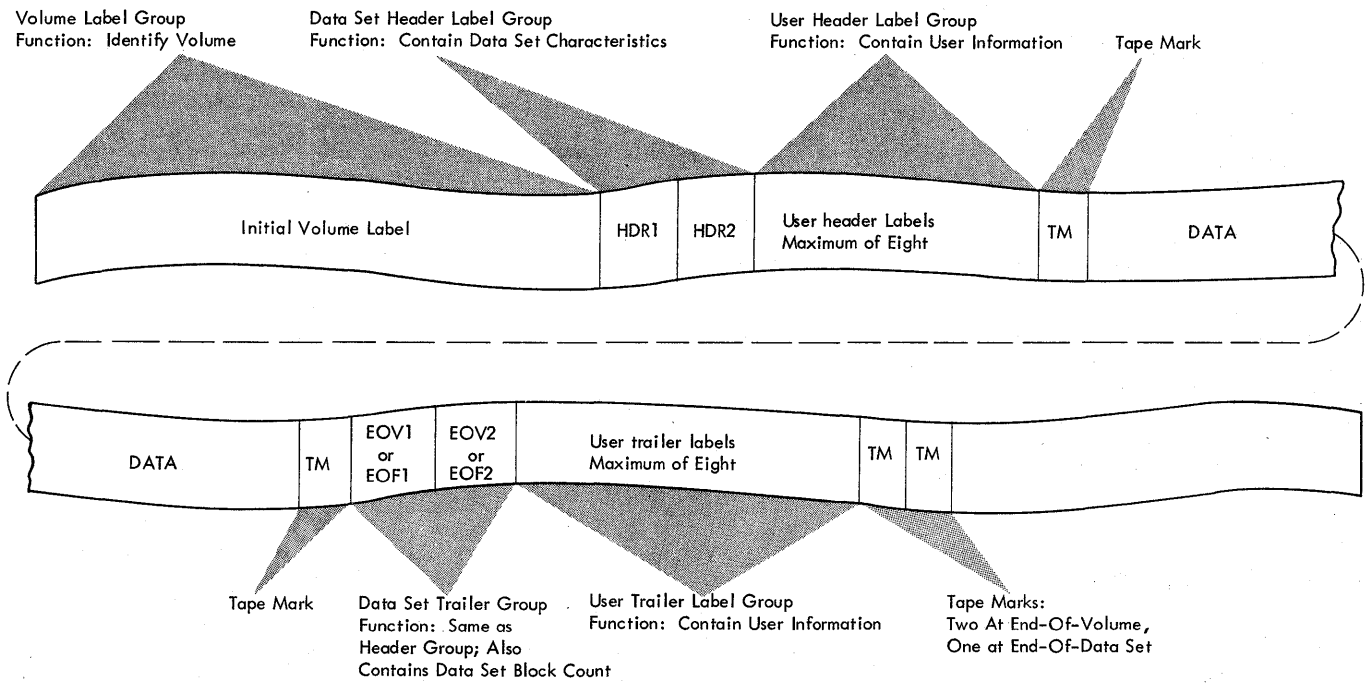


Figure 14. Standard Magnetic Tape Label

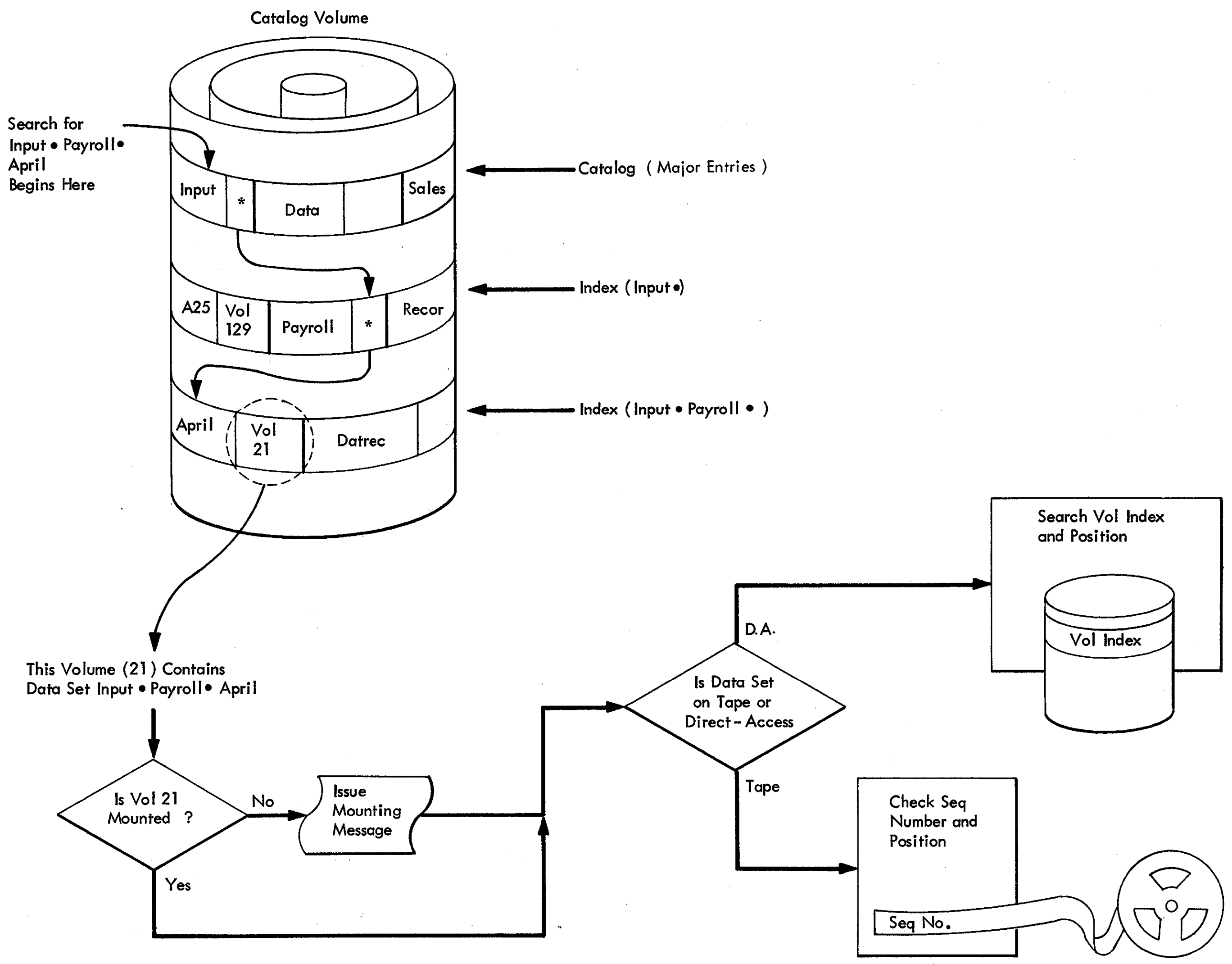

Catalog Volume
Search for Input • Payroll • April Begins Here
Input
*
Data
Sales
Catalog (Major Entries)
Index (Input •)
A25
Vol 129
Payroll
*
Recor
Index (Input • Payroll •)
April
Vol 21
Datrec
This Volume (21) Contains Data Set Input • Payroll • April
Is Vol 21 Mounted ?
No
Yes
Issue Mounting Message
Is Data Set on Tape or Direct-Access
D.A.
Tape
Search Vol Index and Position
Vol Index
Check Seq Number and Position
Seq No.

Figure 15. Data Set Retrieval Through the Catalog

assigned a simple name in the form of a generation and version number, e.g., G0032V00, which represents generation 32, version 0. The next data set name which is assigned automatically is G0033V00. The external name qualifies the generation and version number. This automatic naming permits you to refer to this data set by its absolute name (A.YTDPAY.G0033V00) or by a relative name, e.g., A.YTDPAY(0). This latter refers to the latest cataloged version. A.YTDPAY(+1) identifies a new data set to be added to the group, and A.YTDPAY(-1) identifies the next-to-the-latest generation.

When the index for the generation data group is established, you specify how many generations you want saved. The generations have constant attributes. As a new generation is cataloged, the oldest generation is either automatically destroyed or is deleted from the catalog. Alternatively, you may specify that all old generations of a full generation data group series be deleted from the catalog when the succeeding generation is added, so that the new entry effectively becomes the newest and only member of the series.

When the generation data group index is established, a model data set label is built for it. This model is used for each succeeding generation to supply uniform attributes.

For a complete discussion of generation data groups, see the Utilities publication.

Collecting Data Set Information

We have discussed the facilities for supplying data set information and the central collection point for the information. Only one link is missing from the chain of communication between you and data management: the information must be collected and physically placed in the data control block. Data management's OPEN routines do this collecting and perform important housekeeping operations necessary to begin I/O operations. The OPEN routines are brought into storage by the OPEN macro instruction in assembler language programs, and by compiler-generated instructions in programs written in higher-level languages.

The OPEN routines first search the data control block to determine what information is already present -- placed there during assembly -- and what is still missing. They go to your DD statement for information to fill vacant data control block fields, and finally they go to the data set label. (Labels do not yet exist for new data sets, so the OPEN routines begin to create them and fill them with information from the other sources.)

Once a data control block field is full, it is not changed by OPEN routines. (These fields can be changed dynamically in an assembler language program.) If the DD statement or the label contains information already in a data control block field, the new information is disregarded and the field remains unchanged. (See Figure 16.)

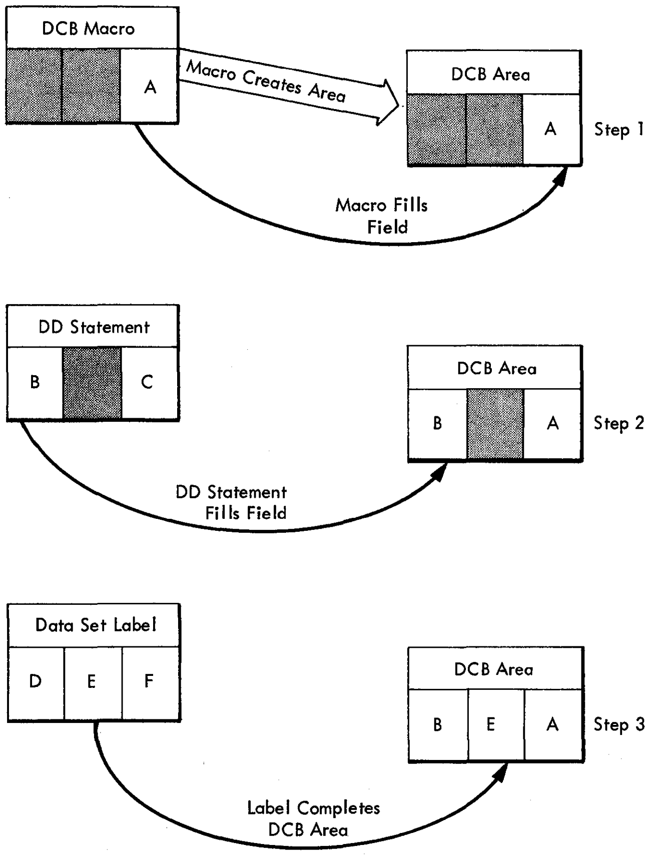


Figure 16. How OPEN Fills a Data Control Block

When you are through processing a data set, its related data control block is restored to its original condition, i.e., information the OPEN routines put in it is deleted. Construction of labels for new data sets is completed, and existing labels may be updated. The data control block is restored so that it may be used for other data sets to be processed.

Other Functions of OPEN

In addition to completing the communication chain between you and data management, the OPEN routines do all the initialization necessary to begin actual data transfer.

Data Set Security: The data set security facility of the operating system prevents unauthorized manipulation of protected data sets. To be protected, a data set must be recorded on a magnetic tape volume with standard labels, or on a direct access device, which always has a standard DSCB. A flag is set "on" in the label to signal protection. The OPEN routines test the flag and, if it is on, they request a password from the operator. Unless he enters the proper password, the OPEN routines cannot continue, and, therefore, the data set cannot be used. If the operator enters an incorrect password (he has two chances), the program terminates abnormally.

Volume Checking: The initiator program of job management is responsible for directing the computer operator to mount specific volumes. The OPEN routines check the volume serial of direct access volumes and standard-label tapes to verify that the proper volume was mounted. If an error is found, the OPEN routines issue mounting instructions again.

Buffer Creation: The creation and manipulation of I/O buffers is a subject discussed later in this section. You should be aware, however, that the OPEN routines are capable of obtaining buffers for you if you do not choose another method of acquiring them.

Buffer Priming: When you use the queued access technique to manipulate sequential records, the OPEN routines anticipate your input requests and fill your input buffers automatically. Before you issue the first GET for the data set, the records are being read into main storage.

ACCESSING DATA

After the OPEN routines finish processing, data management is ready to permit accessing of data. Data management can access data organized in many ways. The following paragraphs explain these organizations.

Sequential: The records of a sequential data set are consecutive, as are the records on magnetic tape. Sequential organization is the only one possible for tape drives, printers, and card readers and punches. The location of each record in a sequential data set does not depend on its contents. Normally, no record can be read or written until all preceding records are read or written, but special facilities are available to permit arbitrary positioning. Records cannot be lengthened, shortened, or deleted without rewriting the subsequent portion of the data set. Both the queued and the basic access technique may be used for sequential data sets. (See Figure 17.)

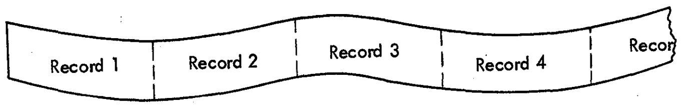


Figure 17. Data Organized Sequentially

Partitioned: The partitioned organization is simply a convenient way to arrange sequential data on a direct access device. A partitioned data set consists of independent groups of sequentially organized data sets, each identified by a member name in the directory. (Earlier we referred to a partitioned data set as a library.) Each member consists of sequentially organized records identified by a single, simple name and, optionally, by alternate names called aliases. The directory is a list of the member and alias names and contains the starting address of each member; the address associated with an alias can be the location of any record in a member. Directory entries are maintained in order of the collating sequence of member names. (See Figure 18.)

Indexed Sequential Organization: Neither the sequential nor the partitioned organization allows convenient access to any particular record without first reading the preceding records or without maintaining a special, separate record that gives the position of the desired record. The indexed sequential organization overcomes this difficulty by making the location of each record in the data set depend on the contents of a key portion of each record. If you supply the contents of the record's key, the system can find the record directly.

To create an indexed sequential data set you must sort the records on a key. These ordered records are then written on a direct access device. Each time a track is filled, the highest key on the track is stored in a track index. When all the tracks of a cylinder are filled, the highest key on the cylinder is stored in a cylinder index.

To read a record, you supply the record's key. Data management then searches the cylinder index for the cylinder containing that key. When the cylinder is located, its track index is searched for the track containing the key of the desired record. Finally, the track is searched for the record containing the key you specified, and the record is retrieved.

This organization lets you insert records without rewriting the entire data set. For example, suppose you wish to insert a record with a key of 599. Further suppose that each track of your data set is completely filled by two records; the new record belongs on the track between records of key value 598 and 600. Since the track is already full, the new record is written after record 598, and record 600 is rewritten on an overflow track. The track index is marked to indicate that an overflow track exists. If there are no overflow tracks available or if they are all full, the record is not written and you are notified.

The indexed sequential organization may be used with both access techniques. Records may be processed sequentially using the queued technique, or individual records may be processed directly using the basic technique. (See Figure 19.)

Direct: If the organizations above do not meet your specific requirements, you can organize records on a direct access device using direct organization, which allows nonsequential data processing.

To read or write a block, you must specify an actual address (i.e., device, cylinder, track, and block position) or an address that is relative to the beginning of the data set. If your blocks contain a key, you can have the data management routines search for a block with the key you specify. The search begins at an actual or relative address that you supply.

The direct organization is permitted on direct access devices only. The queued access technique cannot be used.

Telecommunications: Transmission of data to and from a telecommunications device is considerably different from transmission to and from local I/O devices. However, once telecommunication messages are in input or output buffers, they are similar to conventional records and therefore, can be handled with the queued or basic technique (QTAM or BTAM).

Telecommuncations data transmission techniques can be used by a central computing installation to process jobs entered from remote locations. Remote Job Entry (RJE) is a program that uses the basic technique (BTAM) to provide access to the data processing facilities of a central system for authorized users at remote terminals. RJE provides users with the facility to execute jobs submitted from remote terminals. Execution is under the control of operating system job management routines.

For further information on RJE see IBM System/360 Operating System: Remote Job Entry, GC30-2006.

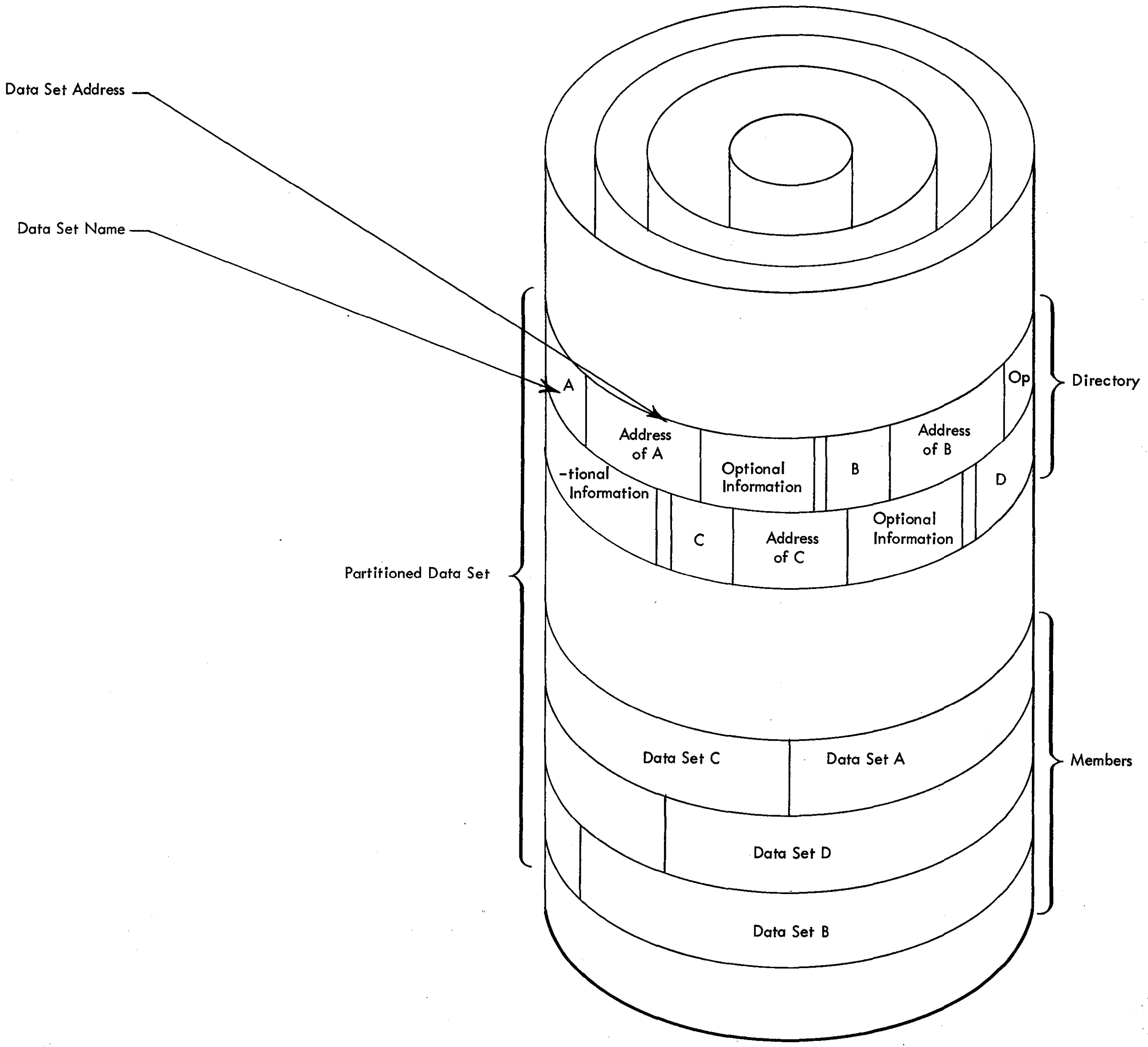


Figure 18. Partitioned Data Set

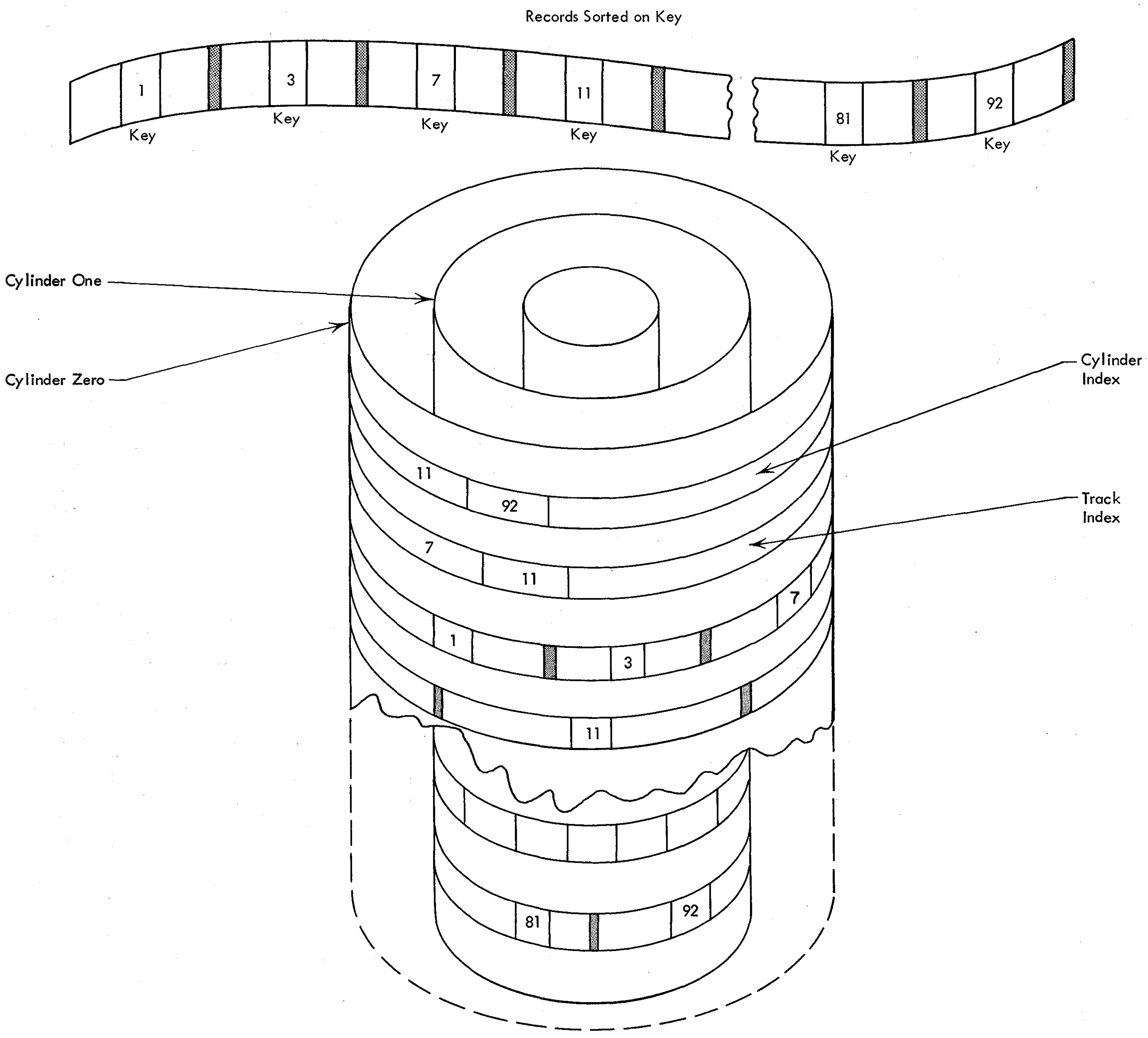


Figure 19. Indexed Sequential Data Set

MANAGING BUFFERS

An important part of I/O operations is providing areas of main storage to hold data after input and before output. Data management provides facilities that enable you to structure main storage for buffers, schedule these buffers to receive input or output, and make the data in the buffers available to your program.

Allocating Main Storage for Buffers

Before main storage can be used for buffers, information must be specified describing the size and number of buffers required. The process of storing this information is called constructing a buffer pool.

Three methods are provided for constructing buffer pools. The first method is to declare an area to be used as a buffer pool. The area is either defined at assembly time or acquired by a GETMAIN macro instruction. The second method is to obtain a buffer pool dynamically at execution time. The third method is to let the OPEN routines automatically obtain main storage and construct the pool for you. A special macro instruction, BUFFER, is available when you use the queued access technique for telecommunications data.

Buffer Scheduling

Simple Buffering: With simple buffering, one or more buffers are taken from the pool assigned to a data set. These buffers are long enough to hold the maximum length block. They are used only for the data set to which they are assigned, e.g., a buffer assigned to an input data set cannot be used for an output data set.

Exchange Buffering: Exchange buffering can be used for input or output, or both. It is available only with the queued sequential access method (QSAM). Access methods are described under "Data Processing Techniques" in this section. Each buffer is large enough to hold one block; the buffer may be divided into segments, each containing one record. When you request a record, the segment containing that record is logically isolated from the buffer and made available to you. However, you must give the buffer a piece of your work area. This piece is logically added to the buffer to restore it to the proper size. When you want to write the record, you give the segment containing the record to the output buffer, in exchange for an equal-sized piece of the output buffer that you can use as a work area.

The advantage of exchange buffering is that records are not physically moved in main storage. A record is written into an input buffer segment, worked on while it is in the segment, and written from the segment. In contrast, simple buffering requires at least one data move, as we will see in the discussion of transmittal modes. Figure 20 illustrates exchange buffering in the substitute mode.

Chained Segment Buffering: This assignment technique is used for telecommunications applications. Since the length of telecommunications messages is unknown, buffer segments are assigned dynamically, during data transfer. Messages will not normally fit in one segment, so address chaining techniques are used to connect the physically separate segments that contain a message.

Dynamic Buffering: This technique is used only for the access techniques that allow multiple READ requests to be queued. In such cases, a buffer is not assigned to each READ request before the request is executed. Instead, a buffer is automatically taken from a pool and assigned just before data transfer begins. The buffer remains in use until it is returned to the pool. It is returned automatically when its contents are written, or under programmer control with a macro instruction. After returning to the pool, the buffer can be dynamically assigned to another active READ request. The advantage of dynamic buffering is that relatively few buffers are needed since the READ requests waiting in the queue do not monopolize buffers.

Transmittal Modes

The following discussion applies only to the queued access technique.

Move Mode: When you issue a GET macro instruction using this mode, an input record from an input buffer is physically moved into your work area. A PUT macro instruction moves an output record from your work area or input area to an output buffer.

Locate Mode: When you issue a GET in this mode, you receive the starting address of a record. You can process the record in the buffer, or move it to your work area. A PUT gives you the address of an output buffer area; you must move the record to the buffer yourself, or tell the system to use it for the next GET-move.

Data Mode: When you issue a GET macro instruction using this mode, the data portion of an entire record will be moved from the input buffer to your work area. This record is composed of the data portions of one or more segments within the data set. A PUT macro instruction using this mode moves the contents of your work area into the output buffer and records the entire record as one or more segments within the data set.

Substitute Mode: When you use exchange buffering you receive data in the substitute mode. It is similar to the locate mode in that a GET gives you the address of an input record. You may work with the record as though it were in your work area, but you must give the buffer an area of main storage equal in size to the one you take. With a PUT, you give the output buffer the area of main storage containing your output record and, in exchange, you receive a piece of output buffer equal in size. Data is never physically moved with this mode. (See Figure 20.)

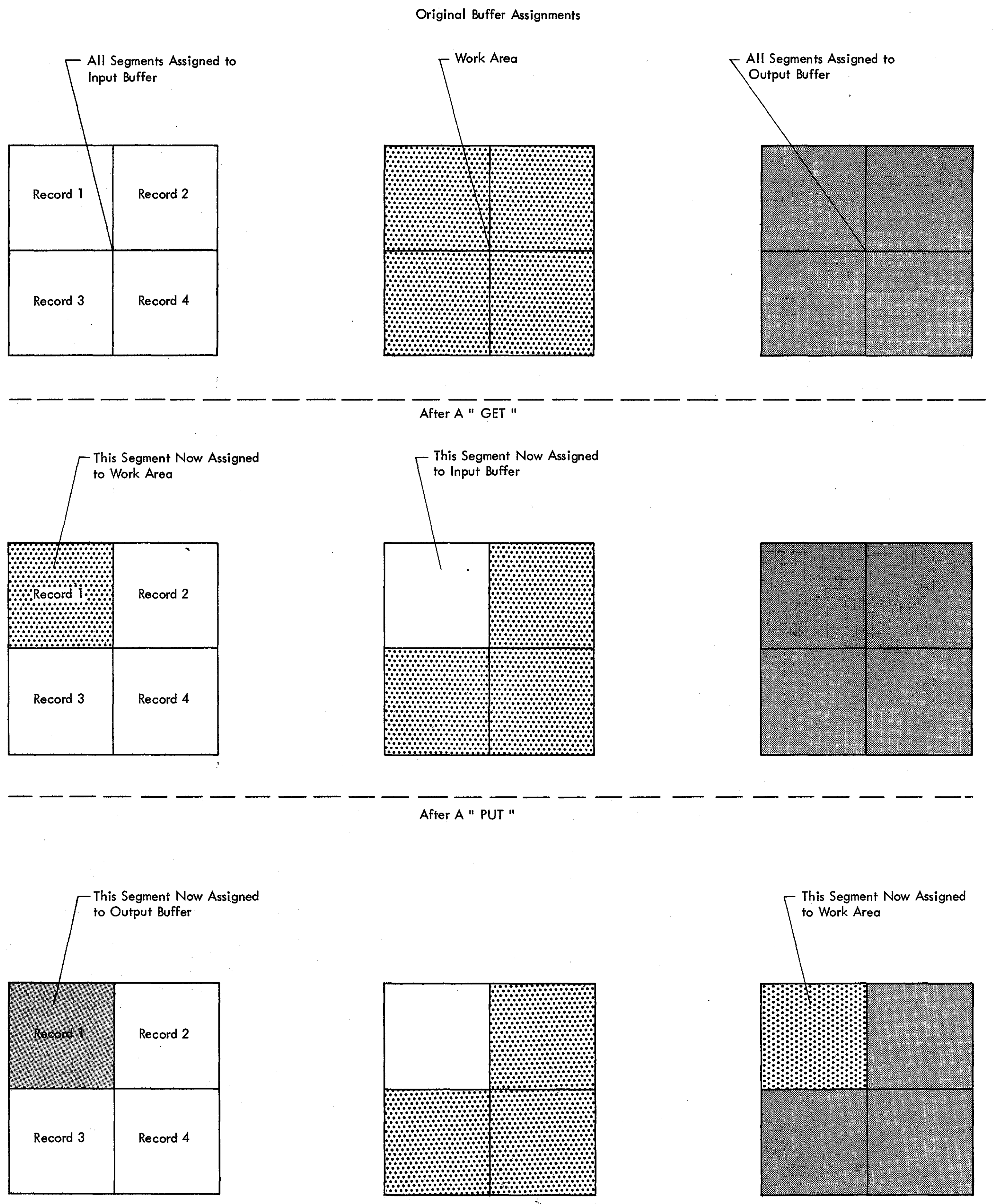

Original Buffer Assignments
All Segments Assigned to Input Buffer
Work Area
All Segments Assigned to Output Buffer
Record 1
Record 2
Record 3
Record 4
After A " GET "
This Segment Now Assigned to Work Area
This Segment Now Assigned to Input Buffer
Record 1
Record 2
Record 3
Record 4
After A " PUT "
This Segment Now Assigned to Output Buffer
This Segment Now Assigned to Work Area
Record 1
Record 2
Record 3
Record 4

Figure 20. Exchange Buffering -- Substitute Mode

DATA PROCESSING TECHNIQUES

The operating system allows you to concentrate your efforts on processing the records read or written by the data management routines. Your main responsibility is to describe the data set to be processed, the buffering techniques to be used, and the access method. An access method can be defined as the combination of data set organization and the technique used to process the data. As previously defined, data access techniques are divided into two categories -- queued and basic.

Access methods are identified primarily by the data set organization to which they apply. For instance, we speak of a basic access method for direct organization (BDAM). Nevertheless, an access method identified with one organization can be used to process a data set usually thought of as organized in a different manner. For example, BSAM and QSAM can easily be used to process partitioned and direct organization data sets. Thus, a data set considered to be in a direct organization is created using the basic access method for sequential organization (BSAM). It is processed using the basic direct access method (BDAM). If the queued access technique is used to process a sequential data set, the access method is referred to as QSAM.

The basic access methods are used for all data organizations, while the queued access methods apply only to sequential and indexed sequential data sets and telecommunications, as shown in Table 4.

Table 4. Data Access Methods

Data Set Organization	Access Technique	
	Basic	Queued
Sequential	BSAM	QSAM
Partitioned	BPAM	
Indexed Sequential	BISAM	QISAM
Direct	BDAM	
Telecommunications	BTAM	QTAM

A summary of access methods is shown in Table 5.

Table 5. Access Method Summary

Organization	Sequential		Partitioned	Indexed Sequential			Direct
				QISAM			
Access Method	QSAM	BSAM	BPAM	LOAD	SCAN	BISAM	BDAM
Primary macro instructions[1]	GET, PUT, PUTX	READ WRITE	READ,WRITE FIND,STOW	PUT	SETL,GET, PUTX	READ WRITE	READ WRITE
Synchronization of program with I/O	Automatic	CHECK	CHECK	Automatic	Automatic	WAIT	WAIT
Record format transmitted	Logical F,V Block U	Block F,V,U	Block (Part of member) F,V,U	Logical F,V	Logical F,V	Logical F,V	Block F,V,U
Buffer creation and construction	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPCOL Automatic	BUILD GETPOOL Automatic
Buffer technique	Automatic Simple Exchange	GETBUF FREEBUF	GETBUF FREEBUF	Automatic, Simple	Automatic Simple	GETBUF, FREEBUF Dynamic FREEDBUF	GETBUF, FREEBUF Dynamic FREEDBUF
Transmittal modes (work area/buffer)	Move, data, locate, substitute			Move, Locate	Move, Locate		

[1]All macro instructions introduced in this table are defined in the publication IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, GC28-6647.

OPERATING SYSTEM VOLUME STATISTICS

Operating system efficiency is affected by the condition of volumes stored on a medium subject to deterioration with use, such as magnetic tape. During use, tape is stretched, flexed, and rubbed, causing its oxide coating to crack or to be eroded. Eroded particles of oxide, fingerprints, and dust contaminate its surface, multiplying erosion and breaking contact between tape and the read/write station. In due time, there will be failures in the read/write process.

A rapidly rising rate of read and write errors, if detected, would signal the probability of a deteriorating tape. If the failure rate could be monitored, it would be possible to judge the condition of the volume and rescue its contents. This could be done before system performance could be seriously affected by reconditioning, by transfer to a different volume, or by a combination of both processes.

Read and write errors can be monitored by a facility called Operating System Volume Statistics. This facility has two options: Error Statistics by Volume (ESV); Error Volume Analysis (EVA). Both options can be specified at the same time.

ESV causes the system to collect statistics for each tape volume during any interval that the volume is open. However, the volumes must be labeled (or be an unlabeled tape whose serial number has been identified to the system) and only an abridged set of these statistics is collected if the ESV records are to be printed at the console rather than on a line printer. These statistics include:

- The volume serial number.
- The CPU serial number.
- The System model number.
- The date and time of day this set of statistics was collected.
- The address of the unit on which this volume was mounted and the channel through which it was operating.
- The number of temporary read or write errors that occurred.
- The number of permanent read or write errors that occurred.
- The number of noise blocks encountered.
- The number of erase gaps written while trying to correct write errors.
- The number of cleaner actions initiated while trying to correct read or write errors.
- The number of Start I/O operations encountered.
- The bit density of the volume (in bits per inch).
- The physical record length of the volume for fixed length records; the length value is forced to zero for undefined or variable length records, and when the EXCP access method issued.

The system operator can select one of two standard output formats at system generation. One of these options, called SMF (for System Management Facilities; SMF is discussed in Section 5 of this publication) causes the ESV statistics to be entered, with other system information, into one of the SMF data sets. The user can provide his own access method, specify his own record format, and select his own recording data set. If he chooses to do this, he must specify ESV=CON at system generation time in order to obtain unit control blocks (UCBs) in the proper format for collecting volume statistics.

The EVA option requires the system operator to specify two minimum values (at system generation), one for the number of temporary read errors and one for the number of temporary write errors. If the number of read or write errors for a volume currently being accessed equals the corresponding values specified by the system operator, the system will print a message to this effect at the console. EVA can be used both for labeled and unlabeled volumes.

Language Comparison: Data Management Facilities

All data management facilities are available to users of the assembler language. Table 6 indicates which of the facilities are available to users of higher-level languages. The table assumes the use of the job control language.

• Table 6. Language Comparison: Data Management Facilities

Data Management	ALGOL	COBOL			FORTRAN			PL/I	RPG
		E	F	ANS[1]	E	G	H		
Automatic Buffer Pool Construction	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Basic Access Technique	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Buffer Control	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Buffer Pool Construction	No	No	No	No	No	No	No	No	No
Chained Scheduling for I/O Operations	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Concatenating Sequential Data Sets	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Control of Confidential Data: Password Protection	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Multi-Volume Data Set Processing	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Entering a Generation Data Group in the Catalog	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Exchange Buffering	No	No	No	No	No	No	No	No	No
Processing a Direct Data Set	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Processing a Partitioned Data Set	No	No	No	No	No	No	No	No	No
Processing an Indexed Sequential Data Set	No	Yes	Yes	Yes	No	No	No	Yes	Yes
Queued Access Technique	No	Yes	Yes	Yes	No	No	No	Yes	Yes
Sequential Data Sets: Device Control	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Sequential Data Sets: Device Independence	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Simple Buffering	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Record Overflow	No	No	No	Yes	Yes	Yes	Yes	Yes	No
Write Validity Check	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes

[1]American National Standard COBOL.

Section 4: Job Management

Because the purpose of the operating system is to keep the computing system as busy as possible, the work to be done must be available when the system is ready to handle it. The job management programs direct and control the flow of one or more jobs through the computing system. The job control statements provide the communication link to the system.

A job is a total processing application comprising one or more related processing programs, such as a weekly payroll, a day's business transactions, or the reduction of a collection of test data. Jobs comprise one or more job steps: that unit of work associated with one processing program. The sequence of control statements and data submitted to the operating system on an input unit especially activated for this purpose is called the input stream.

This section will expand the job management processing block into more detail. In general, the job management functions are:

1. Analysis of the input stream: scanning the input data to identify control statements; interpreting and analyzing the control statements; preparing the necessary control tables that describe each job to the system.

2. Allocation of I/O devices: ensuring that all necessary I/O devices are allocated; ensuring that direct access storage space is allocated as required; ensuring that the operator has mounted any required tape and direct access volumes.

3. Overall scheduling: selecting jobs for execution, either on a sequential or priority basis.

4. Transcription of input data onto, and user output from, a direct access device.

5. Communication between the operator and the system.

What the User Must Do

The user tells the system about his job and job steps through the job control language.

Using the job control language, you must provide job and program information, data characteristics, and device requirements at the time the program is executed rather than when you assembled or compiled it. Other facilities of the language allow you to:

- Copy existing data set names, control statements, and control blocks with a back reference facility to reduce recoding.

- Retrieve data sets by name using the system catalog, eliminating the necessity of identifying the unit type and volume serial numbers.

- Optimize use of channels, units, volumes, and direct access space.

- Pass data sets from one step to another to reduce mounting and retrieval time.

- Share data sets between two or more jobs operating independently in multiprogramming environments.

Job Control Statements

Communication between the operating-system user and the job scheduler is effected by six job control statements:

1. Job statement
2. Execute statement
3. Data Definition statement
4. Command statement
5. Delimiter statement
6. Null statement

Parameters coded on these statements aid the job scheduler in regulating the execution of jobs and job steps, retrieving and disposing of data, allocating input/output resources, and communicating with the operator.

Summary of Job Control Statements

The job statement (the JOB statement) marks the beginning of a job and, when jobs are stacked in the input stream, marks the end of the control statements for the preceding job.

The execute statement (the EXEC statement) marks the beginning of a job step and identifies the first load module to be executed, or the cataloged procedure.

The data definition statement (the DD statement) describes the data sets to be used in the job step.

The command statement is used by the operator to enter commands through the input stream. Commands can activate or deactivate system input and output units, request printouts and displays, and perform a number of other operator functions.

The delimiter statement and the null statement are markers in an input stream. The delimiter statement is used, when data is included in the input stream, to separate the data from subsequent control statements. The null statement is used to mark the end of a job whose last DD statement defines a data set in an input stream.

We shall briefly consider the first three control statements. For full information on the job control language, see the publication IBM System/360 Operating System: Job Control Language, GC28-6539.

Uses of the JOB, EXEC, and DD Statements

The job statement identifies your job and may contain accounting information for use by your installation's accounting routines, give conditions for early termination of the job, and regulate the display of job scheduler messages. With priority schedulers, you can use additional parameters to assign job priority, to assign a job class (the job class determines where your job will be placed on the input work queue. The priority parameter determines the job's initiation priority within its job class), to request a specific output class for job scheduler messages (this applies to PCP also), and to specify the amount of main storage to be allocated to the job. The job statement may also be used to specify step restart for all steps in the job.

The execute statement may provide job step accounting information, give conditions for bypassing the job step, and pass control information to a processing program. With priority schedulers, additional parameters allow you to assign a time limit for the execution of the job step and to specify the amount of main storage to be allocated. The execute statement may also be used to specify step restart for a particular step.

The data definition statement describes a data set and requests the allocation of input/output resources. DD statement parameters identify the data set, give volume and unit information and disposition, and describe the data set's labels and physical attributes.

Applications that require many control statements and are used on a regular basis can be considerably simplified through the use of cataloged procedures. A cataloged procedure is a set of job control statements that are placed in a partitioned data set known as the procedure library. (The procedure library is a system data set named SYS1.PROCLIB.)

Before putting a cataloged procedure on the procedure library, you may want to test it. This can be done by converting the procedure to an in-stream procedure. An in-stream procedure is a set of job control statements placed in the input stream that can be used any number of times during a job by naming that procedure in an execute (EXEC) statement. Another advantage to in-stream procedures is that they can give you the facility of a cataloged procedure without being placed on the procedure library. After testing the procedure, you may keep it in card form and simply insert it in the input stream whenever you want to use it. For a detailed description of in-stream procedures, see the publication IBM System/360 Operating System: Job Control Language Reference, GC28-6704.

What the System Does

Job management contains different functional areas. We have described these areas briefly; now we can discuss them in more detail -- as facilities.

First, we will examine the functions performed by the job scheduler and master scheduler. This will pave the way for a discussion of sequential and priority scheduling, and a closer examination of what you can specify through the job control language.

The Job Scheduler

The job scheduler prepares jobs for execution and schedules their execution on either a sequential or priority basis. In general, the job scheduler portion of the control program performs three types of functions: read/interpret, initiate/terminate, and the handling of output writers.

Read/Interpret: Each reader/interpreter is responsible for reading one input stream. The reader/interpreter reads job control statements, analyzes their contents, and builds tables that are used during initiation and execution of job steps. In a priority scheduling system, the reader/interpreter ensures that the control information is placed on an input work queue by priority and transcribes data from the input stream to a direct access device.

Initiate/Terminate: The initiator/terminator selects (either from the input stream or the input work queue) the next job step to be executed. The initiator/terminator analyzes the I/O device requirements of job steps, allocates devices to them, issues volume mounting instructions, and verifies that the volumes were mounted on the correct device. (For magnetic tape volumes, the OPEN routines verify this.)

As mentioned, the initiator/terminator can issue volume mounting instructions to the operator. When the operator has to perform setups, time is lost. To help minimize the time lost by operator setup, an optional feature is available for Automatic Volume Recognition.

Automatic Volume Recognition: The operator can premount labeled volumes on any available tape or disk device; the job scheduler records the identification of each volume and the device used. When a particular volume is needed for job setup, the table is searched. If the needed volume is already mounted, the usual procedure of issuing a volume mounting message is bypassed. This feature is particularly advantageous in production installations where work schedules normally are set in advance and follow a repeated pattern. In this situation, the operator usually knows which volumes are to be used and the sequence in which they will be used.

Write: During task execution in a multiprogramming system, output data sets may be stored on a direct access device, or they may be written to an output device while the job is executing. Direct system output writers control the writing of output data sets directly to the output device during execution of the job. System output writers can transcribe the data to a system output device (usually a printer or punch). Each system output device is controlled by an output writer task. Moreover, output devices can be grouped into usage classes. For example, a single printer may be designated as a class for high-priority, low-volume printed output; two other printers may be designated as a class for high-volume printed output.

The DD statement allows output data sets to be directed to a class of devices and places a reference to the data on the output work queue. Because the queue is maintained in priority sequence, the system output writers can select jobs in the output work queue on a priority basis.

In systems with input and output work queues, the system output writers are the final link in the chain of control routines. While the execution of one task is proceeding, the output of another task can be written. This increases throughput, that is, the total volume of work performed by the system over a period of time. At two intermediate stages of the work flow, data is accessible from direct access devices, without any unit record requirements. These stages occur when the job has just entered the input work queue, and when the job is completed and placed on the output work queue. At each of these stages, priorities are used to place important work ahead of less important work that might have been prepared earlier. This improves turnaround time, that is, the time elapsed from submitting a problem to receiving an answer.

A multiprogramming system permits managing the concurrent execution of system input readers, system output writers, direct system output writers, and user jobs.

The Master Scheduler

The master scheduler handles operator commands and messages. Messages to the operator from a user's program are initiated by use of the write-to-operator (WTO) macro instruction or the write-to-operator-with-reply (WTOR) macro instruction. You can, therefore, supply information to the operator by using WTO or request information from the operator by using WTOR.

The operator himself can issue various commands. These commands are of the following types:

- Job action commands cause a change in the status of a job (for example, canceling or suspending a job, or modifying its priority).
- System action commands cause changes in the actions taken by the job scheduler. These commands may inform the system of a new device to be used in the input stream, or of a device that is no lcnger available for allocation.
- Information requests allow the operator to inquire about the status of the system or of certain jobs.
- Information entries allow the operator to provide the system with the current date and time, to enter information in the system log, and to reply to system or program requests for information (as a consequence of a WTOR macro instruction).

Operator commands, though they are normally entered into the system from a device such as a console typewriter, can also be placed as separate statements in an input stream.

Summary of Job and Master Scheduler Functions

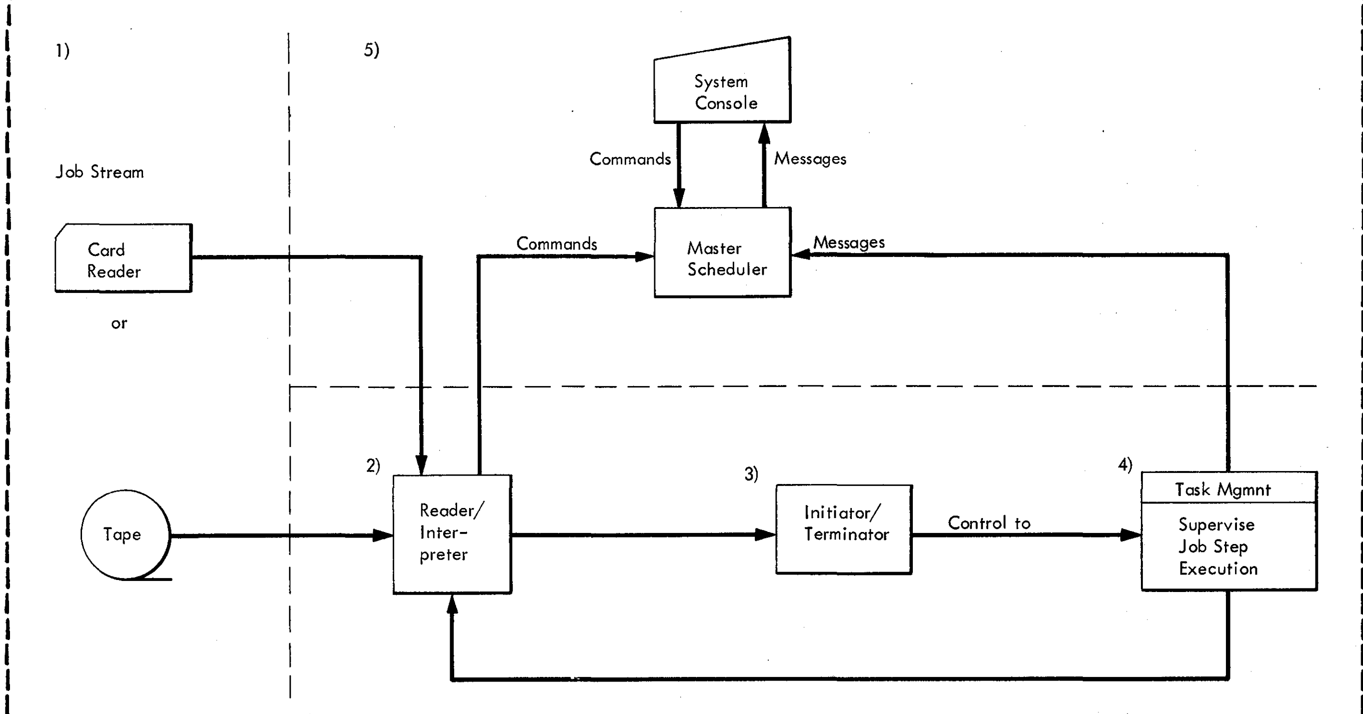
The input stream is read and analyzed by the job scheduler portion of the control program. The job scheduler allocates the I/O devices needed, and then requests the task management programs to initiate execution of the job step defined in the control statements. The master scheduler carries out operator commands which control or inquire about system functions.

By selecting optional scheduler features, you can tailor job management's capabilities to your requirements.

SCHEDULING IN PCP

In PCP, the reader/interpreter scans the control statements for one job step at a time. The initiator allocates I/O devices and notifies the operator of any physical volumes to be mounted; the job step is then ready for execution. Figure 21 depicts scheduling in PCP.

The commands handled by the master scheduler depend on the control program configuration, as discussed in the IBM System/360 Operating System: Operator's Reference, GC28-6691.



1. Your programs, in the form of jobs or job steps defined through the job control language, may enter the system in the input stream from a card or tape device. Input data may be entered into the system with the control statements.
2. The reader/interpreter reads in the control statements for <u>one</u> job step.
3. The initiator/terminator allocates the required I/O devices, notifies the operator of volumes to be mounted (if any), and requests the task management programs to supervise execution of the named job step.
4. The task management programs turn control over to the first load module and supervise its execution.
5. The master scheduler accepts and takes action on commands.

Figure 21. Scheduling in PCP

SCHEDULING IN MFT

In PCP, one job at a time is brought from the input stream into main storage. That job uses all of main storage beyond an area set aside for the control program routines that make up the nucleus of the system. No other job can be brought into storage until the first job is terminated.

In MFT, on the other hand, main storage beyond the nucleus area is separated into one or more areas called partitions. The number and size of these partitions can be redefined at any time. Each partition can service as many as three job queues. The priority of the queues is based on the order in which they were initially specified (at system generation, system initialization, or during operation). That is, if a partition is assigned to service work in job class A and B, all A jobs are scheduled into that partition first; B jobs are scheduled only when there are no more A jobs. Additionally, several partitions may be assigned to service the same job class queues to keep the partitions busy.

The MFT scheduler consists of a master scheduler and a job scheduler. The MFT job scheduler reads input job streams and enqueues jobs on one of fifteen available job queues, corresponding to the class specified in the JOB statement. Position on a queue is determined by the priority specified in the JOB statement. Jobs of equal priority are enqueued on a first-in, first-out (FIFO) basis. Jobs are dequeued from the input queues and initiated according to their place on the queue. (Note: The priority parameter is only a scheduling priority, as such, it has no meaning after the job is initiated. See the "Task Management" section for further details.)

For information on operator actions see the Operator's Guide publication.

SCHEDULING IN MVT

In MVT, jobs are not executed as encountered in the input stream. Instead, control information associated with a job enters an input work queue that is kept on a direct access device. The input on each queue is arranged according to the job class, the initiation of a job within a queue is determined by the priority within the class.

Use of these queues, which can be fed by more than one input stream, permits the system to react to job classes and priorities within the class; and delays caused by the mounting and demounting of I/O volumes. The priority is then applied to the tasks themselves and, in multiprogramming operations, is used to resolve contention between tasks for the system's resources.

Figure 22 depicts scheduling in MFT and MVT.

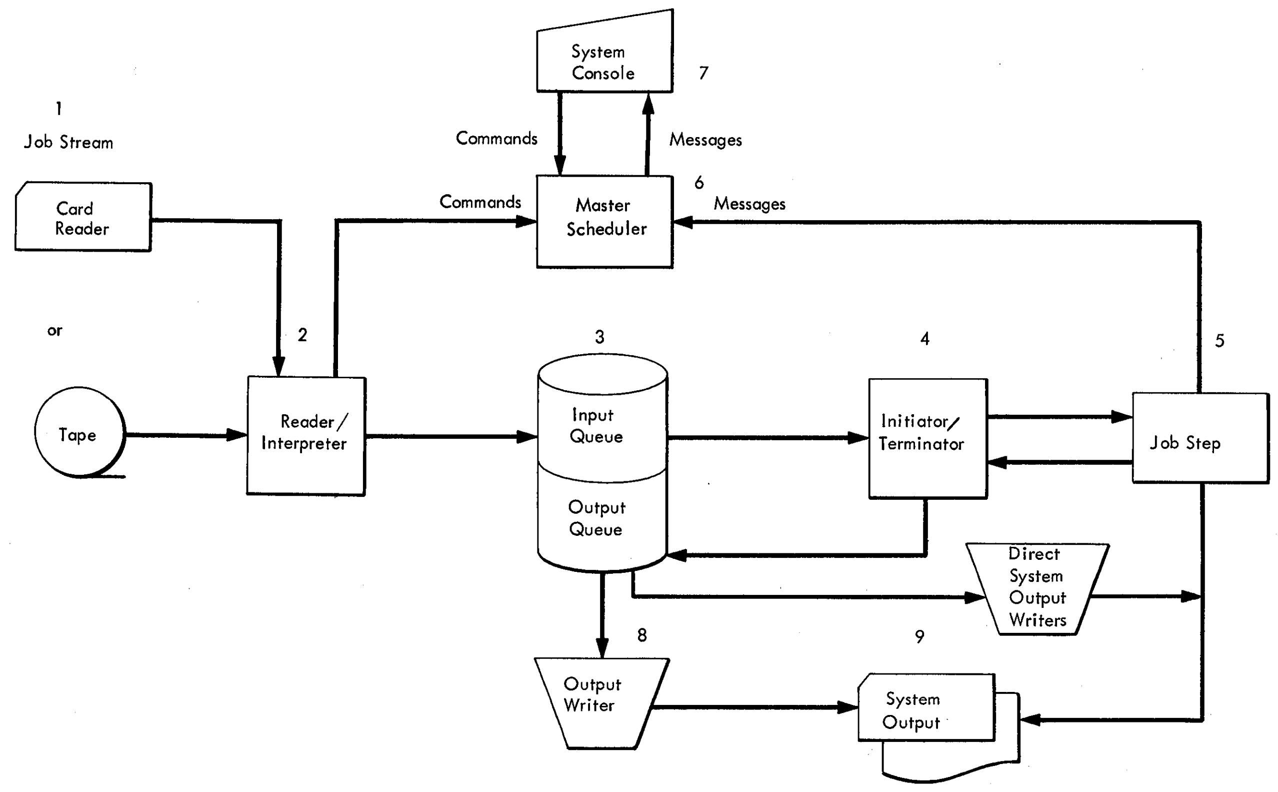
MULTIJOB INITIATION

The MVT initiator/terminator permits multijob initiation. Under operator control, a maximum number of 15 jobs can be underway at the same time. Each job selected is run one step at a time. Jobs are selected from the input work queue and initiated as long as all the following conditions are met:

- Availability of data sets.
- Availability of main storage.
- Availability of I/O devices.
- Jobs are in the input work queue, ready for execution.
- An initiator is available to initiate the job.

Language Comparison: Job Management Facilities

All job management facilities are available to the users of any programming language. The system responds to the requests you make when describing your jobs through the job control language.



1. Your programs, defined as jobs or job steps by the job control language, enter the system through an input stream from a card or tape device. (MFT also supports an input stream from certain direct access devices.)

2. The reader/interpreter reads in control statements for one or more jobs and places them on the input work queue. The input on each queue is arranged by job class, the initiation of a job within a queue is determined by the priority within the class.

3. The job with the highest priority is selected for execution by the initiator/terminator.

4. The initiator/terminator turns your job step over to the task management programs, which supervise its execution.

5. The master scheduler accepts and takes action on commands.

6. Output is written (by job priority) when the job has terminated and while other jobs are being processed, if output data sets are being processed by the system output writer. If output data sets are being processed as direct system output, then the output data sets are written while the job is executing.

Figure 22. Scheduling in MFT and MVT

Section 5: Task Management

Your programming goal is to get work done. The task management programs supervise execution of all work done in the system. They receive a job step from the job scheduler, which initiates a task; the task then may compete for the resources of the system. The task management programs allocate these resources on a job class and priority basis in a multiprogramming environment, or as requested in a single-task environment. When execution is completed, the job scheduler is notified and again takes control.

What the User Must Do

All work submitted for processing must be formalized as a task -- the smallest independent unit of work that can compete for the resources of the system. It is task management's purpose to provide all the requested resources; i.e., both machine resources, such as CPU control, and programming resources, such as a member of the subroutine library. In a PCP environment, it provides these resources as requested. In an MFT or MVT environment, it provides them on a priority basis.

To make effective use of the task management facilities you should be concerned with the following areas:

1. Establishing priorities
2. Creating tasks
3. Synchronizing events
4. Allocating main storage
5. Protection of storage
6. Passing and sharing main storage
7. Establishing intervals
8. Terminating tasks
9. Time slicing

Establishing Priority

In a multitasking environment, the order in which jobs are selected for execution, and requests for resources are resolved by job class and priority. Initially, you specify a job's class and priority in the JOB statement. Classifying jobs allows you to control the types of jobs running concurrently in the system. The operator can modify the scheduling priority up to the time the job is actually selected for execution.

In MVT and MFT with subtasking, dispatching priorities may even be changed by the program during execution: you can lower/raise the priority of an active task or any of its subtasks (tasks created by a job step during execution). When the job scheduler initiates a job step, the job priority within a class is used to establish a dispatching priority and a limit priority. The dispatching priority is used by the resource managers to resolve requests for main storage and CPU resources; the limit priority is used to control dynamic priority assignments. (Note: In MFT without subtasking the dispatching priority is determined by the storage address of the partition in which the task is being executed: that is, the higher the storage address of a partition, the higher the dispatching priority of all work in that partition. Therefore, without subtasking, limit priority has no meaning in MFT.)

Creating Tasks

Effectively, you are the one who creates a task. This is done when:

1. The operator performs the IPL operation or uses commands that start components (reader/interpreter, etc.) of the job scheduler; or
2. The system processes your EXEC statement; or
3. In MVT and MFT with subtasking, an ATTACH macro instruction is executed in your program. Creating a new task by means of the ATTACH macro instruction (subtasking) is the most dynamic control you can exercise. This facility is available with MVT and optionally with MFT. MFT systems with the ATTACH option are referred to as MFT with subtasking.

The ATTACH macro instruction causes the control program to create a new task and indicates the entry point in the program to be given control when the new task becomes active. The new task is a subtask of the originating task; the originating task is the task which was active when the ATTACH macro instruction was issued. The limit and dispatching priorities of the new task are the same as those of the originating task, unless modified in the ATTACH macro instruction.

Resource queue elements represent requests that a resource be made available to a routine in a task. They are created only for active tasks. Similarly, task control blocks (TCBs) which contain all the control information required to supervise a task and which represent subtasks, are created only for active tasks as a result of the ATTACH macro instruction. The originating task can wait for the completion of a subtask, just as it can wait for the completion of other events.

Synchronizing Events

Event synchronization is delaying task execution until some specified event occurs. For example, don't do anything until the last output record is written.

An active task can enter a waiting state directly by a macro instruction, or indirectly in anticipation of some event. After the event occurs, notification is made. If the event is governed by the control program, as it is when performing I/O, the supervisor makes the notification; otherwise you must do it by use of a macro instruction. (You can delay execution until a number of events have occurred.)

Another form of synchronization allows tasks to share resources in a serially reusable way. (See Section 2 for a detailed description of serially reusable programs.) For example, the resource may be a table that is updated during the execution of many tasks. To avoid errors that would arise from simultaneous updating, each task must complete its use of the table before another task gains access to it. To control access to such a resource, you can create a queue of all tasks requiring access, and then limit their access to a one-at-a-time basis.

Allocating Main Storage

The supervisor controls and allocates storage space dynamically, that is, when requested by a task or the control program itself. In your jobs, you can make explicit requests to allocate main storage and to release main storage. There may also be an implicit request for main storage when a program has to be brought from a library. The supervisor must allocate main storage space for the program. In turn, when the program is no longer in use, there will be an implicit release of main storage.

Your request for main storage can be for either fixed or variable amounts. You can also make the request conditional or unconditional (i.e., the task cannot proceed without the requested space).

You can significantly expand the dynamic area by adding the IBM 2361 Core Storage unit to your current processor storage. This unit is designed to provide large capacity storage and, when it is included in your system, main storage consists of both processor storage and the 2361. That is, all processor and 2361 storage is directly addressable. If necessary, you can distinguish between processor storage and 2361 storage, through the use of Main Storage Hierarchy Support, which is provided with the IBM 2361 Models 1 and 2. Processor storage is identified as hierarchy 0 and the 2361 is identified as hierarchy 1. This identification is made through the REGION parameter in either the job or execute statement; certain macro instructions are also extended so that you can specify hierarchy 0 or 1, as desired. For full information on hierarchy identification through job control statements, see the Job Control Language publication. For the macro instructions affected, see the Supervisor and Data Management Macro Instructions publication.

Protection of Storage

Main storage is protected in blocks of 2048 (2K) bytes. Each block can be given a storage protection key in the range of 0 through 15. Protection keys 1-15 are used for jobs, and key 0 is used for the control program. (Every task in one job operates with the same protection key.)

Storage protection is a vital element in a multitasking environment. Jobs are executed in their own regions, and other jobs cannot violate other regions. As a benefit, program testing may be carried on at the same time as productive work.

Passing and Sharing Main Storage

In an MVT environment, areas of storage can be passed or shared between tasks. Since the area of storage no longer required by a subtask may still be needed by the originating task, your jobs may call for the creation of subpools. A subpool is all the 2K blocks of main storage allocated for a particular task under one label (called the subpool number). When the subtask is attached, a subpool can be made available to it by passing or sharing. When a subpool is passed, termination of the subtask results in release of the subpool. When a subpool is shared, termination of the subtask does not result in release of the subpool.

Establishing Intervals

The interval timer is optional on some models of System/360 and standard on others. The control programs make use of the timer in two ways: to ensure that the maximum time for a job step specified in the EXEC control statement is not exceeded, and to make the job step time available for accounting procedures. You can use this interval timer:

- To query the supervisor at any time during program execution; the response is date and time.
- To request that the supervisor communicate with your program after a stated period of time. Intervals are requested in real time (actual time elapsed) or task time (the time the task is actually using the CPU, not including waiting time). You can request that a task be placed in a wait state until a real time interval is completed. You can also request that the task be allowed to continue, but that at the end of either a real-time or task-time interval, control be given to a specified entry in your executing module.
- To ascertain how much of the time requested has elapsed or to cancel a previously specified interval.

Some of the possible applications for these facilities are:

- Time and date "stamping" of messages, data sets, and printouts. The telecommunications package (QTAM) uses the timer in this way.
- Restarting a task after a predetermined time. Telecommunications line-polling can be done on a periodic basis, rather than continuously, during low-traffic hours.
- Program execution analysis and program debugging. Phases of a long program can be timed individually under a variety of conditions. In program debugging, the timer can be used to limit the amount of time spent in executing each section of the program, thus allowing a single test run to continue in spite of loops or other time consuming action that might occur unexpectedly.

In both MFT and MVT, the interval timer can be used to limit the time that each job may execute by including the optional job step CPU timing feature. If you do select this feature, you can include your own accounting routine or the System Management Facilities option to process this information. (When you select the System Management Facilities option, you must also select the job step CPU timing option.) The supervisor times each job step and passes this value to your System Management Facilities or accounting routine. The accumulated value for the entire job is also passed to the System Management Facilities or accounting routine.

<u>Terminating Tasks</u>

The system is notified when a task has completed execution whether normally or abnormally. Any program operating on behalf of a task can discontinue task execution abnormally. The MFT and MVT control programs then free resources and if included, can process this information further. program removes the TCB. In MFT with subtasking, all TCBs created as a result of an ATTACH are also removed. However, in MFT with <u>and</u> without subtasking, the TCBs set during system generation (one TCB per partition), are not removed, but rather made non-dispatchable. Although abnormal termination of a task causes abnormal termination of all its subtasks, it is possible for a subtask to terminate abnormally without causing termination of the originating task. (You can request a dump by use of an appropriate macro instruction. The dump is placed in the data set described by the DD statement you provide.)

<u>Program error exits</u> are requested by the set program interrupt exit (SPIE) macro instruction. The programmer may specify the types of program interruptions his subroutine will handle, and the types the supervisor is to handle. The programmer most likely will want to handle conditions such as overflow and underflow from arithmetic operations, and let the supervisor take normal action for the others, such as execution of a privileged instruction and violation of storage protection.

<u>An abnormal end of task exit</u> is set up by the specify task abnormal exit (STAE) macro instruction. This exit is taken if the task is abnormally terminated internally or externally. An internal termination is one resulting from the execution of the abnormal end of task (ABEND) macro instruction. This may be issued by the user's program after the determination is made that an uncorrectable error has occurred. An external termination is one initiated by the supervisor program, for example, in the event of a storage protection violation or the expiration of the specified time for the job step.

<u>Time Slicing</u>

Time slicing in an optional feature available to users of MFT and MVT. At system generation time, your installation may specify that all tasks with a certain priority (MVT) or that all tasks within a specific group of partitions (MFT without subtasking), or within a certain range of priorities (MFT with subtasking) are to share the use of the CPU for an equal, predetermined length of time. The installation can modify the group of tasks or partitions to be time sliced, the length of the time slice, and in MVT the priority of the time-sliced tasks. This can be done at system initialization time and in MFT, it can also be done with the DEFINE command.

When a member of the time-slice group has been active for a certain length of time, it is interrupted and control is given to another member of the group which will, in turn, have control of the CPU for the same amount of time. In this way, all member tasks or partitions are given an equal slice of CPU time and no task or partition within this group can monopolize the CPU. In MVT only tasks in the group are time-sliced, and they are time-sliced only when the priority level of the group is the highest priority level that has a ready task. Dispatching of tasks continues within the group until: all tasks are in a waiting state, or a task of higher priority than the one assigned to the group becomes ready.

In MFT, only partitions that are assigned to the time-slice group will be time sliced, and they are time sliced only when the first partition in the group is the highest-priority, ready task. Dispatching of the partitions continues within the group until all the partitions are in a waiting state or until a partition with a higher priority is in a ready state.

A task in the system that is not defined within the time-slice group is dispatched under the current priority structure: that is, the task is dispatched only when it is the highest priority task on the input work queue that is ready.

What the System Does

The system responds to your requests in two general ways:

1. It handles resource allocation.
2. It supervises execution of the task.

Resource Allocation

System resources, such as control of the CPU and main storage allocation, are assigned only to tasks. This assignment takes place on a priority basis or as requested. Even with a single-task control program, some resource allocation is necessary and, of course, with a multitask control program, it is required. Therefore, the task management programs must supervise allocation, keep track of all assignments, and ensure that resources are freed upon task completion.

To supervise its allocation, each resource has a manager. For example, the manager of the CPU, which is the most important resource allocated to a task, is called the task dispatcher. Figure 23 illustrates the logical relationship between tasks in the system and the resource managers that manage queues.

In a multitasking environment, more than one task may be contending for the same resource at the same time. Hence, queueing of requests is an essential part of resource allocation. When main storage and/or the CPU becomes available, it is given to the ranking member of the queue. Rank is determined by priority or, in the case of equal priority, by the order of entry onto the queue.

To keep track of assignments, the control program maintains queues that represent unsatisfied requests for resources and tables that identify available resources.

Supervising a Task

To supervise each task, the control program groups all control information that pertains to that task in a TCB. There is, therefore, one TCB for each task in the system and it contains such information as register contents and the location of storage areas allocated to the task. In MVT, the TCB is built when the task is created, that is, when it is attached by the initiator portion of the job management programs. The task given control by the initiator/terminator is called the job step task. In MFT without subtasking, TCBs are created at system generation. These TCBs may be made active or inactive as the result of redefining partitions during operation, but no TCBs are created. In MFT with subtasking, the system TCBs and a job step TCB for each partition are created during system generation. These TCBs may be made active or inactive as the result of redefining partitions. New TCBs may be created whenever the user executes an ATTACH macro instruction to give control to a subtask.

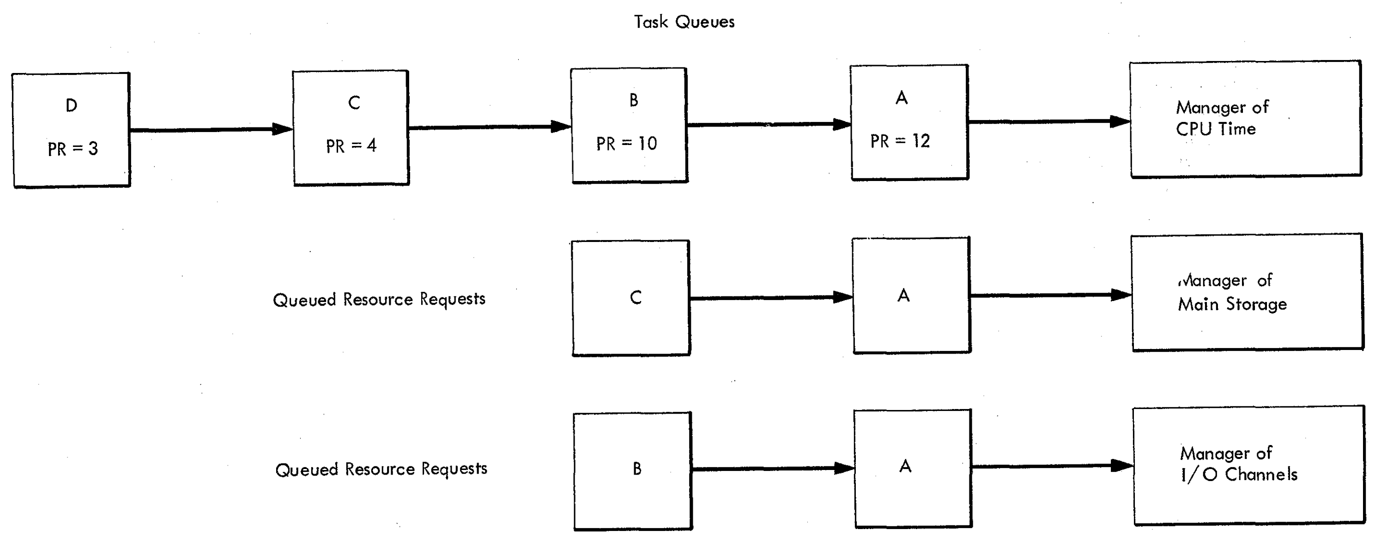


Figure 23. Resource Queues

Once control is passed to the task, the task management programs must keep track of the task's current state. The task's current state depends on the task's readiness to use the CPU. If the task can make immediate use of the CPU, it is ready. If the task is using the CPU, it is active. Otherwise, the task is waiting. That is, waiting is the opposite of ready. The first two states are self-explanatory, but the concept of a waiting state needs more explanation.

A task can enter the waiting state directly or indirectly. It enters the waiting state directly via the WAIT macro instruction, and indirectly as a result of other macro instructions. For example, an indirect wait may occur as a result of a GET macro instruction that requests the next input record. If the record is already in a main storage buffer, no waiting occurs; otherwise, the GET routine issues a WAIT and the task is delayed until the record is available.

The completed use of a resource is always indicated by an interruption, whereupon the appropriate resource manager takes control. Therefore, if an active task -- task 1 -- is interrupted, it will only get control back (after the supervisor processes the interruption) if all other ready tasks are of equal or lower priority. If, however, a higher priority task -- task 2 -- is now ready, it is given control, regardless of the point at which task 1 was interrupted. This does not mean that task 1 is put in the waiting state; it is still ready. Allocating resources by priority extends beyond giving control to a ready task when all others are waiting: it resolves the conflict of who should get control when there are two or more ready tasks.

Note: Through the use of the TIME parameter in the execute statement, you can find out how much CPU time a job step used. This value is exclusive of wait and I/O time. A job step that uses job step CPU timing is canceled if every task in the step is in a wait state for more than thirty consecutive minutes. (Through the System Management Facilities -- SMF -- option, CPU and wait time may be extended.) To invoke job step CPU timing, you must have specified the interval timer. At system generation time, you can include the SMF option, or your own accounting routine to process the information. (If you selected SMF, you must also have selected the job step CPU timing feature and your accounting routine must be added to the appropriate routine.) The supervisor will calculate the amount of time each job step has control of the CPU. This value is then passed, along with the accumulated value for the entire job, via an exit, to the SMF or user-supplied accounting routine, which can process this information further.

The Environments of a Task

Although the requirement that all work be performed under task control has no exception, the manner of controlling tasks is subject to considerable variation. The most significant choice of options available to an installation is that choice between single-task and multitask control programs -- either with a fixed or variable number of tasks. In a single-task environment, no more than one task can exist at any one time. On the other hand, several tasks may coexist in the multitask environment and compete for resources on a priority basis. Since both environments are concerned with task control, a program that is written for the single-task environment and follows normal system conventions will work equally well in the multitask environment.

A SINGLE-TASK ENVIRONMENT (PCP)

In a single-task environment, the job scheduler programs are tasks. Each job step is executed as part of this task, and, since it is the only task in the system, it can have all available resources. The job step's program can, of course, have any of the program structures previously mentioned, except a dynamic structure with more than one task per job step.

A brief look at execution in a single-task environment will help to clarify these concepts.

The control program first locates the load module that you specified in your EXEC statement. It allocates main storage space according to program attributes stated in the library directory entry for the load module, then loads the program into main storage. Once the load module is available in main storage, control is passed to the entry point. If the load module fetched is the first subprogram of a dynamic serial structure, the subsequent load modules required are fetched in the same way as the first, with one exception: if the needed module is reusable or reenterable (that is, self-initializing so that any portion modified in the course of execution is restored before it is reused), and a copy is already in main storage, that copy may be used for the new requirement. When the job step is complete, the supervisor informs the job scheduler whether the completion was normal or abnormal.

A MULTITASK ENVIRONMENT

Although the resource allocation function is not absent in a single-task environment. The task management programs must assign resources to tasks, keep track of all assignments, and ensure that resources are properly freed upon task completion. If several tasks are waiting for the same resource, queueing of requests is required. By clearly distinguishing among tasks, the supervisor can allow tasks to share resources where advantageous to do so.

Before discussing the more apparent benefits of a multitasking environment, let's take a look at one that might easily escape us. Multitasking reduces your system overhead time, even if you are running only single-task applications. You will recall that we have used the terms "single-task control program" and "multitask control program." The significance here is that in a multitasking environment, the control program itself is multitasking. Thus, you are deriving time-saving benefits from it, even though you are running only single-task applications.

The visible benefits include fast throughput in job-shop operations. This is achieved by allowing concurrent operation of input readers, output writers, and your jobs. It is possible to handle a wide variety of telecommunications activities, each of which is characterized by many tasks often in the waiting state. Also, you can use the dynamic structure for your complex problems. Hence, the various segments of your program can share system resources concurrently and thus optimize the use of these resources. Under operator control, multitask operations permit job steps from several different jobs to be established as concurrent tasks.

Up to 15 jobs can be competing for resources concurrently under MVT and MFT. Each job, consisting of one or more processing programs (steps), is selected for execution from an input work queue by its priority within a class. Just as in PCP, the steps of each job are executed sequentially. Each region (in MVT) or partition (in MFT) is a single, contiguous area of main storage and is assigned a unique storage protection key. The supervisor uses protection key zero. Thus, 15 regions or partitions of main storage can be assigned to processing programs at any one time. (Note: In MFT, storage protection is optional; therefore, a protection key is assigned only when the feature is included in the system.)

A step being executed can communicate with succeeding steps in the same job through condition codes and passed data sets. It cannot pass data to a succeeding step by leaving it in its region.

A job, by definition, is independent of all other jobs. This is an essential characteristic in a multiprogramming environment, where jobs are executed on a priority basis with no attempt to synchronize the execution of one job with another. A job step cannot communicate directly with a competing step in a different region or partition. It cannot cause a new region or partition to be established. Commands given by the operator, or, in MVT, supplied in the job streams, control the number of regions or partitions to be used.

As mentioned, MFT and MVT users can select the time-slicing facility at system generation time and modify it at system initialization time (MFT users can also modify the time-slicing specification via the DEFINE command). Time-slicing of equal priority tasks (in MVT) or of a group of partitions (in MFT) prevents one task or partition from monopolizing the CPU. This facility is especially useful when one or more tasks or partitions are involved conversation-type applications in which the user is interacting with the application program through a terminal.

Multprogramming With a Fixed Number of Tasks (MFT)

In an MFT environment, storage is divided into two major areas: the system areas and the dynamic area. The dynamic area is our concern.

The dynamic area is divided into a number of discrete areas called partitions (the maximum number of partitions is 52). The user defines the number of partitions in the dynamic area at system generation time or at initial program loading (IPL) time. He can redefine them at any time. As many as 52 partitions can exist. Each partition can contain one active task at a time; therefore, as many as 52 tasks can be executing concurrently. Of the 52 partitions, up to fifteen may be used for executing jobs. Each of these 15 partitions may also contain, in addition to a job, a transient reader and as many direct system output writers as desired. However, there can only be a total of three readers started in the system at one time. In addition, up to 36 system output writers may operate in the system. If the system has the subtasking option, the total number of tasks and subtasks executing concurrently can be as large as the limiting value of 255. (Note: the number 255 must include the number of system tasks, the number of attached system subtasks, and the number of user tasks and subtasks.)

MFT allows the user to direct jobs to a given partition or group of partitions through the CLASS parameter of the JOB statement. Up to three alphabetic job classes (A-O) are assigned to each partition at system generation time; these job classes may be modified at system initialization or by using the DEFINE command. Using the CLASS parameter on the JOB statement enables the user to define the type of job that each partition may process. As an additional feature , the MFT control program offers partition definition. With this feature, the operator can change the number, size, and job class of any or all partitions in the system at any time after initial program loading.

In MFT without subtasking, the highest storage address is the highest priority partition. The preceding rule does not hold true in MFT systems with subtasking, because the user can change the priority of problem programs by using system macro instructions.

Jobs are scheduled into partitions through the use of the CLASS parameter in the JOB statement, in conjunction with the PRTY (priority) parameter. With the optional storage protection feature, each of the 15 possible jobs is protected from damage by the other jobs, and the system areas are protected from all problem programs.

All partitions are independent with respect to job scheduling and initiation. Jobs are scheduled into the first available problem program partition that services the job class specified in the JOB statement. Execution then proceeds concurrently for all tasks in the system. That is, execution is not simultaneous or overlapped or alternated in a fixed pattern. A task gains control when another task must wait for the completion of some event, such as an input/output operation. Only then is a lower priority task allowed to proceed. When a higher priority task is ready to resume, the lower priority task's processing is suspended and control of the CPU is returned to the higher priority task. (The priority of each task is determined by the partition in which it resides.) The high-priority task then proceeds until another event causes it to relinquish control. The relinquishing of control by one task and another task's receipt of control is called a task switch.

If you select the subtasking option during system generation, then a job step may attach one or more subtasks using the ATTACH macro instruction. The subtasks may be assigned priorities using the parameters of the ATTACH macro intruction. In addition, you may alter the priority of your subtask by using the change-priority (CHAP) macro instruction. All tasks must terminate before the job step can terminate.

In MFT, when you select the time-slicing facility, you should be aware of the following. The time-slice group is composed of a group of contiguous partitions. All tasks scheduled into these partitions will be time-sliced. Also, since each partition in the system is assigned to at least one job class, and since a job is scheduled into a partition according to the job class specified in the CLASS parameter on the JOB card , careful consideration should be given to the job class assigned. If you want the job to be time sliced, direct it to a partition (via the CLASS parameter) that has been assigned to a time-slice group.

In MFT with subtasking, any task that has a dispatching priority falling within the range of priorities assigned to a time-slice group will be considered a member of that group. A task can have a dispatching priority equal to that of the time slice group as a result of the priority parameter on the JOB statement or by specifying the appropriate parameters in the ATTACH or the change-priority (CHAP) macro instruction. Therefore, when using an ATTACH or CHAP macro instruction if the subtask is assigned a priority number that designates a time-slice group, the new task is included in the group regardless of whether the original task is.

Use of this control program requires that of consideration be given to the way the user defines different types of jobs and directs those jobs to partitions consistent with the job's characteristics. Taking some simple cases: CPU-bound jobs can be directed to low priority partitions so that they do not interfere with efficient processing of jobs that do not require the CPU as often. Telecommunications jobs can be directed to the highest-priority partitions so that system response time to the terminal user is minimal.

For further information on MFT, see the publication: IBM System/360 Operating System: MFT Guide, GC27-6939.

Multiprogramming With a Variable Number of Tasks (MVT)

Although MVT offers the same advantages (discussed under "A Multitask Environment") as MFT, there are significant differences between the two programs, especially in their use of storage and the additional facilities available through MVT.

Within MFT and MVT, a certain amount of main storage is reserved for control program modules. The remainder of storage, the dynamic area, is available to user programs. In MFT, the dynamic area of storage is divided into a number of discrete areas called partitions. In MVT, the dynamic area is divided into regions.

The MVT configuration reads one or more input streams and schedules the jobs (classed on an input work queue) according to priority. Each job initiated operates in a region and up to 15 independent jobs can be performed concurrently. The job steps within a single job are performed in sequential order, since one step may depend on the successful completion of another. However, unlike PCP, where each step is limited to a single task, a job step in MVT (which is itself considered to be a task) may attach one or more subtasks using the ATTACH macro instruction. The job step task and its subtasks execute independently within the same region and use the same storage protection key. All tasks must terminate before the step can terminate. Note that in MVT, any task that has a dispatching priority number equal to the time-slicing dispatching priority number will be a member of the time-slice group. A task can have a dispatching priority equal to that of the time-slice group as a result of the priority parameter on the JOB statement or by specifying the appropriate parameters in the ATTACH or change-priority (CHAP) macro instructions. Thus, when using the ATTACH macro instruction, if the sub task has a priority number that designates a time-slice group, the new task is included in the time-slice group regardless of whether originating task is.

Region size is determined by the REGION parameter on the JOB or EXEC card. If supplied on the JOB card, each step of the job will be given a region of that size regardless of any region size specified in the EXEC cards. If the region parameter is omitted from the JOB card, each EXEC card can specify the region size desired for that step. An installation-specified default region size will be used if neither the JOB nor the EXEC card has a REGION parameter. Once a region is established for a job step, no additional storage can be allocated to that region, unless the rollout/rollin feature is available.

Rollout/Rollin (RO/RI)

An additional feature available to the MVT user is rollout/rollin. Rollout/rollin allows the temporary, dynamic expansion of a particular job beyond its originally specified region. When a job needs more space, rollout/rollin attempts to obtain unassigned storage for the job's use. If there is no such unassigned storage, another job is rolled out -- i.e., is transferred to auxiliary storage -- so that its region may be used by the first job. When released by the first job, this additional storage is again available, either (1) as unassigned storage, if that was its source, or (2) to receive the job to be transferred back into main storage (rolled in).

System Management Facilities

Also available to the MVT and MFT user are data collection routines and exit linkages provided by the System Management Facilities (SMF) option. Through SMF data collection routines, this option can be used as a system resource distribution and evaluation tool. By providing your own exit routines at the appropriate locations, this option can be used in a monitoring capacity. Since the data collection and exit facilities are independent of one another once SMF is included in the system at system generation time, they may be used in combination or separately.

SMF data collection routines gather job and direct access and volume information for management information programs. They provide the manager and system programmer with the information necessary to make a variety of analyses. SMF routines can be used to determine each job step's use of the CPU, I/O devices, and storage. They can be used to determine data set activity for each problem program and also to acquire volume usage information for direct access devices. Output created by the SMF routines can also be used to create and maintain inventories on direct access and tape devices.

The manager and system programmers can use this information in various ways. In general, they can measure system usage against their own standards of efficiency and performance and in comparison to those at other installations. They then have a data base (recorded data in a permanent format) to use to improve their installation's standards.

SMF is not, however, confined to after-the-fact analysis. This option also allows the user to write exit routines; these routines can monitor a job or job step at various points during its processing cycle, that is, from control statement analysis to termination of the job. (All linkages for these exits are supplied when the option is included in the system at system generation time.) Therefore, by adding installation routines at the appropriate exits, the installation manager can enforce standards of identification, priority, resource allocation, and maximum execution time. The maximum number of logical records written to a system output device can be limited through use of the OUTLIM parameter on the SYSOUT DD statement.

Here's an example of using both facilities provided by SMF. By using and analyzing the information obtained by the data collection routines the installation manager determines the average time each job step uses the CPU. In general, he finds that job steps exceeding this time limit are in a loop or unending wait state. Time is being wasted and overall efficiency impaired. Therefore the average is used to establish a time limit for each job or job step running on the system; a job exceeding its expected time limit will be terminated. However, there must be some way to allow a job to exceed its expected time limit. Therefore, a routine is coded for the time limit exit; this allows him to extend the run time for selected jobs, such as the inventory program at year's end.

Step Restart

Step restart is similar to checkpoint restart in purpose and effect. However, with step restart, no CHKPT macro instructions need be issued to initiate the restart. The programmer may specify that he wants step restart in effect for all steps in his job (by using a special parameter on the JOB statement) or that he wants it in effect for a particular step (by using a special parameter on the EXEC statement). Restart of an abnormally terminated job step may be automatic (if the job or job step has an eligible ABEND code and the operator consents) or deferred, where a deferred restart involves resubmitting the job.

Language Comparison: Task Managemen Facilities

All task management facilities are available to users of the assembler language, except that MVT is not available to the users of Assembler E. Table 7 indicates which of the facilities are available to users of higher-level languages.

•Table 7. Language Comparison: Task Management Facilities

Task Management Facilities	ALGOL	COBOL			FORTRAN			PL/I	RPG
		E	F	ANS[1]	E	G	H		
Allocating Main Storage	No	No	No	No	No	No	No	Yes	No
Creating Tasks	No	No	No	No	No	No	No	Yes	No
Establishing Priorities	No	No	No	No	No	No	No	Yes	No
MFT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
MVT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Passing and Sharing Main Storage	No	No	No	No	No	No	No	No	No
Protection of Storage	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

[1]American National Standard COBOL.

Section 6: Recovery Management

The most efficient use of your programming and machine resources is impaired when you cannot minimize the duration and effect of unscheduled interruptions caused by a machine malfunction. To effect higher availability of the computing system you must have records that detail the environment of the system at the time the machine malfunction occurred. Recovery Management facilities, which are designed to support Models 65, 65 Multiprocessing, and 85 of the IBM System/360, help you increase computing system efficiency by recording environmental data at the time of the machine malfunction and, in some cases, by providing an analysis of this data.

Recovery from a machine malfunction is possible at one of four levels. First, an interrupted instruction or I/O operation may be successfully retried (instruction retry is part of the recovery management process for Model 65, but is handled directly by the machine recovery facilities for Model 85). If this is not possible, the system operation may be continued by repairing program damage to prevent further interruptions or by associating the damage with a particular task to allow selective termination of the affected job. The system operation may, if necessary, be restarted by an IPL procedure that uses system job and data queues preserved by system restart facilities. Or, recovery management error records can facilitate rapid diagnosis and repair of a non-recoverable machine malfunction.

The following Recovery Management facilities record (on the SYS1.LOGREC data set) the environment of the system at the time of a machine malfunction:

- System Environment Recording (SER) is a set of control program routines that record machine malfunctions in the CPU and channels of Models 40, 50, 65, 75, 91/95, and 195. SER includes two model-dependent programs called SER0 and SER1. Your installation may have either SER0 or SER1, depending on the model and storage size. See the *System Generation* publication for the level of SER provided as a default during system generation, and the alternate choices available.

- Machine-Check Handler for Model 65 (MCH/65) analyzes the error and attempts a recovery by retrying the failing instruction, if possible. If retry is not possible, or if it is unsuccessful, MCH will attempt to repair the malfunction, or isolate it to a task, or both. This facility is optional for MFT and MVT in a Model 65, but standard with Model 65 multiprocessing.

- Machine-Check Handler for Model 85 (MCH/85) is standard programming support for MFT and MVT; it records the machine failure environment and, in the case of an unsuccessful machine retry, analyzes the error and attempts repair and/or isolation and termination of the affected task.

- Channel-Check Handler (CCH) is an optional feature (exception: it is standard for MVT with Model 65 multiprocessing and Model 85) which analyzes the error and produces an interface that aids in setting up for a retry of the failing operation by the Input/Output Supervisor. CCH is available with Model 65 and higher.

SER0, SER1, MCH/65 and MCH/85 are mutually exclusive facilities that receive control when a machine check is detected. CCH receives control when certain channel check conditions occur. However, if CCH is not present in the system, one of the other Recovery Management facilities chosen at system generation time will receive control and write an error record for the channel failure. CCH routines can be loaded dynamically at IPL or NIP time.

Note: Not all the Recovery Management facilities can recover from an error at each of the four levels.

In addition, two facilities exist that provide the capability to bypass various I/O errors. Both facilities are optional support for MFT and MVT. Both are automatically included for Model 65 Multiprocessing.

- Alternate Path Retry (APR) allows an I/O operation that has developed an error on one channel to be retried on another channel (if another channel is assigned to the device performing the I/O operation). APR also provides the capability to VARY a path to a device online or offline. While not model - dependent, APR only performs its function usefully in systems that have alternate paths and include the Channel Check Handler.

- Dynamic Device Reconfiguration (DDR) allows a demountable volume to be moved from one device to another, and repositioned if necessary, without abnormally terminating the affected job or reperforming IPL. A request to move a volume may be initiated by either the operator or the system, for system residence or non-system residence devices.

What the User Must Do

Recovery Management processing is independent of user parameters and other system facilities. Recovery Management facilities are initiated as the result of an unscheduled interruption caused by a machine malfunction.

To ensure more efficient use of the Recovery Management facilities, the programmer must code refreshable modules. A refreshable module cannot be modified by itself or by any other module during execution; i.e., a refreshable module can be replaced by a new copy during execution by a recovery management routine without changing either the sequence or results of processing.

You must specify the refreshable attribute (REFR) in the parameter field of the linkage editor EXEC statement. If an entire load module is to be marked refreshable, each control section within the load module must be refreshable.

What the System Does

When a machine malfunction causes an unscheduled interruption, control is passed to one of the Recovery Management facilities. (Table 8 illustrates the capabilities of each of the Recovery Management facilities.) These programs record the system environment when a machine malfunction occurs in the CPU, main storage, or channels. An analysis of this environmental data determines the level at which recovery is feasible.

Before Recovery Management releases control, it formats and records on the primary system residence volume the data associated with the malfunction. The dynamic output from the environment recording and Recovery Management facilities is recorded on SYS1.LOGREC, a data set on the system residence volume used exclusively for error records. The data contained on SYS1.LOGREC is edited and printed by the Environment Record Editing and Printing (EREP) system utility program. (For more specific information on the EREP system utility program see the publication, IBM System/360 Operating System: Utilities, GC28-6586.)

SER0

System Environment Recording, Option 0, (SER0) is the least complex option of the Recovery Management facilities. SER0 determines the type of malfunction and, if possible, writes the error record on SYS1.LOGREC. The system is placed in the wait state and a message is issued to the operator requesting him to reload the operating system. (The Initial Program Load (IPL) procedure must be followed to reinitialize the system.)

If the error record cannot be written, the system is placed in the wait state and a message is issued to the operator requesting the use of the stand-alone diagnostic program System Environment Recording, Editing, and Printing (SEREP).

SER1

System Environment Recording, Option 1 (SER1), performs the same data collection functions as SER0. SER1 also performs a selective termination analysis that attempts to associate the error with a specific task. If a task/error relationship is established and if additional failures do not occur after SER 1 has filled in the record entry:

1. The affected task is terminated abnormally.
2. The record entry is formatted and written onto SYS1.LOGREC.
3. System operation continues.

If a task/error relationship is not established or if additional failures occur after the record is complete or if I/O errors prevent the reading of the header record on SYS1.LOGREC, the system is placed in a wait state and the operator is issued a message requesting him to reload the operating system.

MCH/65

The Machine-Check Handler for Model 65 (MCH/65) is a more complex option of the Recovery Management facilities that support multiprogramming versions of the operating system (MFT and MVT). It is standard to MVT with Model 65 multiprocessing.

•Table 8. Synopsis of Recovery Management Capabilities

Facility / Capability	SER0	SER1	MCH	CCH
Determine error type (CPU/CHAN)	Yes	Yes	Yes	No
Record environment	Yes	Yes	Yes	Yes
Analyze malfunction	No	Yes[1]	Yes[2]	Yes
Repair program damage	No	No	Yes	No
Retry interrupted instruction	No	No	Yes[3]	No
Initiate selective termination of the affected task	No	Yes[1]	Yes[2]	No
Generate error record for SYS1.LOGREC	Yes	Yes	Yes	Yes
Place system in wait state	Yes	Yes	Yes	Yes
Provide information for error recovery procedures	No	No	No	Yes

[1]Limited.
[2]Extensive testing.
[3]Initially handled by machine recovery facilities in Model 85.

When there is a failure in the CPU or in main storage, MCH/65 receives control and records the system environment. The MCH/65 modules analyze this data and determine the level at which recovery is feasible.

If recovery is not possible at the functional level (by repairing the failure and/or retrying the interrupted instruction), MCH/65, like the SER1 option, attempts, at the system level, to associate the damage with a particular task thereby allowing the selective termination of the affected job. If recovery is successful at either the functional or system level, system operation continues.

However, recovery must be attempted at either the system-supported restart or system repair level when any one of the following occur:

1. The failure affects a critical task.
2. Pertinent data is indeterminate.
3. The failure cannot be corrected.

Recovery at the system-supported restart level consists of a re-IPL procedure, using system job, and data queues that have been preserved by system restart facilities.

Recovery at the system repair level involves placing the system in the wait state. (Exception: For Model 65 Multiprocessing, a permanent storage failure in a non-critical operating system component (e.g., problem program storage) can be logically removed by Storage Reconfiguration and normal system operation can continue.) Use of pertinent diagnostic data formatted by MCH/65 and written onto SYS1.LOGREC facilitates repair of the machine malfunction.

MCH/85

The Machine-Check Handler for the Model 85 (MCH/85) differs only in that machine recovery facilities handle instruction retry. If the machine-check interruption is successfully retried by the machine recovery facilities, MCH/85 is entered solely to format an error record (and, in some cases, to perform error analysis to aid the customer engineer) and to place the record on SYS1.LOGREC.

If the interruption is not successfully retried, by machine recovery facilities, MCH/85 receives control and attempts the following functions, essentially the same as the equivalent MCH/65 functions: (1) identification and analysis; (2) repair or task termination; (3) the formatting of an error record.

High availability of the Model 85 is further enhanced through the combined efforts of MCH/85 and an operator command, MODE, by which he may control the method of error recording for machine malfunctions.

CCH

When a channel failure is detected, the Channel-Check Handler (CCH) receives control from the Input/Output Supervisor. The two major functions performed by CCH are the completion of the error routine interface bytes and the completion of the record entry. The analysis performed by CCH aids the appropriate IBM-supplied error routine (Error Recovery Procedure -- ERP) in setting up for a retry of the failing operation by the Input/Output Supervisor.

The error record CCH formats contain the environment of the channel failure and is written onto SYS1.LOGREC by a routine within IOS.

APR

When Alternate Path Retry is not in the system, the retry of channel errors occurs randomly on the available paths. APR ensures that a different channel will be tried (if one exists, is online, and ready) on retry of channel-detected errors. This is done by marking failing paths offline and indicating the offline paths to the I/O Supervisor. Since IOS will check the status of the path before attempting to initiate I/O to that path, time will not be spent trying paths that have reported not-operational or have been varied offline.

APR also allows the operator to vary paths online or offline. This facility can be used to vary a path online that previously reported not-operational, or to vary a path offline that has been receiving channel-detected errors.

APR supports alternate subselector channel paths on the 2870 channel and alternate selector channel paths. It does not support teleprocessing.

DDR

Dynamic Device Reconfiguration can be requested by the operator any time during execution, or by the system after a permanent error for all supported demountable volumes. DDR monitors the validity of the requests and responses of the operator, and requires the operator to complete system-requested swaps before initiating his own.

Operator-requested DDR is initiated by the use of the SWAP command. System-requested DDR receives control, following a permanent error, through OBR/SDR for non-system residence devices and through Transient Error Fetch, the Error Fetch Sequence, Finch, or the resident DASD ERP for system residence devices.

DDR supports the 2400, 2311, 2314, and 2321. Shared direct access data sets may not be swapped, except to themselves. DDR may be applied across channels. It can be requested only by the operator for readers, printers, and punches during "intervention-required" conditions. Two volumes may be completely swapped following a permanent error, or a volume can be demounted for cleaning and then remounted on the same device.

Error Recovery Procedures and the Online Test Executive Program

To put operating system recovery support in better perspective, the following paragraphs discuss Error Recovery Procedures and the Online Test Executive Program.

Error Recovery Procedures

Error Recovery Procedures (ERPs) are designed to maintain device performance and to ensure that all the routines that implement these procedures provide a uniform type and quality of information. IBM supplies the routines, which are device dependent: they attempt error recovery for particular device types according to the error recovery procedures. The routines:

- Define intermittent and unrecoverable errors.
- Detect and attempt recovery from errors encountered during read or write operations.
- Detect and recover from errors encountered during control operations.

Use of error recovery procedures may differ from installation to installation. For one, at system generation time, the installation will select only those routines needed for the devices in its system. For another, the standard number of retries defined for each recovery routine may not meet an installation's particular requirements. Hence, the standard number can be modified to suit installation needs.

Once selected, recovery routines reside in the SVC library, except for the portion of the direct access error routine needed for the system residence device and for frequent, unusual conditions (such as, end-of-cylinder, head switching, alternate track procedures, etc.), which resides in main storage. Thereafter, whenever an I/O interruption occurs and an error is detected that requires error recovery procedures, the system determines if an IBM-supplied routine is to be used. If so, the appropriate routine is brought into main storage. After it receives control, the error routine determines the type of error and, where possible, attempts to retry the channel program and to recover from the error. At the completion of error processing -- successful or unsuccessful -- the routine causes termination of the I/O request, notifies the user of completion and returns control to the task supervisor.

Online Test Executive Program

An online test system provides a set of programs (OLT's) that can be used to test I/O devices, control units, and channels within the operating system environment. These programs, together with the Online Test Executive Program (OLTEP), form the Online Test System (OLTS).

OLTEP schedules and controls the activities of OLTS. It also provides communication with the operator. CLTEP resides on SYS1.LINKLIB, is called by standard job control statements, and is under control of the operating system at all times.

Essentially, OLTS allows a user to test I/O devices concurrently with the execution of programs. Testing an I/O device ordinarily does not interfere with system input and output, although the unit being tested must be made unavailable for operating system use (that is, prior to testing a device, the user must vary it offline so that it is not accessible to the operating system). After OLTEP is called, it notifies the operator that it is active and provides continuing communication with him during testing.

Tests may be run to diagnose I/O errors, verify repairs, verify engineering changes, or just to periodically check devices. OLTEP has built in safeguards to ensure the protection of data in external storage. During testing, operating system error recovery procedures are invoked only for recovery from a seek check on a direct access device. They are bypassed for all other online test operations.

Language Comparison: Recovery Management Facilities

Recovery Management facilities are available to all programming language users, if his operation system includes Recovery Management support.

Section 7: Multiprocessing

The term "multiprogramming" implies the concurrent execution of two or more tasks by a single CPU. The term "multiprocessing" generally implies two or more interconnected processors or CPUs. More specifically, in the operating system context, multiprocessing implies the simultaneous execution of two or more tasks by two CPUs operating under the same control program, sharing the same main storage, and with each CPU able to communicate with the other without manual intervention. Furthermore, in a multiprocessing system, devices are accessible from either CPU through the use of two-channel switches. Those devices in the system that do not have the two-channel switch capability (logically or physically connected to only one CPU) depend on CPU to CPU communication if the non-connected CPU is to have access to those devices. (However, a device without the two-channel switch cannot be accessed from the non-connected CPU.)

Advantages

What are the advantages of a multiprocessing system? The resources of a multiprocessing system have a higher degree of availability; they allow flexibility in the use of resources; and, since tasks can be executed simultaneously, there is a potential increase in throughput. How, then, are these advantages achieved?

In a system with a single CPU (a uniprocessing system), the failure of the CPU means that none of the system's facilities will be available to you until the failure is corrected. Failure of a system resource other than the CPU (channel, control unit, or I/O device)...and sometimes main storage...means, at best, a degraded mode of operation. To achieve availability, the installation must be configured with a redundancy of most system resources, including CPUs and main storage. In other words, this user should have two of everything.

Installing two completely independent uniprocessing systems, each consisting of a single CPU with associated main and secondary storage does solve the problem of availability. When one component in a system fails, the user goes to the other system. However, this solution achieves availability at the price of flexibility. By dividing the system into two totally different subsystems, all resources of the failing system become unavailable, not just the faulty resource. Furthermore, in an installation having two independent, uniprocessing systems, it is possible that certain jobs cannot be run on one of the systems, because that system lacks sufficient storage space or a required secondary storage device. At the same time, the other system might have a surplus of main or secondary storage. Multiprocessing allows pooling of resources to improve flexibility without sacrificing availability. Note that this includes pooling of CPUs, which was not possible prior to multiprocessing.

The Model 65 Multiprocessing System

To alleviate these problems and to achieve availability, flexibility, and increased throughput, the Model 65 Multiprocessing system was developed. In this System, resources such as main and secondary storage, and CPUs themselves can be pooled. Thus improved use of total resources, along with interprocessor communication and supporting hardware features are the basis of this system. More specifically, the Model 65 Multiprocessing System has the following characteristics:

1. It has two CPUs that can communicate without manual intervention.

2. The two CPUs share the same main storage.

3. It uses a version of MVT (called MVT with Model 65 multiprocessing).

4. Two tasks can be executed simultaneously; to prevent access to critical supervisor data by both CPUs at the same time, a programming technique called lockout is used.

5. Most devices are available from either CPU through the use of two-channel switches; devices that do not have the two-channel switch capability (logically or physically connected to only one CPU) depend on CPU to CPU communication for the non-connected CPU to have accessibility. (However, a device without the two-channel switch cannot be accessed from the non-connected CPU.)

6. At IPL time, faulty or missing components are marked offline

automatically by the Nucleus Initialization Program (NIP). Thereafter, reconfiguration is under operator control via use of the VARY command. For multiprocessing, the VARY command function has been expanded to include CPU, channel, and storage elements. The VARY command extensions allow reconfiguration to occur without disruption of normal job processing.

The following sections describe the differences, if any, of data, job, task, and recovery management in a multiprocessing environment.

Data Management in a Multiprocessing Environment

Data management in a multiprocessing environment is the same as discussed earlier in the book, with the exception of I/O Supervisor functions. With two CPUs, these functions have been expanded to keep track of up to twice the maximum number of channels.

Job Management in a Multiprocessing Environment

Basic job management functions are not changed essentially for a multiprocessing environment. There are no changes required to your job control statements. However, command processing routines process: an extended VARY command, which allows you to place a CPU, an area of storage, or a channel offline; a QUIESCE command, which allows you to stop system activity prior to removal of an I/O device from the system. (NOTE: QUIESCE and VARY should not be used in a system that includes active teleprocessing, since the results may be unpredictable.)

Task Management in a Multiprocessing Environment

MVT Model 65 multiprocessing uses the productive capability of two CPUs so that two tasks are executed simultaneously.

Selection of tasks for execution is done in the same manner as in MVT; the highest priority ready tasks are always selected first. However, when execution begins in MVT with Model 65 multiprocessing, the two highest priority ready tasks are selected and made active, one for CPU A and the other for CPU B. Processing of the tasks proceeds as in a uniprocessing system. (NOTE: Any load module that can be processed with MVT on a uniprocessing system can be processed on Model 65 Multiprocessing System _without_ changing the code, the job control statements, or the data.)

When, for example, a task running on one CPU must suspend operation until the occurrence of some other event (termination of a subtask, I/O completion, etc.) the current control level of the task is flagged as "waiting", and the CPU dispatches the task of next highest priority (not counting the task running on the other CPU). When the awaited event occurs, it may occur on either CPU (unless the event is CPU-bound, as in the case of I/O completions). The task is generally reactivated by the CPU on which the awaited event occurs.

If the reactivated task is one of the two highest-priority tasks, it will be resumed. Resumption will occur on the reactivating CPU unless the other highest ready task is already running or assigned to run on the same CPU. In the latter case, (that is, if the reactivated task displaces the task previously assigned to the other CPU) resumption of the reactivated task is delegated to the other CPU by means of a dispatcher "shoulder tap", an external signal (WRD) which will cause the other CPU to take over resumption of the task.

The above "shoulder tap" is also used when there is no task assigned to the other CPU. In this case, the other CPU leaves its current task in the ready state and searches the input queue for the next highest priority ready task. If this task is of higher priority than its current task, it will be made active and processed. If it is not, the CPU continues processing the current task.

Recovery Management in a Multiprocessing Environment

All functions performed by the Machine Check Handler (MCH) routine, the Channel Check Handler (CCH) routine, the Alternate Path Retry (APR) routine, and the Dynamic Device Reconfiguration (DDR) routines (as described in Section 6) are performed in the multiprocessing environment. In addition, when a failure does occur, the nonfailing CPU is placed into a timed wait loop, pending successful completion of recovery management operations. MCH and CCH are standard in MVT with Model 65 multiprocessing.

Section 8: Summary

The IBM System/360 Operating System introduces your programs to the computing system, initiates their execution, and provides them with all the resources and services necessary for them to do their work. To be effective, the operating system must be general enough to accommodate a variety of applications on a wide range of hardware configurations. It is, therefore, made up of a general library of programs that can be tailored to your requirements. You can select those portions that you need, add your own procedures to them, and update any procedures as your needs change.

For illustrative purposes, the programs and routines that compose the operating system are classified as a control program and processing programs. The three main functions of the control program are to accept and schedule jobs in a continuous flow (job management); supervise on either a sequential or parallel basis each unit of work to be done (task management); and simplify retrieval of all data, regardless of the way it is organized and stored (data management). The processing programs consist of language translators (such as the FORTRAN compiler), service programs (such as the Linkage Editor), and problem programs (such as your programs). You use the processing programs to define the work that the computing system is to do, and to simplify program preparation.

The elements of the operating system are shown in Table 9.

However, the most important facility of the operating system is its unity. This unity establishes a direct line of communication between you and the operating system and, within the system, between the control program and the processing programs. The net result of this effective communication is a reduction in the time from submitting a problem to receiving a solution.

The ability of all these elements to work together is what allows the operating system to increase the productivity of an entire computer installation and, in turn, allows the installation to get its work done with both efficiency and economy.

•Table 9. Operating System Elements

Control Program Elements		
Job Management	Data Management	Task Management
Languages	Service Programs	Application Programs
ALGOL	Independent Utilities	User Written
Assemble	Data Set Utilities	
COBOL	System Utilities	
FORTRAN	Linkage Editor	
PL/I	Sort/Merge	
RPG	TESTRAN	
	Loader	
	7094/M85 Integrated Emulator	

Figure 24 depicts the final form of the illustration we've been developing since the introduction.

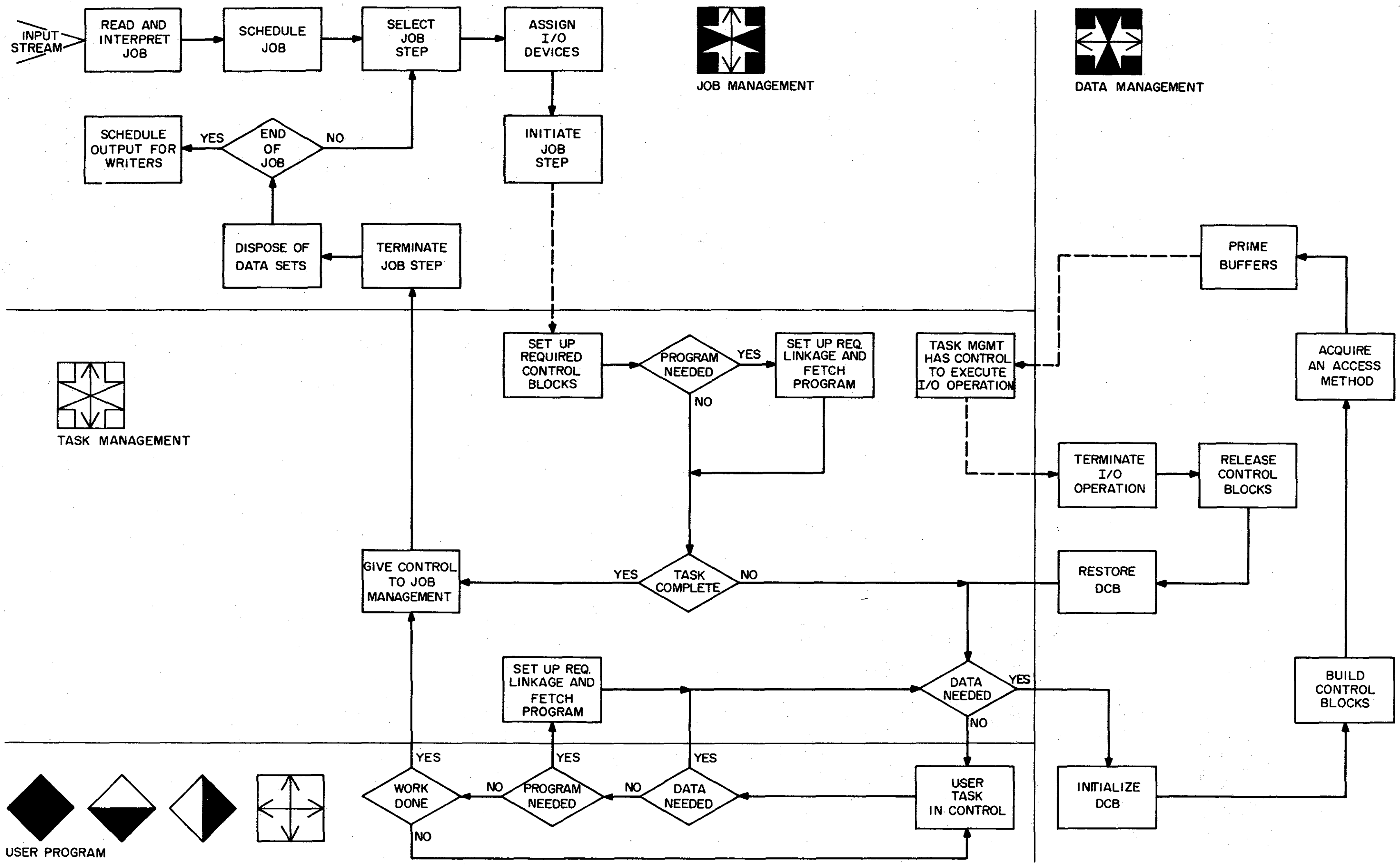


Figure 24. Conceptual Flow of System Responses to User Requests

Glossary

<u>access method</u>: Any of the data management techniques available to the user for transferring data between main storage and an input/output device.

<u>address constant</u>: A value, or an expression representing a value, used in the calculation of storage addresses.

<u>alias</u>: An alternate name that may be used to refer to a member of a partitioned data set; an alternate entry point at which execution of a program can begin.

<u>allocate</u>: To grant a resource to, or

<u>Alternate Path Retry (APR)</u>: Allows an I/O operation that has developed an error on one channel to be retried on another channel (if another channel is assigned to the device performing the I/O operation). APR also provides the capability to VARY a path to a device online or offline.

<u>asynchronous</u>: Without regular time relationship; hence, as applied to program execution, unpredictable with respect to instruction sequence.

<u>attach (task)</u>: To create a task control block and present it to the supervisor.

<u>attribute</u>: A characteristic; e.g., attributes of data include record length, record format, data set name, associated device type and volume identification, use, creation date, etc.

<u>automatic restart</u>: A restart that is initiated either by issuing a CHKPT macro instruction or through the use of special parameters on either the JOB or EXEC statements and that takes place during the current run, that is, without resubmitting the job. An automatic restart is always dependent on an eligible ABEND code and the operator's consent.

<u>auxiliary storage</u>: Data storage other than main storage.

<u>basic access method</u>: Any access method in which each input/output statement causes a corresponding machine input/output operation to occur. (The primary macro instructions used are READ and WRITE.)

<u>batch processing</u>: (See stacked job processing.)

<u>block (records)</u>:
1. To group records for the purpose of conserving storage space or increasing the efficiency of access or processing.
2. A physical record so constituted, or a portion of a telecommunications message defined to be a unit of data transmission.

<u>block loading</u>: The form of fetch that brings the control sections of a load module into contiguous positions of main storage.

<u>buffer (program input/output)</u>: A portion of main storage into which data is read, or from which it is written.

<u>catalog</u>:
1. The collection of all data set indexes maintained by data management.
2. To include the volume identification of a data set in the catalog.

<u>cataloged</u>: The quality attributed to a data set whose name and location are stored in the system catalog.

<u>cataloged data set</u>: A data set that is represented in an index or hierarchy of indexes which provide the means for locating it.

<u>cataloged procedure</u>: A set of job control statements that has been placed in a special data set named SYS1.PROCLIB and that can be retrieved by naming it in an execute (EXEC) statement.

<u>channel</u>: A hardware device that connects the CPU and main storage with the I/O control units.

<u>Channel-Check Handler (CCH)</u>: Is an optional feature (exception: it is standard for MVT with Model 65 multiprocessing and Model 85); it analyzes the error and produces an interface that aids in setting up for a retry of the failing operation by the Input/Output Supervisor.

checkpoint:
1. A point at which information about the status of a job step can be recorded so that the job step can be restarted.
2. To record such information.

Checkpoint/Restart: A facility of the operating system that can minimize time lost in reprocessing a job step that terminated abnormally due to a program or system failure. This restart may begin from a checkpoint or from the beginning of a job step.

checkpoint restart: A restart that is initiated by issuing a CHKPT macro instruction. The restart may be automatic (depending on an eligible ABEND code and the operator's consent) or deferred, where deferred involves resubmitting the job.

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

contents directory: A series of queues that indicate the routines either in a given region of main storage or in the link pack area.

concatenated data set: A collection of logically connected data sets.

control block: A storage area through which a particular type of information required for control of the operating system is communicated among its part.

control dictionary: The external symbol dictionary and relocation dictionary, collectively, of an object or load module.

control program: A collective or general term for all routines in the operating system that contribute to the management of resources, implement the data organization or communications conventions of the operations.

control section: The smallest separately relocatable unit of a program; that portion of text specified by the programmer to be an entity, all elements of which are to be loaded into contiguous main storage locations.

control volume: A volume that contains one or more indexes of the catalog.

CPU (central processing unit): The unit of a system that contains the circuits that control and perform the execution of instructions.

data base: Recorded data in a permanent format.

data control block: A control block through which the information required by access routines to store and retrieve data is communicated to them.

data definition name (ddname): A name appearing in the data control block of a program which corresponds to the name field of a data definition statement.

data definition (DD) statement: A job control statement that describes a data set associated with a particular job step.

Data Generator: A data set utility program that creates multiple data sets within one job for the sequential and partitioned access methods.

data management: A general term that collectively describes those functions of the control program that provide access to data sets, enforce data storage conventions, and regulate the use of input/output devices.

data mode: When you issue a GET macro instruction using the data mode, the data portion of an entire record will be moved from the input buffer to your work area. This record is composed of the data portions of one or more segments within the data set. A PUT macro instruction using this mode moves the contents of your work area into the output buffer and records the entire record as one or more segments within the data set.

data organization: A term that refers to any one of the data management conventions for the arrangement of a data set.

data set: The major unit of data storage and retrieval in the operating system, consisting of a collection of data in one of several prescribed arrangements and described by control information that is accessible by the system.

data set control block (DSCB): A data set label for a data set in direct access storage.

data set label (DSL): A collection of information that describes the attributes of a data set, and that is normally stored with the data set; a general term for data set control blocks and tape data set labels.

deferred entry: An entry into a subroutine that occurs as a result of a deferred exit from the program that passed control to it.

deferred exit: The passing of control to a subroutine at a time determined by an asynchronous event rather than at a predictable time.

deferred restart: A restart that is initiated either by issuing a CHKPT macro instruction or through the use of special parameters on either the JOB or EXEC statements and involves resubmitting the job.

device-independence: The ability to command input/output operations without regard to the characteristics of the input/output devices.

device name: Usually, the general name for a kind of device, specified at the time the system is generated. For example, 2311 or 2400 or TAPE.

direct access: Retrieval or storage of data by a reference to its location on a volume, rather than relative to the previously retrieved or stored data.

direct system output writer: The job scheduler function that controls the writing of a job's output data sets directly to an output device during execution of that job. This function is available only in a multiprogramming environment.

dispatching priority: A number assigned to tasks, and used to determine precedence for use of the central processing unit in a multitask situation.

dump (main storage):
1. To copy the contents of all or part of main storage onto an output device, so that it can be examined.
2. The data resulting from 1.
3. A routine that will accomplish 1.

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

Dynamic Device Reconfiguration (DDR): Allows a demountable volume to be moved from one device to another, and repositioned if necessary, without abnormally terminating the affected job or reperforming IPL. A request to move a volume may be initiated by either the operator or the system, for SYSRES or non-SYSRES devices.

entry point: Any location in a program to which control can be passed by another program.

Error Recovery Procedures (ERPs): These are standard procedures designed to ensure that all the routines that test particular devices provide a uniform type and quality of information.

Error Statistics by Volume (ESV): An option of the Volume Statistics facility. ESV causes the system to collect statistics for each tape volume in the system at any time that volume is open. Some of the statistics you can collect are: volume and CPU serial number, the number of temporary and/or permanent read and write errors, and the number of Start I/O operations encountered. This option only supports labeled volumes, or unlabeled volumes whose serial numbers have been identified to the system.

Error Volume Analysis (EVA): An option of the Volume Statistics facility. The EVA option requires the system operator to specify two minimum values, one for the number of temporary read errors and one for the number of temporary write errors. If the number of read or write errors for a volume currently being accessed exceeds the values specified by the system operator, the system will print a message to this effect at the console. EVA can be used with both labeled and unlabeled volumes.

event: An occurrence of significance to a task; typically, the completion of an asynchronous operation, such as input/output.

event control block (ECB): A control block used to represent the status of an event.

exchange buffering: A technique using data chaining for eliminating the need to move data in main storage. Control buffer segments and user program work areas is passed between data management and the user program according to the requirements for work areas, input buffers, and output buffers, and according to of their availability.

exclusive segments: Segments in the same region of an overlay program, neither of which is in the path of the other. They cannot be in main storage simultaneously.

execute (EXEC) statement: A job control statement that designates a job step by identifying the load module or cataloged procedure to be fetched and executed.

extent: The physical locations on input/output devices occupied by or reserved for a particular data set.

external reference: A reference to a symbol defined in another module.

external symbol: A control section name, entry point name, or external reference; a symbol contained in the external symbol dictionary.

external symbol dictionary (ESD): Control information associated with an object or load module which identifies the external symbols in the module.

fetch (program):
1. To obtain requested load modules and load them into main storage, relocating them as necessary.
2. A control routine that accomplishes 1.

F-format: A data set format in which the logical records are the same length.

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

generation data group: A collection of successive, historically related data sets.

hierarchy: A division of main storage that provides addressing distinction between processor storage, referred to as hierarchy

IEHATLAS: A system utility program used to recover usable data from a defective track, assign an alternate track, and merge replacement data with the recovered data onto an alternate track.

in-stream procedure: An in-stream procedure is a set of job control statements placed in the input stream that can be used any number of times during a job by naming that procedure in an execute (EXEC) statement.

inclusive segments: Overlay segments in the same region that can be in main storage simultaneously.

index (data management):
1. A table in the catalog structure used to locate data sets.
2. A table used to locate the records of an indexed sequential data set.

initial program loading (IPL): As applied to the operating system, the initialization procedure which loads the nucleus and begins normal operations.

initiating task: The job management task of selecting jobs and preparing jobs for execution.

initiator/terminator: The job scheduler function that selects jobs and job steps to be executed, allocates input/output devices for them, places them under task control, and, at completion of the job, supplies control information for writing job output on a system output unit.

input stream: The sequence of control statements and data submitted to the operating system on an input unit especially activated for this purpose by the operator.

input work queue: A queue of summary information of job control statements maintained by the job scheduler, from which it selects the jobs and job steps to be processed.

installation: A general term for a particular computing system, in the context of the overall function it serves.and the individuals who manage it, operate it, apply it to problems, service it, and use the results it produces.

integrated emulator program: A problem program under the control of an operating system that allows programs written for one system to be executed on another system.

IPL: (See initial program loading.)

job: A total processing application comprising one or more related processing programs, such as a weekly payroll, a day's business transactions, or the reduction of a collection of test data.

job class: A parameter on the JOB statement that allows you to define the type of job to be processed, with a maximum of three types per region or partition. In multiprogramming systems, jobs within a job class are initiated according to their priority numbers.

job control statement: Any one of the control statements in the input job stream that identifies a job or defines its requirements.

job library: A concatenation of user-identified partitioned data sets used as the primary source of load modules for a given job.

job management: A general term that collectively describes the functions of the job scheduler and master scheduler.

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

job queue: (See input work queue.)

job scheduler: The control program function that controls input job streams and system output, obtains input/output resources for jobs and job steps, attaches tasks corresponding to job steps, and otherwise regulates the use of the computing system by jobs. (See reader/inter- preter, initiator/terminator, output writer.)

job (JOB) statement: The control statement in the input job stream that identifies the beginning of a series of job control statements for a single job.

job step: That unit of work associated with one processing program and related data. A cataloged procedure can comprise many job steps.

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.

language translator: A general term for any assembler, compiler, or other routine that accepts statements in one language and produces equivalent statements in another language.

library:
1. In general, a collection of objects (e.g., data sets, volumes, card decks) associated with a particular use, and the location of which is identified in a directory of some type. In this context, see job library, link library, system library.
2. Any partitioned data set.

limit priority: A priority specification associated with every task in an MVT operation, representing the highest dispatching priority that the task may assign to itself or to any of its subtasks.

link library: A generally accessible partitioned data set which, unless otherwise specified, is used in fetching load modules referred to in execute (EXEC) statements and in ATTACH, LINK, LOAD, and transfer control (XCTL) macro instructions.

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

linkage: The means by which communication is effected between two routines or modules.

linkage editor: A program that produces a load module by transforming object modules into a format that is acceptable to fetch; combining separately produced object modules and previously processed load modules into a single load module; resolving symbolic cross references among them; replacing, deleting, and adding control sections automatically on request; and providing overlay facilities for modules requesting them.

load: To fetch, i.e., to read a load module into main storage preparatory to executing it.

loader: A service program that combines the basic editing and loading functions of the linkage editor and program fetch in one job step. It loads object and load modules into main storage for execution; however, it does not produce load modules.

load module: The output of the linkage editor; a program in a format suitable for loading into main storage for execution.

locate mode: A transmittal mode in which data is pointed to rather than moved.

lockout: A programming technique used to prevent access to critical data by both CPUs at the same time. (In a multiprocessing environment.)

logical record: A record from the standpoint of its content, function, and use rather than its physical attributes; i.e., one that is defined in terms of the information it contains.

Machine-Check Handler for Model 65 (MCH/65): Is an optional feature for MFT and MVT (exception: it is standard for MVT with Model 65 multiprocessing) which analyzes the error and attempts recovery by retrying the failing instruction, if possible. If retry is not possible, or if it is unsuccessful, MCH/65 will attempt to repair the malfunction, or isolate the task, or both.

Machine-Check Handler for Model 85 (MCH/85): Is a standard feature for MFT and MVT; it constructs a record of errors successfully retried by machine recovery facilities, and, in the case of an unsuccessful machine retry, analyzes the error and attempts repair and/or isolation and termination of the affected task.

macro instruction: A general term used to collectively describe a macro instruction statement, the corresponding macro instruction definition, the resulting assembler language statements, and the machine language instructions and other data produced from the assembler language statements; loosely, any one of these representations of a machine language instruction sequence.

main storage: All addressable storage from which instructions can be executed or from which data can be loaded directly into registers.

master scheduler: The control program component that responds to operator commands, initiates actions requested thereby, and returns requested or required information; thus, the overriding medium for controlling the use of the computing system.

master scheduler task: The command-processing task of searching a queue of pending commands and of attaching a task to execute these commands.

MFT: Multiprogramming with a fixed number of tasks.

module (programming): The input to, or output from, a single execution of an assembler, compiler, or linkage editor; a source, object, or load module; hence, a program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading.

move mode: A transmittal mode in which data is moved between the buffer and the user's work area.

multijob operation: A term that describes concurrent execution of job steps from two or more jobs.

multiprocessing system: A computing system employing two or more interconnected processing units to execute programs simultaneously.

multiprogramming: A general term that expresses use of the computing system to fulfill two or more different requirements concurrently. Thus, it includes MFT and MVT.

multitask operation: Multiprogramming; called multitask operation to express parallel processing not only of many programs, but also of a single reenterable program used by many tasks.

MVT: Multiprogramming with a variable number of tasks.

name: A 1- to 8-character alphameric term that identifies a data set, a control statement, a program, or a cataloged procedure. The first character of the name must be alphabetic.

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.

nucleus initialization program (NIP): The program that initializes the resident control program. Through it, you may request last minute changes to certain options specified during system generation. The operator makes these changes through the console.

object module: The output of a single execution of an assembler or compiler, which constitutes input to the linkage editor. An object module consists of one or more control sections in relocatable, though not executable, form and an associated control dictionary.

Online Test Executive Program (OLTEP): An operating system facility that schedules and controls the activities on the Online Test System (q.v.) and provides communication with the operator. This program is part of a set that can be used to test I/O devices, control units, and channels concurrently during the execution of programs. See also, "Online Test System."

Online Test System (OLTS): OLTS allows a user to test I/O devices concurrently with the execution of programs. Tests may be run to diagnose I/O errors, verify repairs, verify engineering changes, or just to periodically check devices. See also, "Online Test Executive Program."

operator command: A statement to the control program, issued via a console device, which causes the control program to provide requested information, alter normal operations, initiate new operations, or terminate existing operations.

output stream: Diagnostic messages and other output data issued by the operating system or the processing program on output units especially activated for this purpose by the operator.

output work queue: A queue of control information describing system output data sets, which specifies to an output writer the location and disposition of system output.

overlay: To place a load module or a segment of a load module into main storage locations occupied by another load module or segment.

overlay (load) module: A load module that has been divided into overlay segments, and has been provided by the linkage editor with information that enables the overlay supervisor to implement the desired loading of segments when requested.

overlay segment: (See segment.)

overlay supervisor: A control routine that initiates and controls fetching of overlay segments on the basis of information recorded in the overlay module by the linkage editor.

parallel processing: Concurrent execution of one or more programs.

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

partitioned data set: independent groups of sequentially organized data sets, each identified by a member name in the directory.

path: A series of segments which, as represented in an overlay tree, form the shortest distance in a region between a given segment and the root segment.

PDS Compression: A data set utility that compresses a partitioned data set within its original extent and provides a summary of remaining space.

physical record: A record from the standpoint of the manner or form in which it is stored, retrieved, and moved; i.e., one that is defined in terms of physical qualities.

polling: A technique by which each of the terminals sharing a communications line is periodically interrogated to determine if it requires servicing.

post: To note the occurrence of an event.

priority scheduling system: A form of job scheduler which uses input and output work queues to improve system performance.

private library (of a job step): Any partitioned data set which is neither the link library nor any part of the job library.

problem program: Any of the class of routines that perform processing of the type for which a computing system is intended, and including routines that solve problems, monitor and control industrial processes, sort and merge records, perform computations, process transactions against stored records, etc.

processing program: Any program capable of operating in the problem program mode. This includes IBM-distributed language processors, application programs, service and utility programs and user-written programs.

protection key: An indicator associated with a task which appears in the program status word whenever the task is in control, and which must match the storage keys of all storage blocks the task is to use.

qualified name: A control statement term that comprises one or more names, each qualifying the name that follows it. Levels of qualification are separated by periods. For example, the term stepname.procstepname represents a procedure step name qualified by a job step name.

qualifier: Each component name in a qualified name other than the rightmost (which is called the simple name).

queue control block (QCB): A control block that is used to regulate the sequential use of a programmer-defined facility among requesting tasks.

queued access method: Any access method that automatically synchronizes the transfer of data between the program using the access method and input/output devices, thereby eliminating delays for input/output operations. (The primary macro instructions used are GET and PUT.)

reader/interpreter: A job scheduler function that services an input job stream.

ready condition: The condition of a task that is in contention for the central processing unit, all other requirements for its activation having been satisfied.

real time (interval timer): Actual time.

record: A general term for any unit of data that is distinct from all others when considered in a particular context.

reenterable: The attribute of a load module that allows the same copy of the load module to be used concurrently by two or more tasks.

refreshable: A refreshable module cannot be modified by itself or by any other module during execution; i.e., a refreshable module can be replaced by a new copy during execution by a recovery management routine without changing either the sequence or the results of processing.

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

relocation: The modification of address constants required to compensate for a change of origin of a module or control section.

relocation dictionary: That part of an object or load module which identifies all relocatable address constants in the module.

Remote Job Entry (RJE): This facility provides, for a System/360 with attached communication lines, an efficient and convenient method of entering jobs submitted from remote work stations into the job stream. Once a job is entered into the job stream by RJE, execution of the job proceeds under the supervision of the operating system job management routines. All data sets created by the job are handled by operating system data management routines.

resource: Any facility of the computing system or operating system required by a job or task and including main storage, input/output devices, the central processing unit, data sets, and control and processing programs.

resource manager: A general term for any control program function responsible for the allocation of a resource.

restart: To reestablish the status of a job using the information recorded at a checkpoint.

return code: A value that is by system convention placed in a designated register (the "return code register") at the completion of a program. The value of the code, which is established by user-convention, may be used to influence the execution of succeeding programs or, in the case of an abnormal end-of-task, it may simply be printed for programmer analysis.

return code register: A register identified by system convention in which a user-specified condition code is placed at the completion of a program.

reusable: The attribute of a routine that permits the same copy of the routine to be used by two or more tasks. (See reenterable, serially reusable.)

root segment: That segment of an overlay program that remains in main storage at all times during execution of the overlay program; the first segment in an overlay program.

scatter loading: The form of fetch that may place the control sections of a load module into noncontiguous positions of main storage.

scheduler: (See master scheduler and job scheduler.)

secondary storage: Auxiliary storage.

seek: To position the access mechanism of a direct access device at a specified location.

segment:
1. The smallest functional unit (one or more control sections) that can be loaded as one logical entity during execution of an overlay program.
2. In telecommunications, a portion of a message that can be contained in a buffer of specified size.

sequential scheduling system: A form of the job scheduler that reads one input stream and executes only one job step at a time from that input stream.

serially reusable: The attribute of a routine that, when in main storage the same copy of the routine can be used by another task after the current use has been concluded.

service program: Any of the class of standard routines that assist in the use of a computing system and in the successful execution of problem programs, without contributing directly to control of the system or production of results, and including utilities, simulators, test and debugging routines, etc.

Shared DASD Option: An operating system option that enables independently operating computing systems to share common data residing on shared direct access storage devices. The option is selected at system generation time, available with PCP, MFT, MVT (exception: MVT with Model 65 multiprocessing), and it provides the control program functions needed to control device reservation and release.

short block: A block of F-format data which contains fewer logical records than are standard for a block.

shoulder tap: A processing technique that uses the Write Direct instruction to enable one CPU to communicate with another CPU. (In a multiprocessing environment.)

simple buffering: A technique for controlling buffers in such a way that the buffers are assigned to a single data control block and remain so assigned until the data control block is closed.

simple name: The rightmost component of a qualified name (e.g., APPLE is the simple name in TREE.FRUIT.APPLE).

source module: A series of statements (in the symbolic language of an assembler or compiler) which constitutes the entire input to a single execution of the assembler or compiler.

stacked job processing: A technique that permits multiple job definitions to be grouped (stacked) for presentation to the system, which automatically recognizes the jobs, one after the other. More advanced systems allow job definitions to be added to the group (stack) at any time and from any source, and also honor priorities.

step library: A concatenation of user-identified partitioned data sets used as the primary source of load modules for a given job step.

step restart: A restart that is initiated through the use of special parameters on either the JOB or EXEC statements. The restart may be automatic (depending on an eligible ABEND code and the operator's consent) or deferred, where deferred involves resubmitting the job.

storage block: A contiguous area of main storage consisting of 2048 bytes to which a storage key can be assigned.

storage key: An indicator associated with a storage block or blocks, which requires that tasks have a matching protection key to use the blocks.

Storage Reconfiguration: For MVT with Model 65 multiprocessing a permanent storage failure in a non-critical operating system component (e.g., problem program storage) can be logically removed by Storage Reconfiguration and normal system operation can continue.

substitute mode: A transmittal mode used with exchange buffering in which segments are pointed to and exchanged with user work areas.

subtask: A task that is created by another task by means of the ATTACH macro instruction.

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

supervisor: As applied to the operating system, a routine or routines executed in response to a requirement for altering or interrupting the flow of operations through the central processing unit, or for performance of input/output operations, and, therefore, the medium through which the use of resources is coordinated and the flow of operations through the central processing unit is maintained; hence, a control routine that is executed in supervisor state.

supervisor queue area: The main storage area, adjacent to the fixed area, that 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.

synchronous: Occurring concurrently, and with a regular or predictable time relationship.

SYSIN: A name conventionally used as the data definition name of a data set in the input job stream.

SYSOUT: An indicator used in data definition statements to signify that a data set is to be written on a system output unit.

system input unit: A device specified as a source of an input job stream.

system library: The collection of all cataloged data sets at an installation.

system macro instruction: A predefined macro instruction that provides access to operating system facilities.

System Management Facilities: An optional control program feature that provides the means for gathering and recording information that can be used to evaluate system usage.

system output unit: An output device, shared by all jobs, onto which specified output data is transcribed.

system output writer: A job scheduler function that transcribes specified output data sets onto a system output unit, independently of the program that produced such data sets.

system residence volume: The volume on which the nucleus of the operating system and the highest level index of the catalog are located.

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.PROCLIB: The partitioned data set that contains cataloged procedures.

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

task: A unit of work for the central processing unit from the standpoint of the control program; therefore, the basic multiprogramming unit under the control program.

task control block (TCB): The consolidation of control information related to a task.

task dispatcher: The control program function that selects from the task queue the task that is to have control of the central processing unit and gives control to the task.

task management: A general term that collectively describes those functions of the control program that regulate the use by tasks of the central processing unit and other resources (except for input/output devices).

task queue: A queue of all the task control blocks present in the system at any one time.

telecommunications: A general term expressing data transmission between a computing system and remotely located devices via a unit that performs the necessary format conversion and controls the rate of transmission.

Teleprocessing: A term associated with IBM telecommunications equipment and systems.

test translator: A facility that allows various debugging procedures to be specified in assembler language programs.

text: The control sections of an object or load module, collectively.

throughput: A measure of system efficiency; the rate at which work can be handled by a computing system.

time slicing: an option available to users of MFT and MVT that allows them to designate that all tasks with a certain priority (MVT) or all tasks within a specified group of partitions (MFT) are to share the use of the CPU for an equal, predetermined length of time. Specified at either system generation or system initialization time (MFT users may also modify the specifications through the DEFINE command), this facility allows the user to prevent one task of a given priority or one partitions from monopolizing CPU time to the exclusion of all other tasks of the same priority or other partitions.

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

transmittal mode: The method by which the contents of an input buffer are made available to the program, and the method by which a program makes records available for output.

turn-around time: The elapsed time between submission of a job to a computing center and the return of results.

U-format: A data set format in which blocks are of unspecified or otherwise unknown length.

user: Anyone who requires the services of a computing system.

V-format: A data set format in which logical records are of varying length and include a length indicator; and in which V-format logical records may be blocked, with each block containing a block length indicator.

volume: All that portion of a single unit of storage media which is accessible to a single read/write mechanism.

Volume Statistics: An operating system facility that allows you to monitor read and write errors. Operating System Volume Statistics has two options: Error Statistics by Volume (q.v.) and Error Volume Analysis (q.v.).

volume table of contents (VTOC): A table associated with a direct access volume, which describes each data set on the volume.

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

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.

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

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

Index

Indexes to systems reference library manuals are consolidated in the publication IBM System/360 Operating System: Systems Reference Library Master Index, GC28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

Where more than one page reference is given, the major reference is first.

GC28-6535-7

System/360 OS Concepts and Facilities (S360-20) Printed in U.S.A. GC28-6535-7

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
Concepts and Facilities

Order No. GC28-6535-7

- 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	☐ Operator	☐ Sales Representative
☐ IBM personnel	☐ Programmer	☐ Systems Engineer
☐ Manager	☐ Customer Engineer	☐ Trainee
☐ Systems Analyst	☐ Instructor	Other ....

- Please check specific criticism(s), give page number(s), and explain below:

☐ Clarification on page(s)	☐ Deletion on page(s) ....
☐ Addition on page(s)	☐ Error on page(s) ....

Explanation:

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

GC28-6535-7

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source 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.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Cut Along Line

Fold Fold

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

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY ...

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

Attention: Programming Systems Publications
Department D58

Fold Fold

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]

System/360 OS Concepts and Facilities (S360-20) Printed in U.S.A. GC28-6535-7