File No. S360-20
Order No. GC28-6534-4

OS

IBM Systems Reference Library

IBM System/360 Operating System

Introduction

This publication describes operating systems in general and introduces the IBM System/360 Operating System. The publication is intended for anyone interested in the System/360 Operating System, whether or not he is familiar with other operating systems. It describes the purpose, design objectives, organization, function, and application of the System/360 Operating System, and how it was influenced by previous systems; it also describes System/360 compatibility with System/370.

The operating system consists of programming aids and a control program that schedules and supervises the processing of data. The system is designed for a broad range of applications, including teleprocessing and multiprocessing. It helps a System/360 data processing installation increase productivity by using its resources more effectively.

Fifth Edition (November, 1972)

This is a reprint of GC28-6534-3 incorporating changes released in the following Technical Newsletter:

GN28-2512 (dated January 15, 1972)

This edition applies to release 21 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 and System/370 Bibliography, Order No. GA22-6822, and the current SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, N.Y. 12602. Comments become the property of IBM.

Preface

This publication introduces you to the IBM System/360 Operating System, whether or not you are familiar with other operating systems.

Part 1 gives background information to help the newcomer understand operating systems in general. The first section describes the purpose of an operating system and the resources it requires. The second section describes the evolution of operating systems and why they evolved as they did.

Part 2 describes the System/360 Operating System. It discusses:

- The design objectives of the operating system and how they were achieved.
- The organization and function of the operating system.
- The two configurations of the control program.
- The major functions of the operating system: task management, job management, and information management.
- Program development and management aids.
- Advanced types of data processing (multiprocessing and teleprocessing) and the facilities that the operating system provides for them.

Part 3 contains a numbered bibliography. Superscripted items in text refer to these numbered publications. Further information about those superscripted items may be found in the corresponding publications.

For definition of terms used, see the IBM Data Processing Glossary, GC20-1699.

For more information on the System/360 Operating System, refer to the publications described in the IBM System/360: Bibliography, GA22-6822. The IBM System/360 Operating System: Master Index, GC28-6644, contains a reading plan for System Reference Library publications.

Contents

Illustrations

Figures

IBM System/360 Model 65

PART 1: OPERATING SYSTEMS

This part gives background information on operating systems in general, describing their purpose and how they evolved. If you are already familiar with operating systems, go to Part 2: The IBM System/360 Operating System.

Introduction

When computers were introduced several years ago, they were usually put to work on jobs that had required a great deal of routine human activity. Basic accounting, record keeping, and problem solving were a few of these early applications (Figure 1). By and large, the automatic processing of such jobs proved the speed, economy, and reliability of electronic data processing.

Later, computers entered a more challenging phase of development in which the industry began to devise system applications -- applications that go far beyond the mere mechanization of manual operations. Management information systems, process control systems, medical diagnosis systems, computer-assisted instruction (CAI) systems, and information retrieval systems are a few recent examples (Figure 2).

Today, as a result of this rapid progress, most data processing installations are facing an increase in the number of conventional applications as well as an increase in the scope and complexity of large-scale system applications. To cope with these problems, a data processing system must efficiently apply all of its resources: hardware resources, information resources, and human resources (Figure 3).

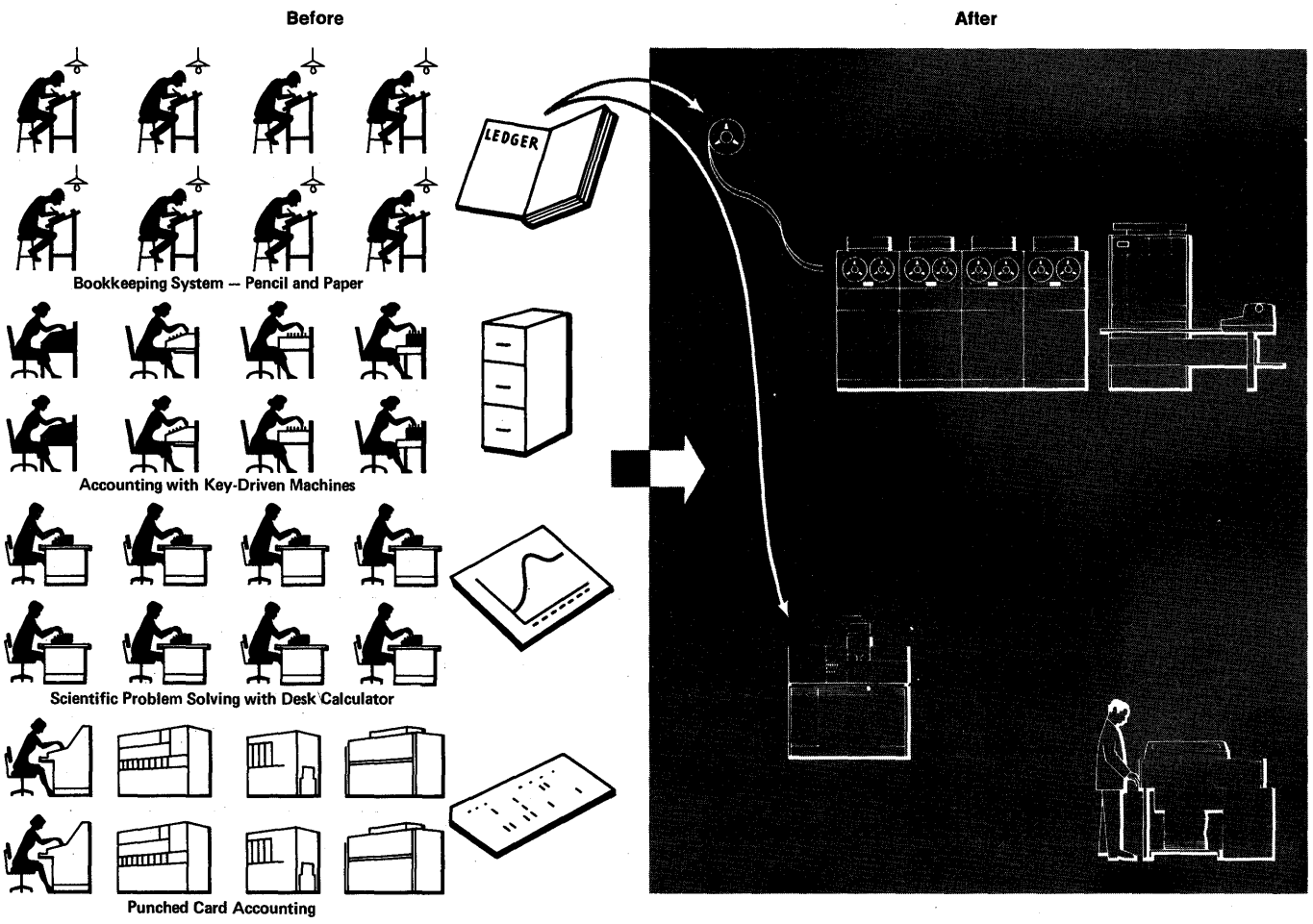


Figure 1. Mechanizing Routine Human Activities

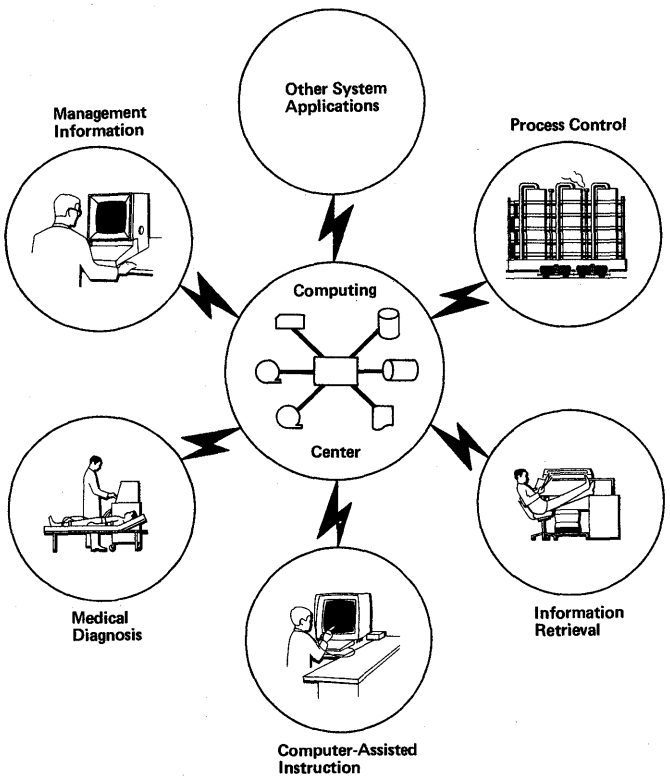


Figure 2. System Applications

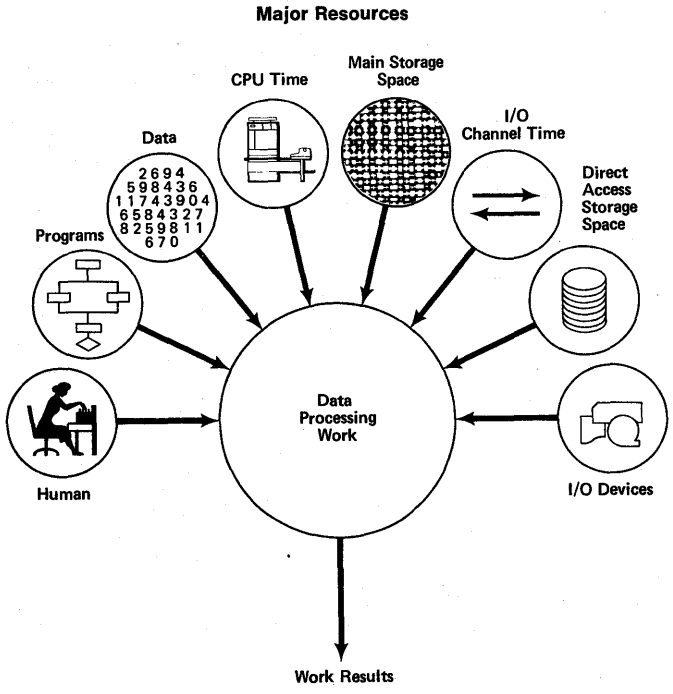


Figure 3. Resources Required to Process Data Automatically

Hardware Resources

To process data efficiently, the physical, or "hardware," components of a computing system must be available when they are needed. The hardware resources at a data processing installation are:

- Time on the central processing unit.
- Main storage space.
- Input/output devices.
- Input/output channel time.
- Direct access storage space.

The major hardware resource is the time available for doing work on the central processing unit (the CPU). The work the CPU performs is to operate on, or process, data. These operations usually consist of the basic arithmetic operations (addition, subtraction, multiplication, and division), transferring data from one storage location to another, converting data from one form to another, and performing simple tests or comparisons to choose between alternative operating sequences.

Individually, these basic operations do not seem very impressive, but the CPU performs them with great speed and reliability, and it can be programmed to perform long, complex sequences of operations without human intervention.

To perform long sequences of operations without human intervention, the CPU must have a resource for storing information in a readily accessible form. Therefore, the CPU has a main storage from which it can quickly obtain information and return results.

Also, there must be some way to quickly enter information into main storage. And, because the capacity of main storage is limited, there must be a way to record -- either for later processing or for use outside the system -- the results that the CPU places in main storage. Therefore, two additional resources are required to process data automatically: one or more input/output devices for reading and recording information and time on one or more channels for transmitting the information to and from main storage.

Devices that feed information into main storage (via a channel) or record information taken out of main storage (via a channel) are referred to collectively as input/output devices. However, they are also storage devices because they store information, whether it be in the form of printed characters on paper, holes in punched cards, or magnetized spots on tape, drums, and disks. Direct access devices, such as magnetic drum or disk units, are

usually called storage devices because they are mainly used for storing information. Other devices such as magnetic tape units, card readers, and printers are called input/output devices, because they are used primarily for entering information into the system and for recording output from the system. Therefore, I/O devices and direct access storage space are considered separate and distinct resources because their primary roles differ.

Information Resources

There are two kinds of information resources at an installation: the data that the CPU processes and the sequences of instructions, called programs, that direct the CPU to perform operations in a particular sequence.

Human Resources

Even more important than hardware and information resources are the human resources at an installation. These consist of the time and talent of the individuals who staff the installation and use the system. Although job responsibilities vary among installations, there are usually three groups of people under the direction of the installation manager:

- System programmers who plan, generate, maintain, extend, and control the system with the aim of improving the productivity of the installation.
- Applications programmers -- the main users of the computing system -- who plan new jobs (applications) for the system and develop the procedures and programs needed to perform them.
- Operations personnel, who receive jobs from the programmers, schedule the order in which jobs are presented to the system, and direct the operation of the system.

To a great extent, the success of a data processing installation depends upon how well these human resources are applied.

The Effective Use of Resources

The resources of a data processing installation represent a considerable investment; it is important, therefore, to use them efficiently. To do this, it is necessary to keep each resource busy doing the kind of work it is best suited for. Hardware and information resources must be readily available so the CPU can be kept busy with the tedious work of processing data. Human resources must be relieved of tasks that the computing system can perform, and dedicated to more creative work: planning new applications, formulating solutions to problems, reacting to changing conditions and unexpected events, and managing the installation.

The productivity of a data processing installation depends on how well its hardware, information, and human resources are selected and employed to do the work at hand. A modern computing system can perform billions of basic operations in a few minutes, and with far greater reliability than a human being. But man must plan all of the work the computing system performs and in doing so must account for every contingency that might arise. In short, a computing system only follows orders. Lacking a program, the system is useless.

But, with a program, a computing system can do much more than process input data to produce output data. By using the few instructions of a computing system in different combinations and sequences, a programmer can create a program that will direct it to do many things that were once done only by human beings. These include translating languages, managing resources, retrieving information, scheduling and supervising work and operating and controlling mechanical devices.

A program, or set of programs, that directs a computing system to perform such operations is called an operating system. An operating system is really an advanced system application of a computing system in the form of organized collections of programs and data. Like other system applications it is designed to handle complex activities, but it differs in the kind of activity it supports. Most system applications support specialized activities outside of the data processing installation, such as banking, process control, or missile design. An operating system is designed to support the activities of the data processing installation itself. In short, an operating system is an application of a computing system, in the form of program and data resources, that is specifically designed for use in creating and controlling the performance of other applications. Its prime objective is to improve the performance of a data processing system and increase facility -- the ease with which the system can be used.

Performance

The overall performance of a data processing system is determined by a combination of three factors. They are throughput, response time, and availability (Figure 4).

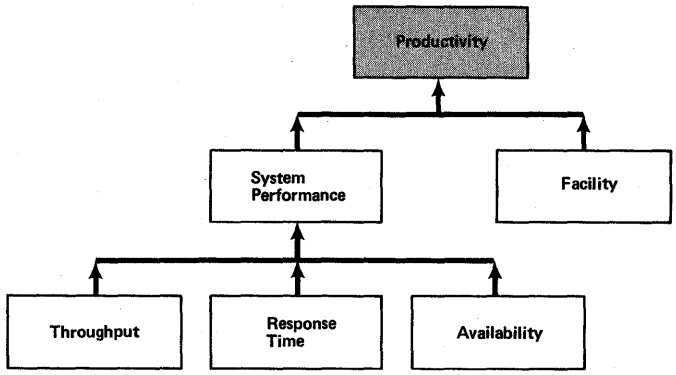


Figure 4. Productivity Factors

THROUGHPUT

Throughput is the total volume of work performed by the system over a given period of time. This is an important factor at any installation; however, it is especially important at a high-volume production installation in which large batches of work are performed in accordance with a flexible time table. This type of operation is typical of many accounting and record keeping applications.

RESPONSE TIME

Response time (sometimes called turnaround time) is the interval between the time a user (or a process-controlling device) submits an item of work to the system for processing and the time he receives results. Response time is especially important where many different people share the use of the system and the overall progress of their work depends on their receiving prompt results from the system. For example, response time is important in a design activity in which a series of calculations is required to complete each design and the designers cannot continue their work until they receive the results of previous calculations. For such activities a decrease in response time increases the pace of the activity and improves human productivity, since less time is spent in idly waiting for results.

AVAILABILITY

Availability is the degree to which a system is ready when needed to process data. The availability of a system is especially important at an installation where a prolonged period during which the system is not available might result in a complete shutdown of the total enterprise.

A high level of availability can often be achieved by including additional hardware resources, such as multiple input/output channels and devices, as well as multiple central processing units. Then if one unit fails it can be immediately replaced by another, thus helping to ensure continued operation of the system as a whole. A system using multiple units of hardware resources, including two central processing units, is described later in the section titled "Multiprocessing."

Facility

Facility is a measure of how easy it is for people to use a data processing system. Facility is achieved in a system mainly by providing the users at an installation with a combination of programming aids, services, and precoded routines that can be employed using appropriate language statements. The general idea is to simplify and speed the job of defining, programming, and scheduling work for the system, thereby making more efficient use of the human resources at an installation.

Facility can also be increased by improving the response time of the system and the degree of interaction between the system and the people who use it. An example of this is a computer assisted instruction (CAI) application in which a dialogue or conversation takes place between a student and the system. In such an application, the student learns from his mistakes as he makes them, thereby speeding up the learning process. A similar approach can be used to improve and speed up problem solving and programming processes as well.

Although operating systems differ in the way they achieve their objectives, they have many common characteristics. In fact, modern operating systems use many concepts and techniques developed in the past. Therefore, the next section traces the evolution of operating systems. This will prepare you for "Part 2: The IBM System/360 Operating System" and help you to understand how and why the system came to be what it is and how it differs from earlier operating systems.

The Evolution of Operating Systems

Modern operating systems, as typified by the System/360 Operating System, evolved in three stages. The first stage began during the early days of electronic data processing, when the major components of an operating system were developed. During the second stage, these components were integrated to form systems and the name "operating system" came into use. The third stage was a union of known techniques with new ideas, aimed at the development of a general purpose system that could improve the productivity of a data processing installation.

The First Stage: Component Development

The first stage began mainly because of language differences between computers and the people who used them. The computer could not understand human languages, and programmers found the precise, restrictive, numerical language of a computer hard to read and write. Futhermore, it was difficult to change a complex program written in "machine language" and each change often triggered a chain of errors.

The language problem was a serious obstacle to the efficient use of human resources. For a time it threatened to limit the growth of electronic data processing. Although a computer could process data rapidly, it had to be fed vast amounts of instructions and data, all in its detailed, cumbersome, numerical language. This required a large staff of programmers who spent their time, not in creative work, but in the tedious translation of applications and problem solutions into machine language.

For example, if a mathematician had a problem to solve -- such as computing a set of trajectories for an artillery shell -- he would usually do it in three steps (top of Figure 5). First, he would analyze the problem and formulate a procedure to solve it. The mathematician had neither the time nor inclination to code his procedure in machine language. Accordingly, he would turn over his procedure and data to programmers. The procedure might be in the form of a flowchart, a mathematical formula, or a series of general instructions; in any event, the procedure was not in a form that was understandable to the computing system.

Therefore, during the second step, it was up to the programmers to translate the procedure into the detailed numerical language of the computing system. Depending on the complexity of the problem, the translation step could take days, weeks, or even months to complete and check out.

The third step consisted of executing the program on the computing system to process the data and produce results. After all of the time and effort devoted to translation, the execution step was often completed in a few minutes. The same general three-step process was followed in developing a business application -- again, far too much time and effort was expended in translating procedures into a form acceptable to the computing system.

Largely as a result of this language problem, two important, interrelated movements began in the data processing industry. One was the development of a set of programming aids that could be used to assist programmers in doing their work. The other was the sharing of program and data resources among the people who used computing systems. These two movements helped correct the imbalance of preparation time and execution time.

PROGRAMMING AIDS

The development of programming aids was begun by system programmers who saw that too much time was being spent catering to the needs of the computing system. Since many routine operations were involved in preparing programs and data for the computing system, why not, they reasoned, use some of the speed and resources of the computing system to do the work? Therefore, the programming aids they developed took the form of programs that could be executed on the computing system. The most important of these were translator programs and input/output control systems.

Translator Programs

Translator programs were designed to translate programs written in a language that human beings could understand into the numerical language of the computing system (a, Figure 6). A number of "machine oriented" assembler languages were the first to appear (b, Figure 6). These were

soon followed by such "human-oriented" languages as FORTRAN (c, Figure 6) and COBOL (d, Figure 6). Programmers could now write their instructions to the computing system in language that were akin to mathematics (FORTRAN) or the language of business (COBOL). With the development of FORTRAN, the mathematician (bottom of Figure 5) now had a reasonable choice. He could define his procedure in mathematical notation (for example, $A=\frac{X}{Y(Z)3}$) and let a professional programmer convert it into a FORTRAN language source program. Or, he could define the procedure directly in FORTRAN (A=X/(Y*Z**3)) with little or no help from the programmer. The source program could then be quickly translated into a machine language object program and executed by the computing system.

Thus programmers were no longer slaves to the needs of the computing system. They could devote more of their time and effort to creative work. The translators not only speeded up the programming process but reduced programming errors, made it much easier to correct errors, produced better documentation of programs, reduced training time, and made life easier for anyone who wanted to use a computing system. The translation process was shortened, the burden of translation was shifted from the programmer to the computing system, and a balance between the two was achieved.

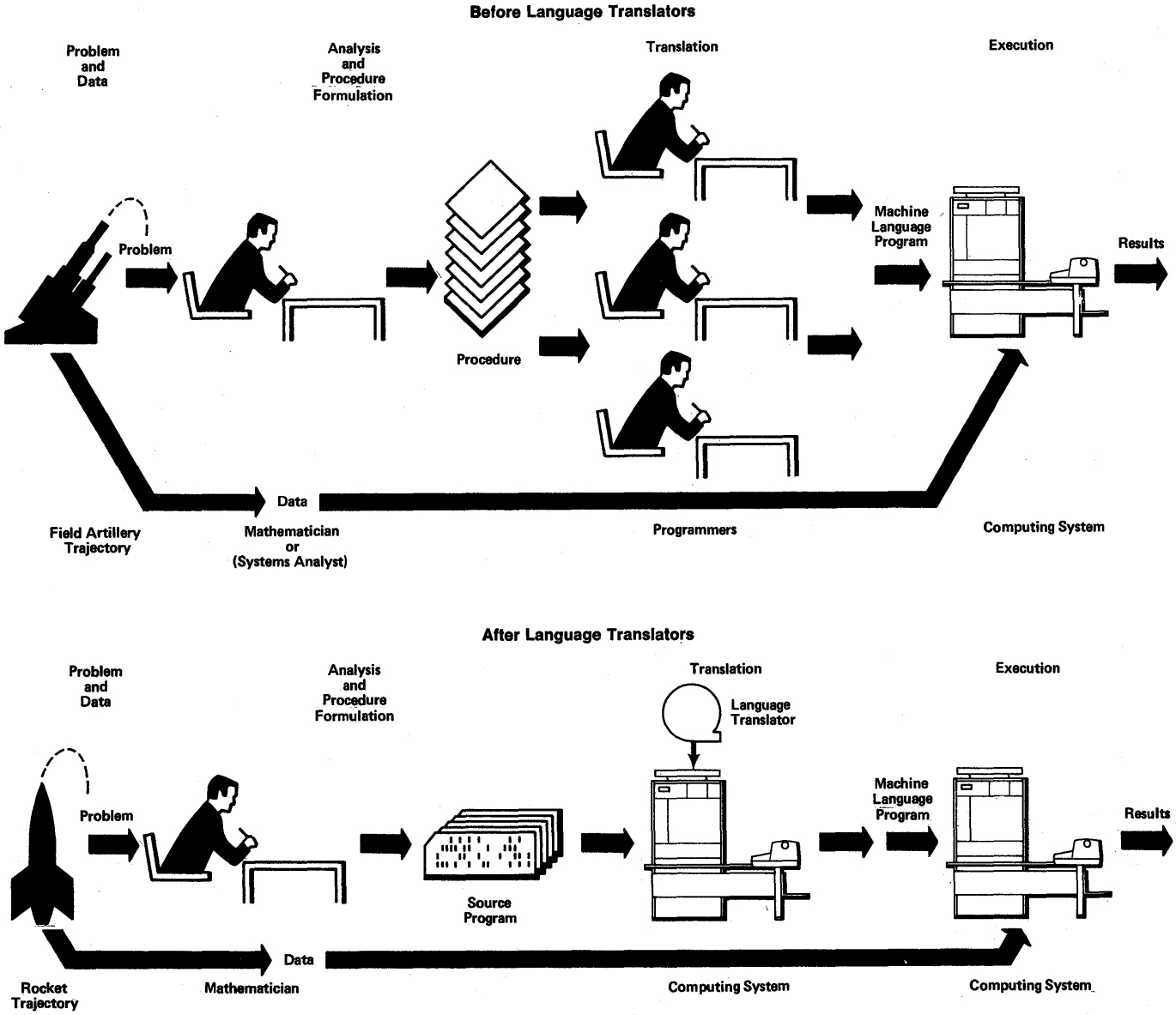


Figure 5. Problem Solving Before and After the Development of Language Translator Programs

a. Machine Language

```
00035  4754 00 2 00000
00036  4734 00 1 00000
00037  0560 00 2 00000
00040  0500 00 1 00000
00041  0040 00 0 01002
00042  0131 00 0 00000
00043  0601 00 1 00000
00044  2 00001 1 00040
00045  4600 00 2 00000
```

b. Assembler Language

```
LOADX1 PXD     0,2
       PDX     0,1
LOADQ  LDQ     0,2
LOADAC CLA     0,1
       TLQ     *+2
       XCA
STOREA STO     0,1
       TIX     LOADAC,1,1
STOREQ STQ     0,2
```

c. FORTRAN Language

```
16 READ(5,20)A,B,C
20 FORMAT(3E10.3)
   IF(A)30,40,30
30 D=B*B-4.*A*C
   IF(D)50,60,70
50 XR1=-B/(2.*A)
   XR2=-B/(2.*A)
   XI1=(SQRT(-D))/(2.*A)
```

d. COBOL

```
IF DEMAND IS GREATER THAN STOCK-ON-HAND,
    ADD DEMAND TO BACK-ORDERS,
    MOVE BACK-ORDERED TO ACTION-CODE;
OTHERWISE,
    SUBTRACT DEMAND FROM STOCK-ON-HAND,
    MOVE ORDER-FILLED TO ACTION-CODE.
```

Figure 6. Programming Languages

Input/Output Control Systems

In addition to the language translators, other important programming aids, called input/output control systems, were developed. These systems were designed to improve the way computing systems performed I/O operations. In the early computing systems, the relatively slow I/O operations and the much faster data processing operations of the CPU could not be performed at the same time (Figure 7). Therefore, the CPU was idle much of the time waiting for the completion of data transfers between I/O devices and main storage. To reduce this idle time, computing systems were soon developed that could perform input, output and data processing operations all at the same time (Figure 8). This represented a significant improvement in the performance of computing systems. However, to take advantage of the improvement the programmer had to make sure that the I/O operations were synchronized with the processing of data; otherwise, the CPU might attempt to process input data before it arrived in main storage or destroy output data before it was transferred to an output device. Therefore, input/output control systems were developed to automatically synchronize I/O operations with data processing.

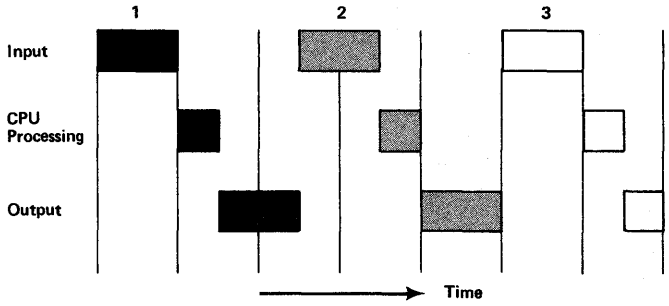


Figure 7. Sequential Input, Processing, and Output

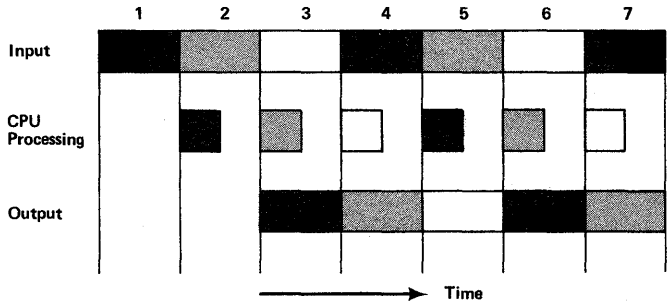


Figure 8. Concurrent Input, Processing, and Output

An input/output control system consisted of an interrelated group of programs that was loaded into main storage along with the processing programs. Using such a system, a programmer merely had to issue a "READ" instruction to obtain the next block of data from an input device or a "WRITE" instruction to send a block of data to an output device. The input/output control system picked up and interpreted the instruction and then initiated and controlled the necessary transfer of data to or from main storage. In the meantime, the CPU could continue processing data.

If each block of input data contained more than one record the programmer merely issued a "GET" instruction to get the next record in sequence. The input/output control system automatically controlled the transfer and storage of data blocks and parcelled out records one at a time from the blocks as they were requested by the processing program (Figure 9). Similarly, to transfer an output record, the programmer merely issued a "PUT" instruction. The input/output control system then picked up and consolidated records into a block before transferring the block to an output device.

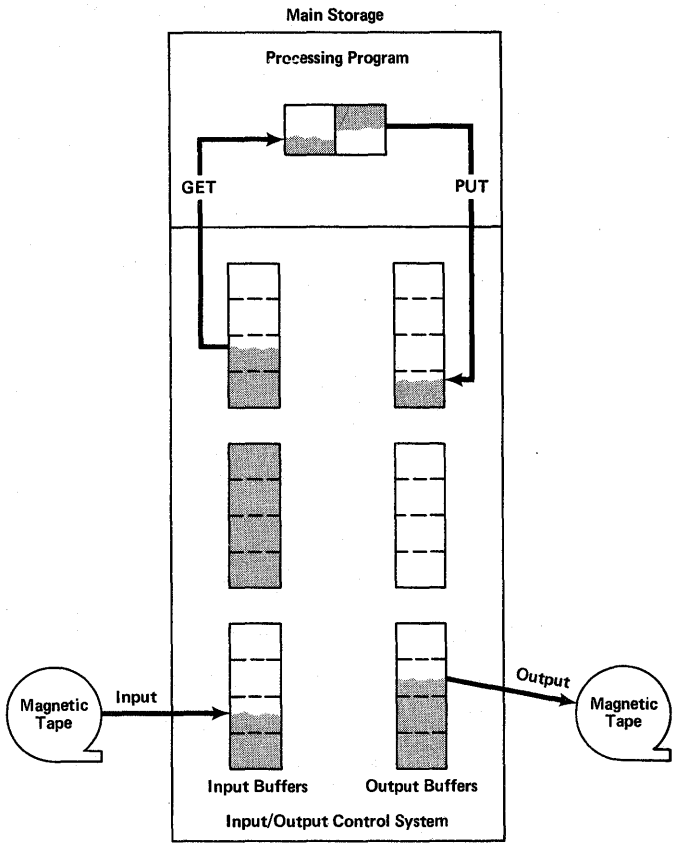


Figure 9. Data Retreival and Recording Using an Input/Output Control System

Input/output control systems assisted programmers in other significant ways. For example, if an error was detected during an input/output operation, the system automatically retried the operation and attempted to recover from the error condition. It also checked labels at the beginning of magnetic tape reels to ensure, among other things, that the correct reel was mounted on the right tape unit. Input/output control systems, as a whole, represented an important step in the evolution of operating systems.

Other Programming Aids

In addition to language translators and input/output control systems, a variety of other programming aids began to evolve. These included programs for generating reports, loading other programs into main storage, combining several programs into a single program, and recording the contents of main storage in a readable form.

PROGRAM AND DATA SHARING

The other major movement that led to the development of operating systems started with the realization that a great many of the program and data resources at an installation could be shared by different users of the computing system, thereby avoiding a great deal of programming and data-gathering duplication. A program used in one job could often be used with little change in another job. Also, the same set of data that was processed during one job could often be processed in a different way during another job. Program and data sharing began as an informal cooperative venture among users within an installation but it soon involved a great many users at different installations. In fact, users of medium and large scale IBM computing systems formed an organization, appropriately called SHARE, to promote the sharing of program and data resources.

Subroutine Sharing

Program sharing took many different forms. One was the sharing of subroutines -- relatively short sequences of instructions that could be incorporated into a program to perform specific functions, such as finding the square root of a number. This type of sharing was often used in conjunction with language translators. Some translators had access to a magnetic tape library of subroutines and incorporated them into a program when instructed to do so by the programmer.

Sharing of Generalized Programs

Another form of sharing was the use of generalized programs for performing common data processing tasks. Typical of these were the generalized sort/merge programs. At many business installations it was found that a large percentage of computing system time was spent in sorting and merging data records into a prescribed sequence.

Rather than design a new program each time a different set of records had to be sorted or merged, many installations reduced the programming effort by using a single general purpose sort/merge program that could be easily modified to process different types of records in different formats and sequences.

Among other shared generalized programs were utility programs, used to perform everyday tasks such as transcribing data from one storage or I/O device to another.

Formal Sharing

As time went by, program and data sharing techniques matured and the sharing became more formalized. For example, many of the generalized programs were refined and improved so that they automatically modified themselves in accordance with

specifications supplied by the programmer. These specifications often developed into formal language statements that a programmer could use to communicate precisely his special data processing needs to the generalized program. At most installations the operations staff maintained, on magnetic tape or in the form of punched cards, a central library of programs and programming aids that could be shared among the members of the installation.

GROWTH IN APPLICATIONS

Largely because of the development of programming aids and the formal sharing of program and data resources, the number of data processing applications grew at a surprising rate throughout the industry. The language barrier between the programmers and the computing system, although not eliminated, had at least been breached. Many installations that had specialized in long-running or often-run jobs could now afford to program small one-shot jobs. Many of the jobs were developed by engineers and others who were not professional programmers. Using the mathematical language of FORTRAN they could now do their data processing work with little or no help from a professional programmer. The professional programmer could devote much more of his time and ingenuity in devising new applications and posing new problems for solution.

As a result, the number of data processing jobs at many installations increased faster than the computing system and its operator could handle them.

Second Stage: Integration and Automatic Operation

The second stage, like the first, began as a result of basic differences between human and hardware resources. In the first stage, they were language differences between computing systems and the people who programmed them. In the second stage, they were differences in speed, reliability, and reaction time between computing systems and the people who operated them.

THE MISCAST ROLE OF THE OPERATOR

As the volume of data processing jobs increased throughout the industry, the differences between the operator and the computing system became more and more apparent and significant. Because of these differences, a computing system spent a large part of its time idly waiting while an operator peformed routine tasks or momentarily pondered what to do next. The problem lay in the simple fact that the operator was too much involved in the mechanics of data processing. In this role, he could not match the data processing speed and reliability of a computing system no matter how swiftly and surely he did his work. An operator often spent more time preparing (setting up) a computing system for a job than the system spent in performing it (Figure 10). Some computing systems, in fact, spent more than half of the work day idly waiting for the operator to do such things as mount magnetic tape reels, place punched cards in a card reader, or manipulate manual controls. Wasted time, due to operator intervention, was especially severe at installations where many small jobs were performed (Figure11A). The steadily increasing speed of computing systems could not compensate for such wasted time because as jobs were processed faster, operator "set-up" time between jobs remained the same. Therefore, an even larger percentage of computing system time was wasted waiting for the operator (Figure 11B).

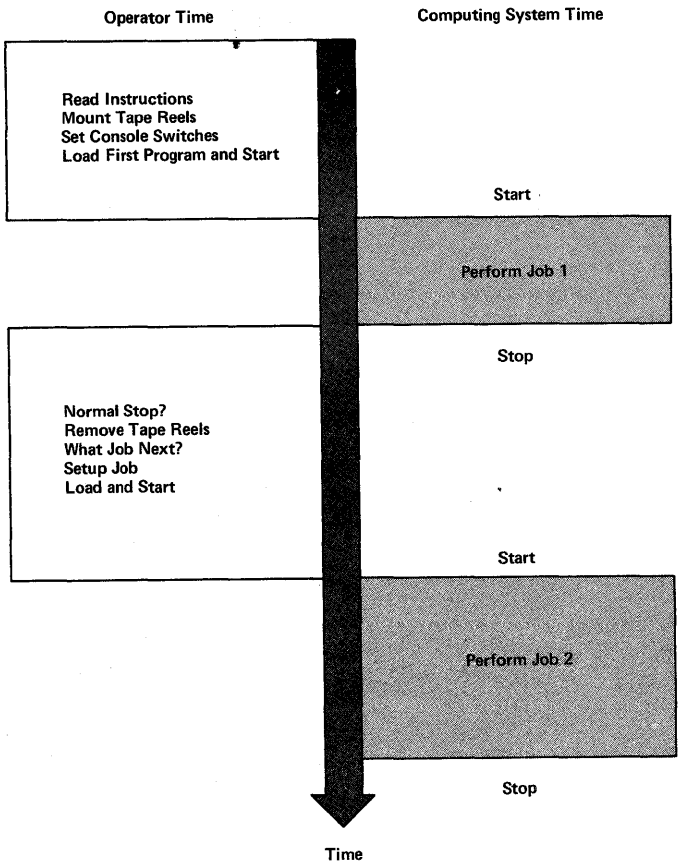


Figure 10. Idle Computing System Time Between Jobs

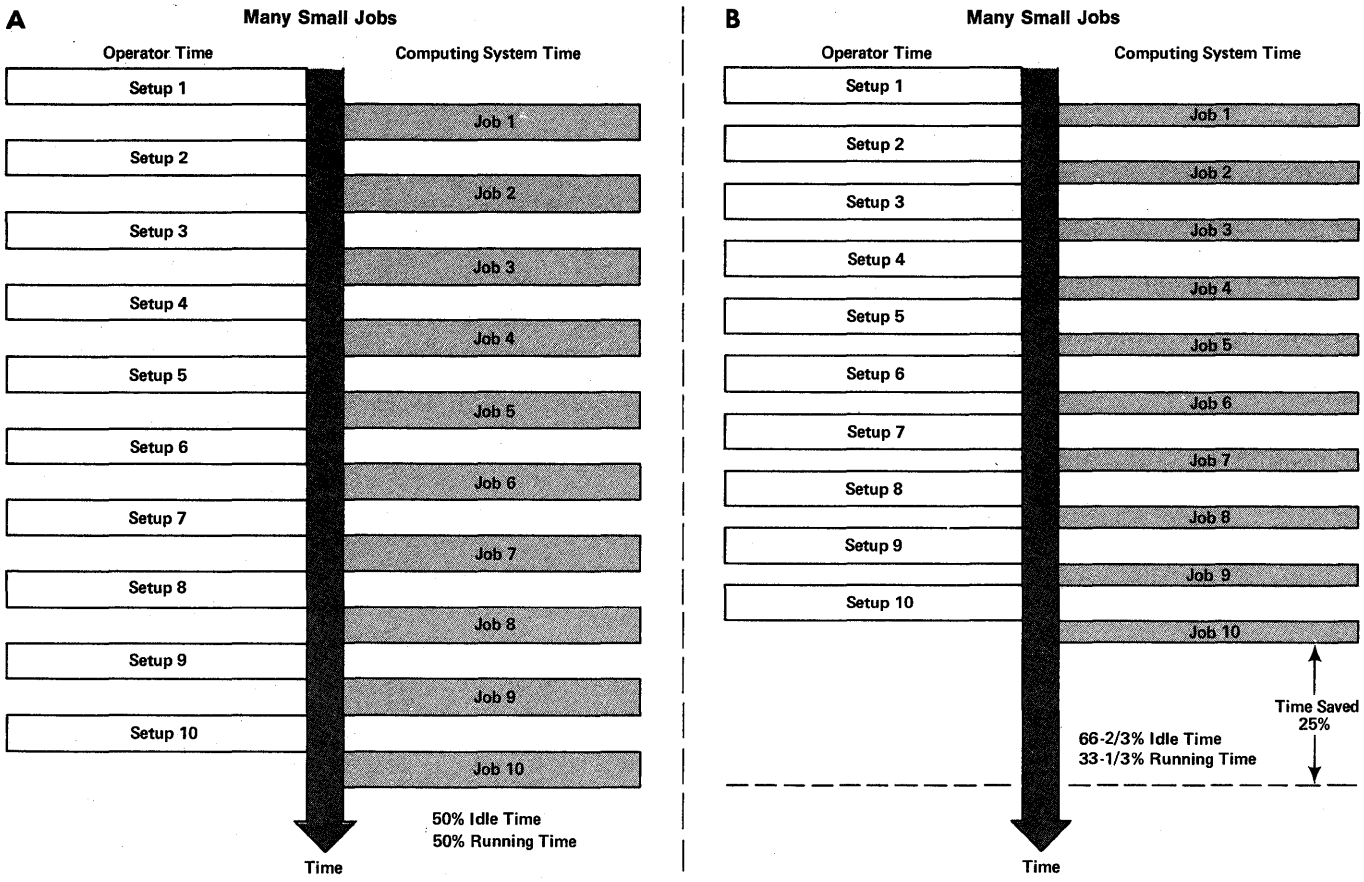


Figure 11. Idle Time When Processing Many Small Jobs

It isn't surprising then, that the emphasis during the second stage in the evolution of operating systems was on applying the fast hardware resources of the computing system to reduce the data processing activities of the operator. The ultimate objective was the non-stop processing of jobs (Figure 12).

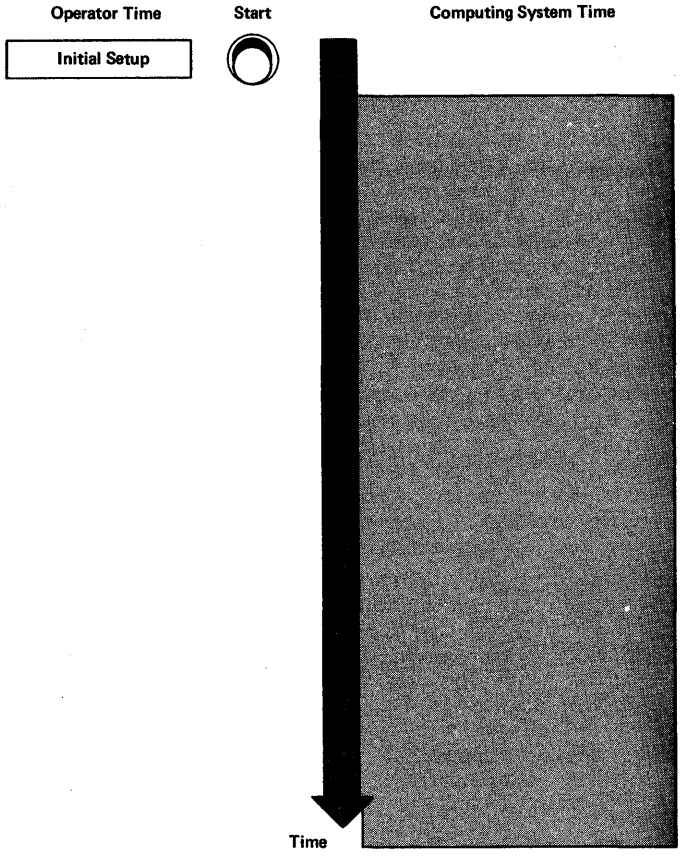


Figure 12. The Ultimate Objective: Non-Stop Processing of Jobs

SYSTEM INTEGRATION OF FIRST-STAGE COMPONENTS

During the second stage many of the programming tools and generalized programs that were developed during the first stage were brought together and placed under the direction of a central control program to form a single integrated system (Figure 13). The original reason for doing this was to improve and, to a large extent, mechanize the operation of the computing system. Hence the name, "operating system."

The operating systems that were developed during the second stage had a lot of common characteristics. Many of their features have survived the test of time and, with refinements, improvements, and extensions, still exist in present day systems. Therefore, it might be helpful at this point to briefly describe a typical operating system, concentrating on those characteristics that were first introduced during the second stage and are now more or less common to all systems. This description can then serve as background for understanding the third stage of development, as typified by the System/360 Operating System.

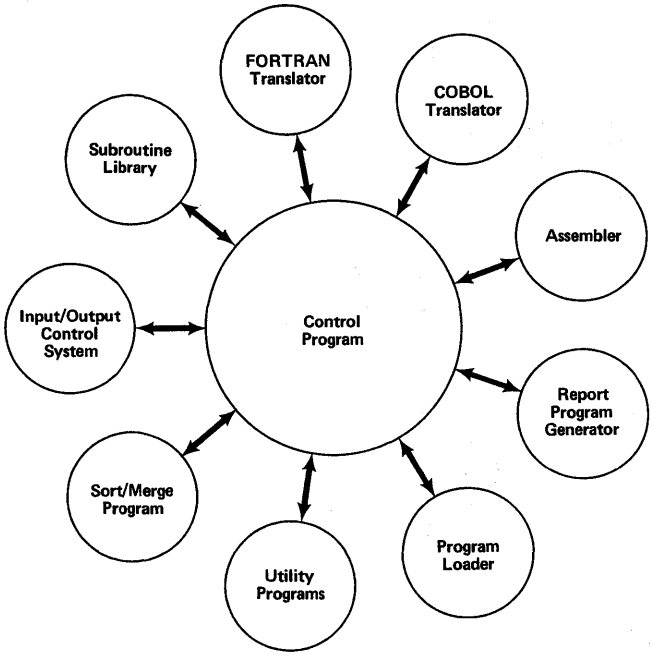


Figure 13. System Integration of First Stage Components

A TYPICAL OPERATING SYSTEM

When operating systems were first introduced, their major, and sometimes only, purpose was to control the performance of a continuous series of independent jobs with as little operator intervention as possible. This was largely accomplished by designing a control program that assumed many of the functions previously performed by the operator. By reducing the degree of human participation in the mechanics of job processing, the control program helped to ensure that jobs were processed faster and more efficiently, and were less subject to human error. It also provided the operator with more time to plan and direct the overall operation of the system.

In various operating systems, the control program was called the system monitor, the executive program, the master program, or some such name. Whatever the name, each had the same basic goal, that is, the non-stop processing of jobs.

A New Control Language For the Programmer and Operator

To reach its goal, the control program had to relieve the operator of his miscast role as a middleman between the programmer and the computer system. Therefore, a control language was established which a programmer could use to bypass the operator and communicate directly to the control program a precise definition of the work (jobs and job steps) he wanted performed. This language consisted of several formal job control statements that could be recorded on punched cards and later read, interpreted, and acted upon by the control program. One statement, usually called the JOB statement, was used to identify and mark the beginning of a job. Another statement, usually called the EXECUTE statement, was used to mark the beginning of a job step (or "job segment" as it was sometimes called) and identify by name a specific program that was to be executed to perform the job step. In most systems each job could consist of one or more such steps (Figure 14).

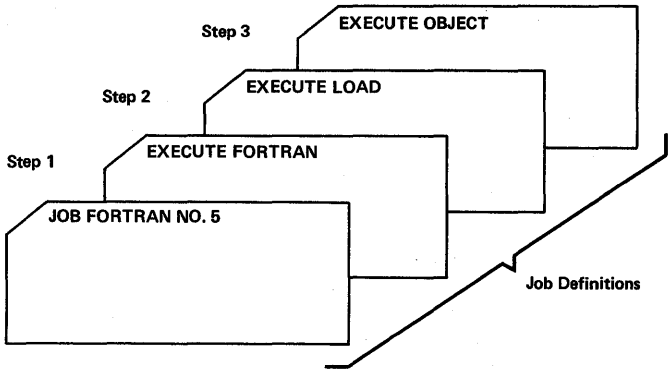


Figure 14. JOB and EXECUTE Control Statements

Although the JOB and EXECUTE statements (or their equivalents) were by far the most important and often-used control statements, most control programs were designed to read, interpret, and react to a number of other statements. A few of these were provided for use by the programmer. Although the exact number and type differed depending on the specific control program, they were generally used in an auxiliary capacity for such things as instructing the control program to relay a message to the operator.

However, most of the control statements were designed for use by the operator in communicating information to the control program and directing its overall operation. These usually included several statements that the operator could use to alert the control program to any changes he wished to make in the status or assignment of I/O devices. Thus, the operator was provided with a formal language, in the form of control statements. With it, he could exercise general control over the system, without necessarily intervening in its automatic operation.

Batched Job Processing

In order to reduce computing system idle time, the control program automatically controlled the transition from one job or job step to another. To do this, it had to have a backlog of jobs available and awaiting processing. Therefore, as job definitions, in the form of punched cards, were received from the programmers, they were placed one behind another to form a batch, or "stack," of job definitions (Figure 15). The job batch was then placed on a computing system input device specifically assigned for that purpose. Because the job definitions were arranged in a continuous series on a common job input device, as soon as one job or job step was completed, the control program could read and initiate the next job. Thus a continuous stream of jobs could be read and processed with a minimum of operator intervention. This technique is referred to as either batched job or stacked job processing.

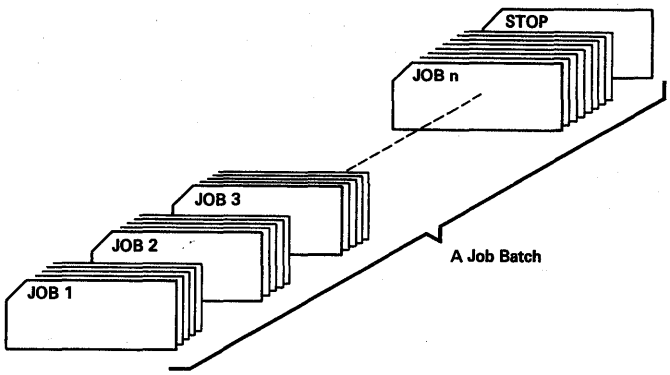


Figure 15. A Batch of Job Definitions

A Common Job Input Device

At some installations, the unit assigned to read the job input stream was a card reader. However, at many installations, especially large ones, each new batch of job definitions was transcribed onto magnetic tape before being read and processed by the control program (a, Figure 16). The transcription was usually done on

a small auxiliary computing system, such as an IBM 1401 Data Processing System. The reason for transcribing job batches in this way was to avoid continually tieing up the larger and more expensive main computing system while relatively slow card reading operations were being performed.

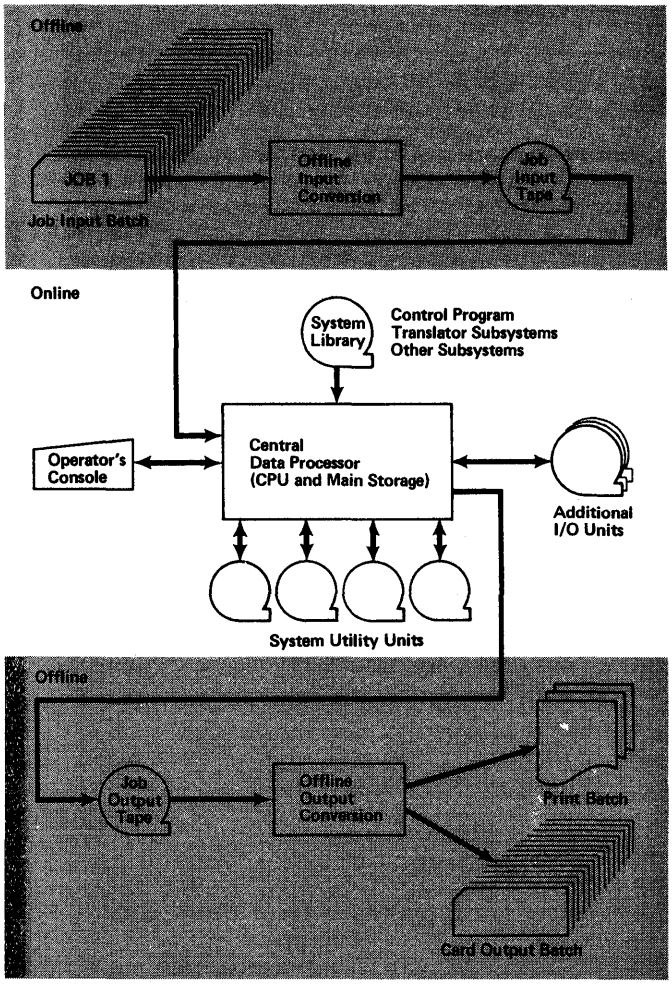


Figure 16. Job Processing at a Typical Operating System Installation

Common Utility I/O Devices

With some systems, advantages were gained by assigning other I/O units to perform specific roles for a series of job steps. For example, several units were often assigned as utility units that could be used for the temporary storage of intermediate data during the course of a job step (b, Figure 16). Since the data they contained at the end of a job step was no longer required, the same units could be used by the next job step without the operator having to change tape reels.

A Common Job Output Device

Just as an input device was assigned and used as a common job input file, most systems used one or two output devices as common job output files for recording output data.

In some systems, a printer or card punch was used for this purpose. However, in most large-system installations each batch of output data from a series of jobs was recorded on magnetic tape in order to avoid the relatively slow printing and card punching operations. The output data was then transcribed into printed or punched card form by a small auxiliary computing system (c, Figure 16). This was usually the same auxiliary system that was used to transcribe the input job batches from card to tape. In some installations, a single magnetic tape unit was used to record job output data that was to be printed as well as data to be punched on cards. The auxiliary computing system separated and directed the two types of data to the proper device during the output operation. Usually the punched card output consisted of object programs produced by language translators. The printed output usually consisted of messages from the operating system, source program listings, storage dumps, and other types of data from specific applications.

Although the offline handling of job input and output increased the rate at which a batch of jobs could be processed, it had its drawbacks. Because a batch of jobs had to be scheduled three times in succession in the overall process of handling jobs, the programmers at many installations found they had to wait a considerable period of time after submitting a job before they received any results. Nevertheless, sharing a common output unit for a series of jobs made it unnecessary for the operator to remove an output tape reel or a deck of cards at the conclusion of each job or job step.

Additional I/O Devices

By assigning I/O units to perform specfic functions and by sharing their use for a series of jobs, many relatively small jobs could be performed without any operator intervention at all. However, operator setup time was not eliminated entirely. Many large job steps required additional I/O units. These had to be set up by the operator before processing of a job step could begin. However, most control programs kept up-to-date records of the exact status of all I/O units. Whenever an

additional tape unit or other device was required to perform a job step, it was assigned from a pool of available units and the operator was given instructions as to which tape reel to mount on which unit. He no longer had to ponder what to do next.

Automatic Step-to-Step Transition

Whenever the control program encountered an EXECUTE statement while processing a series of jobs, it loaded into main storage the program named in the statement. The control program then relinquished control of the CPU to the program. After the program was executed to complete the job step, control of the CPU was returned to the control program. A program that was thus loaded and executed to perform a job step, could be any one of several frequently-used processing programs that were included as an integral part of the operating system. These programs were stored and maintained in auxiliary storage; (storage other than main storage) usually on one or more magnetic tape devices. As a group, they were usually referred to as the system library (Figure 16).

The System Library

The programs contained in the system library included the control program itself, as well as a number of programming aids and generalized programs -- much like those developed during the first stage in the evolution of operating systems. Many, in fact, were versions of these same programs, modified to execute under the direction of a control program. Although the number and variety of programs contained in the system library varied from one system to another, they usually included at least one language translator program, an input/output control program (or system), and a program that could be used in loading other programs into main storage. Many systems contained a much wider selection of general purpose programs that could be used by a programmer in performing one job step or a series of job steps. These included generalized sort/merge programs, utility programs, report program generators, and several different types of language translators. Many of these were designed, furnished, and maintained by the manufacturer of the computing system. However, some users often designed frequently-used programs of their own and incorporated them into the system library as a permanent part of their operating system.

Operating System Subsystems

In order to locate a program specified on an EXECUTE card, the control program maintained a record of the name location of each program stored in the system library. Some of the programs in the system library, such as the language translators, were in effect subsystems of the operating system because they contained a control program (or monitor) of their own which could read, interpret and react to one or more control language statements. The programmer used these statements to define more precisely the job he wanted performed by the subsystem. Such statements were placed behind the EXECUTE statement containing the name of the subsystem (Figure 17). After the main control program loaded a subsystem into main storage and relinquished CPU control, the subsystem could read its control statements from the common job input device, interpret them, and then perform the job step in accordance with specifications.

The specifications differed depending on the subsystem. For a generalized sort/merge program, for example, they included such things as a description of the records to be sorted. For a language translator subsystem, they included such things as whether or not the programmer wanted a printed listing of his source program or whether or not he wanted his program loaded into main storage and executed after being translated. For an Input/Output Control System, they included a description of the input/output data and the way it should be processed.

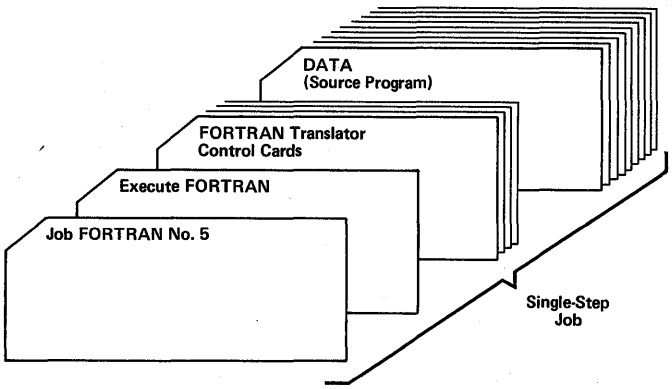


Figure 17. A Single-Step Job Containing Subsystem Control Cards and Data

In some operating systems, a programmer could place input data that was to be processed by a subsystem at the end of a job step definition (following the subsystem control statements, as shown in Figure 17). This data could then be read and processed by the subsystem as it was required. The data could be any of several

types depending on which subsystem was to process it. For example, it could be in the form of source language statements to be translated by a language translator subsystem, or machine language instructions to be loaded into main storage by a loader subsystem and then executed, or data records to be sorted by a sort/merge program.

Two important advantages were gained by placing data to be processed by a subsystem in the job input stream. First, it eliminated the need for an additional I/O device to read the data. Second, it reduced, or eliminated job step setup time by making it unnecessary for the operator to mount a separate tape reel or load a separate card deck containing the data.

The Control Program

The control program of most early operating systems consisted of two parts: the nucleus or basic monitor, and the transitional monitor.

THE NUCLEUS: The nucleus was so called because it always remained in main storage while a series of jobs were processed. It provided common facilities for intercommunication and control among the operator, the control program, and each of the subsystems operating under the control program. It contained information such as the exact status of each I/O device and the time of day. It also contained a number of frequently-used service or utility routines that were required by the control program, but could also be shared and used by subsystems. Typically, these consisted of small routines for loading programs from the system library into main storage, converting data from one form to another, writing messages to the operator, or initiating a main storage dump.

More importantly however, the nucleus contained supervisory routines that were needed to coordinate and control I/O operations. In some systems a complete input/output control system was included in the nucleus. It remained there for immediate use by any program that needed it.

In other systems, to conserve main storage space, only parts of the I/O control system were included in the nucleus. These were key parts that were needed to ensure centralized control of all I/O transfers and to prevent interference among the control program and its subsystems. In some systems, standard error recovery routines were included as well. However, other parts of the input/output control system were loaded into main storage only when they were needed in performing a specific job step.

THE TRANSITIONAL MONITOR: The other major part of the control program was sometimes called the transitional program or monitor. Unlike the nucleus, it occupied main storage only during the interval between one job or job step and another. Its main function was to read, interpret, and react to control statements (JOB and EXECUTE statements) from the programmer and the operator. In doing so, it automatically controlled the transition from one job step to another (by loading and transferring CPU control to the subsystem or program named on an EXECUTE card). Once the transition to a new step was completed, the main storage space occupied by the transitional part of the control program was available for use in performing the job step. After the job step was completed, the transitional monitor was loaded into main storage again to perform its function of initiating the next job step.

An Example: The IBM 7090/7094 (IBSYS) System

The operating system designed for the IBM 7090/7094 Data Processing system was one of the most widely and heavily used of the early operating systems (Figure 18). It was typical of other operating systems of the time in that it employed components that were largely developed during the first stage in the evolution of operating systems. For example, it used the FORTRAN II Processor (a language translator) which was developed by IBM customers before the operating system came into existence.

The IBM 7090/7094 Operating System was unusual, however, in that is contained what amounted to an operating system within an operating system. This was the IBJOB Processor Subsystem, shown in Figure 18, that contained its own control program, or "monitor" as it was then called. The IBJOB Processor subsystem could be used to compile, assemble, load, and execute programs written in FORTRAN IV and COBOL language. It could also be used to assemble, load, and execute programs written in an assembler language, or to load and execute previously assembled object programs. It also provided facilities for combining program segments written in different languages with previously assembled segments to form a single executable object program. Many of the innovations and techniques that were first used in the IBM 7090/7094 Operating System were later used in designing other operating systems.

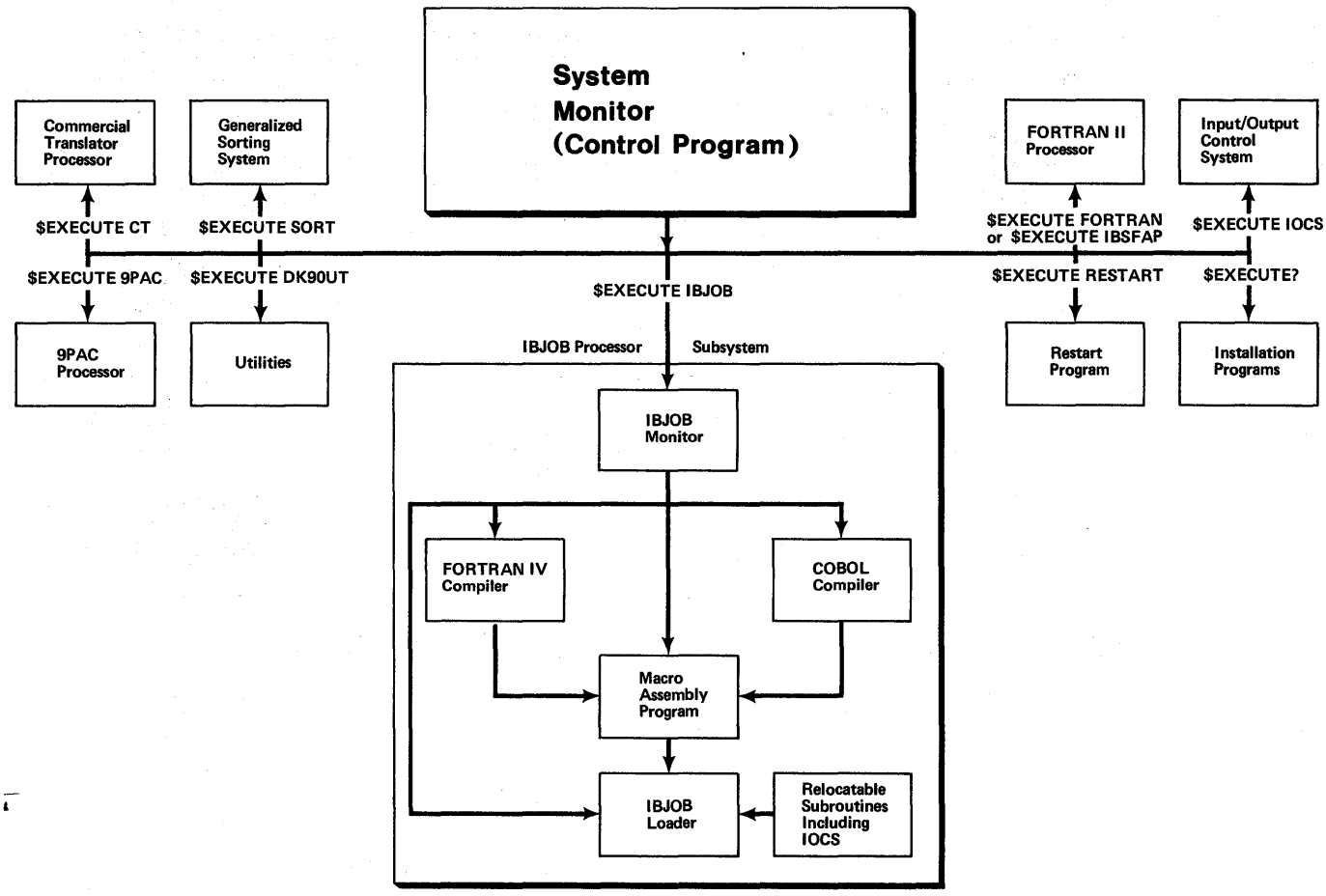


Figure 18. IBM 7090/7094 IBSYS Operating System

BENEFITS FOR LONG-RUNNING JOBS

The second stage extended the application of medium and large scale data processing systems by making it more profitable to use them in designing and executing series of small programs.

When large long-running programs (Figure 19) were executed under the direction of the operating system, the benefits were less, simply because there were fewer transitions and therefore fewer operator interventions to be eliminated. However, these large programs often required a great many man hours to design, test, and maintain. Therefore, to speed up the development process, the work of designing, testing, and maintaining them was often divided among several programmers. The language translators and program loaders in some systems had special provisions that allowed different parts of the same program to be designed, tested, and maintained independently and later combined to form a single program. In some systems, different parts of the same program could be written in different languages. Accordingly, the process of translating, testing and maintaining large programs in pieces often required the initiation and performance of a relatively large number of small jobs and job steps, like those shown earlier in Figure 11. Thus, the benefits to be derived from the automatic transition between jobs and job steps were not necessarily limited to the design and execution of small programs.

In any event, the running time of a typical medium or large scale computing system was improved by using an operating system. Moreover, the productivity of many programmers was improved because they were provided with a variety of centralized programming aids and precoded routines that they could employ singly or in combination without a great deal of difficulty. As a result, the amount of work that could be performed at many installations was significantly increased.

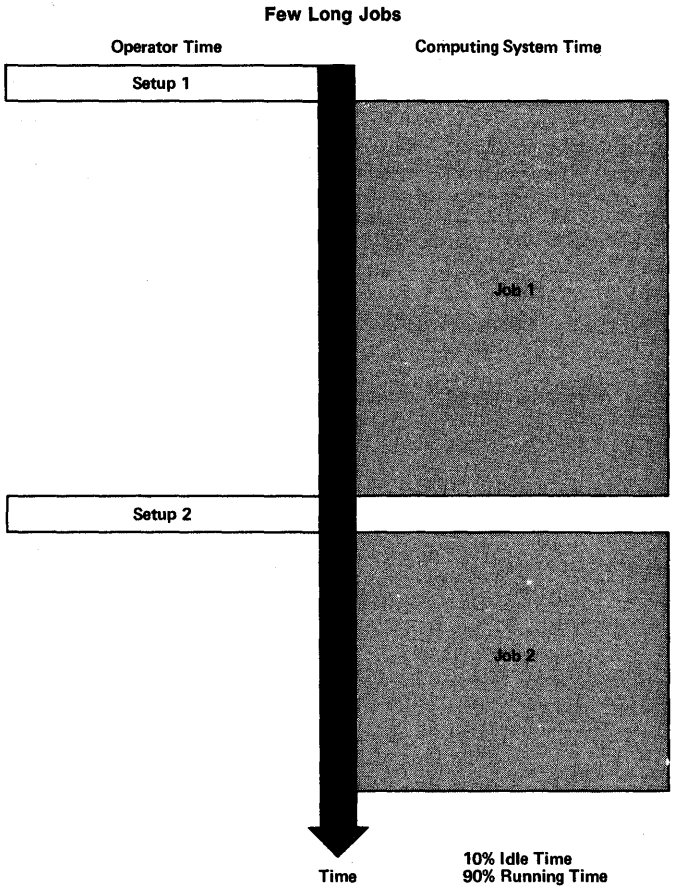


Figure 19. The Running Time of a Computing System When Processing a Few Long Jobs

OPERATING SYSTEM APPLICATIONS

The operating systems that were developed during the second stage were primarily designed for problem solvers -- engineers, scientists, and mathematicians. Most of these early operating systems were sequential, offline applications (Figure 20). They controlled the performance of work in a sequential order, one job or job step at a time, and with rare exceptions had no direct communication with the source or destination of the data that was processed. They were usually tape-oriented systems, relying on magnetic tape for auxiliary storage and fast I/O. Typically, they handled two kinds of jobs:

- Relatively small jobs, performed only once to solve a problem or to process a program being developed, tested, or maintained.
- Large, long-running jobs, performed periodically for commercial purposes, such as payroll accounting and record keeping.

Although some of the more recent systems used direct access storage devices for the system library and for temporary intermediate storage, few took advantage of the ability of these devices to store and access data quickly. Master data (data such as inventory records, personnel records, and payroll information, which represented the current business status of the organization) was maintained in a prescribed sequential order. Transaction data (such as debits, credits, and changes in personnel and payroll information, which represented the activities of the organization since the last update of the master file) was batched and presorted in the same prescribed order before it was processed. Therefore, the second stage operating systems were limited to applications in which it was not necessary to respond immediately to requests for processing and continuously update the master file.

In the meantime, however, a number of online, direct access systems were being developed independently of operating systems. In these applications, immediate response to requests for processing was of extreme importance.

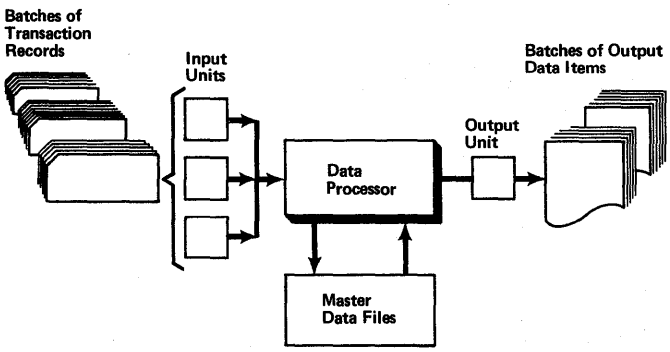


Figure 20. Sequential, Offline Application

Online Direct Access Systems

During the 1950's and early 1960's, a number of online direct access systems were developed independently of the second stage systems that came to be known as operating systems. Whereas operating systems were originally developed for problem solvers -- the scientific and engineering community of users -- online systems were developed for

specialized commercial activities that demanded immediate response to transactions (Figure 21).

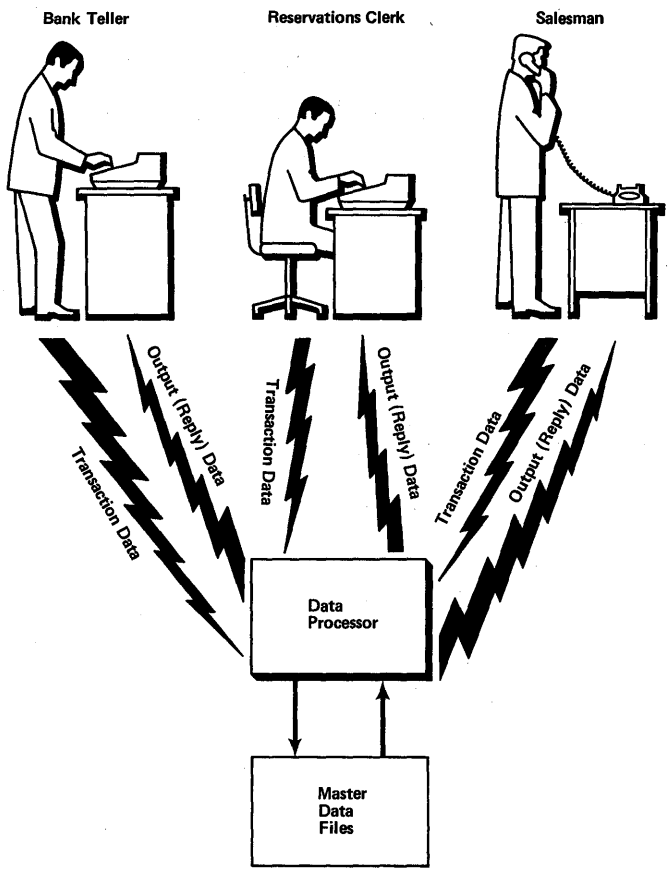


Figure 21. Online Direct Access Applications

In an online, direct access application, the system communicates directly with the source and destination of the data it processes. The data can be sent to or received from local I/O devices or devices at remote locations (by way of telecommunication lines). Therefore, transaction data can be processed as it is received. Also, the master files can be interrogated to produce up-to-date output information (reports, statistics, invoices, etc.) as required.

The records of the master files may or may not be arranged in a prescribed sequence. If they are arranged in a prescribed sequence, they may occasionally be processed sequentially (to prepare a summary report, for example), but usually they are interrogated in a random (non-sequential) order as they are received at the installation. Therefore, direct access storage devices are almost always used to store the master files in an online, direct access application.

AIRLINE RESERVATION SYSTEMS

Online direct access system applications were typified by the early airline reservation systems, in which master files, containing seat inventory records for hundreds of aircraft flights, were stored in direct access storage. By entering pertinent data into the system, ticket agents at widely separated locations could check the availability of space on a particular flight, sell and cancel reservations, and handle similar transactions, all within a few seconds.

However, there were some difficult problems in developing a system for this kind of application. The system had to respond quickly to each transaction and complete it within seconds. It was not possible to accumulate transaction data, sort it into batches, and then process each batch to interrogate or update the master file. Furthermore, the system had to keep the master file continuously up-to-date. A ticket agent and his customer would not tolerate long delays in awaiting confirmation on the availability of space on a particular flight. Moreover, if the master file was not kept up-to-date the agent might sell space already sold by another agent at a different location.

Solving these problems was a formidable undertaking, primarily because of the dynamic nature of an online application.

The Problem of Coping With the Work Load

A transaction usually required several references to the master file. In addition, different types of transactions (inquiries, sales, cancellations, etc.) required different programs to process them. All of these programs could not be in main storage at the same time. Therefore many of the programs had to be stored in direct access storage and brought into main storage each time they were needed to process a transaction. All of this took time. Much time was spent, not in processing transactions, but in locating and gaining access to data and programs in direct access storage. Meanwhile, dozens of other requests to process transactions might have been received.

To add to the problem, transactions occurred unpredictably. There were peak periods of activity, such as on weekends or during morning and evening rush hours, when hundreds or thousands of transactions would have to be processed within minutes. Even a sudden change in the weather could effect

a sharp increase or decrease in the frequency of the transactions. If the transactions were processed one at a time in the order they were received, it would be impossible to keep up with all of them, especially during periods of peak activity. To do so would require communication lines and computing systems that were much faster then those available at the time. Therefore, some means had to be found to handle transactions more quickly. This could only be done by using the hardware and information resources of the installation more effectively. Accordingly, a special purpose control program was designed for the application.

The Solution: The Concurrent Processing of Transactions

The special purpose control program treated each transaction as a separate and distinct work unit and when necessary handled more than one transaction at a time. To do this, the control program had to keep track of hardware and information resources and allocate them as they were required to process each transaction. If the processing of one transaction was temporarily held up to gain access to a program or data in direct access storage, then resources were allocated to start processing a new transaction or to continue processing a transaction that had been started earlier. Thus, several transactions could be processed concurrently to keep pace with new transactions and respond to them within a reasonable period of time.

Each independent transaction required only a fraction of the available resources of the system. Therefore, by processing more than one transaction at a time, by keeping account of their status, and by allocating resources dynamically (as they were required) the control program could keep up with a heavy, fluctuating work load.

This increased the rate at which the computing system could process transactions without extending response time beyond a reasonable limit. This is shown in Figure 22, which compares the serial processing of transactions with the concurrent processing of the same transactions. Note that it takes more time (a fraction of a second or so) to process transaction A concurrently with B and C then to process A alone. However, when A, B and C are processed concurrently, the time required to respond to B and C is greatly reduced and the three transactions are completed in less time. In many online direct access applications it is not unusual for 20 or more transactions to be processed concurrently.

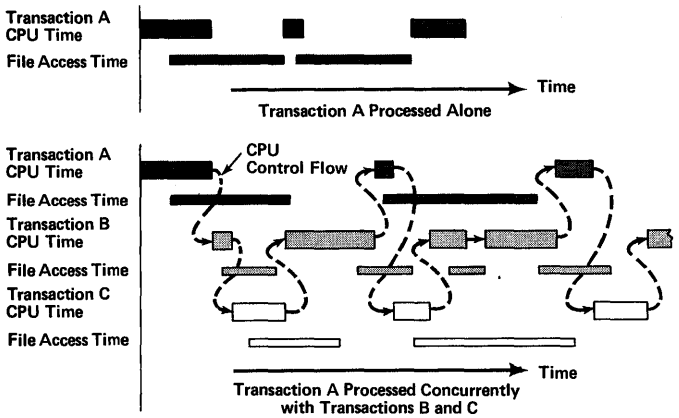


Figure 22. Comparison Between the Processing of Transactions Singly and Concurrently

CONCURRENT WORK TECHNIQUES AND OTHER APPLICATIONS

The technique of performing transactions concurrently made the differences between success and failure in online system applications. It increased the productivity of a commercial installation by making more effective use of its hardware and information resources in processing transactions. The processing of a transaction really represented a data processing task, that is, a definite unit of work performed by the CPU. If techniques could be developed for processing more than one transaction for online direct access applications, why could not similar techniques be used to perform transaction tasks, on other kinds of tasks, for other applications such as operating systems? If so, the productivity of many installations could be further increased. This possibility led to the third stage in the evolution of operating systems.

The Third Stage: A Union of Techniques

The third and current stage in the evolution of operating systems began with the realization that many of the techniques developed in the second stage could be improved and incorporated into a general purpose operating system. These techniques included the methods of designing programs, managing jobs, and managing data that had been developed for the early operating systems, and facilities for the concurrent performance of data processing tasks that had been developed for online direct access systems.

As a result of this union of techniques, the range of application and the overall productivity of operating systems was substantially increased. Operating systems could now be designed to serve a variety of advanced system applications as well as traditional accounting, record keeping and problem solving applications.

One of the most important objectives in designing such an operating system was the concurrent performance of data processing tasks. But two other objectives were of equal importance:

- To design a general purpose system that would satisfy the needs of a variety of users.
- To provide each user with a system tailored to his needs and capable of growth in performance, facility, and application without disruption.

These objectives led to the development of the IBM System/360 Operating System.

PART 2: THE IBM SYSTEM/360 OPERATING SYSTEM

This part describes the IBM System/360 Operating System: its design objectives, organization and function, control program configurations, task management, information management, program development and management, multiprocessing, and teleprocessing.

A General Purpose System

One of the most challenging objectives in designing the System/360 Operating System was to produce a general purpose system that could satisfy the data processing needs of the majority of users at medium and large scale System/360 installations. In the past, discrete systems had been designed to meet the needs of the "Typical User." But experience had shown that there was no such thing as a typical user. Data processing needs differed greatly from one installation to another, and between individual users within an installation.

Any attempt to satisfy such diverse needs with a single system would have been wasteful of resources. Many installations would end up paying a penalty, in storage space and other resources, for facilities they did not need, while other installations would suffer from a lack of needed facilities.

Therefore, instead of designing one or more discrete systems, IBM decided to allow each customer to generate the kind of operating system he required.

The first step in generating a system that satisfies the data processing needs of an installation is to determine exactly what those needs are. To do this a study is usually conducted to answer such questions as: What is the total volume of data processing work that must be done? What types of work must be performed? What is the relative priority of the different types of work? What kind of assistance does each member of the installation need to do his job?

After determining needs, the next step is to select the computing system (hardware) combination and operating system (software) combination that will best satisfy those needs. In selecting the appropriate hardware/software combination the main objective is to improve the productivity of the total installation.

Productivity depends on the performance and facility (ease of use and operation) of the total hardware/software system. These factors, combined with the skill of its members, determine whether an installation has the capability to do the work that must be done.

Investing Resources

The performance and facility, and therefore, the productivity of a total system depends on the hardware, software, and human resources invested in the system. In general, the greater the investment in resources the greater is the performance and facility of the system (Figure 23). However, performance involves other interrelated factors: throughput, response time (called turnaround time in batch job processing), and availability. For a given investment of resources, one factor can be improved only at the expense of others. At some installations, response time and availability are of prime importance; at others, throughput is of greater importance.

Therefore, the main objective in planning and selecting a system should be to invest resources in a way that properly balances these factors and gives the kind of performance and facility that the installation requires.

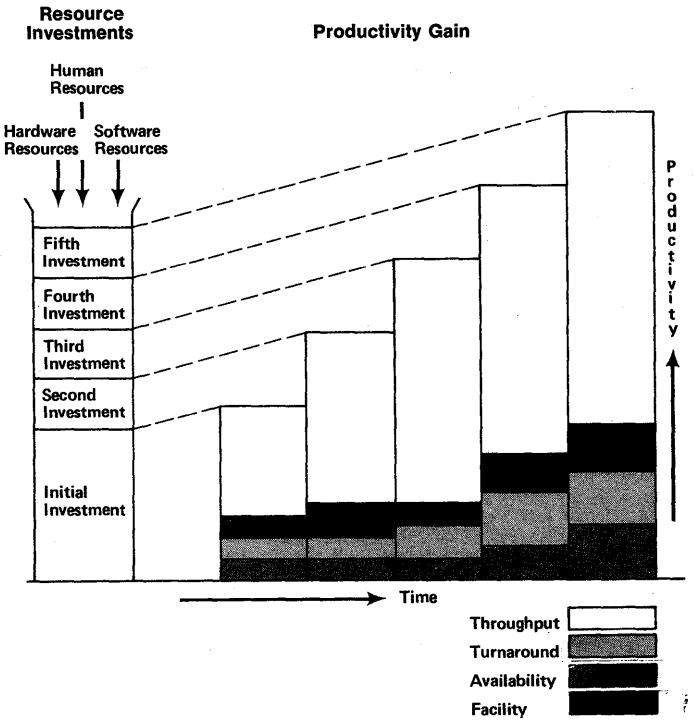


Figure 23. Investing Resources to Increase Productivity

Modular Construction

In order to allow each installation to select the resources it requires, the operating system was designed using a method called modular construction. The total system consists of a large number of parts, or modules, in the form of organized collections of instructions and data (Figure 24). These modules can be assembled and linked together in many combinations to form unique operating systems and can be replaced independently of one another. The really distinguishing characteristic of a module is the fact that it can be replaced independently of other modules. Some of the modules are required in every operating system; others are either optional or alternative. (An installation must select a module from a group of alternative modules but need not select a module from a group of optional modules.)

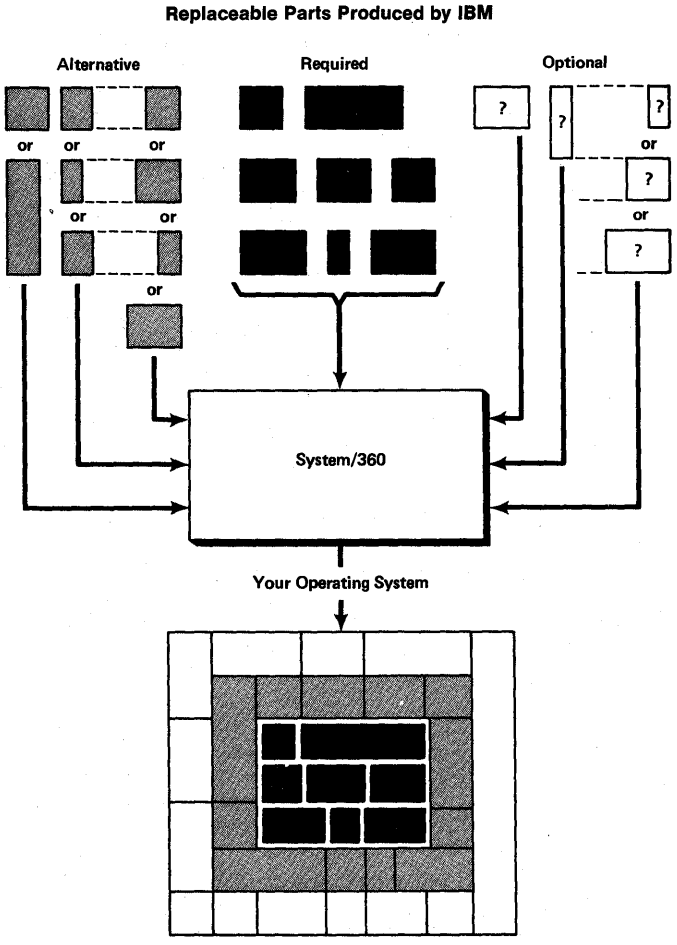


Figure 24. Constructing Your Operating System

The operating system is not assembled before delivery. Instead, IBM makes available to its customers all of the required, optional, and alternative modules it produces. From them, each customer selects and constructs the kind of operating system he requires. To meet the needs of a great many installations, IBM has made available many more facilities than any one installation would require. An installation should select only those facilities it really needs; the selection of unnecessary facilities can result in a costly waste of storage space, CPU time and other resources.

Defining and Generating the System

Once a hardware/software combination has been selected, the next step is to define the combination so that it can be generated automatically. The customer does this with an IBM-supplied system generation language with which he defines the optional and alternative components to be included in his system.[12] Using this definition and a program library of modules supplied by IBM, another IBM System/360 Operating System can automatically retrieve, assemble and link together all of the parts required for the specified system. If the installation does not already have an operating system, IBM supplies a pregenerated system. Although IBM tries to anticipate the needs of its customers, inevitably there are some parts of an operating system that a customer, because of his special needs, may wish to design and supply himself. These may be in the form of either replacements or additions to the system. Such replacements and additions can be integrated into the system when the system is generated. Some can even be incorporated into the system after it is generated. In some cases IBM has anticipated the need for specialized additions and extensions to the operating system by supplying programming aids such as system utility programs, which the system programming staff can use for this purpose.[15]

Tailoring the System to Individual and Daily Needs

Selecting and generating an operating system for an installation is just the first phase of a continual process of tailoring the system to changing needs. Such tailoring is made possible because of the flexibility inherent in the design of the system. However, this flexibility is not provided to allow every user complete freedom to use the system as he chooses. The installation staff must narrow the

choices that can be made by individual users so that the system does not seem overwhelming to those who use it. Flexibility is provided to allow the systems programming and operations staff to react to changing needs and to exercise control over the use and operation of the system. By so doing they can help to ensure that the hardware and software resources of the system are used efficiently, promote the sharing of data and program resources, avoid duplication of effort and help to simplify the use of the operating system. The system programming and operations staff can do this by:

- Selecting default options when the system is generated.
- Selecting options at the beginning of a work period (at initialization).
- Sharing data.
- Storing standard procedures and cataloging them for fast retrieval.
- Controlling the system during operation.

Selecting Default Options

Each processing program provides a number of optional services or features that a programmer can choose prior to using the program. However, when the operating system is generated, the installation staff can specify which options are to be used by default if the programmer does not make a positive choice. For example, at the option of the programmer a language translator may or may not provide a source program listing. At system generation, the installation staff can decide whether or not the listing will be supplied when no choice is made by the programmer. By specifying default options that will best serve the needs of the installation, the systems programming and operations staff can control and standardize the use of processing programs and limit the number of choices that have to be made.

Selecting Options When the Control Program is Initialized

When the operator initializes the control program at the beginning of a work period, he can elect that certain routines, including parts of the control program itself, remain in main storage throughout the work period. By making such choices, based on the type of work that will be performed during the work period, he can increase the performance of the system.

Sharing Data Sets

A system programmer can define and create data sets that can be used, without further definition, by several programmers. This not only saves storage space but saves time in updating duplicate data, and avoids duplication of effort on the part of the programmers.

Cataloging Procedures

When similar jobs are to be performed by several programmers, the system programmer can catalog and store in direct access storage a standard job definition that can later be retrieved and used by each programmer without completely redefining the job. This not only reduces duplication of effort but also helps to standardize the use of the system.

Controlling System Operation

Once the control program is initialized and the system is in operation, the operator can dynamically alter and adjust its operation, based on the flow and type of work and the availability of resources. For example, with certain configurations of the control program he can control the number and type of jobs that are performed concurrently, cancel jobs, and change the priority of jobs. In short, the operations staff has a great deal of high-level control over the operating characteristics of the system. This control can be used to achieve a high degree of productivity. However, to take full advantage of it requires an operations planning activity that analyzes and monitors work requirements and:

- Determines which operations-related options are to be selected when the system is generated.
- Plans which options are to be selected each time the control program is initialized.
- Plans the order in which the work is to be entered into the system.
- Plans the overall operation of the system.
- Establishes standard procedures to be followed by the operator when particular contingencies arise.
- Monitors and reviews the operation of the system to ensure that the resources of the system are being used most effectively.

Controlling the Use of the System

If a system is to achieve the production it is capable of, reasonable control must also be exercised over its use. This is important to the success of any high-production installation. It is doubly important at installations where information and hardware resources can be concurrently shared by more than one user, and where the operations staff has a great deal of flexibility in adjusting the operation of the system to the workload. At such installations, one user, in the course of pursuing his own interests, can adversely affect the interests of other users. For example, one user may assign priority to a job that does not require it, and unduly delay the completion of jobs that do. Another user may request more storage space than his job requires, and perhaps prevent other jobs from being performed concurrently with his job. To prevent individual users from unnecessarily diminishing overall productivity, an installation can:

- Price the use of resources in a way that promotes maximum sharing of resources.
- Establish rules and regulations on the use of the system, for example, rules for assigning priority to jobs or jobs to specific classes.
- Establish standard ways of using the system, appropriate to the types of applications at the installation.
- Check job control statements to ensure that installation rules and conventions are followed.

Effective control over the use and operation of the system/360 Operating System is particularly important because it provides an opportunity to greatly improve performance through efficient sharing of resources.

Growth Without Disruption

From its very beginning, the electronic data processing industry has been marked by extraordinary growth. One aspect of this growth has been an ever increasing number of possible data processing applications. The number and scope of possible applications have grown so rapidly that individual data processing installations have been unable to keep pace in implementing them. Growth at a data processing installation has often been an expensive and painful experience. Therefore, the second major objective in designing the System/360 Operating System was to provide easy installation with the ability to grow without disruption.

Growth in the Past

To date, growth at many data processing installations has been disruptive. If a significant advance in technology or an increase in the number and scope of data processing jobs forced a user to expand the production capacity of his installation, he often had but one choice. That was to replace his old system, or a large part of it, with a new one. Usually, this required that he re-program and modify many of his applications. Sometimes it even required that he reformat all or most of his data. In addition, it required some retraining of the programming and operations staff. Considering the great investment in program, data, and human resources at a typical installation, this kind of disruptive growth proved to be far too expensive. In fact, it sometimes took years for an installation to fully recover. To reduce the number of such painful experiences, many installations either delayed expansion or expanded to a capacity that was far beyond their immediate needs. As a result their growth pattern was somewhat like that shown in Figure 25: either far too much capacity or not enough.

Evolutionary Growth at an Installation

To avoid disruptive growth patterns, the operating system was designed using modular construction (described in the previous section). Modular construction allows a customer to generate a system from a combination of required, optional, and alternative modules and to replace or add modules when necessary. It is basically the same method that is used to construct a System/360 Computing System or System/370 Computing System from a wide selection of central processing units, storage devices, and I/O units. By selecting an appropriate combination of operating system and computing system options, a user can arrive at a software/hardware system whose performance matches the workload (number and scope of applications) at his particular installation. As the workload increases, performance and facility (productive capacity) can also be increased by adding or replacing computing system resources, selecting other operating system options, and using the operating system in different ways. This can be done in small incremental steps in order to keep pace with a steady increase in workload (Figure 26). Thus, periods of extreme underloading or overloading of a system can be avoided.

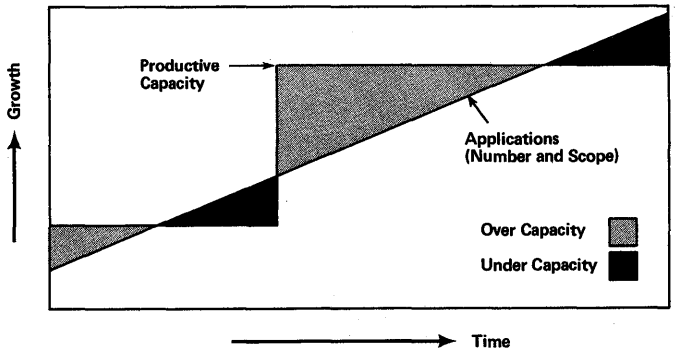


Figure 25. Disruptive Growth

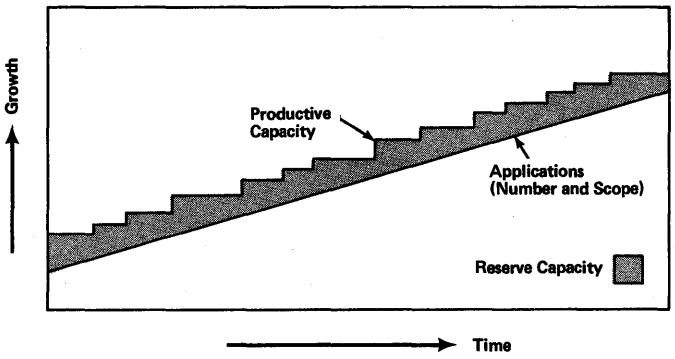


Figure 26. Evolutionary Growth by Incremental Steps

Evolutionary Growth in Improving the System

Modular construction not only makes it possible for an operating system to grow in small evolutionary steps, it also makes it possible for IBM to continue to improve the performance and extend the growth of the

system and pass on the benefits to individual installations. IBM can do this by replacing one module of the system with a new module that performs the same function more efficiently, or by adding new optional modules that an installation can select in order to advance to new applications or to increase overall productivity. Another way is to provide new modules that perform the same basic function but have different performance characteristics. An example of this would be two alternative modules (Figure 27) that perform the same basic function but differ in that one (A) is small but slow and the other (B) is fast but large. Another example would be two program translators, one of which translates programs quickly while the other produces highly efficient object programs. The former could be used for compiling short-running one-shot programs, while the latter could be used for long-running or frequently run programs. One or the other or both might be used at a particular installation. In such ways, the modular construction of the operating system enables IBM to make evolutionary improvements in the design of the system. This has a number of important advantages.

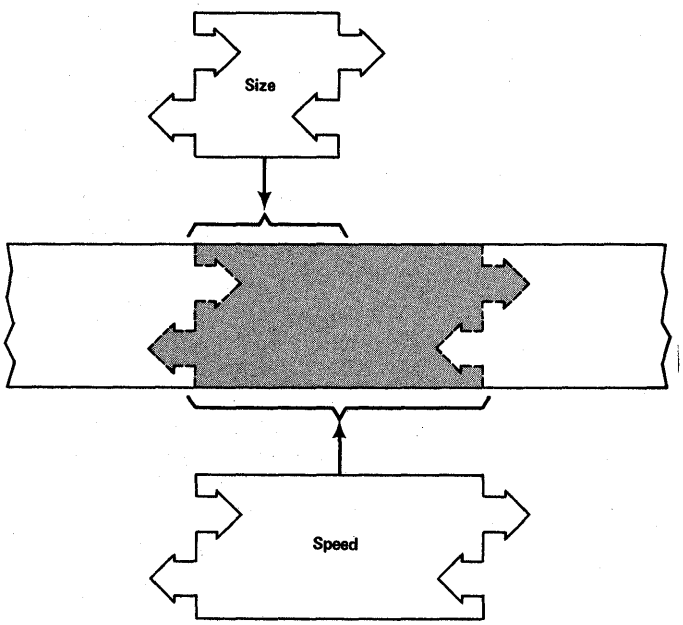


Figure 27. Optimizing Specific Characteristics: Size vs. Speed

GROWTH IN PERFORMANCE

Modular construction makes it possible to continually upgrade the overall performance and facility of the system as a whole. In fact, since the operating system was first introduced, it has gone through a succession of refinements and improvements that have significantly improved its performance and made it easier to use. This has had the general effect of raising the initial productive capacity of the system (Figure 28).

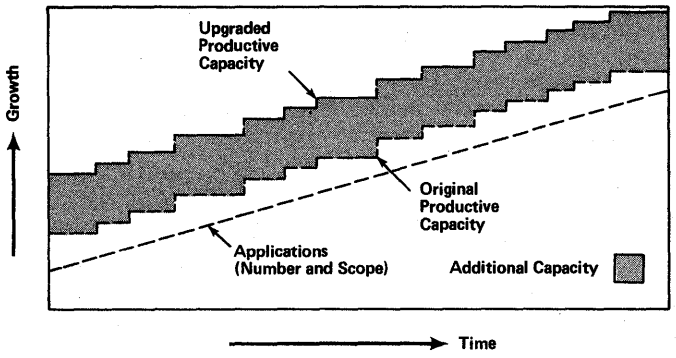


Figure 28. Raising the Initial Productive Capacity of the System Through Design Improvements

GROWTH IN APPLICATION

Modular construction has also made it possible for IBM to extend the productive capacity of the system (as shown in Figure 29), thereby, increasing the number and scope of possible new applications. In fact, the system has already been extended to encompass new system applications that use such techniques as multiprogramming and multiprocessing.

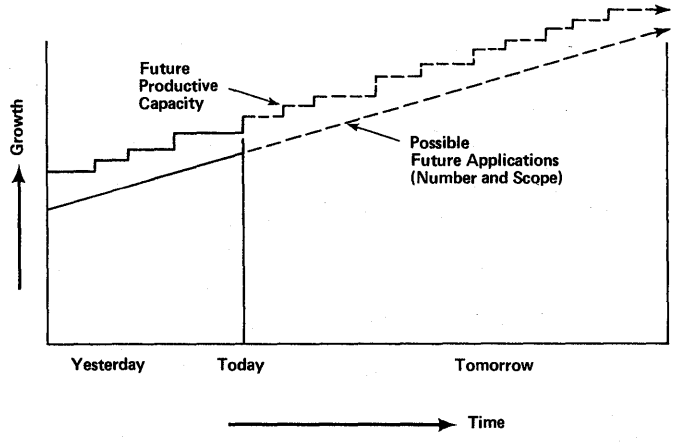


Figure 29. Extending the Production Capacity and Application of the System

TECHNOLOGICAL GROWTH

Because of their modular construction, both the operating system and the computing system can benefit from future developments and improvements in data processing

technology. By taking advantage of new data processing techniques and equipment, IBM, through evolutionary change, can improve the performance and application of the operating system. Over a period of time it can even change the basic design characteristics of the system, often in ways not even contemplated by the original designers.

Other Growth Factors

Growth without disruption is not achieved through modular construction alone, however. There are a number of other factors in the design of the operating system that make it possible for an installation to avoid the growth problems that have plagued the data processing industry in the past.

COMPATIBILITY

To ensure a smooth transition from one configuration to another, the operating system was designed with compatibility as a primary goal. Operating system processing programs (such as language translators and utility programs etc.) that are provided by IBM can be used with any operating system configuration. Also, jobs and programs designed to run under control of one configuration can run under control of a larger configuration.

For example, System/360 is program compatible with System/370 -- except for Models 20, 44, and 67. Programs used for System/360 can function on System/370, except those programs which:

- Use System/360 model dependent features.
- Use PSW bit 12 -- the ASCII bit.
- Use machine-dependent data.
- Use low-address main storage for special purposes.
- Depend on devices or facilities not available in System/370.
- Deliberately cause program exceptions.
- Are time dependent.

DEVICE INDEPENDENCE

Using the operating system, programs can be written in such a way that they are not directly tied to a particular I/O unit. This is an important factor in achieving growth without disruption. By keeping the programs independent of the devices, it is possible to add new I/O equipment without affecting existing programs that might use them. Also, IBM can extend or modify the operating system to handle new types of devices and make them available for immediate use with existing programs.

MULTIPLE TASK MANAGEMENT

Another factor that promotes growth without disruption is the ability of the control program to control the performance of more than one data processing task at a time.

A task is simply work to be accomplished. In System/360, the work to be accomplished is the processing of data by the CPU. To perform a data processing task, the CPU carries out (executes) a series of instructions that is variously called a program, subprogram, routine or subroutine. In short, a data processing task is the work performed by the CPU while executing a program or part of a program. It is an independent unit of work that can compete for the resources of the system.

Because of this ability, the hardware resources of the computing system can be expanded and used efficiently without reprogramming simply by increasing the number of tasks that are performed concurrently. For example, if more main storage space is added to a system, it can be readily used by increasing the number of jobs that are performed concurrently. In such multiple-task configurations, the basic mechanism for allocating and managing the concurrent use of resources already exists. Therefore, the system need not be redesigned or modified each time additional resources are added.

STANDARDS

Another, perhaps not so obvious, way in which the design of the operating system helps to ensure growth without disruption is by establishing standards that can be passed on to and used by customers. These standards ensure coordinated operation of the system. They include:

- Standard data formats.
- Standard data labels.
- Standard ways of linking programs.
- Standard ways of communicating from one program or routine to another.

By adopting such standards a customer can help to ensure that his data, programs, and methods are compatible with one another and with those of present and future versions of the operating system.

The General Organization and Function of Operating System/360

The System/360 Operating System consists of an organized collection of programs that communicate with one another in standard ways. The system is formed of two basic classes of programs: programs that are executed when the CPU is in the supervisor state (when I/O and other certain key instructions can be executed) and programs that are executed when the CPU is in the problem state.

Supervisor State Programs

The supervisor state programs are called, as a group, the system supervisor or supervisor. The supervisor is the service and control center of the operating system (Figure 30). Its primary function is to perform a variety of services requested by the problem state programs, such as allocating storage space, performing I/O operations, loading programs into main storage, and initiating the execution of programs. To perform its function, the supervisor always receives control of the CPU following an interruption of a problem state program. An interruption may result from a specific service request from a problem state program or it may be an automatic interruption initiated by the computing system.

Service Requests

A service is requested by a problem state program through the execution of a supervisor call (SVC) instruction. This results in an interruption of the requesting program and a transfer of control to the supervisor. Usually the request is accompanied by information that the supervisor requires to perform the service. For example, a request to load a program into main storage would be accompanied by the name of the program to be loaded.

Automatic Interruptions

An automatic interruption does not represent a specific request by a problem state program. Rather, it is initiated by the computing system. An automatic interruption results when a significant or unusual event occurs within the computing system, such as the completion of an I/O operation or the detection of an error. The computing system continuously monitors its own operation so that when an event that requires action by the supervisor occurs, the current program is interrupted and CPU control is passed to the supervisor. The interruption network that is built into the computing system relieves the problem state program or the supervisor from continually checking to determine if a significant event has occurred or from wasting time in idly waiting for an event to occur.

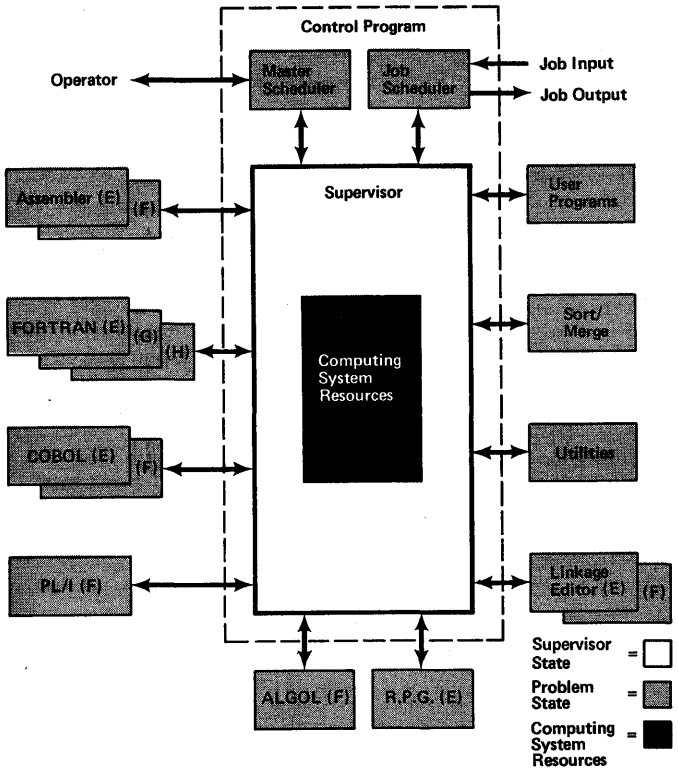


Figure 30. Operating System/360 Supervisor and Problem State Programs

The Effect of an Interruption

An interruption to a problem state program has the effect of placing the CPU in the supervisor state, transferring control of the CPU to the supervisor, and passing on to the supervisor information (in the form of a program status word) indicating the cause of the interruption and the status of the program when it was interrupted. Once the supervisor receives control, it can perform tests to determine exactly what event caused the interruption and then take appropriate action. When the supervisor has completed its action, it can return

control to a previously interrupted program, using the program's status word.

An interruption of a problem state program may be caused by an event that is entirely unrelated to the program. For example, an interruption may result from the completion of an I/O operation for another program. In basic versions of the operating system, the supervisor always returns control of the CPU to the same problem state program that was being executed when the interruption occurred. In versions of the operating system that are designed to control the performance of more than one data processing task at a time, the supervisor may return control to the same or to a different problem state program.[8]

Privileged Instructions

When the supervisor receives control of the CPU as a result of an interruption, it can have the CPU execute certain key instructions, such as I/O and storage protection instructions. These can be executed only when the supervisor is in control, and are, therefore, called privileged instructions. If privileged instructions were executed as part of a problem state program, they could interfere with the execution of another problem state program or with the execution of the supervisor itself. For this reason, these instructions cause an error condition when an attempt is made to execute them in a problem state program.

The Basis of Control

The interruption network and the privileged instructions constitute the basic mechanism that enables the supervisor to service requests by the problem state programs and maintain dynamic control over the performance of work by the computing system. In the course of executing problem state programs, control of the CPU is continually passed back and forth between the programs and the supervisor as the supervisor answers requests for services and responds to events detected by the computing system.

During the intervals when it is in control of the CPU, the supervisor maintains complete control over the allocation and use of the resources of the hardware/software system. To do this, the supervisor keeps a running account of all the programs, data and computing system resources in the system, and their exact status. It also keeps a running account of its own continually changing status. It can thereby maintain continuous control over the activities of the system and prevent one program from interfering with another. By maintaining up-to-date records of everything that happens, the supervisor can coordinate its supervision of the system even though it receives control of the CPU intermittently. The interruption network ensures that the supervisor receives control when necessary and the use of the privileged instructions enables it to prevent interprogram interference.

Key parts of the supervisor always reside in protected main storage (optionally protected for MFT).[8] Some of these, such as the supervisor routines that initiate and control I/O operations, reside permanently in main storage primarily to ensure continuous and coordinated control over the operation of the system. Other parts of the supervisor, such as the supervisor routines that control the allocation of main storage space, reside permanently in main storage because they are used frequently. Supervisor routines that are less frequently used, and whose immediate presence in main storage is not vital to the efficient operation of the system, are usually brought into main storage from a direct access storage device only when they are required to perform specific functions.

The supervisor plays a central and indispensable role as part of the operating system. Therefore, it is more fully described in the sections "Task Management" and "Information Management."

Problem State Programs

The problem state programs that are serviced by the supervisor can be classified as either IBM programs or user (application) programs. Operationally, the IBM and user programs bear the same relationship to one another and to the supervisor. They adhere to established linkage conventions and data formats, and communicate with the supervisor in the same way. All of the services provided by the supervisor are equally available to them. As a group, the only real difference between the IBM-designed problem state programs and the user-designed programs is that the IBM programs are normally designed for general use (in preparing other programs), while the user programs usually are not. In general, the IBM programs are designed to assist:

- Applications programmmers in devising and programming new applications.
- System programmers in generating and maintaining the system and in extending and controlling its use.

- The operations staff in scheduling work and operating the system.

User-designed programs are usually intended for specific applications, such as a payroll application, and are normally prepared and scheduled for execution using the IBM-supplied programs.

The IBM-supplied problem state programs provide the means by which programmers can use the supervisor's services.[9] In addition, they assist the programmers by performing any one of a combination of three main functions: translating language, supplying precoded instruction sequences, and performing specific services. They perform these functions in response to sequences of coded language statements that are written by the programmer. Each program responds to its own combination of statements which together form a language. Using these statements in various combinations and forms, a programmer can communicate his data processing requirements, including requests for services performed by the supervisor.

IBM-supplied programs differ from one another in the type of assistance each provides. Some programs assist the user chiefly by performing specific services. For example, the linkage editor program is used chiefly to integrate individually translated parts or sections of a program. Most of the programs, however, assist the programmer through a combination of translating, providing precoded routines, and performing specific services. A major function of a FORTRAN compiler, for example, is to translate from a notation that is similar to mathematical notation to a form of machine language notation. However, it also provides precoded mathematical subroutines and performs specific services, such as converting data from one form to another. Two of the problem state programs supplied by IBM -- the master scheduler and the job scheduler -- are, like the supervisor, required parts of the system. The others, except for a few system utility programs, are optional. However, the linkage editor and one or more other IBM-supplied programs would normally be included in any system.

The Master and Job Schedulers

The master and job schedulers, together with the supervisor, make up the control program, which to a large extent determines the basic nature of each operating system configuration. The master scheduler controls the overall operation of the computing system-operating system combination. The job scheduler enters work (job) definitions into the computing system, schedules, and then initiates the performance of work under control of the supervisor. In operation, the two differ slightly from other problem state programs in that they can read data from and write data into storage areas that are assigned for use by the supervisor. Because they use the same storage area, the job and master schedulers can quickly pass information to the supervisor concerning the work it is to supervise. In conjunction with the supervisor, the job and master schedulers perform a vital role in scheduling and supervising the performance of work by the computing system. Therefore, they are more fully described in the section "Job Management."

IBM-Supplied Processing Programs

The problem state programs that are provided with the operating system (other than the master and job schedulers) are generally called processing programs to distinguish them from the more special purpose programs provided by the users of the system. Any of the processing programs can be selected and used with any operating system configuration.

A wide selection of IBM-supplied processing programs is available for inclusion in the operating system. These may be supplemented in the future by others supplied by users of the system or by IBM. The processing programs are designed to reduce the time, training, expense, and manpower required to design and code efficient problem-state programs. A programmer, or group of programmers, may use them singly or in combination to process a particular job. They are generally classified as either language translators or service programs although some may both translate and perform specific services, and often supply precoded routines as well.

The language translators and service programs described below are designed for use in combination with one another and with other parts of the operating system. Therefore, the role of each as part of the overall system is described in other sections of the book. In particular, the role of the language translators and the linkage editor is more fully described in the section "Program Development and Management."

LANGUAGE TRANSLATORS

The language translators enable a programmer to define a problem solution or an application in a language that can be

more readily learned and more easily used than the machine language of the computing system. They relieve the programmer from much of detailed work and drudgery involved in programming, and thereby reduce the time required to produce an error-free program. IBM provides translators for six languages. These may be used to define a problem solution or application:

- In a form of mathematical notation (FORTRAN and ALGOL).
- In a concise form of the English language (COBOL).
- In a new programming language (PL/I) having features of both FORTRAN and COBOL as well as new features.
- In a flexible and versatile symbolic language (assembler language).
- In a tabular form (Report Program Generator).
- In a combination of any of the above forms.

For the assembler, FORTRAN and COBOL languages more than one translator is provided (Figure 31). The letters E, F, G, or H in the figure indicate the minimum amount of main storage space that must be available in order to use a translator under operating system control. For example, the assembler language translator F requires a computing system with 65,536 or more bytes of main storage. If more than the minimum main storage space is available, a translator can generally use the additional space to advantage. Translators of a given type may differ in performance characteristics, in the facilities they offer the programmer, or both.

Yes = ■ No = □	Translators			
	E	F	G	H
Assembler	■	■		
FORTRAN	■		■	■
COBOL	■		■	
PL/I		■		
R.P.G.	■			
ALGOL		■		

E = 32,768 bytes
F = 65,536 bytes*
G = 131,072 bytes
H = 262,144 bytes

Figure 31. Language Translators Provided by IBM

Using a particular language, a programmer can avail himself of services provided by the supervisor. However, the full range of services is not available with every language. For example, when using the assembler language, a programmer can program the concurrent performance of multiple data processing tasks; but the same ability is not available when using FORTRAN. This is due in part to the fact that the FORTRAN language evolved before the operating system came into being.

The language translators (and the service programs) are basically no different from other problem state programs. Therefore, the full range of supervisor services are available for their use in translating (or servicing) other programs. Each of the language translators produces object programs (or object modules) in a standard format. With certain exceptions and restrictions, this enables the linkage editor to combine portions of a program written in one language with portions written in another language to form a single program that is ready to be loaded into main storage and executed. Some of the language translators have program testing facilities that can be used to dynamically test a program, or part of a program, in accordance with specifications expressed in the source language. The general characteristics of each language translator are briefly described below. A complete description can be found in the manuals that are provided for each language. (Refer to the IBM System/360 Bibliography, GA22-6822.)

Assemblers

Either an E or F assembler, or both, can be included in an operating system for assembling object programs from source programs written in the assembler language. The major difference between the E and F assemblers is in performance. The F assembler is faster, but requires more main storage space than the E version. The assembler language is an extremely versatile language that can be used to program any type of application. All of the services provided by the control programs are available when using it.

Of the several languages available with the operating system, the assembler language is the one closest to the machine language of the computing system. Each assembler language statement represents either a single machine instruction or a request to the assembler to perform a specific service.

The assembler language can be extended by using the assembler to define for a particular service both a sequence of assembler language statements and a corresponding language statement called a

macro instruction (Figure 32). Once defined, the macro instruction alone can be used to request the service; the macro instruction calls for the execution of the sequence to perform the specific function. This represents an important feature of the assembler that is used extensively.

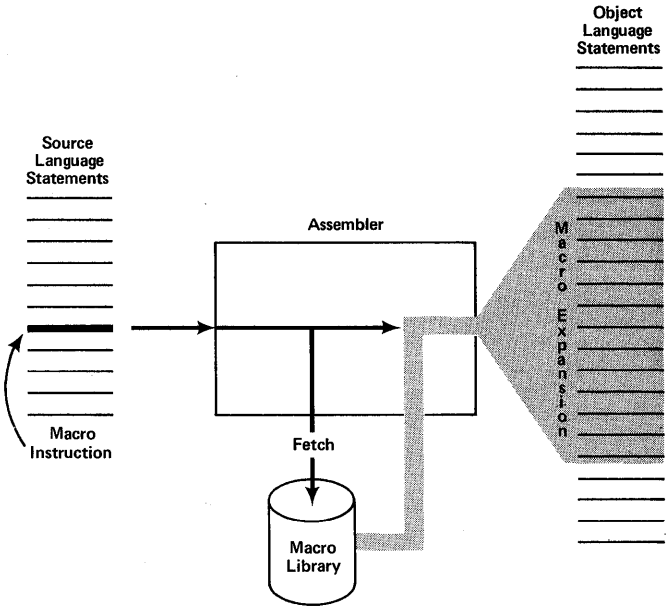


Figure 32. Macro Instruction Expansion

If a macro instruction is to be used frequently, then the instruction sequence it calls can be stored in a macro library associated with the assembler.

The assembler language has been extended by IBM to enable programmers to more easily communicate service requests to the supervisor and to communicate with input/output access method routines. Comprehensive sets of macro instructions are provided for use in the management of data and the sequential or concurrent performance of tasks. The assembler language has also been extended to include macro instructions that, in combination, represent a highly specialized language. For example, a set of macro instructions, called TESTRAN, is provided for use in testing assembler programs. Other sets of macro instructions are provided for designing teleprocessing control programs, for designing programs that display information on a graphic display device, and for generating the operating system itself. Users of the operating system can employ the assembler to create new macro instructions and add them to the complement provided by IBM. In addition, they can, like IBM, use the assembler to create special-purpose languages. These can be designed to meet the specific needs of specialists or professionals within a business or scientific activity. For example, a special-purpose language could be designed for use by civil engineers.[1]

FORTRAN Compilers

Any combination of three compilers (E, G, and H) can be included in the operating system for compiling object programs from source programs written in the FORTRAN Language. The FORTRAN language is a widely used language, developed and refined over a period of years through the combined efforts of IBM, its customers, and the American Standards Association. It closely resembles the language of mathematics, and enables engineers and scientists to define problem solutions in a familiar easy-to-use notation.

The language of the FORTRAN G and H compilers is an extended version of the FORTRAN language as defined by the American Standards Association. The language of the E compiler is an extended version of the Basic FORTRAN Language, also defined by the American Standards Association.

A library of often-used subprograms is provided with each compiler. A program user can specify these subprograms, thereby causing them to be incorporated as part of the program. If both a G and H compiler are included in the system, the two can share a single library of subprograms. The E compiler uses a separate, somewhat smaller library of subprograms. The libraries contain subprograms for performing common mathematical calculations.

As an optional feature, the libraries can also contain subprograms that can be used to transmit data to and receive data from an IBM 1130 computing system at a remote location. This is described more fully in the section "Teleprocessing."

The FORTRAN libraries can also (as an optional feature) contain a number of graphic display subprograms. Using them, a FORTRAN programmer can create programs that display graphic information on one or more IBM 2250 Display Units (Figure 33), an I/O device containing a TV-like display screen and a keyboard. The displays consist of charts, graphs, drawings, and other figures that can be formed from a combination of dots, lines and characters. By using the display subprograms, two-way communication can be established between a FORTRAN-compiled program and an operator at the 2250 Display Unit.

The G Compiler is faster than either the E or H compiler. On the other hand, the H compiler normally produces more efficient

Figure 33. IBM 2250 Display Unit Model 3

infrequently run programs. Although the E compiler is slower and produces less efficient object code than the others, it requires less main storage space.

COBOL Compilers

A COBOL E compiler and an American National Standard COBOL compiler (formerly USAS COBOL) can be included in the operating system for compiling object programs from source programs written in the COBOL Language.

COBOL is a language based on a well-defined restricted form of English. It provides a convenient method of designing programs for commercial data processing applications. COBOL was developed as a cooperative effort by a number of computer manufacturers and users. The USA standard of the language is American National Standard COBOL, X3.23-1968, as approved by the American National Standards Institute. American National Standard COBOL is a compatible subset of CODASYL COBOL, which is the complete definition of the language, as approved by CODASYL (the Conference on Data Systems Language).

An IBM American National Standard COBOL compiler -- which is compatible with the highest level of the USA standard, and which contains a number of IBM extensions to that standard -- can be included in the operating system for compiling object programs from source programs written in the COBOL language. IBM American National Standard COBOL contains many new features not found in previous implementations of COBOL. The COBOL E compiler also can be included in the operating system.

A library of subprograms is provided with each COBOL compiler. The library for IBM American National Standard COBOL can contain graphic display subprograms like those provided for the FORTRAN compilers.

ALGOL Compiler

An ALGOL F compiler can be included in the operating system for compiling object programs from source programs written in the ALGOL language. The ALGOL language is an international algorithmic language used mainly in programming the solution to scientific and technical problems. It is more widely used in Europe than in the United States, and is not as widely used as either FORTRAN or COBOL.

PL/I Compiler

A PL/I F compiler can be included in the operating system for compiling object programs from source programs written in Programming Language I (PL/I). This language incorporates some of the best features of other high level languages as well as a great many new features.

PL/I takes advantage of recent developments in computing system and programming technology. It provides the programmer with an "application-oriented" language for efficiently programming either scientific or commercial applications. It is particularly useful for the increasing number of applications that can best be programmed using a combination of scientific and commercial techniques. These include many of the new systems applications, such as management information systems and command and control systems. The modern features of PL/I enable it to be used for many programming applications for which other compiler languages either cannot be used or can be used only with difficulty.

A library of subprograms is provided with the PL/I compiler. The library can contain graphic display subprograms like those provided for the FORTRAN compilers.

Report Program Generator

The Report Program Generator (RPG) provides the programmer with an efficient, easy-to-use facility for generating object programs that are used to produce reports from existing sets of data. The reports may range from a simple listing of information from a punched card deck to a precisely arranged and edited tabulation of calculated data from several input sources. Several reports can be created concurrently from a single set of data.

SERVICE PROGRAMS

Service programs assist a programmer by providing routines for performing frequently used operations such as editing, linking, and otherwise manipulating programs and data. The service programs consist of linkage editors, a loader, a sort/merge program, a set of utility programs, emulator programs, and a set of graphic programming services.

Linkage Editors

An E and an F linkage editor are provided for combining program segments that were individually compiled or assembled. The linkage editor forms a single program that is ready to be loaded (by program fetch) into main storage and executed. The linkage editor enables changes to be made in a program without recompiling (or reassembling) the complete program; only those sections that are changed need to be recompiled. It also permits division of a program that is too large for the space available in main storage, so that executed segments of the program can be overlaid by segments yet to be executed.

The F linkage editor requires more storage space but is faster than the E linkage editor. It can also handle a more complex overlay structure of program segments. Otherwise, the two are much the same.

Loader

The loader combines the basic editing functions of the linkage editor and the loading function of program fetch in one job step. It loads object modules produced by language translators and load modules produced by linkage editor into main storage for execution. It is designed for high performance loading of modules that do not require the special facilities of the linkage editor and program fetch. The loader does not produce load modules for program libraries.

Sort/Merge Program

The sort/merge program is a generalized program that can be used to sort or merge fixed- or variable-length records in ascending or descending order. The sorting and merging can be performed using magnetic tape and direct access storage devices for input, output, and intermediate storage. The program takes full advantage of the I/O resources that are allocated to it by the control program. The sort/merge program can be used independently of other programs, or it may be used directly by programs compiled by the American National Standard COBOL and PL/I compilers.

Utility Programs

The utility programs provided with the operating system, are divided into three subsets:

- Data set utility programs.
- System utility programs.
- Independent utility programs.

Data Set Utility Programs: These programs are used chiefly by the programmer and operator to:

- Transfer, copy, or merge sets of data from one storage medium or I/O device onto another (sometimes in the process, editing the data or changing its format).
- Edit, rearrange, and update programs and data.
- Compare, print, or punch data.
- Create an input stream.

System Utility Programs: The system utility programs are used chiefly by the system programmer to:

- Change or extend the indexing structure of the system library catalog.
- Print an inventory of the data and programs that are cataloged in the system library.

Independent Utility Programs: The independent utility programs are used with the operating system, but are not an integral part of the system. They are used chiefly by the system programmer to prepare direct access storage devices for use under operating system control.

Emulator Programs

An integrated emulator program, used in conjunction with a compatibility feature, allows object programs written for one system to be executed on another system with little or no reprogramming. The compatibility feature consists of hardware and microprogrammed routines that aid emulation. The emulator program is executed as a problem program under the operating system control program.

Refer to the IBM System/360 Bibliography, GC28-6822, for the order number of the publication describing your integrated emulator.

Graphic Programming Services

A number of services are provided with the operating system for designing and executing programs that communicate with a user at an IBM 2250 Display Unit (Figure 33), or an IBM 2260 Display Station (Figure 34). With these services a program can retrieve information from storage, display the information on the face of a TV-like screen, check the information for accuracy, modify it at the display screen, and return it to storage.

The graphic programming services include:

- An extensive set of graphic design macro instructions.
- Processing routines.
- Data manipulation aids.
- I/O interruption analysis and control routines.
- Error recovery and diagnostic routines.

These services can be used to simplify the job of designing advanced applications in the fields of science, engineering, and business.

Program Products

IBM Program Products provide the user with specialized task-oriented functions. Program Products are designed to operate with other IBM programs.

Some typical IBM Program Products would include:

- Language processors
- Sorts
- Conversion aid programs
- General purpose utilities
- Industry application programs
- General application programs

Program Products are available from IBM for a license fee.

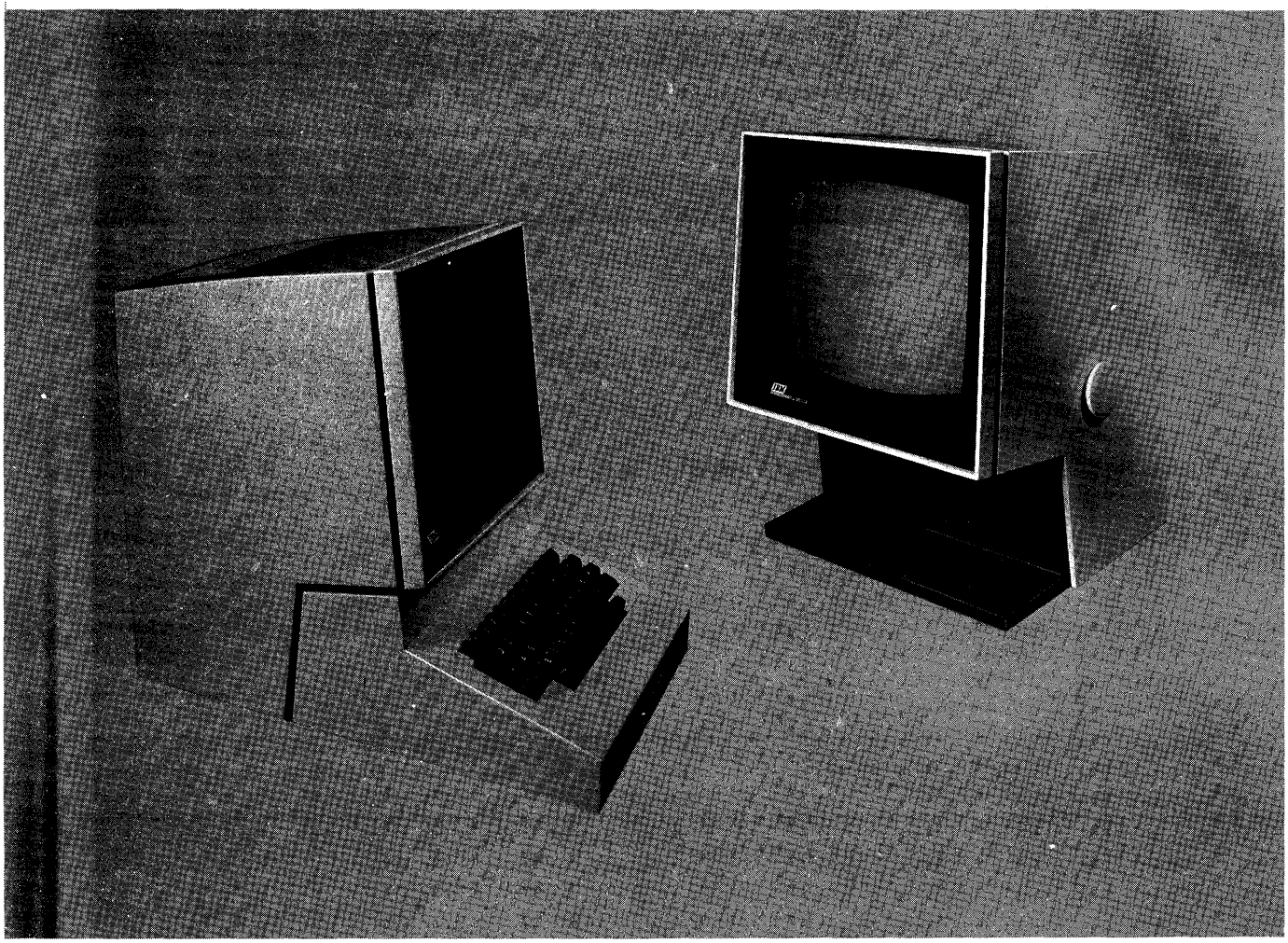


Figure 34. IBM 2260 Display Station, With and Without Alphameric Keyboard

Control Program Configurations

An IBM System/360 Operating System consists of a control program (a supervisor, master scheduler, and job scheduler) together with a number of optional processing programs such as the language translators, utility programs, and sort/merge programs described in the previous section. The processing programs are designed to help the user program solutions to problems and design new applications. They do this by giving the programmer a combination of programming aids, services, and precoded routines that he can use with appropriate language statements. Although the control program also assists the user, its primary functions are to efficiently schedule, initiate, and supervise the work performed by the computing system.

There are two configurations of the control program:

- The multiprogramming with a fixed number of tasks (MFT) configuration.
- The multiprogramming with a variable number of tasks (MVT) configuration.

Each configuration is designed to be used with a particular range of computing system models and main storage sizes. Figure 35 shows the control program configurations that can be used with various CPU models and main storage sizes. For comparison, the figure includes the other System/360 software systems: Basic Programming Support (BPS), Basic Operating System (BOS), Tape Operating System (TOS), and Disk Operating System (DOS). The storage sizes do not include IBM 2361 Core Storage, which can add up to 8,192K bytes to the main storage capacity of a Model 50, 65, or 75 in blocks of 1,024K or 2,048K.

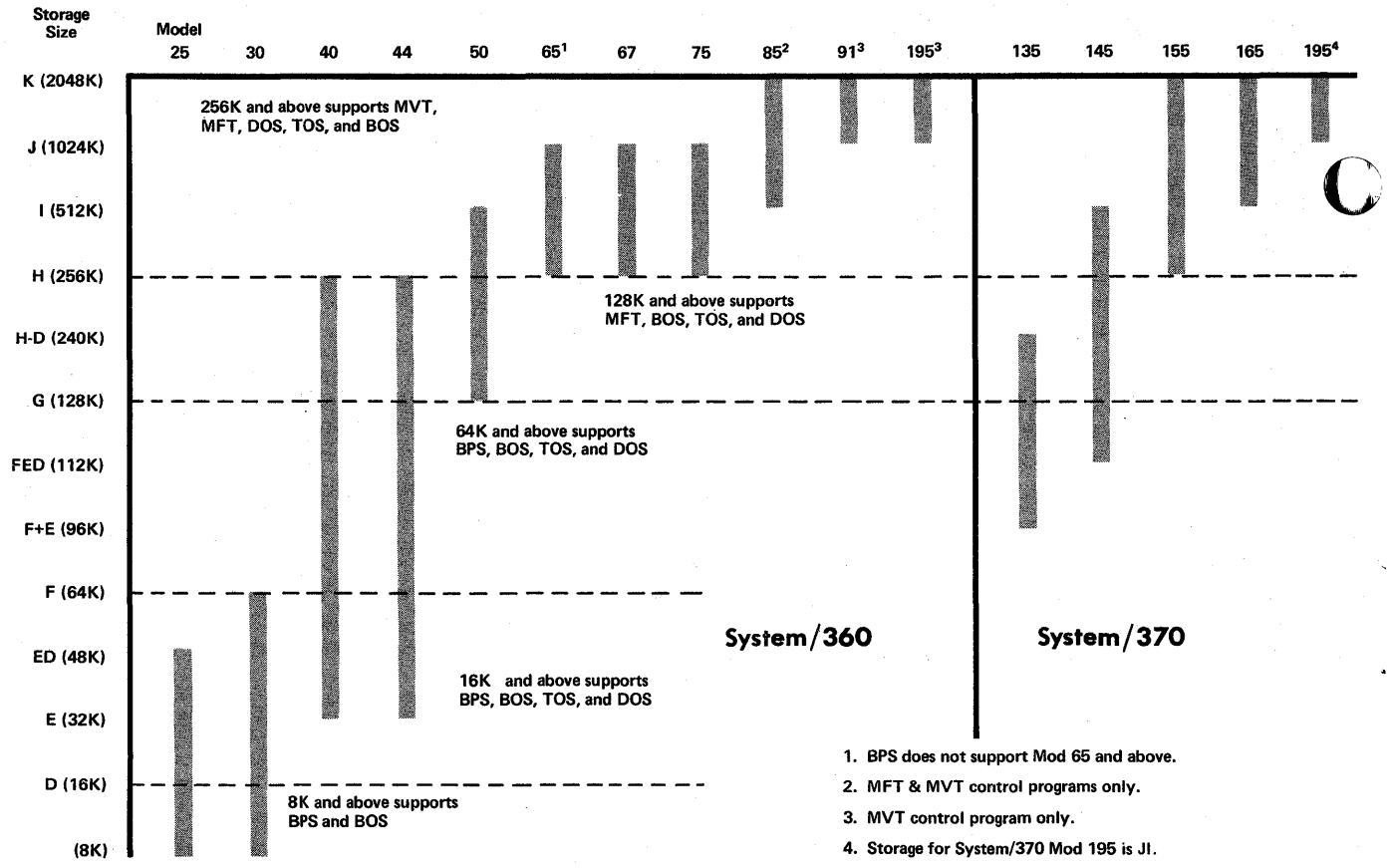


Figure 35. System/360 Software Systems For Various System/360 and System/370 Configurations

All of the IBM-supplied processing programs described in the previous section can be executed under the direction and control of any of the two operating system control program configurations. In addition, any problem state program produced using the processing programs are compatible with any of the two configurations. Furthermore, any jobs or data that can be processed by one configuration can be processed by another, provided the required hardware and software resources are available (Figure 36). Although many of their elements are identical and their general organizations and functions are the same, the significant differences between one operating system control program configuration and another has to do with the way each operates internally. The main difference is the number and types of data processing tasks they can perform at one time. The MVT configuration can control the concurrent performance of a greater number and variety of tasks. It can thereby keep more of the total software/hardware system in productive operation more of the time and significantly increase the volume of work performed by the system over a given period of time.

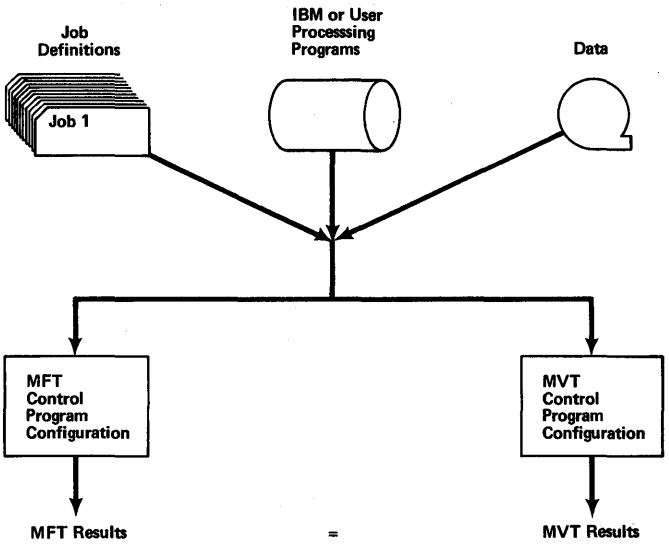


Figure 36. Compatibility of MFT and MVT Control Program Configurations

MFT CONTROL PROGRAM

The MFT control program configuration can control the performance of more than one task at a time. As its name implies, however, it can control a fixed number of tasks concurrently. An MFT control program can read a continuous stream of job or job steps in sequential order. However, it can read jobs from up to three such streams concurrently. Moreover, it can dynamically schedule and initiate the performance of each job based on an assigned priority and class. (To balance the operation of the computing system, jobs can be classed according to the resources they use; for example, whether they are primarily dependent on I/O or CPU time.) Once initiated by the control program, up to 15 job steps, representing the steps of 15 different jobs, can be performed concurrently.

The control program can also concurrently record up to 36 streams of job output data. The reading of job definitions from three input streams, the performance of 15 steps of different jobs, and the recording of 36 job output data streams can all be performed concurrently (Figure 37), provided the total does not exceed 52, and enough computing system resources are available. Although there is a limit to the number of concurrent data processing tasks an MFT control program can handle, from a practical viewpoint, the real limitation would more than likely be the availability of resources to perform the tasks. Moreover, there is an optional feature of the MFT control program that permits each job step to create an unlimited number of additional tasks. This feature is called "MFT with subtasking." Tasks created by a job step are called subtasks and are performed concurrently with each other and with other tasks in the system. Subtasks are dependent on the job step task and must be completed before the end of the job step.

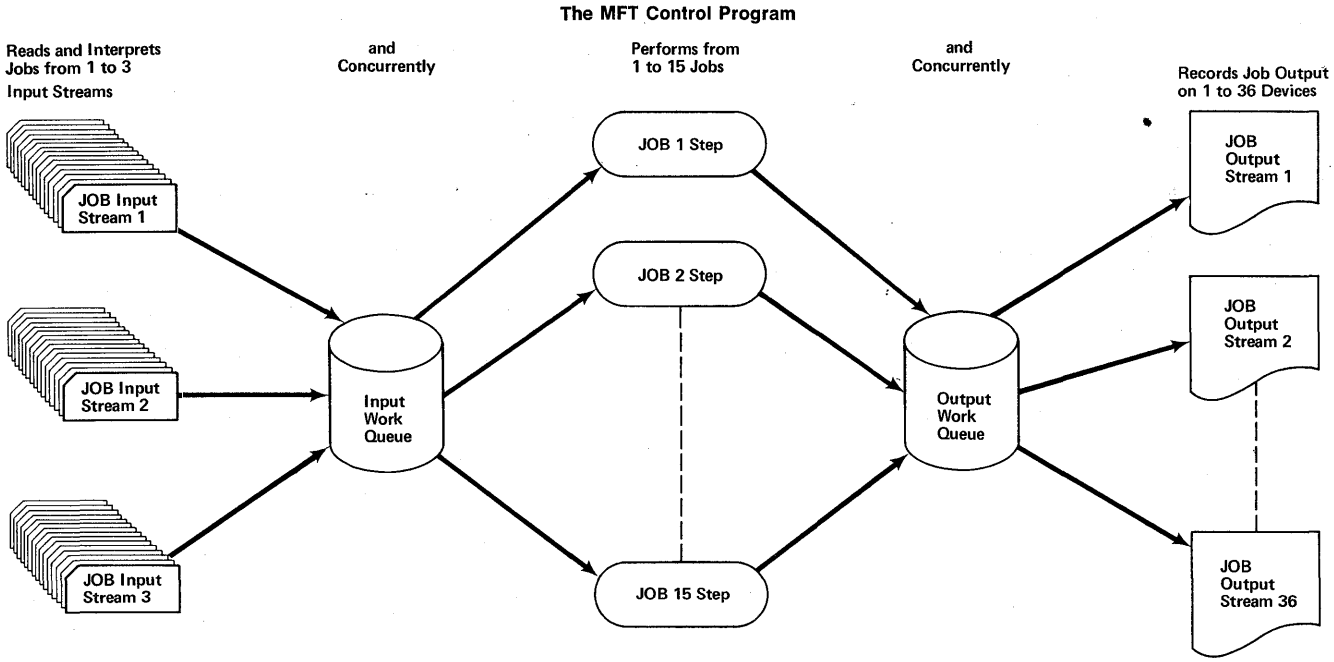


Figure 37. Concurrent Processing of Job Steps and Job Support Tasks by an MFT Control Program

In an MFT configuration, areas of main storage are reserved for the control program, and optionally, for the loader and user-written routines that can be used concurrently by the control program and by any tasks that are being performed. The remainder of main storage is divided into partitions (Figure 38). The size of each partition is set by the operator, and its priority is determined by its position relative to other partitions. Thus, partition P0 is reserved for the highest priority jobs, and Pn the lowest. Once defined, a main storage partition may be assigned by the operator for use in performing from one to three classes of jobs out of a maximum of 15 job classes. When a job is initiated, it is dynamically allocated a particular partition of main storage space depending on the class assigned to the partition and the specific class (and priority within the class) of the job. Other partitions may be assigned by the operator for use in reading and interpreting input streams of job definitions and for recording streams of job output data. When the control program is initialized at the beginning of a work period, the operator can include additional input/output access method routines by loading a secondary nucleus containing those routines. If included, these routines can be used in performing a single job step, or more than one job step concurrently. In an MFT control program configuration, program resources (in the form of access method routines) and data resources, as well as the hardware resources of the computing system, can be shared among concurrent jobs.

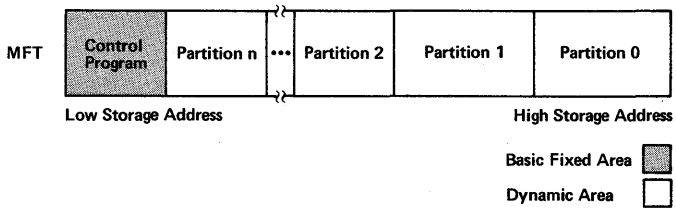


Figure 38. General Organization of Main Storage For the MFT Control Program Configuration

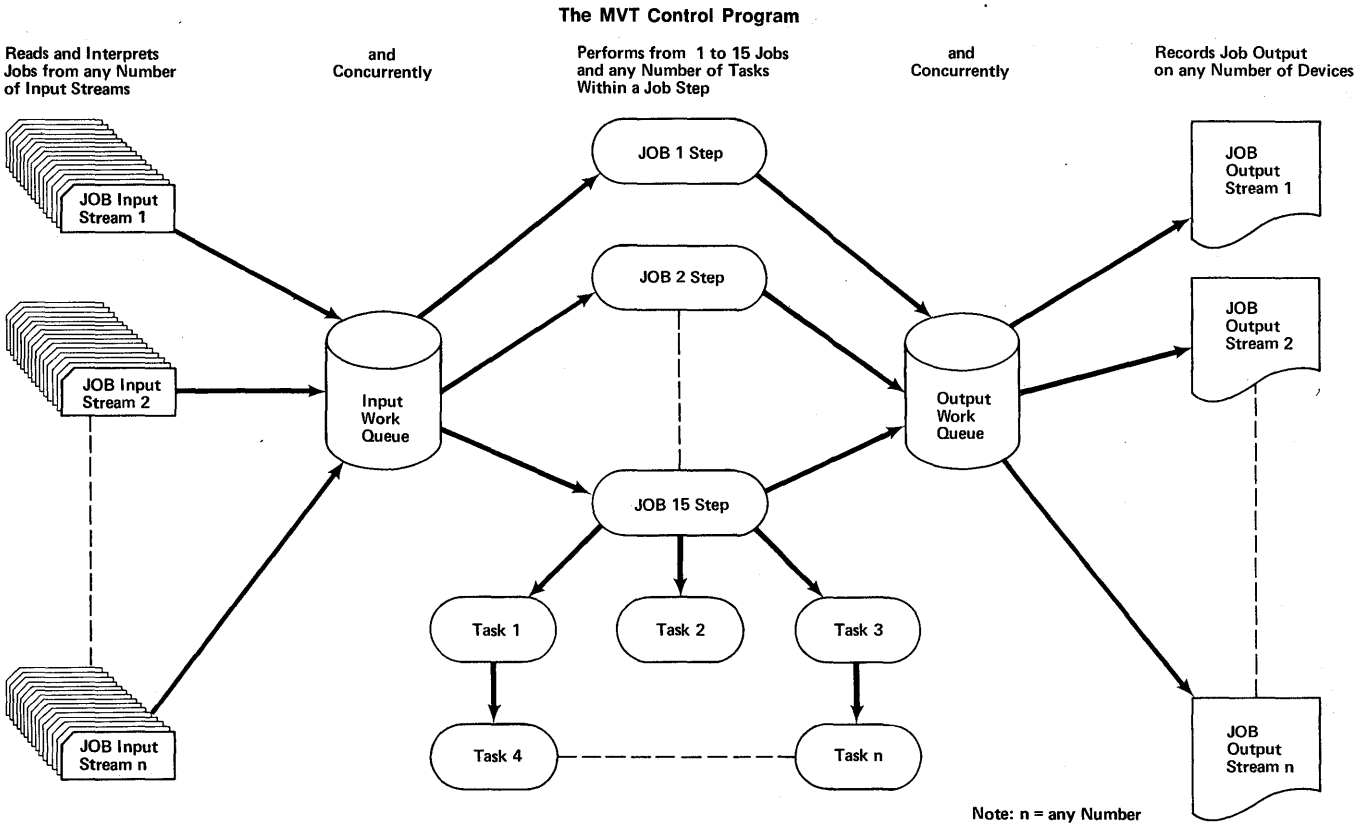


Figure 39. Concurrent Processing, by an MVT Control Program, of Job Steps, Job Support Tasks, and Tasks Within Job Steps

MVT CONTROL PROGRAM

The MVT control program configuration reads one or more continuous streams of jobs, and schedules the jobs in order of priority. With this configuration, up to 15 independent jobs can be performed concurrently. The job steps within a single job are necessarily performed in sequential order because one step may depend on the completion of another. However, within a job step, any number and type of data processing tasks can be initiated (Figure 39). These tasks are performed concurrently with one another, with tasks initiated by other jobs, and with tasks initiated by the control program and by the operator. Operator initiated tasks include job-support tasks for reading any number of job input streams and for recording any number of streams of job output data. The number of concurrent data processing tasks an MVT configuration can handle is limited solely by the availability of the resources that would be required to perform them.

In an MVT configuration -- in addition to areas of main storage reserved for the exclusive use of the control program -- an area of main storage (called the link pack area) is reserved for program routines that can be used concurrently by the control program and by any jobs that are being performed (Figure 40). These include access method routines as well as other routines designed by IBM or by a user of the system. The remaining storage serves as a pool of storage from which the controlprogram assigns a subpool (or region as it is usually called) to each job step as it is initiated. Once a job step, or a task within a job step, is initiated, it can draw upon and release storage space within its assigned region. Upon 40completion of a job step, the region is returned to the pool where it is available for assignment to other job steps.

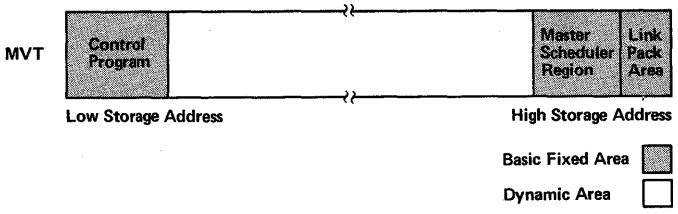


Figure 40. General Organization of Main Storage From the MVT Control Program Configuration

In this configuration, program and data resources, as well as the hardware resources of the computing system, can be shared among concurrently performed jobs, tasks within a job step, and control program tasks.[7]

Major Functions

The remainder of this book discusses the major functions that can be performed using the operating system, briefly describing the purpose of each function, how it is performed, and the major objectives it helps to achieve. Unless otherwise noted, the discussion applies to both control program configurations. The major functions are discussed in the following topics:

- Task management.
- Job management.
- Information management.
- Program development and management.
- Multiprocessing.
- Teleprocessing.

The characteristic of the System/360 Operating System that sets it apart from previous general purpose operating systems is its ability to schedule and supervise the performance of more than one data processing task at a time. It does this through efficient management of system resources.

Resource Sharing

A program is only one of several resources that are needed to perform a data processing task. I/O devices or direct access storage space is required for entering or storing input data and for recording or storing output data. Time is required on I/O channels for transmitting information to and from main storage and for starting and controlling I/O operations. Main storage space is required for storing a series of instructions and the data processed when the instructions are executed. Finally, CPU time is required to execute the instructions and thereby do the work of processing the data.

At most data processing installations, data processing tasks are performed one at a time; that is, a new task is not begun until the current task is completed. The average data processing task requires, at any given moment, only a fraction of the total available resources of the system (Figure 41). Therefore, many parts of the system are often idle for significant periods of time. For example, many tasks, such as data conversion tasks, require only a fraction of the storage space and I/O devices available in the system, and only intermittent use of the CPU for short periods of time.

However, at MFT and MVT installations, the available resources of the system are dynamically allocated and shared among several tasks being performed concurrently. As a result, more of the total computing system is kept in productive operation more of the time (Figure 42). The sharing is not limited to the hardware of the computing system. It includes the sharing of program and data resources as well.

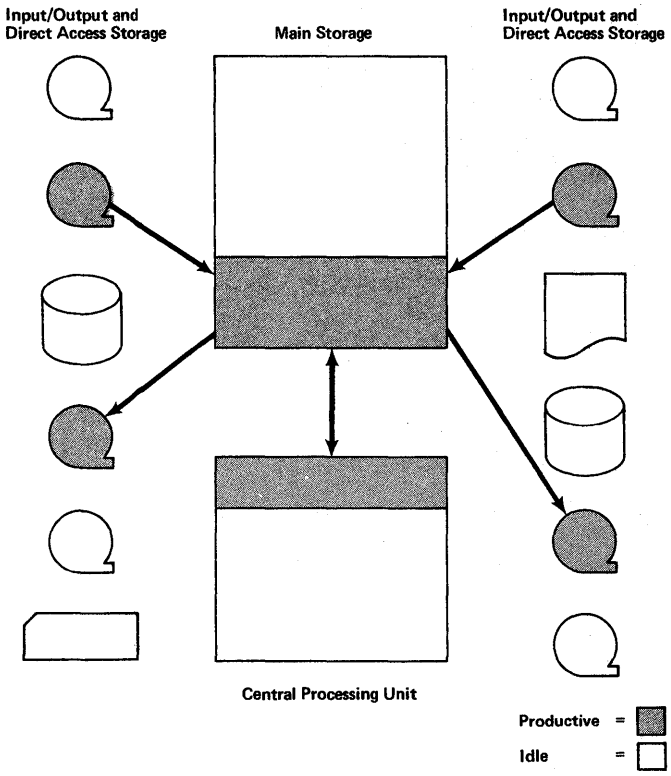


Figure 41. A Single Task System

Program Sharing

At most data processing installations, only one program at a time is executed to perform one task and produce one set of results, as shown in Figure 43. At such an installation there is always a one-for-one correspondence between a program (a series of instructions) and a task (the execution of these instructions by the CPU), and no distinction between them is necessary.

At MFT and MVT installations, however, such a distinction is necessary because a single program can be executed to perform several tasks concurrently and produce several independent sets of results (Figure 46). For example, the CPU can begin to execute a program to process one set of data, and then following an interruption, execute the same program to process another set of data. Thus, several data processing tasks can be performed concurrently using a single program. Such a program must be reenterable; that is, the program must be

designed so that it is not changed in any way when the CPU executes it. In other words, the execution of one instruction in the program must not change any other instruction in the program.

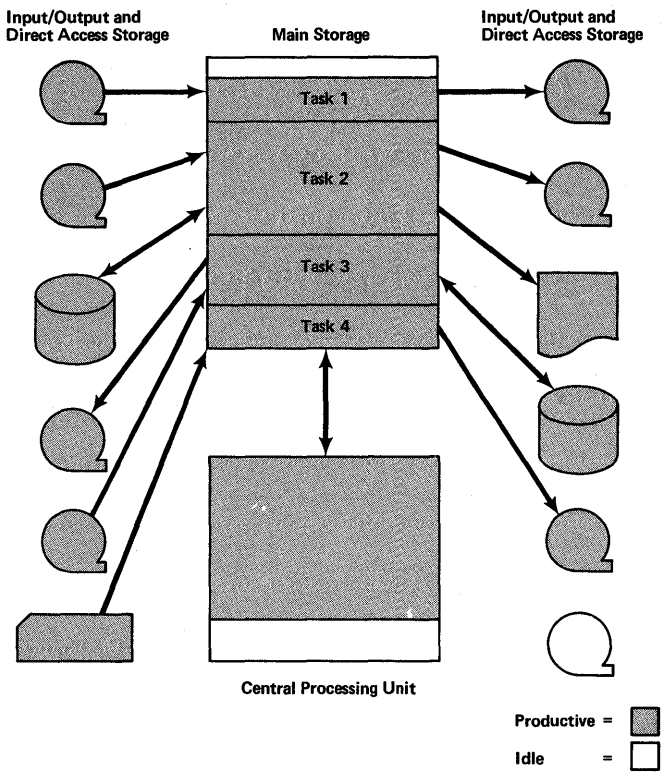


Figure 42. A Multiple-Task System

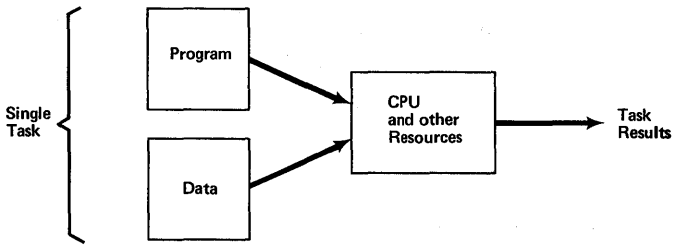


Figure 43. Unshared Information Resources

With an MFT or MVT control program configuration, any reenterable program can be loaded into main storage when the operating system is initialized at the beginning of a work period, and can remain there indefinitely. This avoids continual reloading of frequently used programs. Other reenterable programs can be brought into main storage when required by a specific job step. These programs are loaded into the job step's partition or region of main storage; each can be used concurrently by all tasks that belong to the job step, but not by tasks that belong to other job steps. The concurrent use of a reenterable program saves main storage, because each task does not require its own copy of the program. Once loaded, a reenterable program is available for use by concurrent tasks for as long as it remains intact within the job step's region or partition.

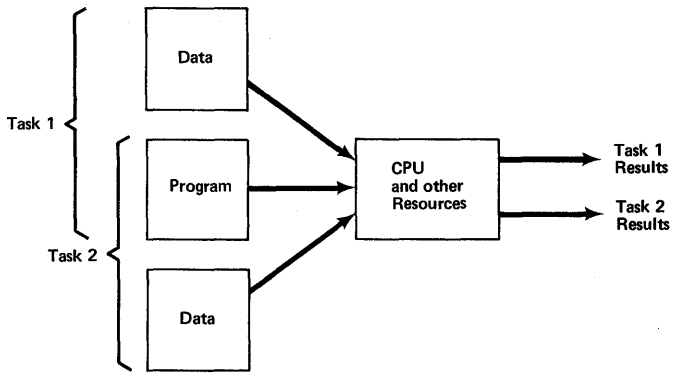


Figure 44. Program Sharing

Data Sharing

MFT and MVT control programs permit the concurrent sharing of data, as well as programs, when performing multiple tasks. Several programs using a common set of input data can be executed to perform different tasks and produce several independent sets of results, as illustrated in Figure 45. For example, two different programs can be executed concurrently to produce two different summary reports derived from the same basic set of data. To do this, the data set must not be subject to change in any way when the tasks are performed.

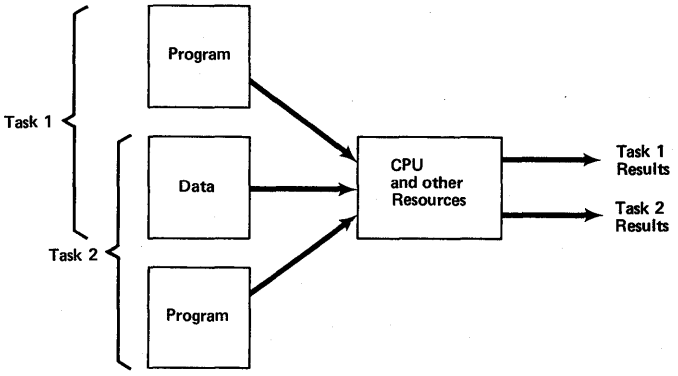


Figure 45. Data Sharing

Resource Management

Although the master and job schedulers initiate the performance of tasks, the tasks are actually performed under control and direction of the supervisor. The supervisor of an MFT or MVT control program manages the concurrent performance of tasks by keeping a running account of all the tasks that are initiated and by scheduling the order in which they are to be performed, based on their relative priorities. It also keeps a running account of all the available resources of the system and allocates the resources as they are required to perform specific tasks.

Some resources, such as reenterable programs in main storage, are always available for immediate allocation to a task. Other resources of the system, such as the CPU, can be allocated to only one task at a time. If a particular resource required for a task is not immediately available, the task is temporarily suspended. The task is then, in effect, placed on a waiting list or queue. At any time, several tasks may be waiting in a queue for such a resource. Usually, tasks in a resource queue are arranged in a priority sequence. However, other sequences are sometimes used to ensure more efficient use of particular resources (I/O channel time, for example). The resource queues, in effect, serve as work reservoirs that absorb fluctuations in demands for resources. They thereby help to ensure that a ready backlog of work is at hand to keep the resources as busy as possible. Whenever practical, the allocation of a resource is deferred until it is actually needed to perform a task, and the resource is released for reallocation as soon as the need has been satisfied. In other words, the resources are usually allocated dynamically, and are not tied up unless they are being used. Thus, within a relatively short period of time, a particular resource may be used over and over again in performing many different tasks.

More precise means of managing resources in an MFT or an MVT control program environment exist for the user. Both the Time Slicing feature and the System Management Facilities (SMF) feature allow system resources to be shared more equally among tasks.

The Time Slicing feature can be used to prevent any task in a group from monopolizing the CPU and thereby delaying the assignment of CPU control to other tasks in the group. The supervisor does this by allocating a uniform interval of CPU time (a time slice) to a task within the group whenever it is assigned control of the CPU. If the task is still active when the interval ends, then the supervisor assigns control to another task.

At an MVT installation, all of the tasks having a predetermined priority are allocated time intervals when CPU control is assigned. At an MFT installation, all of the tasks to be performed using any one of a predetermined group of consecutive main storage partitions are allocated a time interval. (If the MFT control program includes the subtasking option, time intervals are allocated to tasks whose priorities fall within a certain range; the range corresponds to a predetermined group of consecutive main storage partitions, but the priority of any task can vary and thus be independent of the partition where the task is performed.) The time slicing feature is especially useful for graphic display and teleprocessing applications. It can help to ensure a uniform time response to a number of users who are located at local or remote terminals and are sharing the hardware and information resources of an installation.

The SMF feature provides data collection routines and exit linkages for the user. 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; this information is used to make a variety of analyses. Output created by the SMF routines can also be used to create and maintain inventories on direct access and tape devices; establishing a data base (recorded data in a permanent format) against which to make an analysis. 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.

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, standards of efficient resource management -- identification, priority, resource allocation, and maximum execution time -- can be enforced in the system.

Advantages of Multiple-Task Management

The ability of MFT and MVT supervisors to dynamically allocate resources and manage the performance of several tasks has a number of advantages over more conventional methods of data processing:

- It increases the efficiency of the system. Resources that might otherwise be idle when one task is performed can be used to perform other tasks.
- It enables the system to grow without disruptions. The addition of more storage space and other resources can be readily accommodated and used efficiently without reprogramming merely by increasing the number of jobs, or other tasks, to be performed.
- It improves the flexibility of the system. The system can readily adjust to varying demands for resources and changes in the workload. This is particularly significant for teleprocessing applications, such as the airline reservation application, where the workload varies dynamically in unpredictable ways.
- It improves the rate at which the system can respond to work requests. Several tasks can progress in parallel instead of being performed in consecutive order. This too is important in teleprocessing applications, where a system normally must respond to each work request within a short period of time.

Concurrent Tasks Within Job Steps (MVT and MFT With Subtasking)

At an MVT installation, the same task management facilities that initiate and control the concurrent performance of jobs and job support tasks are directly available for use by a customer. The facilities are also available at an MFT installation when the subtasking option is selected at system generation.

Previous control programs, other than the MFT and MVT control programs of the operating system, have provided means for concurrently performing data processing tasks. But these were special purpose control programs, such as the airline reservation system application discussed earlier. In these applications, many of the advantages offered by a multiple task system, such as fast handling of the workload, were essential to the application. Therefore, they justified the high cost of a special purpose control program.

An MVT configuration of the operating system, or the MFT-with-subtasking configuration, is a true general-purpose task management system. It is not restricted to performing specific types of tasks, a group of related tasks, or a fixed number of concurrent tasks, but can, in fact, be used to perform concurrently any number or types of related or unrelated tasks within the limits of available system resources and the user's ingenuity.

The same multiple task management facilities that are used by the control program can also be used by a customer in designing processing programs that can be executed to concurrently perform tasks within a job step. Therefore, the advantage of a multiple task system, heretofore limited to a few specialized applications, can be realized in a wide range of applications. The general purpose task management facilities of the control program can, in fact, be used in innumerable ways by both IBM and its customers to increase the productivity and utility of System/360 and to broaden or extend its application.

At an installation of either MVT or MFT with subtasking, once a job step is initiated by the control program, it may, in turn, initiate the performance of other tasks, which compete for and share the resources of the system with one another and with the steps of other jobs. In fact, a complete and complex hierarchy of tasks can be dynamically initiated and terminated in the process of performing a single job step.

The programmer is provided with a complete set of general purpose task management language statements. With these he can program dynamic control over the concurrent performance of the tasks of a job step. He can, for example, initiate the performance of new tasks as the workload increases and terminate the tasks as the work is completed freeing those

resources. He can also synchronize the performance of one task with that of another or with I/O operation; for example, he can indicate that further performance of one task should await the completion of one or more other tasks or other occurrences such as the end of the time interval.

The programmer can also establish a system of relative priorities among the tasks of a job step. If necessary, he can specify changes in priority based on events that occur as the tasks are being performed. For example, he may raise the priority of a task if it still is not completed after a specific period of time.

The programmer can also establish new resources and, with the help of the control program, control their use in performing concurrent tasks. A new resource may be a self-initializing program that can be used serially, but not concurrently, in performing different tasks, or it may be a table of data that can be used either serially, or concurrently, depending upon whether or not the table can be modified while it is being used to perform a task. The programmer is given the means to ensure that a resource is used serially if it is subject to change and concurrently if it is not.

Main storage may be shared or passed between tasks in an MVT environment by using a subpool -- a 2K block of main storage allocated for a task under the label called a subpool number. Subpools can be shared by other tasks, or they can be passed from the task that created them to another task.[8]

Normally, main storage space requested by a job step program is allocated from a partition or region of main storage that is assigned to the job step when it is initiated. However, an optional feature of the MVT supervisor allows the temporary assignment of an additional region (or regions) of main storage to a job step that has outgrown its previously assigned region. When additional space is requested during a job step, the supervisor attempts to satisfy the request from an unassigned portion of the dynamic area of main storage. If space is not available, then the contents of a region assigned to another job step are transferred to direct access storage (rolled out), and the vacated region is assigned to the step that requires additional space. When the region is no longer needed, its original contents are restored (rolled in) and the delayed step is allowed to continue.

The user indicates which steps can be rolled out and which cannot. However, the relative priority of the steps determines the order in which they are rolled out and whether or not a particular step will be rolled out.

The multiple-task management facilities described in this section provide the basic tools required for many applications that are beyond the capability of a single-task system. These include many telecommunication applications that would otherwise be impractical without a specially designed control program. The same facilities can also be used for more conventional applications. They enable the programmer to design highly efficient production programs that, when executed, result in a high degree of resource sharing among concurrent tasks.

At a System/360 Operating System installation, the actual work of processing data is performed by the computing system under control and direction of the supervisor. However, before any work can be performed by the system, it must be scheduled and initiated by either the master scheduler or the job scheduler. The master scheduler is used by the operator to schedule and initiate the work performed by the job scheduler. The job scheduler, is used to read, interpret, schedule, initiate, record output for, and terminate the steps of a series of jobs that are defined and submitted for processing by the programming staff.

In all configurations of the operating system, the job scheduler is designed to process a continuous series of jobs without unnecessary delays between one job or job step and another. In the MFT and MVT configurations of the operating system, the supervisor is capable of directing and controlling the performance of more than one data processing task at a time. The master and job schedulers for these configurations are designed to take advantage of this capability, and by so doing increase the performance of the system as a whole. This is accomplished in two major ways: by scheduling and initiating the performance of more than one job at a time, and by performing job support tasks concurrently with the jobs.

Non-Stop Job Processing

A job is the major unit of work performed by the operating system. Each job is defined by a series of job control language statements coded by a programmer (see Figure 46). The job definition is provided by a JOB statement containing information concerning the job, such as its name and priority.[4]

Each job consists of one or more steps. These are defined by the programmer and arranged in the order in which they are to be performed. A job step is defined by an EXEC statement containing information such as the name of a program to be executed to perform the job step and (for MVT) the amount of main storage space required to execute the program. The specified program may be a problem state program supplied by IBM, such as a language translator, or it may be a problem state program created by the user of the system, such as a payroll program.

IBM 2361 Core Storage, if included in a system, is allocated to a job step in the same way as processor main storage. A programmer can request that either processor storage or 2361 storage, or both, be allocated for a specific job step.

Individual job definitions can be placed one behind another to form a continuous series or stream of job definitions. These can then be read by the job scheduler and processed without stopping the computing system between jobs or job steps. If operator actions are required for one job, such as mounting tape reels, these can be performed while the resources of the system are being used to process other jobs.

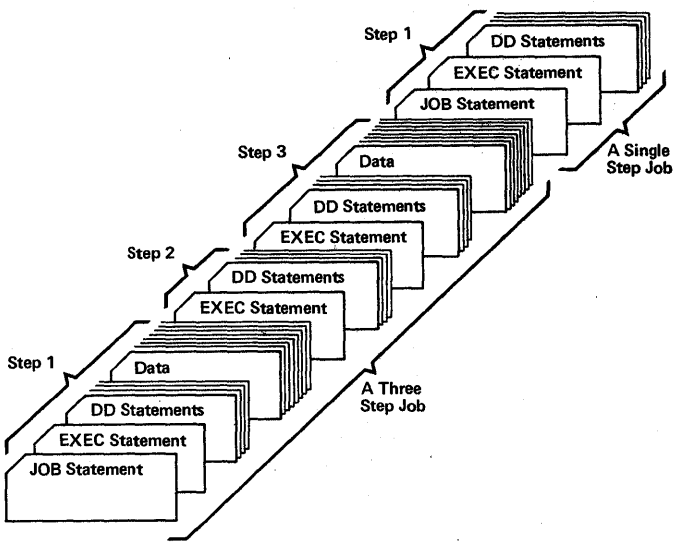


Figure 46. Job Definitions

Any set of data that is processed during a job step must be identified and defined within the definition of the job step using a DD (data definition) statement.[3] A programmer may place input data to be processed during a job step within the job step definition and define the data as being part of the common input stream. Similiarly, output data produced during a job step can be defined as being part of a common job output stream. It can then be printed or recorded on a commonly shared output device. As a result of defining data sets in this way, no operator setup delays are incurred within a job step. Also, any I/O devices that would otherwise have been required for the job step are available for other purposes.

If a series of job step definitions are to be used repeatedly with little or no change, a programmer can store and catalog them in a procedure library maintained in direct access storage by the control program. Thereafter, using single job and job step statements in an input stream, he can direct the job scheduler to pick up the job step definitions from the procedure library. If necessary, the same job statement can override specifications in the job step definitions picked up from the procedure library. Using this feature, a system programmer can predefine standard types of jobs that are commonly performed at an installation.[6,7]

By doing this, the system programmer can eliminate the need for applications programmers to redefine standard jobs each time they are performed. He can also help to ensure that the system is used efficiently and consistently.

The ability to process a continuous series of jobs and job steps with little or no operator intervention is an important characteristic of all configurations of the control program. By reducing the degree of human participation in the mechanics of data processing, the operating system ensures that jobs are processed faster and are less subject to human error. As a result, the total volume of work performed by the system can be increased.

Multiple-Job Processing

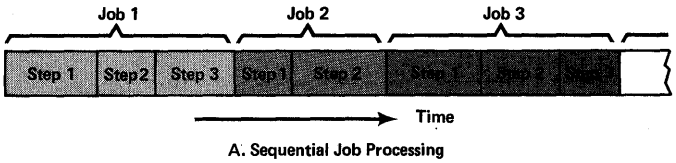
Normally, the steps of a data processing job are logically related to one another to produce a specific end result. In most cases, the steps of a job must be performed in a particular sequence since output produced by one step often serves as input to a succeeding step. For example, a typical job may consist of the following three steps:

- Translating a source program into an object program.
- Linkage editing the object program to produce a program suitable for loading into main storage.
- Loading and executing the program.

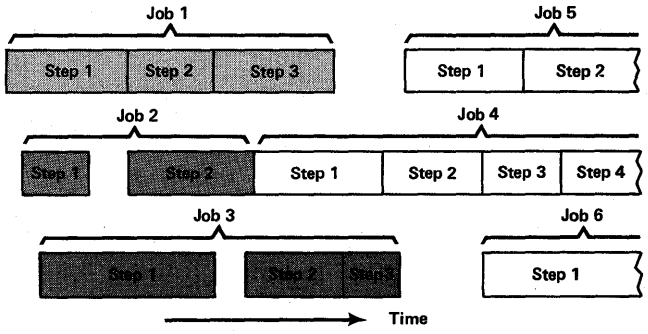
Each of these steps is a necessary part of the complete job. They cannot be performed in any other sequence. Because the steps within a single job may be dependent upon one another, they are always performed in a sequential order. Steps of different jobs are not dependent upon one another, therefore, they can be performed concurrently. In most computing systems, jobs and job steps are performed one at a time in a fixed sequential order as shown in part A of Figure 47. No matter how small the job or how large the system all of the resources of the system are tied up until the step is completed.

With an MFT or MVT control program, these same jobs can be performed either sequentially (as shown in part A of Figure 47) or, if enough resources are available, concurrently as shown in part B of Figure 47. In the latter instance, any one job may take longer to perform because it may be temporarily delayed from time to time awaiting a resource currently being used to perform other jobs. However, because the resources are shared among several jobs, the rate at which the jobs as a whole are performed is significantly increased, resulting in a greater total throughput.

Each job step is actually a task that can be performed concurrently with other tasks, including steps of other jobs, under management of the control program. The control program (both MFT and MVT) can initiate and supervise the concurrent performance of up to 15 steps of different jobs.



A. Sequential Job Processing



B. Concurrent Job Processing

Figure 47. Sequential and Concurrent Job Processing

A set of data in direct access storage can be shared concurrently among several jobs provided it is not changed in any way by the jobs that are sharing it. When the control program encounters a job that will change a data set, it prevents possible conflicts in the use of data set by delaying initiation of the job until all previous jobs that use the data set are completed. Similarly, if a job that

changes a data set is being performed, any succeeding jobs that use the data set are not initiated until the current job is completed. The control program can also delay the initiation and progress of low priority jobs in favor of higher priority jobs. In all other respects, jobs are performed independently of one another and the control program recognizes no direct relationship between one job and another.

Multiple-job processing is particularly suited to data processing installations with a high volume of work and a large number of resources. It enables a large system to perform small jobs as well as large jobs efficiently, and to run jobs with complementary resource requirements concurrently, thereby increasing throughput. It also allows a gradual, systematic expansion of hardware resources without reprogramming. With multiple-job processing, an installation can achieve a high degree of productivity by optimizing the system for particular classes of work and controlling the mixture and load of work.

Concurrent Job Support Tasks

Job definitions, and any input data that accompanies them in an input stream, are usually submitted for processing in the form of punched cards. Also, much of the job output data normally ends up in printed or punched card form. In many installations (particularly large ones), the relatively slow printing and punched card operations are performed by a small offline computer that specializes in data transcription. Although this can improve job processing efficiency by not tying up the main computing system with low speed I/O operations, it can significantly increase job turnaround time. Typically, at such an installation (Figure 48) the following steps are performed when processing jobs:

1. The jobs, in punched card form, are normally arranged in priority order.
2. After enough jobs have been accumulated to form a batch, they are transcribed to magnetic tape.
3. The batch of jobs on the tape is manually scheduled and then processed on the central computing system.
4. After a batch of output data has been recorded on a tape by the central computing system, it is manually scheduled and then converted to printed or punched card form or a combination of the two.
5. The printed and punched card output is manually sorted into various classes and distributed to the individuals that submitted the jobs.

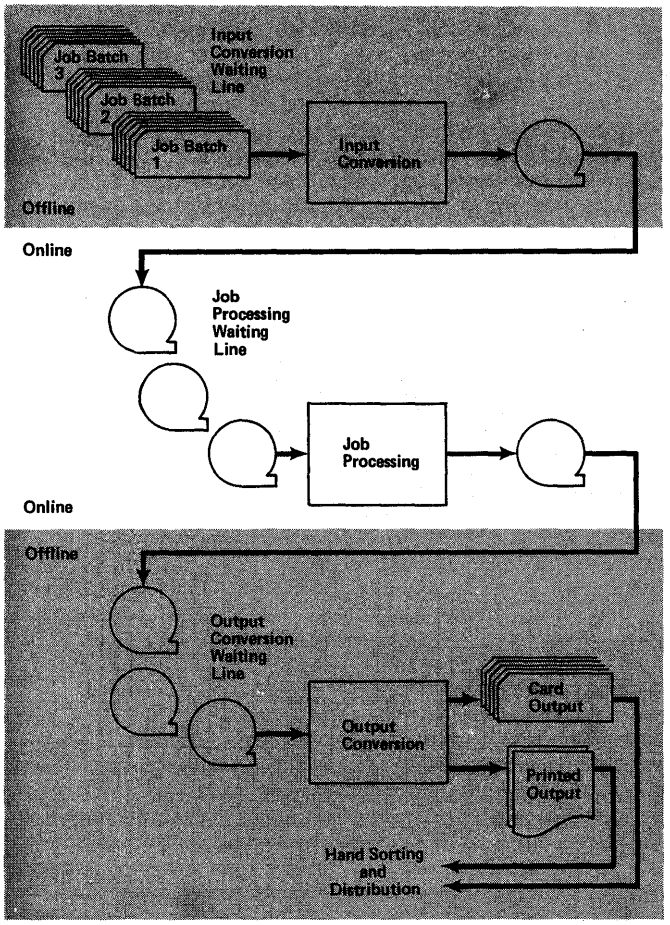


Figure 48. Offline Peripheral Operations

Each of these steps involves considerable human activity and attendant delays. To avoid confusion and inefficiency, the total process requires tight supervisory control and coordination. In such a process, priority jobs can be accommodated only with difficulty or loss of efficiency. As a result, turnaround time at such an installation is measured in hours and days.

To avoid such problems at MFT or MVT installations, operations such as reading job and data cards and printing job output data, are performed by the control program as separate tasks, concurrently with other work. As a result, jobs can be processed automatically, from beginning to end, on the central computing system. Thus, the delays and human activities involved in using offline systems are avoided and turnaround time is reduced significantly.

At MFT and MVT installations the control program can read job definitions and data from one or more job input streams and record job output data on one or more output devices, while initiating and controlling the performance of one or more jobs. As job definitions are read and interpreted, they are placed in an input work queue located in direct access storage. Also, as the jobs are being performed, output data from the jobs is placed in an output work queue. The input and output work queues are roughly equivalent to the waiting lines for job processing and output conversion shown in Figure 48. However, the input and output work queues are automatically maintained in direct access storage by the control program and are an integral part of the system. Thus, one job need not await the completion of a long series of preceding jobs. As soon as one job is placed in an input work queue, it can be initiated by the control program. As soon as the job is completed, any output data it placed in the output work queue can be recorded. Therefore, turnaround time can be reduced from days or hours to minutes.

The MFT and MVT Job and Master Schedulers

The job and master schedulers control the concurrent processing of job and job support tasks. The MFT and MVT job schedulers are divided into three major parts: the reader/interpreter, initiator/terminator, and the output writer (Figure 49). However, each part can be executed concurrently with and independently of the others.

The reader/interpreter program reads jobs and job step definitions from an input stream, analyzes the definitions, and places them in the input work queue.

The initiator/terminator program selects a job from the input work queue and initiates the job and each of its steps. Once a job step is initiated, it is performed as a separate task under the control and direction of the supervisor. While each job step task is being performed, output data may be generated and placed in the output work queue in direct access storage. The initiator/terminator terminates the job and each of its steps as they are completed, and initiates a new job step.

The output writer program reads data from the output work queue and records it on an output device such as a printer or card punch.

The reader/interpreter, the initiator/terminator, and the output writer programs are all executed independently of one another to perform separate and distinct tasks. Therefore, at any one time, any one or all may be active and each can be started or stopped independently of the others.

The master scheduler serves as a two-way communications link between the operator and the system by way of the operator's console. It is used to relay messages from the system to the operator, to execute commands, and to respond to replies from the operator. The operator is provided with a full set of commands which he can use to start and stop job scheduling tasks, log operational information, monitor and control the progess of work performed by the system, and restart the system after a shutdown.

Operation of the system can be planned in advance and operator commands placed in a job input stream. From there they are relayed to the master scheduler for execution as they are encountered by the reader/interpreter. Since the operator can also enter commands by way of the operator's console, he can dynamically control the operation of the system to react to conditions that develop while it is operating.

By issuing commands to the master scheduler, the operator can start or stop reader/interpreter tasks, initiator/terminator tasks, and output writer tasks, based on the type of work to be performed and the availability of resources to perform it. At the beginning of a work period, an operator may start one or more reader/interpreter tasks. Then, after a number of job definitions have been placed in the input work queue, he can start one or more initiator/terminator tasks. At any point, he can stop any of the reader/interpreter tasks thereby releasing their assigned I/O and main storage resources for allocation to job step tasks. Then or later, the operator may start one or more output writer tasks to record the output data sets produced when the job steps are performed.

Any reader/interpreter and output writer tasks that are to be performed at an installation are defined in much the same way as a single step job is defined. These definitions are cataloged in a procedure library where they are available for use in initiating a reader/interpreter or output writer by operator command. Any number of task definitions can be cataloged. Each can define a reader/interpreter or output writer task having a unique set of characteristics. For example, the

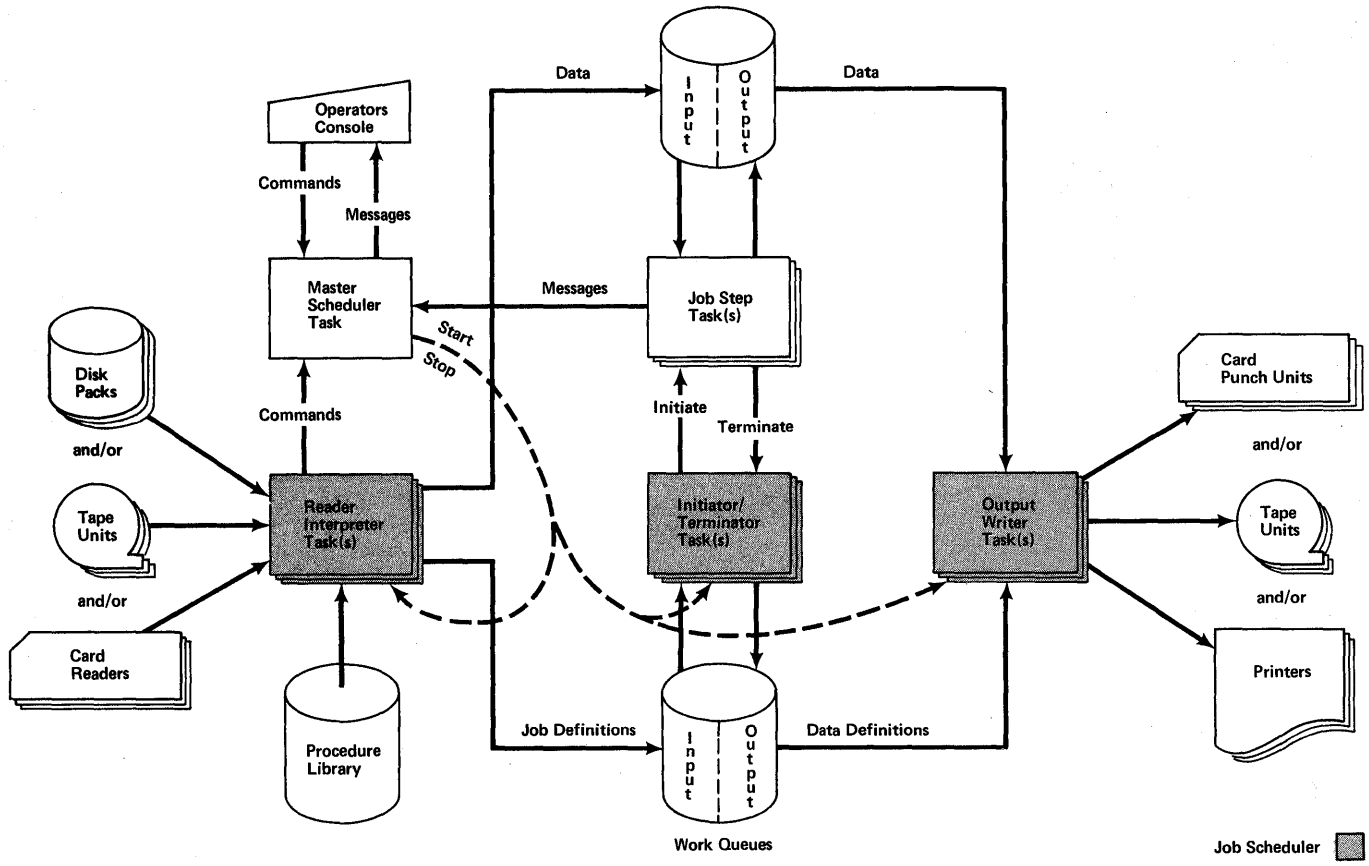


Figure 49. The MFT and MVT Job Master Schedulers

reader/interpreter task definitions can specify different devices and different data set options, as well as different default options to be used when interpreting job or job step definitions.

Many of these specifications can be overridden and respecified when the operator starts the task. Thus, the installation staff is provided with a great deal of flexibility in defining the characteristics of the reader/interpreter and output writer task to be performed. For example, special tasks that satisfy the unique requirements of a particular department or a particular class of jobs can be defined.

At MFT and MVT installations, up to 15 initiator/terminator tasks can be started to control the initiation and termination of up to 15 concurrent jobs. At an MVT installation, any number of reader/interpreter and output writer tasks can be started provided enough resources are available. At an MFT installation, the number of concurrent reader/interpreter tasks is limited to three, and the number of output writer tasks is limited to 36.

Job Priorities

So far as job management is concerned, the main difference between an MFT and MVT control program has to do with the way in which priority is assigned and main storage space is allocated.

At an MFT or MVT installation, each job that is submitted for processing can be assigned a specific priority relative to other jobs.[4] It can also be assigned to any one of several classes of jobs. When the job definitions are read by the reader/interpreter they are placed in the input work queue in accordance with their assigned class and priority. A separate input queue is maintained for each class assigned to the jobs. Within each input queue, the job definitions are arranged in the order of their priority. Output data produced during a job step can be assigned by the programmer to any one of up to 36 different data output classes defined at the installation. When an output writer task is started it can be assigned to process from one to eight different classes of output. A particular output class may represent such things as the priority of

the data, the type of device that may be used to record it, or the location or department to which it is to be sent.

In an MFT installation, any main storage space not reserved for use by the control program is logically divided as specified by the operator into partitions of various sizes. Each partition is assigned by the operator for use in performing either a reader/interpreter or output writer task or a particular class of jobs. The priority of a job step task is determined by the partition to which it is assigned. Each partition is assigned by the operator to one, two, or three classes of jobs. Whenever a new job is initiated it is directed to (or is allocated) a partition that was assigned to its job class. The operator can change the job class or classes to which a partition is assigned, and thereby control the mixture of jobs. In addition, since each partition is assigned a specific priority, he can also control the priority assigned to each class of jobs.[6]

In an MVT installation, any main storage space not reserved for the control program serves as a pool of storage from which a region is dynamically allocated by the control program to each job step or job support task as it is initiated. The size of the region to be allocated to each job step is specified by the programmer in the job or job step definition. The priority of a job is also specified by the programmer. When an initiator/terminator task is started by the operator, it can be assigned to initiate jobs from one through eight input work queues. By classifying jobs and assigning initiator/terminators to initiate specific classes of jobs, it is possible to control the mixture of concurrent jobs; thus, jobs with complementary resource requirements can be performed concurrently. For example, one initiator/terminator can be assigned to a class of jobs requiring a great deal of CPU time and little I/O while another initiator/terminator is assigned to a class requiring little CPU time and a great deal of I/O.[7]

Except for human resources, recorded information is the single most valuable resource of an installation. Information serves as a rational basis for controlling the activities of an enterprise and for making decisions upon which its success depends. In many enterprises, more money is spent in gathering and storing information than in processing it. Yet, most are barely beginning to tap the potential uses of the information resources in which they have so heavily invested.

The Centralization and Growth of Information

Over the last few years, most of the basic accounting, record keeping, and problem solving activities of the typical large enterprise have been computerized. As a result, much of the basic information of the enterprise has accumulated at one or more central data processing installations. Along the way, a great deal of information that was once recorded on punched cards and paper was transcribed in a more condensed form on magnetic tapes, disks, and drums. Although the space required to store existing information has been reduced, the amount of information continues to increase at an explosive rate that threatens to overwhelm those who strive to manage and use it effectively.

Problem of Growth and Centralization

The lack of an effective system for managing the mass of information at an installation often causes a great deal of duplication. It isn't unusual, for example, for several nearly identical sets of information to be independently created, stored, and maintained. This causes severe problems in keeping the information up-to-date and in controlling its use. Without effective control over the use of information, users tend to gather and maintain their own information and avoid consolidating and sharing it in cooperative ventures. As new information accumulates, much of the obsolete information remains and the operations staff finds it hard to cope with the mass of information with which they are entrusted.

Because there is no visible evidence of magnetically recorded information, the operations staff is usually forced to maintain elaborate records on paper in order to classify, catalog, and locate information, control its use and disposition, and find and assign space for storing it. Such efforts are subject to human errors, changes in personnel, and other problems that often arise in human activities.

Opportunities of Growth and Centralization

Although the growth and centralization of information at data processing installations creates management problems, it also creates opportunities. When data is centralized, all of the important records on which the day-to-day activities of an enterprise depend can be combined into a single mass of information. Using mathematical and programming techniques it is possible to derive from this mass new forms of information such as reports and statistics for management information systems. In other words, the same information that is required for traditional accounting and problem solving activities can often be used as an information base for newer and more imaginative ways of running a business or scientific enterprise.

Requirements for a Unified Information Management System

To solve the problems that result from the growth and centralization of data at an installation and take advantage of the opportunities it provides requires a unified information management system. Because requirements differ widely from one installation to another or from one application to another, IBM does not provide a single system that will solve the information management problems of every installation. It does, however, make it possible for any installation to develop an organized and efficient information management system to meet its own particular needs. It does this by providing:

- A consistent way of organizing data.
- A built-in library reference system for use in locating data.
- A combination of methods for storing and retrieving data.

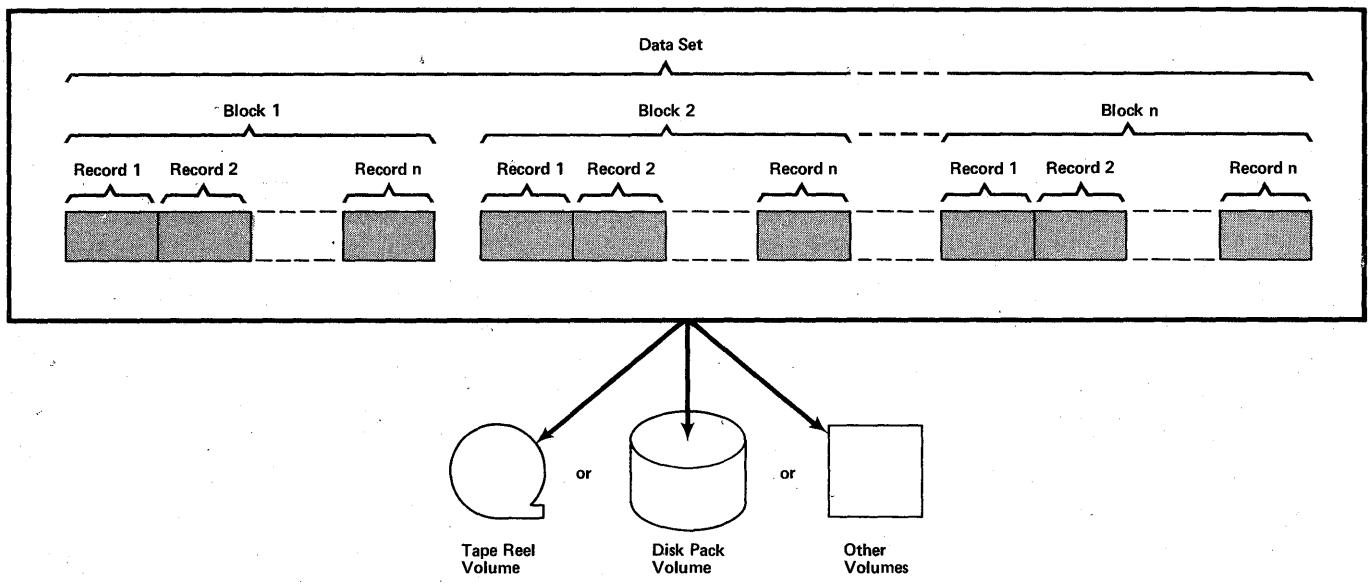


Figure 50. Data Organization

Data Organization

A basic requirement of a unified information management system is that the data be organized in a consistent manner. Otherwise, the data cannot be easily shared by a community of users, nor can the activities of processing programs be coordinated with those of the control programs. In many respects the control program uses traditional methods of organizing data. The smallest division of information that is normally of concern to the control program is a record (Figure 50). A record is formed of one or more fields recorded in an unbroken series, and usually represents an organized body of related data, such as all of the basic accounting information concerning a single sales transaction. A field usually represents a single item of information, such as an account number, the name of a person, or the calculated interest on a loan. In any event, such data fields are not singled out or recognized by the control program.

One record or several records grouped together in an unbroken series form a block. A block of data in auxiliary storage is separated from another block by a gap in the data, and is transferred to or from main storage as a unit. Records may be grouped together to form a block because they represent a logical entity. Or they may simply be grouped together to avoid wasting auxiliary storage space or to reduce the number of separate data transfers between main and auxiliary storage.

The operating system can store and retrieve records that are all the same length or that differ in length (Figure 51). If the records are of differing lengths, a field at the beginning of each record must indicate how long the record is, and a field preceding the first record of each block must indicate the length of the complete block. These two fields, indicating the record and block lengths, are used by access method routines in extracting records from a block. If the lengths of the records of a block are not defined, the complete block can still be stored or retrieved by the control program. However, any consolidation or extraction of records must necessarily be done by the user program that processes the records.

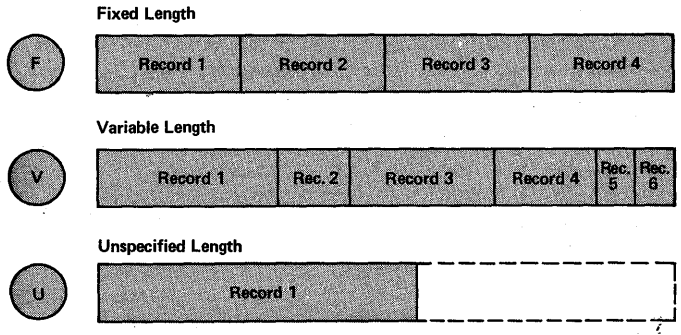


Figure 51. Data Record Formats

In the case of variable-length records, a single record can be divided into segments, with each segment contained in a separate block (Figure 52). A record can

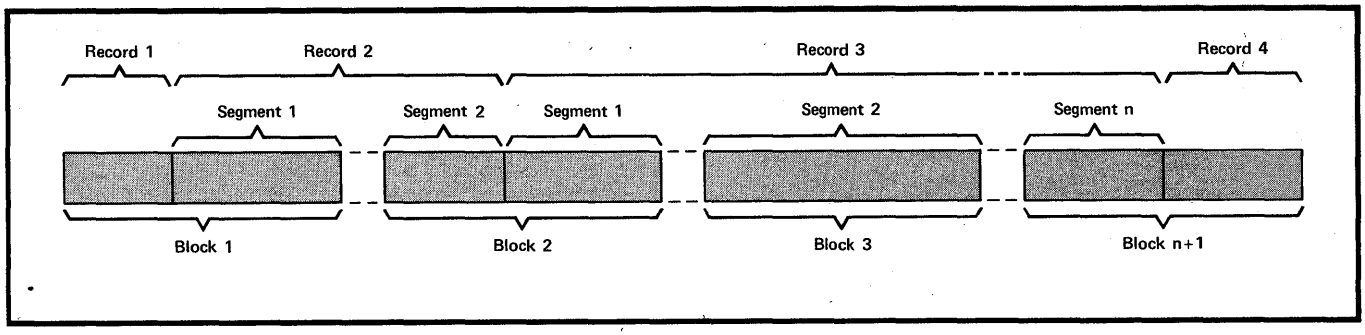


Figure 52. Spanned Variable-Length Records

thus be longer than a block, and may in fact span several blocks. As a result, block length can be defined in such a way as to optimize the use of auxiliary storage, without regard to record length. No special programming is necessary to make use of spanned records, but programs can be written to process either complete records or record segments. Each segment contains a field that defines it length and its relative position within the record. The field indicates whether the segment is the first segment, the last segment, or a middle segment, or whether the segment is a record complete in itself.

One or several related blocks of data separated by gaps form a data set or, as it is often called, a data file. Each set of data represents an organized body of related information, such as all of the information concerning a series or group of sales transactions. Data sets are usually independent of one another, both logically and physically. They may, for example, be stored in different auxiliary storage volumes.

A volume is a section or unit of auxiliary storage space that is serviced by a single read/write mechanism whose operation is entirely independent of any other read/write mechanism. In an operating system installation, a volume may be:

- A reel of tape.
- A disk pack.
- A data cell.
- A drum.
- A section of an IBM 2302 Disk Storage (Model 3 or 4) serviced by a single read/write mechanism.

They are called volumes because, like collections of books, they store related sets of information (data sets). Sometimes, like books, they can be moved from place to place (disk pack and tape reel volumes, for example). Often the volume containing a particular set of data is located by searching through a series of indexes, in much the same way as a book in a library is found. Also, each direct access volume contains a table of contents defining sets of data contained in the volume and tells where they are located.

Library Reference System

In a medium or large scale data processing organization, keeping track of data can be a formidable undertaking. This is true in organizations that attempt to take full advantage of the benefits offered by direct access storage devices.

At any one time a great many data sets or files may exist within the organization. The bulk of these are usually stored on magnetic tape or in direct access storage. Usually each programmer must keep track of the tape and direct access storage volumes on which his data sets are stored. For direct access storage volumes, he must also keep track of exactly where on a volume his data sets are located.

Furthermore, the operations staff at an installation must assign space for storing data sets. They may assign removable tape and disk pack volumes to programmers and, in the case of direct access storage volumes, they may also assign specific areas of storage on particular volumes. In any event, the operations staff must maintain up-to-date records on the assignment of space for storing data and must systematically control its use and disposition.

Over a period of time, as new data sets are created and old data sets are abandoned, the problem of managing the data mass becomes more severe. This happens not only because of an increased number of data sets (both current and obsolete) but also because human beings become too much involved in the mechanics of managing data and more subject to error. As a result,

data sets may be destroyed at times because of mixups in space assignments, and programmers may abandon installation procedures, maintain their own private library of removable volumes, and avoid sharing storage space and data with one another.

Another problem facing many installations is the problem of using direct access storage efficiently. After a time, available space on shared direct access storage volumes tends to become fragmented, and a great deal of space is wasted. Yet any attempt on the part of the operations staff to reorganize and consolidate the data sets on the volumes can be a difficult undertaking involving many individual users.

Because of such problems, the operating system contains a built-in library reference system that is used to classify and locate data sets and allocate space for storing them. In many ways it resembles library reference systems that are used to locate information in book libraries. In the operating system, the control program assumes the role of the librarian. Given the name of a data set, it can identify the volume containing the data set and then locate its position within the volume provided, of course, the data set has been stored and cataloged within the library reference system.

To do this, the control program searches through a catalog consisting of a hierarchy of indexes maintained in direct access storage (Figure 53).[13] The catalog not only serves to direct the control program to the volume containing the data set, it also serves to classify the data set. At any given time there may be a great many data sets stored in auxiliary storage. Therefore, the system must ensure that no two data sets have the same name. This is accomplished by adopting a method of classifying and cataloging data sets that is similar to the Dewey decimal classification method used to classify and catalog books in a library. Instead of numbers, alphameric names of up to eight characters are used to identify a set of data. With this method, a data set name may be referred to as:

DESIGN.ELECTRO.ROBERTS

where ROBERTS is the basic name of a data set that is classified under the name ELECTRO that, in turn, is classified under the name DESIGN. The major class name and each subclass name of a data set corresponds to an index in the catalog. By searching through the corresponding major class index and each corresponding subclass index, in turn, the control program can identify the serial number of the volume in which a data set is stored and, if necessary, instruct the operator to mount the volume on an appropriate device.

Once the control program identifies the volume in which a data set is stored, it can locate the position of the data set within the volume. At the beginning of each direct access volume in the system is a volume label that directs the control program to a table of contents (Figure 53). The table of contents contains the name, description, and location of each data set stored within the volume. By reading and searching through the table of contents, the control program can find the location of a data set.

The table of contents of a volume also contains a record of each unused area in the volume. The record contains the location of each area and its size. Therefore, by searching through the table of contents, the control program can automatically find and allocate unused space on the volume for the temporary or long term storage of data sets.

A tape volume does not contain a table of contents. However, each data set is assigned a sequence number when it is created. Using this sequence number, the control program can locate a particular data set on the tape and advance the tape to the beginning of that data set.

No distinction is made within the library reference system as to the type of data contained in a volume. Data is stored in the same way whether it is a set of job control statements, a source program, an object program, or a set of data blocks to be processed by an object program. The only distinction among the different types of data is in the nature of the data itself and the way in which it is used.

Most of the data sets that make up the operating system and that are used by the operating system in performing its work are cataloged within the library reference system. Once the operating system is generated, a system programmer can extend the catalog using system utility programs specifically provided for that purpose. For example, he can construct a catalog consisting of several levels of indexes, and when more data sets are created, he can extend the catalog to reflect this growth. The catalog structure for a particular operating system may be represented in the form of a chart, as shown in Figures 53 and 54. The data sets that are cataloged can be classified in many different ways -- for example, to reflect the organizational structure of an engineering department that uses the system.

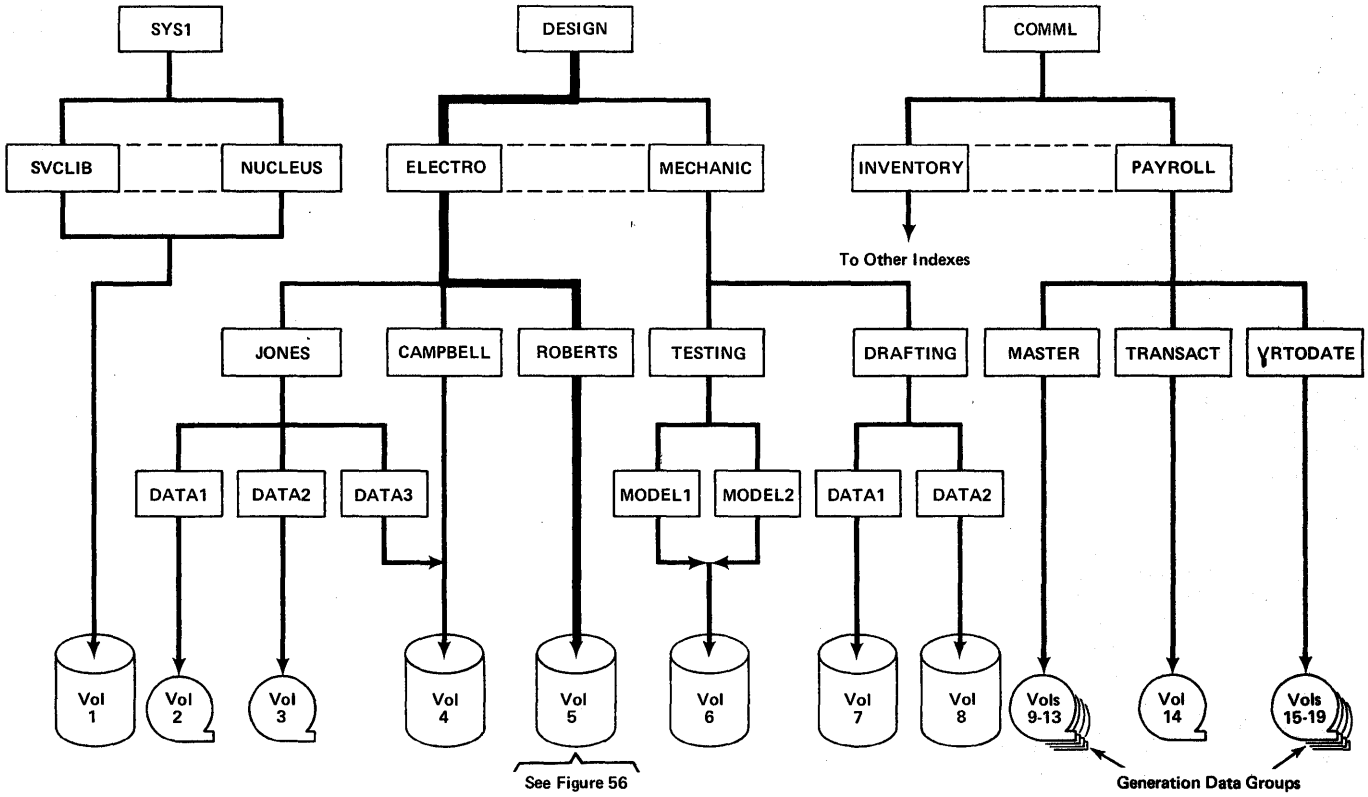


Figure 53. Simplified Diagram of Catalog System for Locating a Volume

Although the highest level index of the catalog is always stored on the direct access volume containing the operating system control program, branches of the catalog can be stored on other direct access volumes, including removable volumes. For example, the index named DESIGN in Figure 53, as well as any subclass indexes, could be stored on a removable disk pack that is assigned for exclusive use by the design department.

In addition to cataloging single data sets in the library reference system, several successive generations (updates) of a data set (called a generation data group) can be cataloged.[3] This method can be used to catalog a data set, such as a year-to-date earnings data set, that is updated weekly by a payroll program. Each generation of the data set may have the same name and be identified relative to the current generation of the data set. With this method of cataloging, the system can automatically keep track of the generations that have been created, and delete obsolete generations as new ones are created.

A special type of direct access data set, called a partitioned data set can also be cataloged (Figure 54).[10] This type of data set is used primarily to store programs, and is therefore often referred to as a program library. It has its own index, called a directory, which is used to locate one or more sequential blocks of data (called members) that are stored in separate partitions of the data set. The major use of partitioned data sets is described in the section titled "Program Development and Management."

The library reference system for locating (and assigning space for) data sets makes it possible for programmers to efficiently share the use of direct access storage devices with a minimum of interference. To help ensure efficient sharing, the system programmer is provided with a comprehensive set of system utility programs with which he can control the classification of data and the use of direct access storage. These programs can be used to create, rename, or delete indexes and data sets; to reorganize and rearrange a library of data sets; and to conduct surveys of the organization and contents of direct access storage.[15]

Methods of Storing and Retrieving Data

There are two major types of auxiliary storage devices commonly used in computing systems: sequential access devices and direct access devices. A device is classified as one or the other depending on the way it stores or retrieves blocks of data.

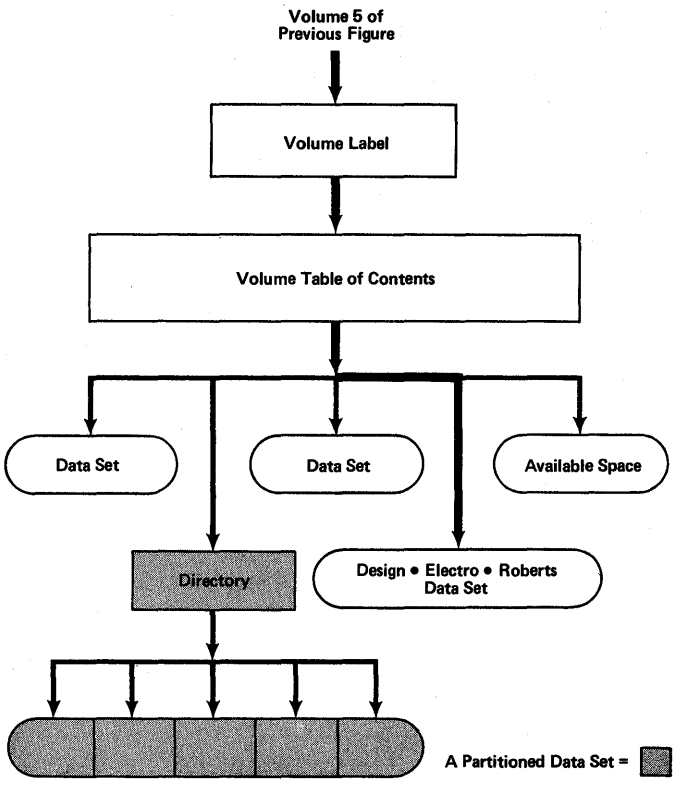


Figure 54. Simplified Diagram of Catalog System For Locating Data Sets Within a Direct Access Volume

Because a sequential access device, such as a magnetic tape unit, stores or retrieves a particular data block in a series, it must scan through all intervening data blocks. This is both time-consuming and wasteful of resources. Therefore, sequential access devices are seldom used when it is necessary to retrieve or store particular blocks of data in a non-sequential order. However, they can be used with a great deal of efficiency when storing or retrieving a succession of data blocks in a fixed sequence.

A direct access device can retrieve a particular block of a data set more directly. Therefore, it can be used in applications where it is necessary to retrieve or update particular data blocks in a non-sequential order. A direct access device can also store and retrieve a continuous series of data blocks, often at a faster rate than a sequential access device. Therefore, a direct access device is frequently used for both direct and sequential processing of data blocks.

To help an installation use these devices effectively, the operating system

provides five basic methods for storing and retrieving data:[10]

- The Basic Sequential Access Method (BSAM), which stores or retrieves data blocks in a continuous sequence using either a sequential or direct access device.

- The Basic Direct Access Method (BDAM), which directly retrieves or updates particular blocks of a data set on a direct access device.

- The Basic Indexed Sequential Access Method (BISAM), which directly retrieves or updates particular blocks of a data set on a direct access device, using an index to automatically locate the data set. The index is stored in direct access storage along with the data set. Other forms of the BISAM method can be used to store or retrieve blocks of the same data set in a continuous sequence. This method is used only with direct access devices, and takes advantage of both their sequential and direct access characteristics.

- The Basic Partitioned Access Method (BPAM), which is usually used to store or retrieve programs. It is described in the next section "Program Development and Management."

- The Basic Telecommunications Access Method (BTAM), which is used only in teleprocessing applications. This access method is described in the section "Teleprocessing."

The proper method to use in processing data depends on the nature and organization of the data and the nature of the application. For example, the indexed sequential method could be used effectively with data that must be updated and interrogated quickly, and periodically processed in a sequential order (to prepare a summary report, for example).

More than one method can be used to process the same data set, provided the organization of the data meets the basic requirements for each of the methods used. For example, the sequential access method could be used to transcribe a data set that is usually processed using the direct or indexed sequential access method.

In addition to the basic access methods, the operating system provides extended versions of the sequential access method, the sequential form of the indexed sequential access method, and the telecommunications access method. These are:

- The Queued Sequential Access Method (QSAM).
- The Queued Indexed Sequential Access Method (QISAM).
- The Queued Telecommunications Access Method (QTAM).
- The Telecommunications Access Method (TCAM).

QTAM and TCAM are discussed in the section "Teleprocessing."

When QSAM or QISAM is used, the system takes advantage of the fact that they store and retrieve blocks and records of a data set in a known sequential order. It does this by forming in main storage a waiting line, or queue, of input data blocks that are awaiting processing. For output data, it forms a queue of data blocks that have been processed and are awaiting transfer to auxiliary storage or an output device. The queues enable the CPU to process one data block while other blocks are being transferred in or out of main storage.

Data records from a block in an input queue are parcelled out, one at a time, as they are requested by the program that processes the data. If an input queue is empty when a record is requested by a program, then the control program automatically holds up further execution of the program until the next data block is received.

When a request is made by a processing program to store a record, the record is consolidated in an output queue with the other records of the block. If the output queue is full when the request is made, the control program automatically holds up further processing of records until a block in the output queue has been transferred.

Each basic or queued access method has a number of optional or alternative variations, such as different ways of transmitting data records to or from a processing program. The options and alternatives differ with each method.[9,10]

Defining Data, Access Methods, and Devices

In order to use an access method to create a new data set, the control program must have three types of information. First, it must have information about the data set, such as the lengths of the records and blocks that it will contain. It must also have information on the access method that will be used to create the data set, including a definition of all the selected optional and alternative variations of the

method, such as the size and organization of the output queue. Then it must know the type or class of I/O or storage devices on which the data set will be recorded.

When using the assembler language, the programmer can define this information within his program at the time he designs it, or he can define it in a data definition (DD) statement of a job step definition just prior to submitting his job for processing. (As previously described, each job step definition must contain a data definition statement for each data set that is used or created during the defined job step.) There are a number of reasons why the programmer may want to defer defining information until he submits his job.

First, by withholding information on the characteristics of the data set and the method to be used to create it, he can change characteristics and access methods each time the program is executed. He can do this merely by changing specifications on the data definition statement before submitting the job for processing, without changing or reassembling the program. Thus, a program that is used to create a data set can be independent of the method used to create it, and the access method can be varied from time to time depending on circumstances. For example, the size of the output queue used to create a data set can be changed depending on how much main storage space is available when the program is executed.

Deferring the definition of the device on which a newly created data set is to be stored or recorded offers a number of important advantages. Each time the program is executed, the data set that is created can be stored or recorded on a different type or class of device. For example, before submitting his job the programmer can indicate on the DD statement whether he wants the data set printed on an online printer or recorded on magnetic tape, on a particular direct access volume or device, or anywhere in direct access storage. This ensures that the program is independent of any particular type of device.

Thus, if a particular type of device is not available when needed, another type may be selected. Also, new types of devices may be added to the system without necessarily changing or reassembling the programs. Device independence makes it possible to run the programs on different computing systems having different complements of I/O and direct access storage devices.

When the programmer uses the assembler language to design a program, he must include in the program any information that is needed to apply an access method. When he uses a high-level language, such as FORTRAN, to design a program, the translator provides information and places it in the program. However, the high-level language translator normally withholds certain information in order to ensure a degree of device independence and flexibility in the use of the program. The withheld information is supplied in the data definition statement of the job step and can be redefined by the programmer.

A data set and the access method employed to create or use it can have a great many variations. Defining and coding their characteristics can be a rather lengthy and painstaking process. Therefore, the control program contains facilities that can be used to avoid repetitive definitions and coding each time a data set is created or used.

One is a facility for cataloging job step and data definitions. If a series of job step and data definitions are used repeatedly by programmers, they can be stored and cataloged in a procedure library maintained in direct access storage by the control program. Thereafter, a programmer can, using a single job and job step statement in an input stream, direct the job scheduler to pick up the job step and data definitions from the procedure library. If necessary he can, in the same statement, temporarily override specifications in the job step and data definitions picked up from the procedure library. Thus a programmer need only define and code changes to a data definition that are required for his particular job.

To further reduce repetition in defining data sets, the control program stores a description of each data set contained within the system.[11] When a new data set is created, its description is stored on the same volume that contains the data set. If it is a magnetic tape volume, the description is stored in a data set label that precedes the data set. If it is a direct access volume, the description is stored in the table of contents of the volume. In either case, once a data set is created it can be used and processed by different programs without being described again each time it is used.

Program Development and Management

Today, most data processing installations have two outstanding problems. The first concerns a shortage of programming talent and a continual increase in the number and scope of new data processing applications. The second problem concerns the effect of machine malfunctions on a system. Because of these problems, many data processing installations find it difficult to cope efficiently with the work at hand. As a result, the overall productivity of an installation often suffers, and the introduction of important new applications is sometimes postponed indefinately. Consequently, various processing programs and recovery routines have been developed to cope with these problems.

A Unified Program Development System

To help combat such problems, the operating system provides a variety of facilities in the form of processing programs that are designed to reduce the time, expense, and manpower required to program new applications. For the most part, these facilities consist of the language translators and service programs, which were described briefly in a previous section. Although each of these is important in its own right, they are really designed to be used together. When combined with the facilities of the control program, they serve as a unified system for developing and managing the use of programs. The overall objective of such a system is to reduce the total time and effort required to program and maintain an application from the time it is conceived until it becomes obsolete.

A particular application may be a short one-shot affair such as computing the interest on a loan, or a large complex, long-running application that requires most of the resources of an installation. Therefore, the operating system is designed to assist groups of programmers working cooperatively on a single project or application, as well as individual programmers working independently on many different applications.

The operating system provides such assistance in four ways. First, it provides a method for designing and constructing programs and subprograms in the form of replaceable parts, or modules. This method, called modular construction, is the same basic method that is used to construct the operating system. Second, it provides means for storing programs and parts of programs in organized libraries in direct access storage where they are immediately available for automatic retrieval, yet are still subject to modification. Third, it provides means for dynamically loading programs and subprograms into main storage as they are required to perform tasks. Fourth, the Checkpoint/Restart facility provides the programmer with the means to prevent a complete restart of a program in case of an error.

MODULAR CONSTRUCTION

Although a programmer can write a program and have it translated and executed without a hitch, this is an exception, rather than the rule. Most programs are tested, modified, and recompiled several times before they are put to use in performing useful work. It may take months to completely develop and check out a program. Even then, it may go through an evolutionary process of improvement, extension, and updating that can, and often does, last until it is nearly obsolete.

Therefore, the language translators and the linkage editor are designed so that they can be used in combination to construct programs and subprograms in the form of modules that are logically interconnected but can be modified separately. This makes it unnecessary for a programmer to retranslate and test his complete program each time a part of it is modified. Only those parts (or modules) that are affected by a modification need be retranslated.

This method of construction also allows the work involved in developing a program to be divided up among several programmers (Figure 55). Many programs are much too large and complex for a single individual to design alone in a reasonable period of time. This is especially true of the large, complex programs that are usually required for advanced system applications.

A program module represents a complete program or part of a program that can be modified or replaced without affecting other programs. It may range in size from a single instruction to a large program that requires all of the available space in main storage.

There are three types of program modules: source modules, object modules, and load modules. Each of these represents a different stage in the development of a program.

A source module is the input to a language translator for a particular translation. It is actually a sequence of language statements which the programmer has decided to consider as a replaceable entity.

An object module is the output of a language translator for a particular translation. It contains a program or part of a program in the form of machine language instructions.

A load module is produced by the linkage editor. It can be produced from one or more object modules or a combination of object modules and other load modules.

A source module can contain references to instructions and data in other modules. It can also contain instructions and data that are referred to by other modules. Yet each source module can be translated individually, and not combined with other modules until sometime later. Therefore, as a part of the translation process, each translator prepares a record of all references to or from other modules. This record is represented by the interlocking arrows in Figure 55. It is appended to the object module produced by the translator and is used later by the linkage editor to consolidate the module with other modules.

To produce a load module, the linkage editor resolves all cross-references among the input modules from which the load module is to be formed. It does this by replacing each reference with the address of the item referred to. The linkage editor also produces and appends to the load module a consolidated record of all cross-references. These include cross-references that have already been resolved as well as unresolved references to modules that have yet to be incorporated into the load module. Because a load module contains this record, the linkage editor can be used to combine it with object modules or other load modules, and to delete or replace previously consolidated modules.[5]

After all cross-references have been resolved, a load module can be loaded anywhere in main storage by the control program. The final assignment of storage addresses is completed as part of the loading process.

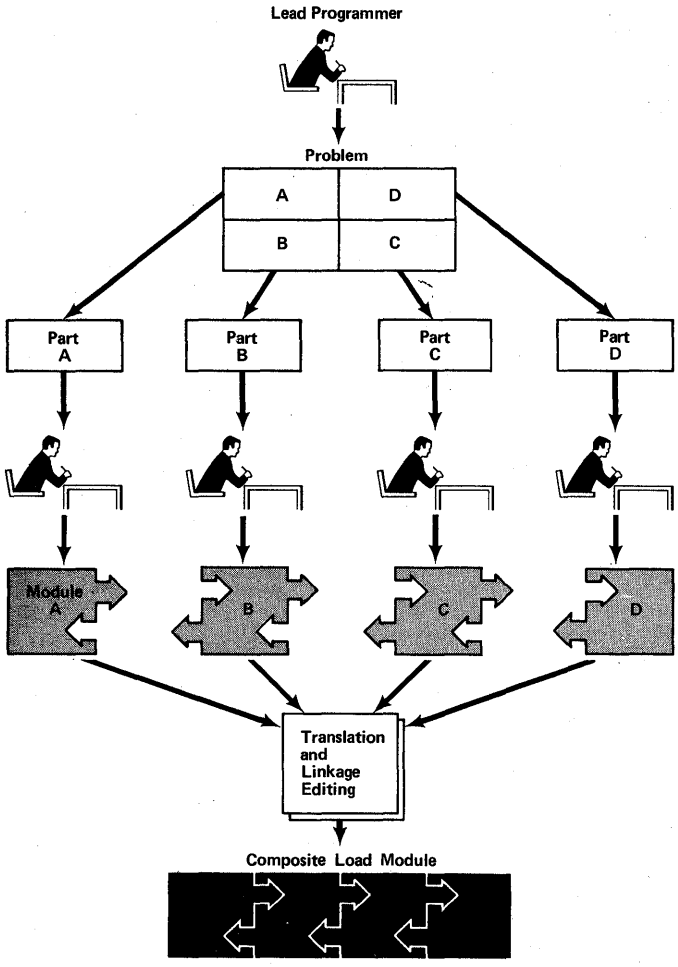


Figure 55. Program Design

There are several reasons why a programmer may want to divide and subdivide a program into modular parts:

- He may, sooner or later want to replace one part with another part. This may be done because the original part contained an error, or had to be updated or improved.
- He may want to develop and test different parts of his program at different times and gradually build a complete program.
- He may want to divide up the workload, that is, assign responsibility for developing and maintaining different parts of a program to different programmers.
- He may want to use different language translators to design different parts of his program. For example, he may want to use the FORTRAN language to design a part of his program that requires the use of mathematical

techniques, and the assembler language to design parts that require either a great deal of flexibility or services of the supervisor that are not available through the FORTRAN language.

A programmer should evaluate these considerations in deciding whether to divide his program into modules. As a general rule he should divide his program along functional lines into self-contained modules to reduce the number of cross-references between one module and another.

ORGANIZED PROGRAM LIBRARIES

In the operating system, programs or parts of programs are usually stored in program libraries maintained in direct access storage. These libraries include programs that make up the operating system, as well as programs created by the user of the system.

A program library is actually a special type of data set called a partitioned data set that is stored in direct access storage and can be cataloged like any other data set. A partitioned data set, as its name implies, is divided into independent partitions. Each partition contains a program, or part of a program, in the form of one or more sequential data blocks. Each program library contains a built-in directory (or index) that the control program can use to locate by name a particular program or part of a program stored in the library.

The control program provides an access method, called the partitioned access method, that the programmer can use to create program libraries for storing and retrieving programs. However, most applications programmers use libraries that have already been created by a system programmer or by the control program.

Using direct access storage for programs libraries has a number of distinct advantages. Once a program module is entered into the system, it can be stored, translated, tested, modified, retranslated, and combined with other modules, without storing it on punched cards. Thus, programmers need not maintain large, bulky card files and can avoid relatively slow card reading and punching operations. Another advantage is that programs can be loaded directly into main storage without searching through a long series of programs on magnetic tapes.

In developing and maintaining programs, three libraries are generally used: a source module library, an object module library, and a load module library (Figure 56). These can be used by individual programmers or shared by several programmers working on the same or different projects.

The source module library is used to store source modules while they are being translated and tested. There they are available for updating or correction, if necessary, prior to retranslation. IBM provides a comprehensive set of utility programs specifically designed for this purpose.

As each source module is translated, the resulting object module is stored in an object module library where it is available for processing, or consolidation with other modules, by the linkage editor. The load modules produced by the linkage editor are placed in a load module library. There they are available for loading into main storage and execution under direction of the control program, or for further editing and modification by the linkage editor. Once a load module is tested and fully perfected, a programmer can delete from the library system any source and object modules that were used in developing it. The load module can be further modified, extended, and updated, however, since a composite record is maintained within the module of all cross-references among the modules from which it is formed.

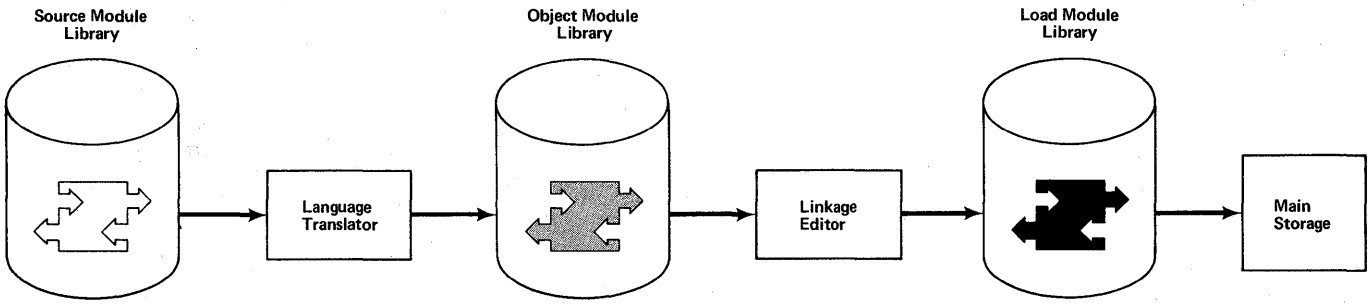


Figure 56. Program Module Libraries

In other operating systems, large or complex programs are often stored and maintained in auxiliary storage in two forms. In one form, final main storage addresses within the programs have been assigned. Therefore, the programs are ready to be loaded into main storage and executed. By the same token, the programs cannot be modified, short of complete retranslation, except by resorting to a makeshift technique called "patching." Patching is generally avoided at well-run installations because it leaves no visible record of a modification. Therefore, the same programs are stored and maintained in auxiliary storage in a different form. In this form, final main storage addresses within the program are not yet resolved; the various parts of the program are still subject to modification without resorting to either patching or complete retranslation. In the System/360 Operating System, a single composite load module is both subject to modification and ready to be loaded into main storage and executed. Therefore, only the load module form of a program need be stored and maintained in auxiliary storage.

DYNAMIC PROGRAM LOADING

Once a load module is perfected and stored in a program library, it is ready to be dynamically loaded anywhere in main storage by the control program, and then executed. This characteristic of a load module is commonly referred to as relocatability (Figure 57).[1] A load module may be a complete program or a part of one or more larger programs. This is true regardless of whether it was originally formed from a single object module or a combination of modules.

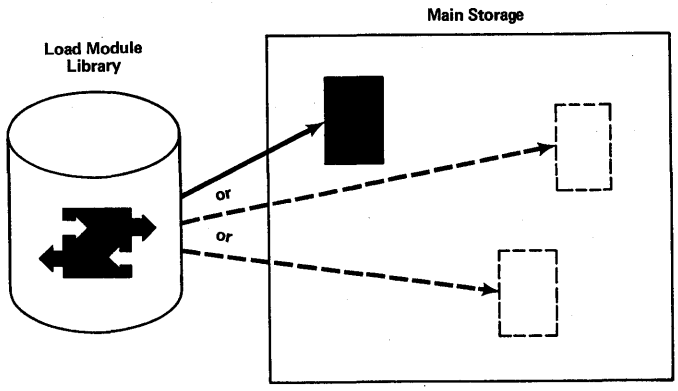


Figure 57. Relocatability

There are a number of reasons why a programmer may divide and store his program as separate load modules:

- He may want to share parts of his program with other programmers.
- He may anticipate that a particular part of his program will be deleted or replaced in the future.
- He may want to have major parts of his program executed as separate job steps, in which case data generated during one step may be passed to a succeeding step.
- He may want to load a particular part of his program into main storage only when a certain event occurs, such as the expiration of a time period or, in a teleprocessing application, the arrival of a message from a remote location.
- He may want to load different parts of his program into main storage at different times in order to conserve storage space. In such an instance, parts of a program that are yet to be executed can be brought into main storage to replace parts that have already been executed.
- If he is using an MVT operating system, he may want to have different parts of a program executed concurrently as separate and distinct tasks.

The job scheduler and the problem-state programs whose execution it initiates are separate and distinct programs. However, they and their subprograms must adhere to standard linkage conventions so that control can be passed between them in a consistent manner. As a result, any programs and subprograms that are stored in the load module libraries can be shared and used for different applications. These include programs and subprograms of the operating system itself as well as those designed by a user.[5]

The degree of program sharing depends largely on the installation and how well the use of the system is controlled and coordinated. The sharing might be limited to a few programmers or applications or it might extend to all of the programmers at an installation.

CHECKPOINT/RESTART FACILITY

When an error is detected within the computing system an interruption normally occurs, and CPU control is transferred to an error recovery routine, which attempts to repeat the operation that was being performed when the error was detected. If the error does not recur, it is assumed that the error is not permanent, and CPU

control is returned to the production program at the point at which it was interrupted. Thus, if it is possible to recover from the error, no time is wasted in the middle of a production run in correcting the fault that caused the error.

In some cases however, it is not possible to continue the production run at the point at which it is interrupted by an error. The error may have been caused by a permanent failure or it may have altered or destroyed data or instructions that are required to continue with the production run. To avoid restarting the production run from the very beginning when such situations arise, an optional Checkpoint/Restart facility is provided with the operating system. Using this facility it is possible to design production programs, particularly long ones, with convenient rerun points (checkpoints). A programmer initiates a checkpoint by requesting the supervisor, via an SVC instruction, to record in auxiliary storage a checkpoint data set containing all of the information necessary to restart the production program from the point at which the checkpoint is taken. Then, no matter where processing is interrupted during the production run, it can be restarted at the last checkpoint using the checkpoint data set to reconstitute main storage, load registers, position tapes, etc.

This technique can also save time when a program is interrupted by operator intervention for another of higher priority or for any other reason. At the option of the operator, a restart can be either automatic or deferred. If automatic, the production program is restarted immediately at the last checkpoint. If deferred, the production program is resubmitted sometime later for processing, and is restarted then at the last checkpoint.

Using the Checkpoint/Restart facility, it is also possible to restart predesignated job steps in the event a job step is interrupted and cannot continue. Like a checkpoint restart of a production program, a step restart can be automatic or deferred at the option of the operator. The EXECUTE statement of the job step definition indicates whether a job step is subject to restarting.[2]

A Means of Recovery

A failure of the system, whether during the development of new programs or while processing MFT or MVT jobs, can result in a loss of productivity and diminish the effectiveness of the system. To protect against, or at least to diminish the effects of, a failure, reliability, availability, and serviceability (RAS) facilities interact with the control program. RAS facilities attempt to retry or repair machine malfunctions that result in system failure. One means available for RAS implementation is the Recovery Management Support (RMS) for both System/360 and System/370.[6,7]

Recovery management routines fall into two distinct categories: Those that record the environment at the time of the machine malfunction; and those that exist to bypass various I/O errors. Recovery management routines of the first class include:

- System Environment Recording (SER) routines provide the user with the environment of the CPU and the channel at the time of the failure.
- Machine Check Handler (MCH) routines, according to the CPU model, analyze, record, retry, and if possible isolate the machine malfunction.
- Channel Check Handler (CCH) analyzes channel errors for device-dependent error routines, and constructs a permanent record -- the channel inboard error record -- for the I/O Supervisor.
- Error Recovery Procedures (ERPs) analyze intermittent and unrecoverable errors, detect I/O errors, and attempt their retry.

Recovery management routines of the second class include:

- Alternate Path Retry (APR) allows an I/O operation that has failed on one channel to be retried on another alternate channel.
- Dynamic Device Reconfiguration (DDR) allows the operator to move a movable volume -- tape drive or disk -- at system or operator request, to another device.

Multiprocessing

Multiprocessing is a technique whereby the work of processing data is shared among two or more interconnected central processing units (or computing systems). A combination of a main computing system that specializes in the processing of jobs, and a separate offline "satellite" computer that specializes in transcribing input and output data for the same jobs might be considered a multiprocessing system since the combination contains two central processing units that indirectly communicate with one another. However, in this kind of multiple CPU installation, the communication between one CPU and another is achieved wholly through operator intervention. In the case of the offline satellite example, the communication consists of the operator removing a tape reel from a tape unit connected to one computer and mounting it on a tape unit connected to the other computer. Therefore, such a combination is not usually considered a true multiprocessing system.

CPU-to-CPU Communication

In a true multiprocessing installation, one CPU may communicate with another in a combination of ways. At one extreme, communication may be represented by a few control signal lines that are used to broadly synchronize the operation of one CPU with that of another. Then again, the communication may consist of sharing direct access storage devices or main storage units among two or more CPUs. Another form of communication can be achieved by using a channel-to-channel adapter. This device enables blocks of data to be transferred quickly from the main storage of one CPU to that of another. Communication can also consist of transferring data over telecommunication lines from one CPU to another at a remote location.

Advantages of Multiprocessing

There are a number of reasons why multiprocessing might be employed at an installation. Multiprocessing can increase availability, increase production capacity of a system, ensure more efficient use of resources, and allow two or more CPUs to share the same data.

INCREASED AVAILABILITY

Multiprocessing can help to ensure a high level of availability for a system. Availability is the degree to which a system is ready when needed to fulfill its role in an activity. An example of such an activity is a missile launching, where the unavailability of a computing system for use in guiding the missile could delay launching or cause it to fail. For such applications, more than one CPU is employed so that one can quickly replace another in the event of a failure. Some other applications in which availability is of prime importance are: airline reservation system applications, process control applications, banking system applications, and air or ground traffic control system applications.

INCREASED PRODUCTION CAPACITY

Multiprocessing can also increase the overall data processing capacity of a system. This is especially important for solving large scientific or engineering problems in such fields as theoretical physics or aircraft design. Problems such as these often require weeks or months of computation before a solution is arrived at. The increased data processing capacity provided by multiple processing units can drastically reduce the time required to solve such problems.

MORE EFFICIENT USE OF RESOURCES

In systems that can perform multiple data processing tasks concurrently, multiprocessing can result in more efficient use of hardware resources. By pooling the resources of two computing systems, it is sometimes possible to perform more work, such as job steps, concurrently than when two separate and distinct systems are employed. With separate computing systems, a job or other unit of work that is ready to be initiated on one system may be delayed for the lack of a resource that is available, but idle, on the other system.

DATA SHARING

Multiprocessing can make it possible for two or more central processing units to share sets of data maintained in direct access storage. This can help to ensure more efficient and consistent processing of the data. Moreover, by reducing or eliminating redundant data, it can conserve

direct access storage space and reduce the time and effort required to retrieve the data and keep it up to date.

Operating System Support of Multiprocessing

The IBM Operating System/360 supports two general purpose multiprocessing applications. One employs two to four System/360 Computing Systems that share direct access storage devices. The other employs two computing systems that share all of main storage and most I/O devices.

In addition to these two general purpose applications, there are a number of special purpose multiprocessing applications supported by the operating system. These involve the use of central processing units at remote locations that communicate with one another by way of telecommunication lines, and are therefore described in the next section "Teleprocessing."

MULTIPROCESSING WITH SHARED DIRECT ACCESS STORAGE DEVICES

As an optional feature, the operating system supports a multiprocessing system containing two System/360 Computing Systems (Figure 58) of the same or different models that share a control unit and up to eight direct access storage devices. Access to a particular device is gained through a two-channel switch that enables the shared control unit to be instantaneously switched between two channels. Each of the two channels is connected to a different computing system. The computing systems gain access to a particular device on a first come first served basis. Either of the major configurations of the control program (MFT or MVT) can be used with either of the two computing systems.

MVT WITH MODEL 65 MULTIPROCESSING

The operating system supports the Model 65 Multiprocessing System through MVT with Model 65 multiprocessing, an extension of MVT. This system uses two identical Model 65 Processing Units (CPUs), called Model 65 Multiprocessors. The two CPUs share all of main storage, which may range from a minimum of 524,288 bytes to a maximum of 2,097,152 bytes, and most I/O devices.[7]

One configuration of the Model 65 Multiprocessing System is shown in Figure 59. In this configuration, the system is physically symmetrical, excluding the 1052 Printer-Keyboards; that is, each CPU has access to any I/O device in the system, except the 1052 attached to the other multiprocessor. This total device accessibility is made possible through use of program-controlled, two-channel (or two-processor) switches on control units. The 2816 Switching Unit and the 2844 Auxiliary Storage Control are also used to achieve total device accessibility, in certain instances.

In other configurations of the Model 65 Multiprocessing System, in addition to the 1052 Printer-Keywords, 1443, 2150, and 2501 unit record equipment and devices supported for graphics and teleprocessing applications can be accessed by only one CPU.

Operating Modes

The model 65 Multiprocessing System can operate in three modes: multisystem (MS), partitioned (PTN), and 65 mode. MVT with Model 65 multiprocessing supports multisystem and partitioned modes only.

Multisystem Mode: When the Model 65 Multiprocessing System operates in multisystem mode, both CPUs share all of main storage as though it were a single unit. They also share most I/O devices.

A single supervisor allocates resources and apportions work between the two CPUs. Because two CPUs share the workload, two entirely different tasks (or two parts of the same task) can be processed simultaneously. Figure 60 shows how four tasks can be handled by two CPUs in multisystem mode.

The control program synchronizes the operation of both CPUs through the Direct Control Feature -- control lines over which the CPUs can communicate -- and through use of a malfunction alert signal. Examples of situations in which activity must be synchronized are:

- One CPU is processing a task and a higher-priority task is ready and waiting to be processed.
- I/O activity cannot be initiated by one CPU; therefore, a check must be made to see if the other CPU can initiate the activity.
- An error condition exists in one CPU.

One requirement for programming support of the Model 65 Multiprocessing System is Recovery Management Support (RMS). The RMS routines help reduce delays, loss of data, and other effects caused by intermittent or persistent hardware failure. If a machine error occurs, the RMS routines attempt to recover from it by retrying the operation that failed and attempt to repair any program damage resulting from the error. If the error is persistent, then the recovery routines alert the operator and provide information to help locate the faulty component.

In most single-CPU systems, when a failure occurs in an I/O device, that device can be placed offline; that is, its use as a system resource can be discontinued. The system can continue to operate. In the Model 65 Multiprocessing System, any noncritical malfunctioning component (one CPU, channels, areas of main storage in multiples of 2048 bytes, and I/O devices) can be placed offline. Thus, the rest of the system can continue to function. This assures a high degree of availability.

Partitioned Mode: When a CPU is operating in partitioned mode, it must have its own main storage, with a minimum of 512K bytes; auxiliary storage; control units; and I/O devices. In this mode, the CPU operates as a separate and distinct system under MVT with Model 65 multiprocessing.

65 Mode: As with partitioned mode, a CPU operating in 65 mode must have its own main storage, with a minimum of 256K bytes; auxiliary storage; control units; and I/O devices. In this mode, the CPU operates as a separate and distinct system under the MFT or MVT configuration of the control program (excluding MVT with Model 65 multiprocessing).

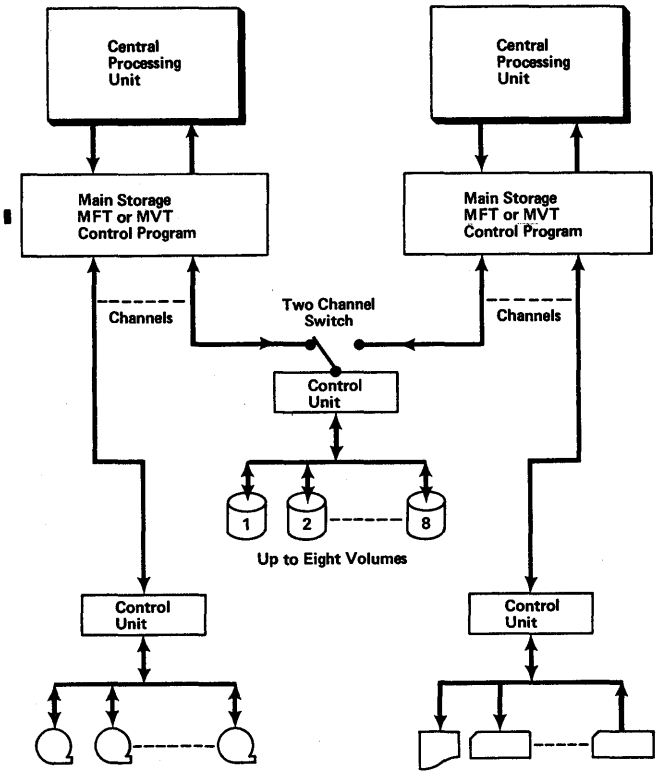


Figure 58. Multiprocessing With Shared Direct Access Storage Devices

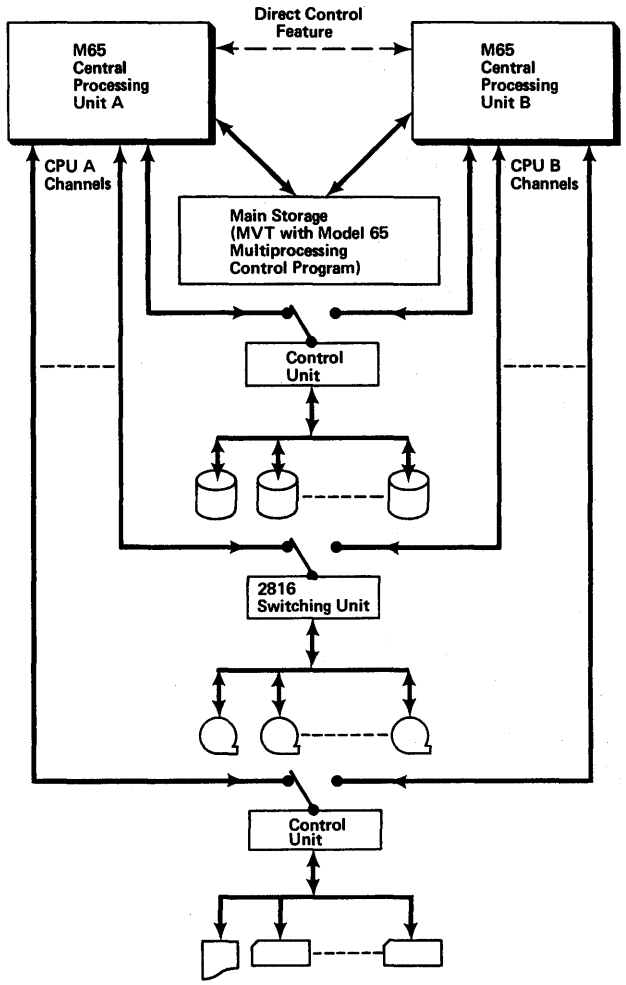


Figure 59. A Symmetrical Configuration of the Model 65 Multiprocessing System

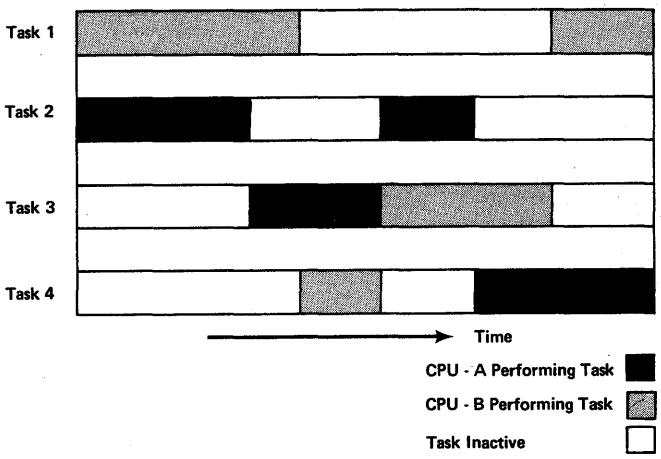


Figure 60. Two CPUs in Multisystem Mode, Balancing the Execution of Four Tasks

Teleprocessing

Teleprocessing refers to a large variety of data processing applications in which data is received from or sent to a central data processing system over communication lines, including ordinary telephone lines. Usually the source or destination of the data is remote from the central processing system, although it can be in the same building. In any event, the source or destination points of the data are often called terminals or (for some applications) work stations.

A terminal, or work station, can have one or a combination of I/O devices. A large variety of such devices are available for use at remote terminals. These include special keyboards, TV-like graphic display devices, printers, card read-punch units, and telephones. In addition, a remote terminal may be represented by another data processing system, in which case the application is not only a teleprocessing application but a multiprocessing application as well.

Teleprocessing applications range from those in which data is received by a central processing system and merely stored for later processing, to large complex system applications in which the hardware and information resources of the central system are shared among a great many users at remote locations.

General Types of Applications

Several general types of teleprocessing applications that are possible with the operating system are briefly described below. There are a number of variations and combinations of these general applications.

DATA COLLECTION

Data collection is a teleprocessing application in which data is received by a central processing system from one or more remote terminals and is stored for later processing. Depending on the specific application, the transfer of data may be initiated either at the terminal or by the central processing system.

An example of a data collection application would be one in which data is received intermittently during the day (as it is generated) and is processed, when convenient, during the second or third shift, perhaps taking advantage of lower data processing rates for the shift. This could be an application in which production workers, upon completion of their jobs, transmit by means of special input devices such data as their ID numbers, the number of work units they completed, and other pertinent data. The central system, after all the data for the day has been collected and stored, could then process it for accounting and production control purposes.

In other applications, data may be accumulated during the day and then placed on an input device, such as a punched card reader. The data could be collected by the central computing system during off-peak hours in order to take advantage of lower communication line rates.

MESSAGE SWITCHING

Message switching is a type of teleprocessing application in which a message received by the central computing system from one remote terminal is sent to one or more other remote terminals. Message switching can be used in a nation-wide or world-wide telegraph system or it can be used by a geographically dispersed business or scientific enterprise to provide instantaneous communication within the enterprise.

REMOTE JOB PROCESSING

Remote job processing is a type of application in which data processing jobs, like those that are entered into the system locally, are received from one or more remote terminals and processed by the operating system.

Two applications of remote job processing are provided as optional features of the MFT and MVT configurations of the operating system control program. These are described later in this section under "Remote Job Entry" and "Conversational Remote Job Entry."

TIME SHARING

Time sharing is a teleprocessing application in which a number of users at remote terminals can concurrently use a central computing system. In this type of application each user at a terminal has the

impression that he is the sole user of the computing system. In reality, however, the resources of the system are shared among several users. When the use of the computing system is momentarily not required by one user, it is available to satisfy the needs of others. Because of its speed, the computing system can respond to the needs of all the users within a few seconds.

Often, in this type of application, a dialogue or conversation is carried on between the user at a remote terminal and a program within the central computing system. The program may be designed to interrogate the user and immediately respond to his replies or requests, or even his mistakes.

General purpose time sharing is provided as an optional feature of the MVT control program. It is described later in this section under the title "Time Sharing Option."

ONLINE PROBLEM SOLVING

Online problem solving is a form of time sharing that has a great many potential applications in the fields of education, engineering, and research. Because the system can respond quickly to the needs of the user, it can directly participate in, and speed up, the problem solving process as well as other similar processes such as program design and learning. Thus, if a user, in the course of designing a program, makes a mistake, he may be immediately alerted by a program in the central computing system to take corrective action. Therefore, he need not wait until the complete program is compiled and tested before the mistake is detected. Similarly, in a computer assisted instruction (CAI) system - an important variation of this type of application - a student is immediately informed of, and learns from, his mistakes as he makes them.

A specific application of online problem solving is provided as an optional feature of the MFT and MVT configurations of the control program. This application also has certain features of a remote job processing application. It is described later in this section under the heading "Graphic Job Processing."

INQUIRY AND TRANSACTION PROCESSING

Inquiry and transaction processing is a teleprocessing application in which inquiries and records of transactions are received from a number of remote terminals and are used to interrogate or update one or more master files maintained by the central computing system. With this application, the system can directly participate in and control various commercial, scientific, and military activities as they are being carried on.

One of the earliest examples of this type of application is the airline reservation system application described previously. A similar application is one in which the system is used to service a geographically dispersed banking activity. In such an application, master files containing account records for thousands of depositors are stored in direct access storage. By entering pertinent data into the system, tellers at remote locations can use a special I/O device to check balances, update passbook records, and handle similar transactions within a matter of seconds.

Other examples of this type of teleprocessing application are information retrieval systems, management information systems, and inventory control systems.

An intriguing variation of an inquiry and transaction processing application is one in which a telephone is used as the sole means of input and output at a terminal. Although the telephone is normally used to communicate with people, it can also be used to communicate with a central computing system. A simple example of this would be one in which a salesman wished to check the delivery time for a particular product. He could do this, even while in a customer's office, by first dialing the central computing system, and then entering (by dial or touch buttons) a transaction code and the stock number of the product. After interrogating the master file to determine the delivery time for the product, the computing system could then, using an audio response device, such as the IBM 7770 Audio Response Unit, compose and return a verbal message to the salesman informing him of the delivery time for the item. The salesman could then discuss this with the customer and transmit the order to the central system or make other inquiries, again by way of the telephone. Thus the services of a central computing system can be as close as the nearest telephone.

Message Control and Message Processing Programs

A teleprocessing program for most applications is normally divided along functional lines into two parts: a message control program and one or more application programs (traditionally know as message processing programs). Message is the traditional name for a unit of information that is transferred to or from a remote terminal by way of telecommunications lines. A message may consist of one or more segments. A single-segment message is usually composed of two parts: the message header followed by the message text. The message header contains control information concerning the message, such as the source or destination code of the message, the message priority, and the type of message. The message text consists of the actual information that is routed to a user at a terminal or to a program in the central computing system that is to process it. In general, the information in the message header is used for control and routing purposes by a message control program, and the information in the message text is processed, if necessary, by an application program.

MESSAGE CONTROL PROGRAMS

The main function of a message control program is to control the transmission of information between an application program in the central computing system and I/O devices at remote terminals. In this respect it performs much the same function as access method routines that are used to control the transmission of information between an ordinary processing program and local I/O devices. For this reason, routines that are provided by IBM for use in creating a message control program are also called access method routines. There are three sets of such routines: the queued telecommunication access method (QTAM), the telecommunications access method (TCAM), and the basic telecommunication access method (BTAM). Although they are called access method routines, they differ from other access methods routines in a number of respects, especially in the way in which they are assembled to form a composite set of routines for controlling the transmission of I/O information. The access method routines for an ordinary processing program are assembled and linked together by data management routines in the operating system control program as they are requested through the use of an OPEN macro instruction. The routines that make up a message control program on the other hand, are assembled by an assembler language translator as a separate program when the message control program is created, in much the same way as the operating system control program is generated.

Queued Telecommunications Access Method

The queued telecommunications access method (and the telecommunications access method, described following this explanation) can be used to create message control programs for a variety of teleprocessing applications ranging from message switching or data collection to high volume inquiry and transaction processing.

To design a message control program for such applications can be a very difficult and time consuming undertaking requiring a specialized knowledge of teleprocessing equipment and techniques. Therefore, to simplify and speed the creation of a message control program, IBM provides a special message control language in the form of assembler language macro instructions. These macro instructions can be used to select specific modules from a comprehensive set of message control and editing modules. These modules can be adjusted to meet specific needs and then linked together to form a complete message control program. The macro instructions relieve the programmer assigned to the teleprocessing application of the detailed, intricate, and specialized programming that is usually required for such an application. They are specifically designed for easy use in describing the communication line procedures, line configurations, buffer lengths, polling (terminal interrogation) procedures, and types of message translation and editing required for a particular application. Using the message control macro instructions a complete message control program for a teleprocessing application can be described and assembled in days rather than months.

The message control program serves as an intermediary between the I/O devices at remote terminals and the application programs that process messages (Figure 61). It enables the terminals to be referred to indirectly, in much the same way as local I/O devices are referred to, using such standard macro instructions as GET, PUT, OPEN, and CLOSE. It automatically performs detailed functions, such as sending or receiving messages, allocating buffers, translating message codes, formatting messages, and checking for errors.

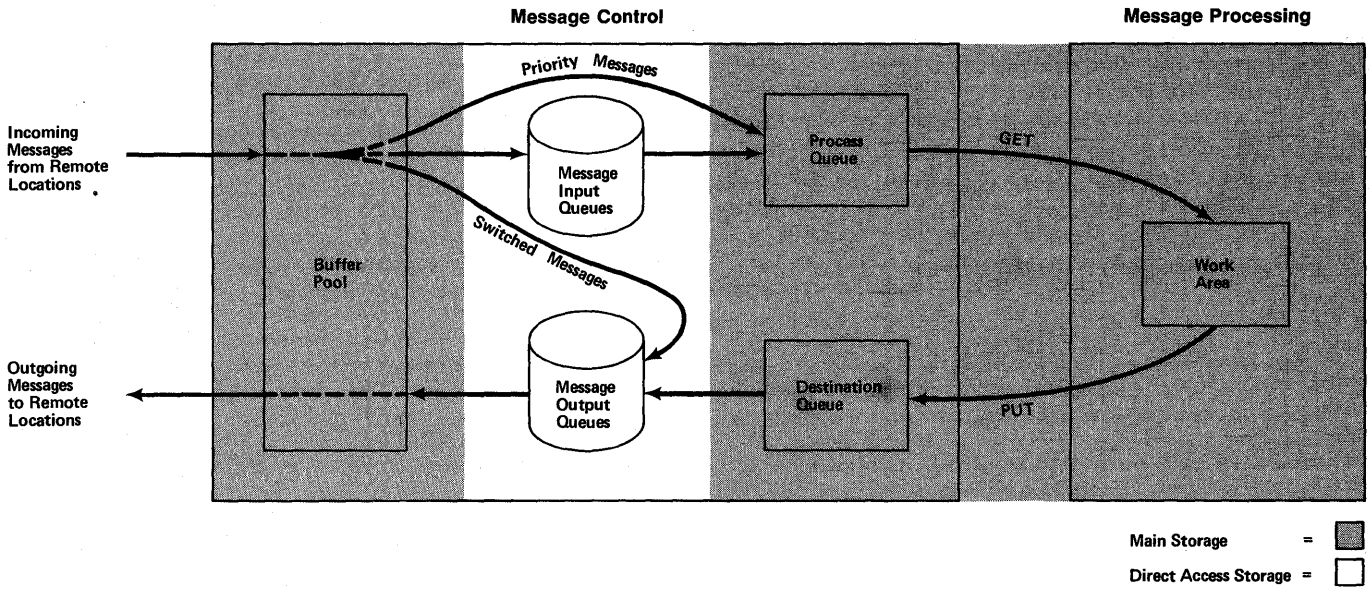


Figure 61. Simplified Diagram of Message Control Using the Queued Telecommunication Access Method

A message control program can be executed as a separate task independently of any application program. As input messages are received, they are routed (after translating, checking, editing, etc.) to one or more message queues in main storage or direct access storage. Application programs take them from there as in ordinary processing. When a message is to be sent to a terminal by an application program, it is placed on an output queue in direct access storage. (The telecommunications access method places messages on a destination queue in either direct access storage or main storage.) The message is then sent by the message control program to its destination. In the case of message switching and data collection applications, a special purpose application program may not be required: the message control program can route an inbound message directly to an appropriate output queue. A telecommunications job can be entered into the system in the same way as any other job. The job scheduler of the operating system, therefore, can be used to allocate any I/O device and direct access storage space required for message logs and message queues, and to prepare and schedule the job for processing. A message control program, and any application program associated with it, can be entered into the system as separate jobs; or they can be combined and entered as a single job.

With MFT and MVT operating system configurations, more than one job can be run concurrently. Therefore, other jobs can share the physical resources of the system with a teleprocessing job and thereby improve efficiency, especially during periods when message traffic is low.

Telecommunications Access Method

The telecommunications access method (TCAM) is similar to QTAM, but offers a wider range of device and program support. For example, TCAM supports local terminals connected directly to the computing system, as well as remote terminals connected through communication lines. For remote terminals, TCAM supports both the start-stop and binary synchronous methods of data transmission; binary synchronous support permits the use of faster terminals than are available with QTAM. In fact, with TCAM, a terminal may be an independent computing system -- another System/360 or an IBM 1130.

The preceding discussion of QTAM applies generally to TCAM. However, a TCAM application program can use either GET and PUT, or READ and WRITE macro instructions to receive and send messages.

To take advantage of TCAM facilities, QTAM application programs can easily be converted to TCAM. TCAM facilities include:

- Online testing of teleprocessing terminals and control units.
- Input/output error recording.
- Program debugging aids.
- Network reconfiguration facilities.

The facilities for network reconfiguration permit great flexibility in controlling the telecommunications network. The network can be modified by the systm operator, by a user at a terminal, or by an application program.

Basic Telecommunications Access Method

The basic telecommunications access method (BTAM) is designed for limited applications that do not require the extensive message control facilities or QTAM or TCAM, or for applications that require special facilities not normally found in most applications.

The BTAM facilities provide tools that would be required to design and construct almost any teleprocessing application. These include facilities for creating terminal lists and performing the following operations:

- Polling terminals.
- Answering.
- Receiving messages.
- Allocating buffers dynamically.
- Addressing terminals.
- Dialing.
- Creating buffer chains.
- Changing the status of terminal lists.

When the basic telecommunications access method is used, READ and WRITE macro instructions, rather than GET and PUT, are used by an application program to retrieve and send input and output messages.

MESSAGE PROCESSING PROGRAMS

A message processing program is an application program that processes or otherwise responds to messages received from remote terminals. In designing the program, all of the facilities of the operating system are available including the language translators, service programs, and the data, program, and task management facilities of the system. The processing of messages can be performed sequentially as a series of single tasks or more than one message can be processed concurrently. In many applications, a message processing program requires access to data or routines stored in local direct access storage. In such applications it is possible to process several messages concurrently as separate tasks. As the processing of one message is delayed while access is being gained to direct access storage, another message can be processed. By processing several messages concurrently, the total message throughput of the system can be significantly increased. Since many of the messages in such applications require identical processing, a single reenterable program in main storage can be used to perform each of several concurrent tasks, and thereby save main storage space and program loading time. The general purpose task management facilities of the MVT control program are particularly appropriate to this type of application. They allow the system to be used for many high-message-volume applications that would otherwise be impractical without a specially designed control program.

Basically the same I/O macro instructions (OPEN, CLOSE, GET, PUT, READ, WRITE) are used in a message processing program (when using QTAM, TCAM or BTAM) as are used in other application programs. Therefore, application programs can be designed more or less independently of the devices that are used to transmit data, whether local or remote. The system can make a gradual transition from processing work entered locally, to processing work received from remote locations, with a minimum of disruption.

Specific Teleprocessing Applications Provided by IBM

IBM has designed a number of specific teleprocessing applications and has made them available as optional features. These are applications that are of interest to a significant number of customers. To date, they include the following:

- Remote job entry.
- Conversational remote job entry.
- Time sharing.
- Graphic job processing.
- IBM System/360 - 1130 data transmission for FORTRAN.

Except for time sharing, these are applications of the basic telecommunications access method (BTAM). Time sharing is an application of the telecommunications access method (TCAM).

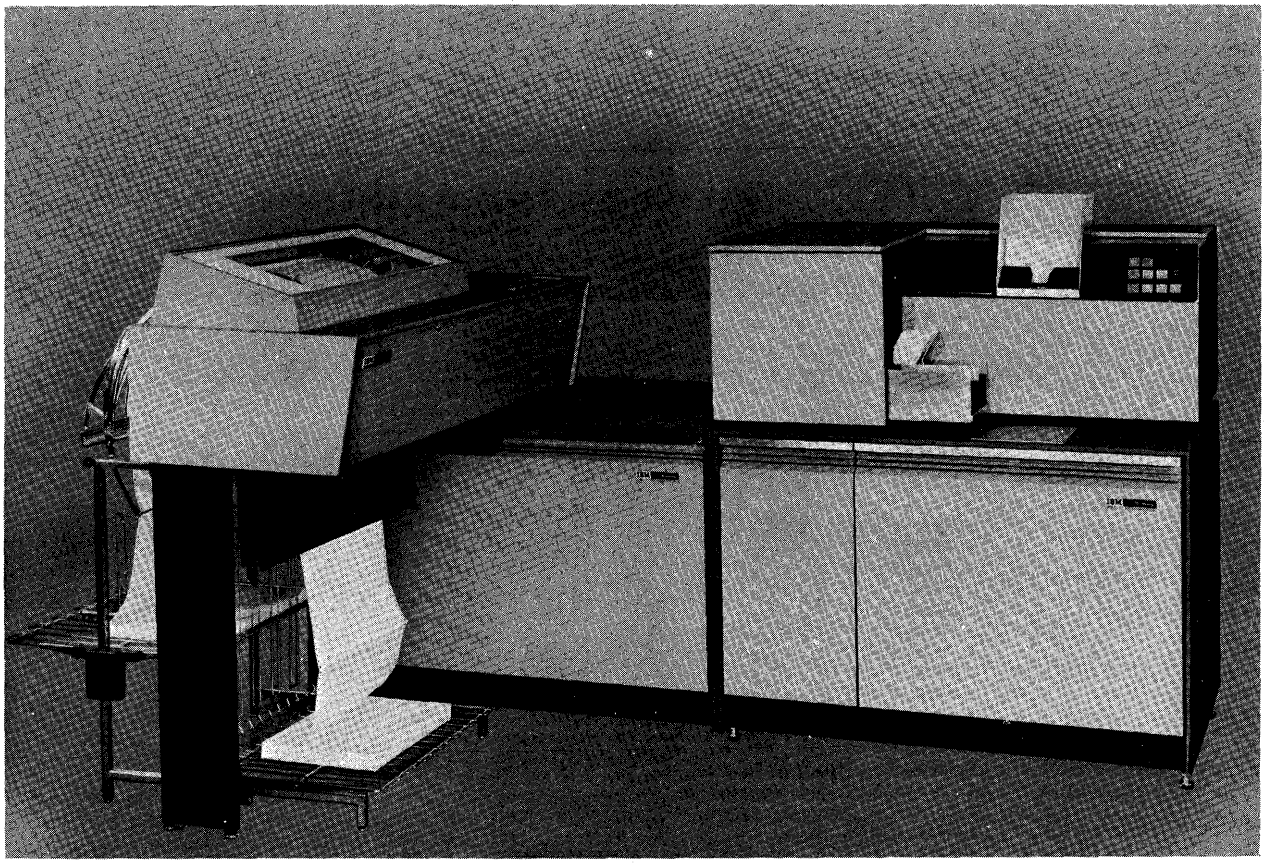


Figure 62. IBM 2780 Data Transmission Terminal

REMOTE JOB ENTRY

Remote job entry is an optional feature of an MFT or MVT operating system. It is a type of teleprocessing application in which jobs (like those entered into the system locally) are received from one or more remote locations. The jobs may be entered via such input devices as punched card readers and magnetic tape units. These may be attached to any of the following work stations:

- Another IBM System/360.
- An IBM 1130 Computing System.
- An IBM 2770 Data Communication System.
- An IBM 2780 Data Transmission Terminal (Figure 62).

Immediately or on command, output from a job can either be directed to the terminal from which the job originated, or transmitted to one or more other terminals. It can also be printed or otherwise processed by the operating system, or cataloged and stored (for later retrieval) in the operating system library. Data that is processed by the job can either be entered along with the job itself or can be retrieved from the system library. All of the operating system facilities that are available to the local programmer, such as the language processors, the service programs, and the data, job and task management facilities, are also available to the programmer at the remote location. Anything a programmer can specify locally he can specify at the remote location because the operating system is specifically designed for use at remote locations as well as for local use. The data cataloging and management facilities of the system, for example, enable individual programmers to compile, store, test, update, recompile, load, and execute programs within the confines of the operating system without resorting to the use of punched cards, or without specific knowledge of the I/O configuration of the system.

Jobs that are received from remote locations are placed by the remote job entry program into a job input queue in a format acceptable to the job scheduler. From there, the jobs are picked up and initiated by the initiator/terminator of the job scheduler in the same manner as for local jobs. The remote job entry program is executed by the central computing system as a separate task, much like a combined reader/interpreter and output writer.

CONVERSATIONAL REMOTE JOB ENTRY

Like remote job entry, conversational remote job entry is an optional feature of an MFT or MVT operating system. It enables remote users to enter jobs for batch processing, using terminals that resemble ordinary office typewriters. Users enter jobs conversationally, by carrying on a dialog with the central computing system.

Remote job input consists of programs and data that the user creates at a keyboard terminal. Typed lines of program source statements, data, and job control statements are collected within the system; there is thus no need for keypunching, and there is no wait for operator handling or card reading. Simple error correction procedures enable the user to enter data correctly and easily. Optional facilities are available for checking the syntax of FORTRAN and PL/I statements as they are entered, allowing errors to be corrected before the statements are compiled.

Because data is transmitted directly between the central processor and the terminal, job turnaround time is greatly reduced. To submit a job for execution, the user just selects the program, data, and job control statements that are to be entered in the job stream. When the job is completed, the user can examine the output at any terminal.

Remotely submitted jobs are initiated, executed, and terminated in the same manner as jobs that are submitted locally. A remote user thus has available the same batch processing facilities that are available to a local user. For example, a remote user can enter data from a terminal and have it stored at the central installation for use at a later date. Stored data can be retrieved easily for online display and modification, and can be used as job input to the operating system. A user can update stored data by inserting, replacing, deleting, or changing single typed lines or groups of lines. Stored data can be shared by many users, but is protected against unauthorized access or modification.

In addition to facilities for job preparation, job entry, retrieval of job output, and manipulation of programs and data, conversational remote job entry can provide the terminal user with information about the status of his data sets and the status of jobs that he has submitted. There is also a message facility for two-way communication between terminal users and the operator of the central computing system.

TIME SHARING OPTION

Time sharing is an optional feature of the operating system with MVT. The Time Sharing Option (TSO) makes the facilities of the operating system available to programmers at remote terminals to develop, test, and execute programs conveniently, without the job turnaround delays typical of batch processing. It gives those who may not be programmers the use of data entry, editing, and retrieval facilities. It also allows the management of an installation to dynamically control the use of the system's resources from a terminal.[14]

In general, a time-sharing system differs from a batch processing system in three ways:

1. A terminal user concurrently shares the resources of a computing system with other terminal users.

2. A terminal user can enter his problem statements and other input into the system as he develops them, and he receives results quickly.

3. A terminal user is constantly aware of the progress of his job. He is prompted for information the system needs to execute his job, he quickly receives responses to his requests for action, and he is notified immediately of errors the system detects, so that he can take corrective action at once.

TSO is not necessarily intended to be used as a dedicated time-sharing system, that is, a system on which only time-sharing operations take place. Time sharing, or foreground operations, can take place concurrently with batch or background operations. If there are periods when TSO is not needed in the system, time sharing operations can be stopped, and the system will then process background jobs in the usual way with MVT and TCAM.

The Telecommunications Access Method, or TCAM, handles all I/O between remote terminals and jobs in the system. TCAM

distinguishes between time sharing applications, with emphasis on quick response back to the calling terminal, and other teleprocessing applications, where emphasis may be on routing and formatting of messages between one remote terminal and others. Both types of applications can operate simultaneously in the same system.

An important feature of TSO is the dynamic allocation of data sets for time sharing users. Dynamic allocation allows data sets to be created, deleted, concatenated, or separated without allocation at the beginning of the job step. A user can thus defer definition of his data sets until he requires them.

Working at the Terminal

A remote terminal has a keyboard for entering input and a typewriter-like printer or a display screen for output. Devices that can be used as terminals include:

- IBM 2741 Communication Terminal.
- IBM 1050 Data Communication System.
- IBM 2260 Display Station.
- IBM 2265 Display Station.
- AT&T Teletypes Model 33 and 35 KSR.

During a typical session, the user enters a series of commands to define and perform his work. The commands provided with the system handle data and program entry, program invocation in either the foreground or the background, program testing, data management, and session and system control. IBM Program Products are available to support problem solving, data manipulation, and text formatting, to provide terminal-oriented language processors, and to make these processors more convenient to use from the terminal.

Commands specifically tailored to an installation's needs can be written and added to the command language or used to replace IBM-supplied commands. Any load module can be established as a command and executed simply by keying in the program name at the terminal. Load modules not defined as commands can be invoked in the foreground with the CALL command.

The terminal user can also submit jobs to the background job stream. Commands similar to those used for the Conversational Remote Job Entry facility are used to create job control language describing the job, and to submit it to the batch job stream. The user can request notification of job completion at his terminal, and can have job output directed either to his terminal or to a device at the computer site.

System Control

Once an installation has generated a system that includes TSO, time sharing operations can be started and stopped at any time by the system console operator. The operator can specify how many regions of main storage are to be assigned to time sharing users.

Each foreground main storage region handles many active foreground jobs, although only one job is actually in the region at any moment in time. A foreground job is assigned to a main storage region and has access to the system's resources for a short period of time called a time slice. At the end of the job's time slice, or if the job enters the wait state for terminal I/O, the main storage image of the job (that is, programs, work areas, and associated control blocks) is stored on a direct access device and another job is brought into the same region of main storage and given a time slice. The process of copying job images back and forth between main and auxiliary storage is called swapping. Writing an image to axuiliary storage is a swap out; reading one into main storage is a swap in.

A time slice must be long enough to perform a meaningful amount of processing, but not so long that the time between successive slices prevents quick response to conversational users. At the same time, time slices cannot be so short and frequent that system overhead for swapping and task switching becomes excessive. Balancing these factors depends on the number and type of jobs the system is processing. A solution for one job mix is not necessarily suitable for another job mix. The TSO time sharing algorithms -- the formulas used to calculate the division of time among jobs -- are based on several variables, most of which can be specified by the installation to tune the system for their particular workload.

The management of an installation can shift most of the responsibility for controlling the time sharing system from the operator at the system console to users at remote terminals, called control terminals. A control terminal user can alter the system configuration to meet changing work loads. For instance, he can assign an extra region of main storage to time sharing operations during peak periods, and then release it to be used for batch operations during slack periods. Such changes require no shutdown of TSO and are not noticed by the users of other regions. Even the starting and stopping of TSO operations is accomplished without shutting down the system or affecting background operations.

GRAPHIC JOB PROCESSING

Graphic job processing is an optional feature of an MFT or MVT operating system. The graphic job processor is a program that enables users at remote IBM 2250 Display Units (Figure 63) to quickly and conveniently define and start jobs that are processed by the operating system. The display unit may be used to communicate directly with the System/360, or communicate with the System/360 by way of an IBM 1130 Computing System.

A user of the graphic job processor need not be familiar with the job control language of the operating system. Instead of a user defining a job in the form of job control statements, information about the job is elicited from him by means of a series of displays on the screen of the TV-like 2250 graphic display tube. A sample of such a display is shown in Figure 64. The user responds to the displays by entering requested information and selecting options using an alphameric keyboard, a light pen, or both. The graphic job processor then converts the information about the job into job control statements that are passed to the operating system to initiate the job.

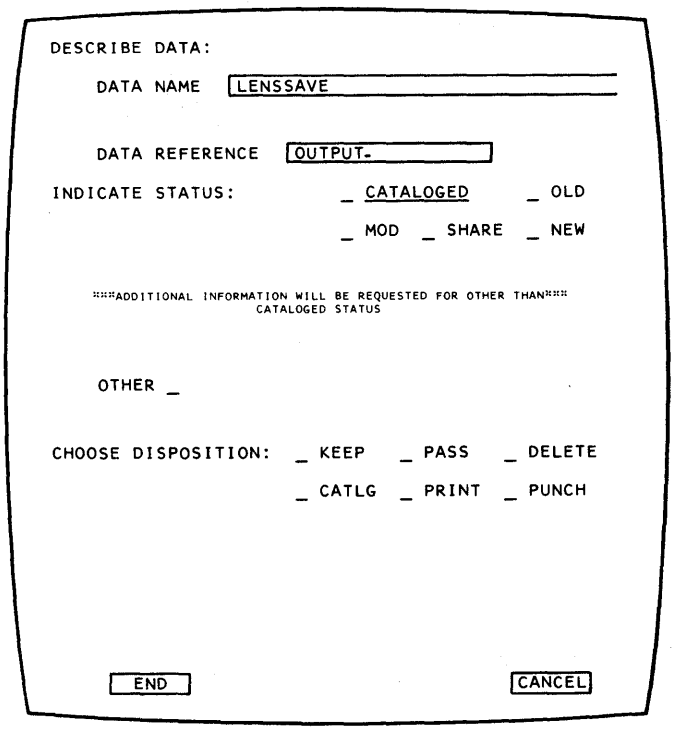


Figure 63. A DESCRIBE DATA Display For the Graphic Job Processor

The graphic job processor enables the user to:

- Identify himself to the system (LOG ON).
- Define a single job step (SPECIFY JOB STEP).
- Identify data to be used in a job step (DESCRIBE DATA).
- Start the processing of a job (BEGIN JOB).
- Execute a cataloged procedure (BEGIN PROCEDURE).
- Communicate with the system operator (WRITE MESSAGE).
- Enter 80-character data records, actual job control statements, and other program control statements (ENTER DATA).
- Cancel a job currently being defined (CANCEL JOB).
- Complete interaction with the 2250 and prepare the 2250 for the next user (LOG OFF).
- Repeat previously completed operations (RECALL).
- Name an 1130 program that is to be run in conjunction with a program in the IBM System/360 Computing System (SPECIFY 1130 PROGRAM).

The last operation applies only when an 1130 Computing System is used.

A System/360 installation allows up to 15 users at separate display units to process jobs independently of one another.

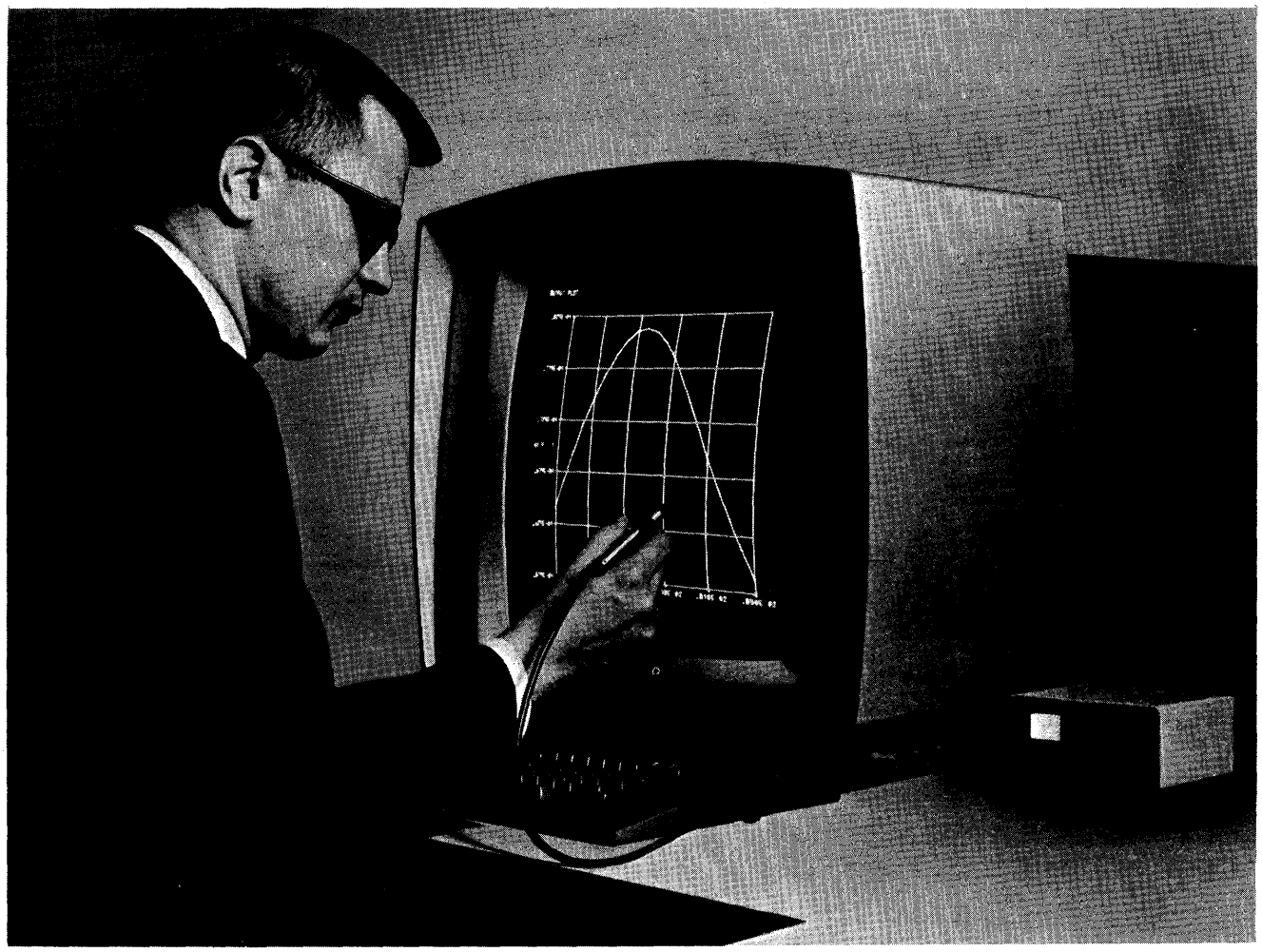


Figure 64. Using a Graphic Display Program on a 2250

Communication and control between each user and the operating system is established and maintained by the graphic job processor. The processor responds to a user by displaying messages on the 2250 screen. A printed record of the job control operations performed by a user at the display unit can be provided upon request.

A job that is defined at a display unit can be placed in a job input queue for batch processing (independently of the graphic job processor) using main storage space and other resources not assigned to the graphic job processor. This type of job is often called a background job. Alternatively, a foreground job can be defined and executed immediately using main storage and other resources assigned to the graphic job processor.

A foreground job would normally result in the execution of a graphic display program that would interact with the user at the display console, as shown in Figure 64. Thus, the graphic job processor can enable engineers, designers, and other non-programmers to execute and use graphic display programs for a variety of graphic display applications such as optical design (Figure 65) auto design, and civil engineering applications.

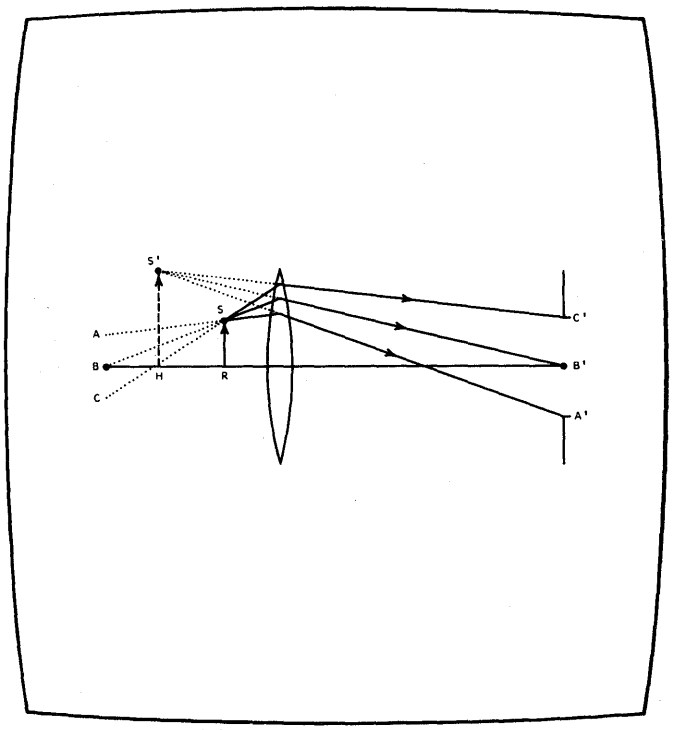


Figure 65. A Typical Optical Design Application Display

SYSTEM/360-1130 DATA TRANSMISSION FOR FORTRAN

A set of optional subroutines, provided for FORTRAN IV programmers, can be used to transmit data between a program being executed under control of the System/360 Operating System, and a program being executed under control of the Disk Monitor System of the IBM 1130 Computing System. The same subroutines can also be called using the assembler language.

The data transmission subroutines make it possible for an 1130 program to use the high speed computing ability and large storage capacity of the System/360. Thus, they can be used to increase the flexibility and efficiency of an 1130 application.

Separate sets of transmission subroutines are available for the System/360 and the 1130. These routines enable a programmer to transmit data from one system to the other without a detailed knowledge of telecommunications programming.

The data transmission subroutines enable a programmer using either system to:

- Initialize the communication lines.
- Transmit and receive data via the lines.
- Test the status of a previously requested transmit or receive operation.
- Initiate routines in the other system.
- Terminate the communication link between the System/360 and 1130 data transmission programs.

In addition, System/360 transmission subroutines enable the programmer to terminate the execution of an 1130 mainline program. Conversion subroutines are included in each set to reconcile differences in the FORTRAN data formats of the System/360 and the 1130. These subroutines can be called only by a System/360 program. They perform the following conversions:

- 1130 integer to System/360 integer, and vice versa.
- 1130 standard-precision real numbers to System/360 standard-length real numbers, and vice versa.
- 1130 extended precision real number to System/360 double-precision real numbers and vice versa.

Part 3, the bibliography, contains the titles of all the publications referenced in this manual.

IBM System/360:

1. Assembler Language, GC28-6514

IBM System/360 Operating System

2. Advanced Checkpoint/Restart Planning Guide, GC28-6708
3. Job Control Language Reference, GC28-6704
4. Job Control Language Reference, GC28-6704
5. Linkage Editor and Loader, GC28-6538
6. MFT Guide, GC28-6939
7. MVT Guide, GC28-6720
8. Principles of Operation, GA22-6821
9. Supervisor Services and Macro Instructions, GC28-6646
10. Data Management Services, GC26-3746
11. System Control Blocks, GC28-6628
12. System Generation, GC28-7554
13. Data Management for System Programmers, GC28-6550
14. TSO Planning Guide, GC28-6698
15. Utilities, GC28-6586

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.

GC28-6534-4

IBM System/360 Introduction Printed in U.S.A. GC28-6534-4

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM System/360 Operating System
Introduction

GC28-6534-4

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

What is your occupation? _____
Number of latest Technical Newsletter (if any) concerning this publication: _____
Please indicate in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. Elsewhere, an IBM office or representative will be happy to forward your comments.

Cut or Fold Along Line

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 81
Poughkeepsie
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

Cut or Fold Along Line

IBM System/360 Introduction Printed in U.S.A. GC28-6534-4