A brief outline of the structural elements of OS/360 *is given in preparation for the subsequent sections on control-program functions.*

Emphasis is placed on the functional scope of the system, on the motivating objectives and basic design concepts, and on the design approach to modularity.

The functional structure of OS/360

Part I Introductory survey

by G. H. Mealy

The environment that may confront an operating system has lately undergone great change. For example, in its several compatible models, SYSTEM/360 spans an entire spectrum of applications and offers an unprecedented range of optional devices.[1] It need come as no surprise, therefore, that OS/360—the Operating System for SYSTEM/360—evinces more novelty in its scope than in its functional objectives.

In a concrete sense, OS/360 consists of a library of programs. In an abstract sense, however, the term OS/360 refers to one articulated response to a composite set of needs. With integrated vocabularies, conventions, and modular capabilities, OS/360 is designed to answer the needs of a SYSTEM/360 configuration with a standard instruction set and thirty-two thousand or more bytes of main storage.[2]

The main purpose of this introductory survey is to establish the scope of OS/360 by viewing the subject in a number of different perspectives: the historical background, the design objectives, and the functional types of program packages that are provided. An effort is made to mention problems and design compromises, i.e., to comment on the forces that shaped the system as a whole.

Basic objectives

The notion of an operating system dates back at least to 1953 and

MIT's Summer Session Computer and Utility System.[3] Then, as now, the operating system aimed at non-stop operation over a span of many jobs and provided a computer-accessible library of utility programs. A number of operating systems came into use during the last half of the decade.[4] In that all were oriented toward overlapped setup in a sequentially executed job batch, they may be termed "first generation" operating systems.

A significant characteristic of batched-job operation has been that each job has, more or less, the entire machine to itself, save for the part of the system permanently resident in main storage. During the above-mentioned period of time, a number of large systems—typified by SAGE, MERCURY, and SABRE—were developed along other lines; these required total dedication of machine resources to the requirements of one "real-time" application. It is interesting that one of the earliest operating systems, the Utility Control Program developed by the Lincoln Laboratory, was developed solely for the checkout of portions of the SAGE system. By and large, however, these real-time systems bore little resemblance to the first generation of operating systems, either from the point of view of intended application or system structure.

Because the basic structure of OS/360 is equally applicable to batched-job and real-time applications, it may be viewed as one of the first instances of a "second-generation" operating system. The new objective of such a system is to accommodate an environment of diverse applications and operating modes. Although not to be discounted in importance, various other objectives are not new—they have been recognized to some degree in prior systems. Foremost among these secondary objectives are:

- Increased throughput
- Lowered response time
- Increased programmer productivity
- Adaptability (of programs to changing resources)
- Expandability

throughput

OS/360 seeks to provide an effective level of machine throughput in three ways. First, in handling a stream of jobs, it assists the operator in accomplishing setup operations for a given job while previously scheduled jobs are being processed. Second, it permits tasks from a number of different jobs to concurrently use the resources of the system in a multiprogramming mode, thus helping to ensure that resources are kept busy. Also, recognizing that the productivity of a shop is not solely a function of machine utilization, heavy emphasis is placed on the variety and appropriateness in source languages, on debugging facilities, and on input convenience.

response time

Response time is the lapse of time from a request to completion of the requested action. In a batch processing context, response time (often called "turn-around time") is relatively long: the user gives a deck to the computing center and later obtains printed re-

sults. In a mixed environment, however, we find a whole spectrum of response times. Batch turn-around time is at the "red" end of the spectrum, whereas real-time requirements fall at the "violet" end. For example, some real-time applications need response times in the order of milliseconds or lower. Intermediate in the spectrum are the times for simple actions such as line entry from a keyboard where a response time of the order of one or two seconds is desirable. Faced with a mixed environment in terms of applications and response times, OS/360 is designed to lend itself to the whole spectrum of response times by means of control-program options and priority conventions.

productivity

For the sake of programmer productivity and convenience, OS/360 aims to provide a novel degree of versatility through a relatively large set of source languages. It also provides macro-instruction capabilities for its assembler language, as well as a concise job-control language for assistance in job submission.

adaptability

A second-generation operating system must be geared to change and diversity. SYSTEM/360 itself can exist in an almost unlimited variety of machine configurations: different installations will typically have different configurations as well as different applications. Moreover, the configuration at a given installation may change frequently. If we look at application and configuration as the environment of an operating system, we see that the operating system must cope with an unprecedented number of environments. All of this puts a premium on system modularity and flexibility.

Adaptability is also served in OS/360 by the high degree to which programs can be device-independent. By writing programs that are relatively insensitive to the actual complement of input/output devices, an installation can reduce or circumvent the problems historically associated with device substitutions.

expandability

As constructed, OS/360 is "open-ended"; it can support new hardware, applications, and programs as they come along. It can readily handle diverse currency conventions and character sets. It can be tailored to communicate with operators and programmers in languages other than English. Whenever so dictated by changing circumstances, the operating system itself can be expanded in its functional capabilities.

Design concepts

In the notion of an "extended machine," a computing system is viewed as being composed of a number of layers, like an onion.[5,6] Few programmers deal with the innermost layer, which is that provided by the hardware itself. A FORTRAN programmer, for instance, deals with an outer layer defined by the FORTRAN language. To a large extent, he acts as though he were dealing with hardware that accepted and executed FORTRAN statements directly. The SYSTEM/360 instruction set represents two inner layers, one when operating in the supervisor state, another when operating in the problem state.

The supervisor state is employed by OS/360 for the *supervisor* portion of the control program. Because all other programs operate in the problem state and must rely upon unprivileged instructions, they use *system macroinstructions* for invoking the supervisor. These macroinstructions gain the attention of the supervisor by means of SVC, the supervisor-call instruction.

All OS/360 programs with the exception of the supervisor operate in the problem state. In fact, one of the fundamental design tenets is that these programs (compilers, sorts, or the like) are, to all intents and purposes, problem programs and must be treated as such by the supervisor. Precisely the same set of facilities is offered to system and problem programs. At any point in time, the system consists of its given supervisor plus all programs that are available in on-line storage. Inasmuch as an installation may introduce new compilers, payroll programs, etc., the extended machine may grow.

In designing a method of control for a second-generation system, two opposing viewpoints must be reconciled. In the first-generation operating systems, the point of view was that the machine executed an incoming stream of programs; each program and its associated input data corresponded to one application or problem. In the first-generation real-time systems, on the other hand, the point of view was that incoming pieces of data were routed to one of a number of processing programs. These attitudes led to quite different system structures; it was not recognized that these points of view were matters of degree rather than kind. The basic consideration, however, is one of emphasis: programs are used to process data in both cases. Because it is the combination of program and data that marks a unit of work for control purposes, OS/360 takes such a combination as the distinguishing property of a *task*. As an example, consider a transaction processing program and two input transactions, A and B. To process A and B, two tasks are introduced into the system, one consisting of A plus the program, the second consisting of B plus the program. Here, the two tasks use the same program but different sets of input data. As a further illustration, consider a master file and two programs, X and Y, that yield different reports from the master file. Again, two tasks are introduced into the system, the first consisting of the master file plus X, and the second of the master file plus Y. Here the same input data join with two different programs to form two different tasks.

In laying down conceptual groundwork, the OS/360 designers have employed the notion of multitask operation wherein, at any time, a number of tasks may contend for and employ system resources. The term *multiprogramming* is ordinarily used for the case in which one CPU is shared by a number of tasks, the term *multiprocessing*, for the case in which a separate task is assigned to each of several CPU's. Multitask operation, as a concept, gives recognition to both terms. If its work is structured entirely in the form of tasks, a job may lend itself without change to either environment.

In OS/360, any named collection of data is termed a *data set.* A data set may be an accounting file, a statistical array, a source program, an object program, a set of job control statements, or the like. The system provides for a cataloged library of data sets. The library is very useful in program preparation as well as in production activities; a programmer can store, modify, recompile, link, and execute programs with minimal handling of card decks.

System elements

As seen by a user, OS/360 will consist of a set of language translators, a set of service programs, and a control program. Moreover, from the viewpoint of system management, a SYSTEM/360 installation may look upon its own application programs as an integral part of the operating system.

translators

A variety of translators are being provided for FORTRAN, COBOL, and RPGL (a Report Program Generator Language). Also to be provided is a translator for PL/I, a new generalized language.[7] The programmer who chooses to employ the assembler language can take advantage of macroinstructions; the assembler program is supplemented by a macro generator that produces a suitable set of assembly language statements for each macroinstruction in the source program.

service programs

Groups of individually translated programs can be combined into a single executable program by a linkage editor. The linkage editor makes it possible to change a program without re-translating more than the affected segment of the program. Where a program is too large for the available main-storage area, the function of handling program segments and overlays falls to the linkage editor.

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

Included in the service programs are routines for editing, arranging, and updating the contents of the library; revising the index structure of the library catalog; printing an inventory list of the catalog; and moving and editing data from one storage medium to another.

the control program

Roughly speaking, the control program subdivides into master scheduler, job scheduler, and supervisor. Central control lodges in the supervisor, which has responsibility for the storage allocation, task sequencing, and input/output monitoring functions. The master scheduler handles all communications to and from the operator, whereas the job scheduler is primarily concerned with

job-stream analysis, input/output device allocation and setup, and job initiation and termination.

supervisor

Among the activities performed by the supervisor are the following:

- Allocating main storage
- Loading programs into main storage
- Controlling the concurrent execution of tasks
- Providing clocking services
- Attempting recoveries from exceptional conditions
- Logging errors
- Providing summary information on facility usage
- Issuing and monitoring input/output operations

The supervisor ordinarily gains control of the central processing unit by way of an interruption. Such an interruption may stem from an explicit request for services, or it may be implicit in SYSTEM/360 conventions, such as in the case of an interruption that occurs at the completion of an input/output operation. Normally, a number of data-access routines required by the data management function are coordinated with the supervisor. The access routines available at any given time are determined by the requirements of the user's program, the structure of the given data sets, and the types of input/output devices in use.

job scheduler

As the basic independent unit of work, a job consists of one or more steps. Inasmuch as each job step results in the execution of a major program, the system formalizes each job step as a task, which may then be inserted into the task queue by the initiator-terminator (a functional element of the job scheduler). In some cases, the output of one step is passed on as the input to another. For example, three successive job steps might involve file maintenance, output sorting, and report tabulation.

The primary activities of the job scheduler are as follows:

- Reading job definitions from source inputs
- Allocating input/output devices
- Initiating program execution for each job step
- Writing job outputs

In its most general form, the job scheduler allows more than one job to be processed concurrently. On the basis of job priorities and resource availabilities, the job scheduler can modify the order in which jobs are processed. Jobs can be read from several input devices and results can be recorded on several output devices—the reading and recording being performed concurrently with internal processing.

master scheduler

The master scheduler serves as a communication control link between the operator and the system. By command, the operator can alert the system to a change in the status of an input/output unit, alter the operation of the system, and request status information. The master scheduler is also used by the operator to alert the job scheduler of job sources and to initiate the reading or processing of jobs.

The control program as a whole performs three main functions: job management, task management, and data management. Since Part II of this paper discusses job and task management, and Part III is devoted entirely to data management, we do not further pursue these functions here.

System modularity

Two distinguishable, but by no means independent, design problems arise in creating a system such as OS/360. The first one is to prescribe the range of functional capabilities to be provided; essentially, this amounts to defining two operating systems, one of maximum capability and the other a nucleus of minimum capability. The second problem is to ascertain a set of building blocks that will answer reasonably well to the two predefined operating systems as well as to the diverse needs bounded by the two. In resolving the second problem, which brings us to the subject of modularity, no single consideration is more compelling than the need for efficient utilization of main storage.

As stated earlier, the tangible OS/360 consists of a library of program modules. These modules are the blocks from which actual operating systems can be erected. The OS/360 design exploits three basic principles in designing blocks that provide the desired degree of modularity. Here, these well-known principles are termed parametric generality, functional redundancy, and functional optionality.

parametric generality

The degree of generality required by varying numbers of input/output devices, control units, and channels can be handled to a large extent by writing programs that lend themselves to variations in parameters. This has long been practiced in sorting and merging programs, for example, as well as in other generalized routines. In OS/360, this principle also finds frequent application in the process that generates a specific control program.

functional redundancy

In the effort to optimize performance in the face of two or more conflicting objectives, the most practical solution (at least at the present state of the art) is often to write two or more programs that exploit dissimilar programming techniques. This principle is most relevant to the program translation function, which is especially sensitive to conflicting performance measures. The same installation may desire to effect one compilation with minimum use of main storage (even at some expense of other objectives) and another compilation with maximum efficacy in terms of object-program running time (again at the expense of other objectives). Where conflicting objectives could not be reconciled by other means, the OS/360 designers have provided more than one program for the same general translation or service function. For the COBOL language, for example, there are two translation programs.

For the nucleus of the control program that resides in main storage, the demand for efficient storage utilization is especially

functional optionality

pressing. Hence, each functional capability that is likely to be unused in some installations is treated as a separable option. When a control program is generated, each omitted option yields a net saving in the main-storage requirement of the control program.

The most significant control program options are those required to support various job scheduling and multitask modes of operation. These modes carry with them needs for optional functions of the following kinds:

- Task synchronization
- Job-input and job-output queues
- Distinctive methods of main-storage allocation
- Main-storage protection
- Priority-governed selection among jobs

In the absence of any options, the control program is capable of ordinary stacked-job operation. The activities of the central processing unit and the input/output channels are overlapped. Many error checking and recovery functions are provided, interruptions are handled automatically, and the standard data-management and service functions are included. Job steps are processed sequentially through single task operations.

The span of operating modes permitted by options in the control program can be suggested by citing two limiting cases of multitask operation. The first and least complicated permits a scheduled job step to be processed concurrently with an initial-input task, say A, and a result-output task, say B. Because A and B are governed by the control program, they do not correspond to job steps in the usual sense. The major purpose of this configuration is to reduce delays between the processing of successive job steps: tasks A and B are devoted entirely to input/output functions.

In the other limiting case, up to n jobs may be in execution on a concurrent basis, the parameter n being fixed at the time the control program is generated. Contending tasks may arise from different jobs, and a given task can dynamically define other tasks (see the description of the ATTACH macroinstruction in Part II) and assign task priorities. Provision is made for removal of an entire job step (from the job of lowest priority) to auxiliary storage in the event that main storage is exhausted. The affected job step is resumed as soon as the previously occupied main-storage area becomes available again.

In selecting the options to be included in a control program, the user is expected to avail himself of detailed descriptions and accompanying estimates of storage requirements.

system generation

To obtain a desired operating system, the user documents his machine configuration, requests a complement of translators and service programs, and indicates desired control-program options—all via a set of macroinstructions provided for the purpose. Once this has been done, the fabrication of a specific operating system from the OS/360 systems library reduces to a process of two stages.

First, the macroinstructions are analyzed by a special program and formulated into a job stream. In the second stage, the assembler program, the linkage editor, and the catalog service programs join in the creation of a resident control program and a desired set of translators and service programs.

Summary comment

Intended to serve a wide variety of computer applications and to support a broad range of hardware configurations, OS/360 is a modular operating system. The system is not only open-ended for the class of functions discussed in this paper, but is based on a conceptual framework that is designed to lend itself to additional functions whenever warranted by cumulative experience.

The ultimate purpose of an operating system is to increase the productivity of an entire computer installation; personnel productivity must be considered as well as machine productivity. Although many avenues to increased productivity are reflected in OS/360, each of these avenues typically involves a marginal investment on the part of an installation. The investment may take the form of additional personnel training, storage requirements, or processing time. It repays few installations to seek added productivity through every possible avenue; for most, the economies of installation management dictate a well-chosen balance between investment and return. Much of the modularity in OS/360 represents a design attempt to permit each installation to strike its own economic balance.

CITED REFERENCES AND FOOTNOTES

1. For an introduction to SYSTEM/360, see G. A. Blaauw and F. P. Brooks, Jr., "The structure of SYSTEM/360, Part I, outline of the logical structure," *IBM Systems Journal* **3**, No. 2, 119–135 (1964).
2. The restrictions exclude MODEL 44, as well as MODEL 20. The specialized operating systems that support these excluded models are not discussed here.
3. C. W. Adams and J. H. Laning, Jr., "The MIT systems of automatic coding: Comprehensive, Summer Session, Algebraic," *Symposium on Automatic Coding for Digital Computers*, Office of Naval Research, Department of the Navy (May 1954).
4. In the case of the IBM 709 and 704 computers, the earliest developments were largely due to the individual and group efforts of SHARE installations. The first operating systems developed jointly by IBM and SHARE were the SHARE Operating System (SOS) and the FORTRAN Monitor System (FMS).
5. G. F. Leonard and J. R. Goodroe, "An environment for an operating system," *Proceedings of the 19th National ACM Conference* (August 1964).
6. A. W. Holt and W. J. Turanski, "Man to machine communication and automatic code translation," *Proceedings Western Joint Computer Conference* (1960).
7. G. Radin and H. P. Rogoway, "NPL: highlights of a new programming language," *Communications of the ACM* **8**, No. 1, 9–17 (January 1965).