# Introduction and Overview of the Multics System

*F. J. Corbató*
*Massachusetts Institute of Technology*
*Cambridge, Massachusetts*
*and*
*V. A. Vyssotsky*
*Bell Telephone Laboratories, Inc.*
*Murray Hill, New Jersey*

**Multics** (Multiplexed Information and Computing Service) is a comprehensive, general-purpose programming system which is being developed as a research project. The initial Multics system will be implemented on the GE 645 computer. One of the overall design goals is to create a computing system which is capable of meeting almost all of the present and near-future requirements of a large computer utility. Such systems must run continuously and reliably 7 days a week, 24 hours a day in a way similar to telephone or power systems, and must be capable of meeting wide service demands: from multiple man-machine interaction to the sequential processing of absentee-user jobs; from the use of the system with dedicated languages and subsystems to the programming of the system itself; and from centralized bulk card, tape, and printer facilities to remotely located terminals. Such information processing and communication systems are believed to be essential for the future growth of computer use in business, in industry, in government and in scientific laboratories as well as stimulating applications which would be otherwise undone.

Because the system must ultimately be comprehensive and able to adapt to unknown future requirements, its framework must be general, and capable of evolving with time. As brought out in the companion papers[1-5], this need for an evolutionary framework influences and contributes to much of the system design and is a major reason why most of the programming of the system will be done in the PL/I language.[6] Because the PL/I language is largely machine-independent (e.g. data descriptions refer to logical items, not physical words), the system should also be. Specifically, it is hoped that future hardware improvements will not make system and user programs obsolete and that implementation of the entire system on other suitable computers will require only a moderate amount of additional programming.

The present paper attempts to give a detailed discussion of the design objectives as they relate to the major areas of the system. Some of the highlights of the subsequent papers are: a virtual memory system for each user involving two-dimensional addressing with segmentation and paging; the dynamic linking of program segment cross-references at execution time to minimize system overhead; the routine use of sharable, recursive, pure procedure programming within the system as the normal mode of operation; the pooled use of multiple processors, memory modules, and input-output controllers; and multiprogramming of all resources and of multiple users. Automatic management of the complex of secondary storage media along with backup, retrieval, and maintenance procedures for the stored information will be provided by a file system. Further, it is expected that most of the software of the system will be almost identical in form to user programs. The system will incorporate automatic page-turning for both user and system programs alike.

# Introduction

As computers have matured during the last two decades from curiosities to calculating machines to information processors, access to them by users has not improved and in the case of most large machines has retrogressed. Principally for economic reasons, batch processing of computer jobs has been developed and is currently practiced by most large computer installations, and the concomitant isolation of the user from elementary cause-and-effect relationships has been either reluctantly endured or rationalized. For several years a solution has been proposed to the access problem.[7-9] This solution, usually called time-sharing, is basically the rapid time-division multiplexing of a central processor unit among the jobs of several users, each of which is on- line at a typewriter-like console. The rapid switching of the processor unit among user programs is, of course, nothing but a particular form of multiprogramming.

It is now abundantly clear that it is possible to create a general-purpose time-shared multiaccess system on many contemporary computers (especially after minor but basic modifications are made). Already two major and extensive systems have been created, one on the IBM 7094[10,11] and one on the Q-32 computer.[12] In addition, there have been numerous smaller scale systems, the most notable being on the DEC PDP-1,[13,14] the IBM 7094,[15] the GE 235,[16] the DEC PDP-6,[17] and the SDS 930,[18] as well as somewhat more limited versions of time-sharing on the RW-400,[19,20] and the CDC G21,[21] the Johnniac,[22] and the IBM 7040.[23] As time goes on, surveys of implemented systems are being made [42,24] and "score cards" are being kept.[25]

The impetus for time-sharing first arose from professional programmers because of their constant frustration in debugging programs at batch processing installations. Thus, the original goal was to time-share computers to allow simultaneous access by several persons while giving to each of them the illusion of having the whole machine at his disposal. However, at Project MAC it has turned out that simultaneous access to the machine, while obviously necessary to the objective, has not been the major ensuing benefit.[26] Rather, it is the availability at one's fingertips of facilities for editing, compiling, debugging, and running in one continuous interactive session that has had the greatest effect on programming. Professional programmers are encouraged to be more imaginative in their work and to investigate new programming techniques and new problem approaches because of the much smaller penalty for failure. But, the most significant effect that the MAC system has had on the MIT community is seen in the achievements of per sons for whom computers are tools for other objectives. The availability of the MAC system has not only changed the way problems are attacked, but also important research has been done that would not have been undertaken otherwise. As a consequence the objective of the current and future development of time-sharing should extend way beyond the improvement of computational facilities with respect to traditional computer applications. Rather, it is the on-line use of computers for new purposes and in new fields which should provide the challenge and the motivation to the system designer. In other words, the major goal is to provide suitable tools for what is currently being called machine-aided cognition.

More specifically, the importance of a multiple access system operated as a computer utility is that it allows a vast enlargement of the scope of computer-based activities, which should in turn stimulate a corresponding enrichment of many areas of our society. Over two years of experience indicates that continuous operation in a utility-like manner, with flexible remote access, encourages users to view the system as a thinking tool in their daily intellectual work. Mechanistically, the qualitative change from the past results from the drastic improvement in access time and convenience. Subjectively, the change lies in the user's ability to control and affect interactively the course of a process whether it involves numerical computation or manipulation of symbols. Thus, parameter studies are more intelligently guided; new problem-oriented languages and subsystems are developed to exploit the interactive capability; many complex analytical problems, as in magnetohydrodynamics, which have been too cumbersome to be tackled in the past are now being successfully pursued; even more, new, imaginative approaches to basic research have been developed as in the decoding of protein structures. These are examples taken from an academic environment; the effect of a multiple access system on business and industrial organizations can be expected to be equally dramatic but experience in this area is still very limited. It is with such new applications in mind that the Multics system has been developed. Not that the traditional uses of computers are being disregarded. Rather, these needs are viewed as a subset of the broader more demanding requirements of the former.

To meet the above objectives, issues such as response time, convenience of manipulating data and program files, ease of controlling processes during execution and above all, protection of private files and isolation of independent processes become of critical importance. These issues demand departures from traditional computer systems. While these departures are deemed to be desirable with respect to traditional computer applications, they are essential for rapid man-machine interaction.

## System Requirements

In the early days of computer design, there was the concept of a single program on which a single processor computed for long periods of time with almost no interaction with the outside world. Today such a view is considered incomplete; for the effective boundaries of an information processing system extend beyond the processor, beyond the card reader and printer and even beyond the typing of input and the reading of output. In fact they encompass as well what several hundred persons are trying to accomplish. To better understand the effect of this broadened design scope, it is helpful to examine several phenomena characteristic of large service-oriented computer installations.

First, there are incentives for any organization to have the biggest possible computer system that it can afford. It is usually only on the biggest computers that there are the elaborate programming systems, compilers and features which make a computer "powerful." This comes about partly because it is more difficult to prepare system programs for smaller computers when limited by speed or memory size and partly because the larger systems involve more persons as manufacturers, managers, and users and hence permit more attention to be given to the system programs. Moreover, by combining resources in a single computer system, rather than in several, bulk economies and therefore lower computing costs can be achieved. Finally, as a practical matter, considerations of floor space, management efficiency and operating personnel provide a strong incentive for centralizing computer facilities in a single large installation.

Second, the capacity of a contemporary computer installation, regardless of the sector of applications it serves, must be capable of growing to meet a continuously increasing demand. A doubling of demand every two years is not uncommon.[27] Multiple-access computers promise to accelerate this growth further since they allow a man-machine interaction rate which is faster by at least two orders of magnitude. Present indications are that multiple-access systems for only a few hundred simultaneous users can generate a demand for computation exceeding the capacity of the fastest existing single-processor system. Since the speed of light, the physical sizes of computer components, and the speeds of memories are intrinsic limitations on the speed of any single processor, it is clear that systems with multiple processors and multiple memory units are needed to provide greater capacity. This is not to say that fast processor units are undesirable, but that extreme system complexity to enhance this single parameter among many appears neither wise nor economic.

Third, computers are no longer a luxury used when and if available, but primary working tools in business, government, and research laboratories. The more reliable computers become, the more their availability is depended upon. A system structure including pools of functionally identical units (processors, memory modules, input/output controllers, etc.) can provide continuous service without significant interruption for equipment maintenance, as well as provide growth capability through the addition of appropriate units.

Fourth, user programs, especially in a timesharing system, interact frequently with secondary storage devices and terminals. This communication traffic produces a need for multiprogramming to avoid wasting main processor time while an input/output request is being completed. It is important to note that an individual user is ordinarily incapable of doing an adequate job of multiprogramming since his program lacks proper balance, and he probably lacks the necessary dynamic information, ingenuity or patience.

Finally, as noted earlier, the value of a timesharing system lies not only in providing, in effect, a private computer to a number of people simultaneously, but, above all, in the services that the system places at the fingertips of the users. Moreover, the effectiveness of a system increases as user-developed facilities are shared by other users. This increased effectiveness because of sharing is due not only to the reduced demands for core and secondary memory but also to the cross-fertilization of user ideas. Thus a major goal of the present effort is to provide multiple access to a growing and potentially vast structure of shared data and shared program procedures. In fact, the achievement of multiple access to the computer processors should be viewed as but a necessary subgoal of this broader objective. Thus the primary and secondary memories where programs reside play a central role in the hardware organization and the presence of independent communication paths between memories, processors and terminals is of critical importance.

From the above it can be seen that the system requirements of a computer installation are not for a single program on a single computer, but rather for a large system of many components serving a community of users. Moreover, each user of the system asynchronously initiates jobs of arbitrary and indeterminate duration which subdivide into sequences of processor and input/output tasks. It is out of this seemingly chaotic, random environment that one arrives at a utility-like view. For instead of chaos, one can average over the different user requests to achieve high utilization of all resources. The task of multiprogramming required to do this need only be organized once in a central supervisor program. Each user thus enjoys the benefit of efficiency without having to average the demands of his own particular program.

With the above view of computer use, where tasks start and stop every few milliseconds and where the memory requirements of tasks grow and shrink, it is apparent that one of the major jobs of the supervisor program (i.e., "monitor," "executive," etc.) is the allocation and scheduling of computer resources. The general strategy is clear. Each user's job is subdivided into tasks, usually as the job proceeds, each of which is placed in an appropriate queue (i.e., for a processor or an input/output controller). Processors or input/output controllers are in turn assigned new tasks as they either complete or are removed from old tasks. All processors are treated equivalently in an anonymous pool and are assigned to tasks as needed; in particular, the supervisor does not have a special processor. Further, processors can be added or deleted without significant change in either the user or system programs. Similarly, input/output controllers are directed from queues independently of any particular processor. Again, as with the processors, one can add or delete input/output capacity according to system load without significant reprogramming required.

# The Multics System

The overall design goal of the Multics system is to create a computing system which is capable of comprehensively meeting almost all of the present and near-future requirements of a large computer service installation. It is not expected that the initial system, although useful, will reach the objective; rather the system will evolve with time in a general framework which permits continual growth to meet unknown future requirements. The use of the PL/I language will allow major system software changes to be developed on a schedule separate from that of hardware changes. Since most organizations can no longer afford to overlap old and new equipment during changes, and since software development is at best difficult to schedule, this relative machine-independence should be a major asset.

It is expected that the Multics system will be published when it is operating substantially and will therefore be available for implementation on any equipment with suitable characteristics. Such publication is desirable for two reasons: First, the system should withstand public scrutiny and criticism volunteered by interested readers; second, in an age of increasing complexity, it is an obligation to present and future system designers to make the inner operating system as lucid as possible so as to reveal the basic system issues.

The accompanying papers describe in some detail how the Multics system will meet its objectives. However, it is useful, in establishing an overview, to touch on the highlights and especially on the design motivation.

## Design Features Of The Hardware

The Multics system objectives required equipment features that were not present in any existing computer. Consequently it was necessary to develop for the Multics system the GE 645 computer. The GE 635 computer was selected for modification to the GE 645 inasmuch as it already satisfied many of the crucial requirements. In particular, it was de signed to have multiprocessors, multiple memory modules, and multiple input/output controllers. Thus, the requirements of modular construction for reliability and for ease of growth were amply met. The communication pattern is particularly straight forward since there are no physical paths between the processors and the input/output equipment; rather all communication is done by means of "mailboxes" in the memory modules and by corresponding interrupts. Furthermore, major modules of the system communicate on an asynchronous basis; thus, any single module can be upgraded without any changes to the other modules. This latter property is useful in that one of the ways in which system capacity (and cost) may be regulated is by changing either the speed or number of memory modules. Of course further adjustment of system capacity is possible by varying the number of processor units or the configuration of drum and disk equipment. In any case, one obtains the important simplification that a single supervisor program can operate without substantial change on any configuration of equipment.

Figure 1 illustrates the equipment configuration of a typical Multics system. All central processors (CPU) and Generalized Input/Output Controllers (GIOC) have communication paths with each of the memory modules. When necessary for maintenance or test purposes, the system can be partitioned into two independent systems (although each of the drum, disk and tapes must belong to one of the two systems). The remote terminals can dial either of the two GIOC through the private branch exchange, which is not shown in the figure.

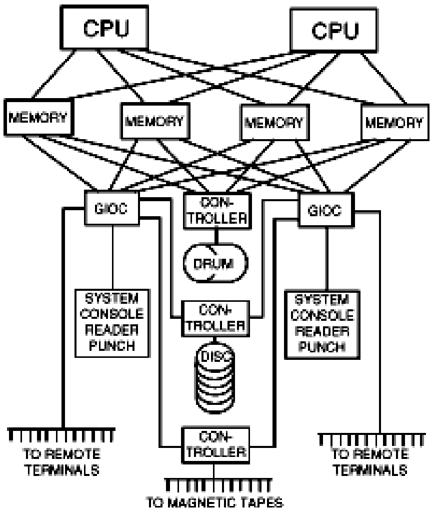


Figure 1. Example of Multics system configuration.

The most novel feature in the GE 645 is in the instruction addressing. A two-dimensional addressing system has been incorporated which allows each user to write programs as though there is a virtual memory system of large size. This system is organized into program segments (i.e., regions) each of which contains an ordered sequence of words with a conventional linear address. These segments, which can vary in length during execution, are paged at the discretion of the supervisor program with either 64- or 1,024-word pages. This dual page size allows the supervisor program to use more effective strategies in the handling of multiple users. Paging, first introduced on the Atlas computer,[28] allows flexible dynamic memory allocation techniques as well as the sensible implementation of a one-level store system. To the user in the Multics system, page addressing is invisible; rather, it is the segments which are explicitly known to him and to which he is able to refer symbolically in his programs. These notions were first suggested by Holt,[29] further developed by Dennis,[30,31] Dennis and Glaser,[32] Forgie,[33] and others.[34,35] The value of segmentation and paging has since been widely discussed during the past year and has gained broader acceptance.[36-39] The explicit hardware implementation details of segmentation and paging for the Multics system are discussed in the companion paper by Glaser, Couleur and Oliver.[1]

Because two-dimensional addressing is rather new, it is useful to clarify the reasons for it. The major reasons for segments are:

1. The user is able to program in a two- dimensional virtual memory system. Thus, any single segment can grow (or shrink) during execution (e.g., in the GE 645, each user may have up to a quarter million segments, each including up to a quarter million

words).
2. The user can, by merely specifying a starting point in a segment, operate a program implicitly without prior planning of the segments needed or of the storage requirements. For example, if an error diagnostic segment is unexpectedly called for, it is brought in automatically by the supervisor; it is never brought in unless needed. Similarly, elaborate computations which branch into many different segments in a data-dependent way use segments only as needed.
3. The largest amount of code which must be bound together as a solid block is a single segment. Since binding pieces of code together (sometimes called "loading") is a process similar to assembling or compiling, the advantage of being able to prepare an arbitrarily large program as a series of limited-overhead segment bindings is significant. The saving in overhead is comparable to that in FORTRAN when one uses multiple subprograms instead of a single large combined block of statements. If the combined block is used, not only does the compilation process become particularly cumbersome but the eradication of programming errors in all the different sections requires more compilation time.
4. Program segments appear to be the only reasonable way to permit pure procedures and data bases to be shared among several users simultaneously. Pure procedure programs, by definition, do not modify themselves. Therefore a supervisor program can minimize the core memory requirements of a collection of user programs by supplying only one copy of a jointly used pure procedure. Nearly all of the Multics system as well as most of the user programs will be written in this form. One consequence is that there will be no clear-cut demarcation between user programs and system programs; instead the demarcation will depend largely on the responsibility for maintenance.

Pages are a separate feature from segments and have further and distinct advantages.

1. The use of paged memory allows flexible techniques for dynamic storage management without the overhead of moving programs back and forth in the primary memory. This reduced overhead is important in responsive time-shared systems where there is heavy traffic between primary and secondary memories.
2. The mechanism of paging, when properly implemented, allows the operation of incompletely loaded programs; the supervisor need only retain in main memory the more active pages, thus making more effective use of high-speed storage. Whenever a reference to a missing page occurs, the supervisor must interrupt the program, fetch the missing page, and re-initiate the program

without loss of information.

A critical feature in the segment and paging hardware is the descriptor bit mechanism which controls the access of processors to the memory. These bits essentially allow hardware "fire-walls" to be established within the programming system which assist the isolation of hardware or software difficulties. Besides controlling the usual properties such as read-only, data-only, etc., one descriptor bit allows a segment to be declared "execute-only." The presence of this bit allows procedures to be transferred to and executed but never read by user programs. This feature will be of interest to commercial service bureaus, and in application areas where privacy of program procedure is essential (e.g., a class-room grading program). Another property of the descriptors is that they allow most of the supervisor modules to be written with the same descriptors as user programs; most system programs thereby do not have access to privileged instructions, the inadvertent use of which can cause drastic machine misbehavior. This feature is especially pertinent when it is recognized that timesharing systems are real-time systems with behavior which it is difficult to duplicate or repeat. Consequently, all possible compartments and protection mechanisms that one can have are of value.

For effective operation of the Multics system, a drum with a high transfer rate is needed. The drum provided with the GE 645 meets the requirement and allows convenient and efficient management of a high rate of input/output requests. In particular, requests are organized by the supervisor program into queues in core memory and are fetched from these queues by the drum controller asynchronously of the processors. Because of the queues and because drum record sizes are commensurate with core memory page sizes, it is straightforward to program for continuous input/output transmission without latency delays.

Disk input/output requests are also organized into queues and are fetched from core memory by the generalized input/output controller. This controller is discussed in more detail in the paper by Ossanna et al.[4] Again, because the supervisor is contending with a statistical mix of user and super visor requests for information to and from disk, it is expected that latency delays between requests will be negligible. Because the transmission capacity to the disk is large, system performance is expected to be unhampered by input/output bottlenecks. Since the Multics system will be used as an information processor in a wide range of applications, it is important that a readable character set be used. The standard character set will be the recently proposed ASCII code which has 128 codes and includes upper and lower case letters.[40] This set, which contains 95 printing graphics, can be reasonably represented on contemporary input/output consoles. Line printers capable of printing the 95 graphics will be standard equipment.

## Design Features Of The Software

An important aspect of the software is the subroutine and linkage conventions which are associated with the use of the segment and paging hardware. The following features are incorporated.

1. Any segment has to know another segment only by symbolic name. Intersegment binding occurs dynamically as needed during program execution. Intersegment binding is automatic (i.e., not explicitly programmed by the user) and the mechanism operates at high efficiency after the first binding occurs.
2. Similarly, a segment is able to reference symbolically a location within another segment. This reference binds dynamically and automatically; after binding occurs the first time, program execution is at full speed.
3. It is straightforward for procedures to be pure procedures, capable of being shared by several users.
4. Similarly, it is straightforward to write recursive procedures (i.e., subroutines capable of calling on themselves either directly or indirectly by a circular chain of calls).
5. The general conventions are such that the call, save, and return macros used to link one independently compiled procedure to another do not depend on whether or not the two procedures are in the same segment.
6. Each user is provided with a private software "stack" for temporary storage within each subroutine. Of course, any user can choose to ignore this storage mechanism, but it is available and does not have to be added as an afterthought by a subsystem designer.

In addition, there is basically only one kind of calling sequence, thus avoiding much confusion. System programming is done with the same facilities, tools, etc., available to the ordinary user, and system programs do not have to be written with special forethought. It is anticipated that the system will be open-ended and will be largely created by the users themselves; many of the useful languages and subsystems will undoubtedly be contributed without solicitation. For this reason supervisor and user programs are constructed with similar form, and processes such as paging do not distinguish between user and supervisor programs. (Of course, a few key pieces of the supervisor are locked in core memory.) Thus there is no intrinsic limit on the size of the supervisor program nor on the complexity or the features which it may have. The avoidance of a size limitation will be of major value as the system services grow.

It is important to recognize that the average user of the system will see no part of the segmentation and paging complexity described in the paper by Glaser et al. Instead he will see a virtual machine with many system characteristics which are convenient to him for writing either single programs or whole subsystems. As a subsystem writer he must be able to make the computer appear to have any particular form ranging from an airline reservations system, to an inventory control system, from a management gaming machine, to even a "FORTRAN machine" if so desired. There are no particular restrictions on the kinds of new systems or languages which can be imbedded.

Further features which should ultimately appear in the system are:

1. the ability to have one process spawn other processes which run asynchronously on several processors (thus improving the real-time response of the overall process);
2. the ability for data bases to be shared among simultaneously operating programs. In addition the system will include all the major features of the present Project MAC system such as interconsole messages and macro-commands. The latter allow users to concatenate sequences of console-issued commands as short programs thereby forming more elaborate commands which can be used with a single name and parameter call.

Another feature of the system is that it will include batch processing facilities as a subset. In particular, users will start processes which may have n terminals attached, with n= 1 for individual man-machine interaction, and n= 0 for running an absentee-user program, the latter case corresponding to batch processing. A user will be able to transform conveniently a process back and forth between the zero and one terminal states. In addition, for the purposes of teaching machines and gaming experiments, it will be possible to attach to a process an arbitrary number of additional terminals.

The supervisor will, of course, do scheduling and charging for the use of resources. Scheduling policies will be similar but more general than those currently in the MAC system; for batch processing, jobs should be scheduled so that a user will be able to obtain a quotation of maximum completion time. The time accounting done by the system will be accurate to a few microseconds. In particular, the system will "fight back" by charging for exactly what equipment is used (or others are prevented from using). In this way, orderly system expansion will be possible since the particular equipment charges which are collected will always allow further acquisition of equipment. In addition the system will incorporate hierarchical control of resource allocations and accounting authorizations. A project manager will be able to give computing budgets to group leaders who in turn will be able to delegate flexibly and straightforwardly sub-budgets to team leaders, etc. An important aspect of this resource allocation and budgeting is the ability of any member of the hierarchy to reallocate flexibly those resources over which he has control. With control of the resource allocation and administrative accounting decentralized, the operation of systems which serve hundreds of persons becomes manageable.

In a similar way, system programming is decentralized. For example, the maintenance of the system might not be entirely under the control of a single group; instead particular translators might be delegated to independent subgroups of system programmers. This isolation and distribution of responsibility is considered mandatory for the growth of large, effective systems. Hierarchical and decentralized accounting and system programming is made possible by a highly organized file system which controls the access rights to the secondary memory of the system and thus to the file copies of the vital procedures and data of the system.

# Design Considerations In The File System

The file system is a key part of a time-sharing or multiplexed system. It is a memory system which gives the users and the supervisor alike the illusion of maintaining a private set of segments or files of information for an indefinite period of time. This retention is handled by automatic mechanisms operated by the supervisor and is independent of the complex of secondary storage devices of different capacity and access. A scheme, such as is described in the paper by Daley and Neumann,[3] where all files of information are referred to by symbolic name and not by address, allows changes in the secondary storage complex for reasons either of reliability or capacity. In particular, the user is never responsible for having to organize the movement of information within the secondary storage complex. Instead the file system has a strategy for arranging for high speed access to recently used material.

Of considerable concern is the issue of privacy. Experience has shown that privacy and security are sensitive issues in a multi-user system where terminals are anonymously remote. For this reason, each user's files can be arranged to be completely private to him. In addition, a user may arrange to allow others to access his files selectively on a linking basis. The linking mechanism permits control over the degree of access one allows (e.g., a user may wish a file to be read but not written). The file system al lows files to be simultaneously read but automatically interlocks file writing. The file system is designed with the presumption that there will be mishaps, so that an automatic file backup mechanism is provided. The backup procedures must be prepared for contingencies ranging from a dropped bit on a magnetic tape to a fire in the computer room.

Specifically, the following contingencies are provided for:

1. A user may discover that he has accidentally delete a recent file and may wish to recover it.
2. There may be a specific system mishap which causes a particular file to be no longer readable for some "inexplicable" reason.
3. There may be a total mishap. For example, the disk-memory read heads may irreversibly score the magnetic surfaces so that all disk-stored information is destroyed.

The general backup mechanism is provided by the system rather than the individual user, for the more reliable the system becomes, the more the user is unable to justify the overhead (or bother) of trying to arrange for the unlikely contingency of a mishap. Thus an individual user needs insurance, and, in fact, this is what is provided.

# Design Considerations In The Communication And Input/Output Equipment

A design feature of the system is that users can view most input/output devices uniformly. Thus a program can read from either a terminal or a disk file, or output can be sent either to a file or to a punch, a typewriter, or a printer. In particular, the user of the system does not have to rewrite his program to change these assignments from day to day or from use to use. The symmetric use of equipment is, of course, highly desirable and makes for greater simplicity and flexibility.

A typical configuration of the Multics system will contain batch processing input/output devices such as card readers, punches and printers and these normally will be centrally located at the main computing installation. For remote users there will be terminals such as the Model 37 Teletype which uses the revised ASCII code with upper and lower case letters. The Model 37 Teletype also can operate on the TWX network of the Bell System. It will therefore be possible for many of the 60,000 TWX subscribers to be, if authorized, users of a Multics installation. An additional standard terminal for the Multics system will be a modified version of the IBM 1052 console. This unit (and all other terminal devices which do not have the ASCII character set) will have software escape conventions, defined to allow unambiguous input or output of the complete ASCII character set. The escape conventions are general and allow even primitive devices (in a graphic sense) to communicate with the system. The IBM 1052 terminals, which basically use the Selectric typewriter mechanism, are operated with a special typeball, prepared for Project MAC as a compromise subset of the ASCII graphics.

For those users who wish to have remotely located satellite substations capable of punching and reading cards and line printing, there are a variety of options available. Because the design of the General Input/Output Controller is relatively flexible, it is possible to use the GE 115, the Univac 1004, or virtually any other similar subcomputer as a terminal, provided one is prepared to implement the necessary interface program modules within a Multics system. At present none of these terminals are completely satisfactory since the full 128-code revised ASCII character set is not standard and excessive use of the software escape mechanism is required for printing.

In general, the area of remote terminal equipment is considered to be in an early state of development. Equipment innovations are expected, as it becomes evident that systems are capable of supporting their use. Terminals with graphical input/output are highly desirable although at present costly. The initial approach of the Multics system will be such that there will be no standard graphical input/ output terminal although several special projects are being attempted. The system viewpoint initially will be that all graphical input/output will be with small, dedicated computers capable of handling the immediate interrupts. These small computers may multiplex a few terminals and in turn appear to be not too demanding to the main system. Thus the main system interrupt load will not become excessive. In a similar way the need for real-time instrumentation such as in monitoring experimental apparatus is expected to be handled initially on a nonstandard basis. The philosophy is the same as with graphical input/output, namely, to employ small, dedicated computers for handling the real-time interrupts so as to draw upon the main system for major processing of information in a more leisurely way.

## General Considerations

It is expected that the ultimate limitation on the exploitation of the Multics system will be the knowledge which the user has of it. As a consequence, documentation of what the system contains is considered to be one of the most important aspects of the system. For this purpose a technique has been developed wherein the main system reference manual is to be maintained on-line in a fashion similar to what is currently being done at Project MAC. This allows any user of the system to obtain a current table of contents with changes listed in reverse chronological order. Thereby he can keep abreast of all system changes. Because the manual text is on-line, one is able to obtain immediate access to the latest changes at any hour or at any terminal. The on-line storage of the text also lets the system documentation group, by using appropriate editing programs, make global revisions whenever necessary. Of course, the distribution of manual revisions will still be handled in the ordinary way in that revised manual sections will be available at document rooms. Furthermore, it should be clear that there is no substitute for a good editor maintaining discipline over the documentation and for intelligent selectivity in the reference material. A documentation technique such as the one given here is believed to be an absolute necessity when users of the system no longer visit a computation center in the course of their daily activities. The user who is 200 miles away from the computer installation should have nearly the same knowledge about the system as the one who is 20 feet away.

Another area of consideration is that of compatibility with batch processing. In the Multics system for the GE 645, it will be possible to use simultaneously, but independently, the GECOS batch processing system; user jobs operating under GECOS should behave exactly as they do on the GE 625 or GE 635 computers. Effort will be made to allow the GECOS user to change conveniently to the Multics frame of operation but there will be no particular attempt made for compatibility between the two systems of basically different design. A user of the GECOS system may continue to use the GECOS system until he is prepared to make a change to the Multics system at his own place, time, and choosing. This, of course, relieves a manager installing a Multics system of the transient effect of several hundred persons changing their computing habits in one day and thus allows distribution of the normal dissatisfaction that arises under such circumstances.

One of the inevitable questions asked of a multiple-access system is what capacity it will have for simultaneous on-line users. The answer, of course, is highly dependent upon what the users are doing. Clearly, if they are requesting virtually nothing, one can have a nearly infinite number of terminals. Conversely, if one person wishes, for a single problem, system resources which equal the entire computing system, it is conceivable, if the scheduling policy allows it, that there can be only one terminal attached to the system. If one assumes that the service requirements are similar to those which have been experienced at Project MAC, then on the basis of simple scaling of processor and memory speed it is expected that the system will be able to serve simultaneously a few hundred users. But it is hazardous to predict any firm numbers; rather the pertinent parameters in a system of this type will always be the cost-performance figures. Performance, of course, is somewhat subjective, but the issues are not those of memory speed, processor speed or input/output speed. Instead the user should judge a system by the quality and variety of services, the response times, the reliability, the overall ease of understanding the system, and the performance with respect to the interface of the system which he uses. For example, pertinent questions for a PL/I user to ask are how costly, on the average, the translator is per statement, how easy it is to debug the language, and how efficiently the object code produced by the translator runs. Here, the object code referred to is that for an entire problem and not just for isolated "kernels"; the efficiency refers to the total resource drain required to execute the problem. and thereby includes the input/output demands as well.

# Conclusions

The present plans for the Multics system are not unattainable. However, it is presumptuous to think that the initial system can successfully meet all the requirements that have been set. The system will evolve under the influence of the users and their activities for a long time and in directions which are hard to predict at this time. Experience indicates that the availability of on line terminals drastically changes user habits and these changes in turn suggest changes and additions to the system itself. It is expected that most of the system additions will come from the users themselves and the system will eventually become the repository of the procedure and data knowledge of the community.

The Multics system will undoubtedly also open up large classes of new uses not only in science and engineering but also in other areas such as business and education. Just as introduction of higher-level programming languages, such as FORTRAN, in creased by an order of magnitude the number of persons using computers, multiple access systems operated as a utility will substantially extend the exploitation of information processing systems to the point of having significant social consequences. Such social issues are explored in a companion paper by David and Fano.[5]

# References


1. E. L. Glaser, J. F. Couleur and G. A. Oliver, "System Design of a Computer for Time-Sharing Applications", this volume.
2. V. A. Vyssotsky, F. J. Corbató and R. M. Graham, "Structure of the Multics Supervisor," this volume.
3. R. C. Daley and P. G. Neumann, "A General Purpose File System for Secondary Storage," this volume.
4. J. F. Ossanna, L. E. Mikus and S. D. Dunten, "Communications and Input/Output Switching in a Multiplex Computing System," this volume.
5. E. E. David, Jr., and R. M. Fano, "Some Thoughts About the Social Implications of Accessible Computing," this volume.
6. "IBM Operating System/360, PL/I: Language Specifications," File No. S360-29, Form C28-6571-1, I.B.M. Corp.
7. C. Strachey, "Time Sharing in Large Fast Computers," Proceedings of the International Conference on Information Processing, UNESCO, June 1959, paper B. 2. 19.
8. J. C. R. Licklider, "Man-Computer Symbiosis," *IRE Transactions on Human Factors in Electronics*, vol. HFE-1, no.

1, pp. 4-11, Mar. 1960.
9. J. McCarthy, "Time-Sharing Computer Systems," *Management and the Computer of the Future* (M. Greenberger, ed.), M.I.T. Press, Cambridge, Mass, 1962, pp. 221-236.
10. F. J. Corbató, M. M. Daggett and R. C. Daley, "An Experimental Time-Sharing System," Proceedings of the Spring Joint Computer Conference, 21, Spartan Books, Baltimore, 1962, pp. 335-344.
11. F. J. Corbató et al, *The Compatible Time Sharing System: A Programmer's Guide*, 1st ed., M.I.T. Press, Cambridge, Mass.,4963.
12. J. Schwartz, A General Purpose time-sharing System, Proceedings of the Spring Joint Computer Conference, 25, Spartan Books, Washington, D.C., 1964, pp. 397-411.
13. J. B. Dennis, "A Multiuser Computation Facility for Education and Research," *Comm. ACM*, vol. 7, pp. 521-529 (Sept. 1964).
14. S. Boilen et al, "A Time-Sharing Debugging System for a Small Computer," Proceedings of the Spring Joint Computer Conference, 23, Spartan Books, Baltimore, 1963, pp. 51-58.
15. H. A. Kinslow, "The Time-Sharing Monitor System," Proceedings of the Fall Joint Computer Conference, 26, Spartan Books, Washington, D.C., 1964, pp. 443-454.
16. "The Dartmouth Time-Sharing System," Computation Center, Dartmouth College, Oct. 19, 1964.
17. "PDP-6 Time-Sharing Software," Form F-61B. Digital Equipment Corp., Maynard, Mass.
18. W. W. Lichtenberger and M. W. Pirtle, "A Facility for Experimentation in Man-Machine Interaction," this volume.
19. G. J. Culler and R. W. Huff, "Solution of Nonlinear Integral Equations Using On-line Computer Control," Proceedings of the Spring Joint Computer Conference, 21, Spartan Books, Baltimore, 1962, pp. 129-138.
20. G. J. Culler and B. D. Fried, "The TRW Two-Station, Online Scientific Computer: General Description," Computer Augmentation of Human Reasoning, Washington, D. C., June 1964, Spartan Books, Washington, D.C., 1965.
21. "Carnegie Institute of Technology Computation Center User's Manual."
22. J. C. Shaw, "JOSS: A Designer's View of an Experimental Online Computing System," Proceedings of the Fall Joint Computer Conference, 26, Spartan Books, Washington, D.C., 1964, pp. 455-464.
23. T. M. Dunn and J. H. Morrissey, "Remote ComputingãAn Experimental System," Part 1; J. M. Keller, E. C. Strum and G. H. Yang, Part 2, Proceedings of the Spring Joint Computer Conference, 25, Spartan Books, Washington, D.C., 1964, pp. 413-443.
24. J. I. Schwartz, "Observations on time-shared Systems," A CM

Proceedings of the 20th National Conference, p. 525 (1965).
25. "Time-Sharing System Scorecard, No. 1 (Spring 1965)," Computer Research Corp., 747 Pleasant St., Belmont, Mass.
26. R. M. Fano, "The MAC System: The Computer Utility Approach," *IEEE Spectrum*, vol. 2, pp. 56-64 (Jan. 1965).
27. P. M. Morse, "Computers and Electronic Data Processing," *Industrial Research*, vol. 6, no. 6, p. 62 (June 1964).
28. T. Kilburn, "One-level Storage System," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 2 (Apr. 1962).
29. A. W. Holt, "Program Organization and Record Keeping for Dynamic Storage Allocation," *Comm. ACM*, vol. 4, pp. 422-431 (Oct. 1961).
30. J. B. Dennis, "Program Structure in a multi-access Computer," Tech. Rep. No. MAC-TR-11, Project MAC, M.I.T., Cambridge, Mass. (1964).
31. J. B. Dennis, "Segmentation and the Design of Multiprogrammed Computer Systems," IEEE International Convention Record, Institute of Electrical and Electronic Engineers, New York, 1965, Part 3, pp. 214-225.
32. J. B. Dennis and E. L. Glaser, "The Structure of Online Information Processing Systems," Information Systems Sciences: Proceedings of the Second Congress, Spartan Books, Washington, D.C., 1965, pp. 1 -11.
33. J. W. Forgie, "A Time- and Memory-Sharing Executive Program for Quick-Response Online Applications," this volume.
34. M. N. Greenfield, "Fact Segmentation," *Proceedings of the Spring Joint Computer Conference*, 21, Spartan Books, Baltimore, 1962, pp. 307-315.
35. "The Descriptor," Burroughs Corp., 1961.
36. "Univ. of Mich. Orders IBM Sharing System," *EDP Weekly*, vol. 6, no. 5, p. 9 (May 24, 1965).
37. B. W. Arden et al, "Program and Addressing Structure in a Time-Sharing Environment" (submitted for publication).
38. *Computing Report for the Scientist and Engineer*, Data Processing Division, I.B.M. Corp., vol. 1, no. 1, p. 8 (May 1965).
39. W. T. Comfort, "A Computing System Design for User Service," this volume.
40. "Proposed Revised American Standard Code for Information Interchange," *Comm. ACM*, vol. 8, no. 4, pp. 207-214 (Apr. 1965).
41. P. A. Crisman, ed., *The Compatible time- sharing System: A Programmer's Guide*, 2nd ed., M.I.T. Press, Cambridge, Mass., 1965.
42. A. L. Samuel, "Time-Sharing on a Computer," *New Scientist 26*, 445 (May 27, 1965) 583-587.

* *Work reported herein was supported (in part) by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01).*

---

*1965 Fall Joint Computer Conference*

---

*thvv@multicians.org*