HIGHLIGHTS OF THE MULTICS SYSTEM

The introductory parts of this Programmers' Manual are divided into two chapters. This first chapter, titled "INTRODUCTION TO THE CONCEPTS OF MULTICS", concentrates on the motivation underlying the concepts of Multics, such as virtual memory, direct addressing of permanent information files, dynamic linking, etc. The second chapter, titled "INTRODUCTION TO THE USE OF MULTICS", discusses how to use the Multics system in practice, with examples of typical ways of using the system. Both of the introductory chapters are intended to be read as a textbook rather than as reference; they provide the background for understanding the detailed reference material which follows Chapter II.

The material of the first chapter is borrowed almost entirely from published papers and internal working documents. This preliminary draft does not include the appropriate credits to the original places of publication; such credits will be added before the manual is widely distributed.

The first section breaks down as follows:

1. Introduction
2. System Requirements
3. The Multics System
4. The Hardware System
5. Overview of Multics Capabilities
6. Languages
7. Reliability and Performance

1. Introduction

Multics (from: Multiplexed Information and Computing Service) is the name of a new general purpose computer system developed by the Computer System Research group at M.I.T. Project MAC, in cooperation with the General Electric Company and the Bell Telephone Laboratories. This system is designed to be a "computer utility", extending the basic concepts and philosophy of the Compatible Time-Sharing System (CTSS, operating now on the IBM 7094 computer) in many directions. Multics is implemented initially on the General Electric 645 computer system, an enhanced relative of the GE 635 computer.*

---

*The immediately following material is drawn largely from reference C.1 of the Bibliography (see Section I.1.2).

One of the overall design goals of Multics 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 sequel, 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 has been done in a subset of the PL/I language. 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.

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. 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.

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. 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 M.I.T. community is seen in the achievements of persons 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 extends 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 provides 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 can in turn stimulate a corresponding enrichment of many areas of our society. Over six 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. Objectively, 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 multiple-access systems on business and industrial organizations can be equally dramatic. 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.

2. 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 large 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. 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 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 time-sharing 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 time-sharing 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.

3. 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 reqirements. 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 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.

An ability to share data contained within the framework of a general purpose time-sharing system is a unique feature of Multics, and is directly applicable to administrative problems, research requiring a multi-user accessible data base, and general application of the computer to very complicated research problems. The attention paid to mechanisms to provide and control privacy is of direct interest for several of the same applications as well as, for example, medical data. Multics can thus be a valuable tool which provides opportunities for important new research in these areas.

4. The Hardware System

The General Electric 645 Computer System is a large-scale, information processing system with most of the features currently found in such systems. If one attempted to classify systems, it would fall in the same general category of size as the GE 635, the Univac 1108, and the IBM Systems 360/65 and 67.

The configuration at M.I.T., shown in figure 1, contains 384k (k = 1024) 36 bit words of core memory (1 microsec. access to 36 bits or 1.3 microsec. access to 72 bits), 2 central processors (1-2 microsec. for most instructions), a high performance paging drum, (moves 1024 words in 2 ms., 16 ms. average latency with queue-driven channel controller), 34 million words of disk storage and a General I/O Controller which handles magnetic tapes, card equipment, and high-speed full ASCII printers, as well as all telecommunications channels. The central processor is built on the GE-635 instruction set, with augmentation to permit control of paging and segmentation hardware.

The configuration of figure 1 will be augmented in September, 1970, with an IBM 2314 replaceable disk pack unit, providing 36 million additional 36 bit words of on-line (and potentially detachable) storage.

5. Overview of Multics Capabilities

Multics offers a number of capabilities which go well beyond those provided by many other systems. Those which are most significant from the user's point of view are described here. Perhaps the most interesting aspect of all is that a single system encompasses all of these capabilities simultaneously.

1. The ability to be a small user of Multics.

   An important difference between Multics and CTSS is that Multics provides a really small user with a proportionally small cost. For example, a student can be handed a limited set of tools, can do limited work (perhaps debugging and running small FORTRAN programs), and expect to receive a bill for resource usage which is substantially smaller than the corresponding CTSS-like user. If all users are small, then of course the number of users can be increased in proportion to their smallness. An underlying consideration throughout the Multics design has been that the simple user should not pay a noticeable extra price for a system which also accommodates the sophisticated user. As an administrative aid, facilities are provided so that one can restrict any particular user to a specific set of tools, and thereby limit his ability to use up resources.

2. The ability to control sharing.

There are a variety of applications of a computer system which involve building up a base of information which is to be shared among several individuals. Multics provides facilities in two directions.

Sharing:

- CTSS-style links to other users' files.
- Ability to move one's base of operation into another user's file directory (with his permission).
- Direct access with uniform conventions to any file stored in the system.
- Ability for two or more users to share a single copy of a file as data in core memory.

Control:

- Ability to specify precisely to whom, and with what access mode (e.g., read, write, and execute permissions are separate and per-user) a piece of data or the entire contents of a sub-directory are available.
- Ability to revoke access at any time. (A flaw in CTSS on this point has been corrected in the Multics design.)

- Ability, using the Multics "protection ring" structure, to force access to a data base to be only via a program supplied by the data base owner. This facility may be used to allow access to aggregate information, such as averages or counts, or specified data entries, without simultaneously giving access to the entire file of raw data, which may be confidential. There are a large number of potential administrative applications of this feature, and as far as is known, Multics is the only general-purpose system which provides it.

3. The virtual memory approach.

In a direction diametrically opposed to the little user is the person with a difficult research problem requiring a very large addressable memory. The Multics File System, with the aid of a high-performance paging drum, provides this facility in what is often called a "virtual memory" of extent limited only by the totality of secondary storage (drums, disks, etc.) attached to the system. An interesting property of the Multics implementation is that a procedure may be written to operate in a very large virtual memory, but core resources are used only for those parts of the virtual memory actually touched by the program on that execution, and disk and drum resources are used only for those parts of the memory which actually contain data. Another very useful property from a programmer's point of view is that files stored in the "file system" are directly accessible to his program by a virtual memory address. This property eliminates the need for explicitly programmed "overlays", "chain links", or "core loads", and also reduces the number of explicitly programmed input and output operations. Following the same style of operation as CTSS, the Multics File System takes on the responsibility for safe keeping of all information placed there by the user. It therefore automatically maintains tape copies of all files which have remained in the system for more than an hour. These tapes can be used to reload any user files lost or damaged as a result of hardware or software failures, and may also be used to retrieve individual files damaged by a user's own programming blunder.

Each user has an administratively set quota of space which limits the amount of storage he can use, although he may purchase as large an amount of space as he would like; additional disk storage can be added to the 645 in large quantities if necessary.

4. The option of dynamic linking.

In constructing a program or system of programs, it is frequently convenient to begin testing certain features of one program before having written another program which is needed for some cases. Dynamic linking allows the execution of the first program to begin, and a search for the second program is undertaken only when (and if) it is actually called by the first one. This feature also allows a user to freely include in his program a conditional call out to a large and sophisticated error diagnostic program, secure in the knowledge that in all those executions of his program which do not encounter the error, he will not pay the cost of locating, linking, and mapping into his virtual memory the error diagnosis package. It also allows a user borrowing a program to provide a substitute for any subroutine called by that program when he uses it, since he has control over where the system looks to find missing subroutines. In those cases where subroutine "A" calls subroutine "B" every time, there is, of course, no need to use dynamic linking (and the implied library search) so facilities are provided to "bind" "A" and "B" together prior to execution.

5. Configuration flexibility.

An important aspect of the Multics design is that it is actually difficult for a user to write a program which will stop working correctly if the hardware configuration is changed. In response to changing system-wide needs, the amount of core memory, the number of central processors, the amount and nature of secondary storage (disks, drums, etc.), and the type of interactive typewriter consoles may change with time over a range of 2 or 3 to 1 but users do not normally need to change their programs to keep up with the hardware. The system itself can adapt to most major configuration changes (e.g., more memory) by re-initiatilizing itself, an operation which takes a few minutes.

6. The Human Interface.

Experience with CTSS has proven that ease of use of a time-sharing system is considerably more sensitive to human engineering than is a batch processing system. The Multics command language has been designed with this aspect in mind. Features such as universal use of a character set with both upper and lower case letters in it and allowing names of files to be 32 characters long are examples of the little things which allow the non-specialist to feel that he does not have to discover a secret code in order to be an effective user. In a similar vein, a hierarchical file system provides a very useful file organization and bookkeeping aid, so that a user need keep immediately at hand only those files he is working with at the moment. Such a facility is of great assistance when attacking complicated or intricately structured problems.

6. Languages

Multics currently provides two primary user languages: FORTRAN IV and PL/I. The FORTRAN compiler is fairly standard with a speed of compilation comparable to or a little slower than that of the extremely fast MAD compiler on 7094 CTSS. It is supported by the usual library of math routines and formatted input/output facilities. FORTRAN IV is probably the best language available for low-budget or student use.

The PL/I compiler for Multics is quite interesting, because it offers a very full selection of language facilities, over 300 helpful error diagnostics, and ability to "get at" the advanced features of Multics all at a reasonable cost. On a "seconds to translate a source language page" basis, the PL/I compiler currently takes about twice as long as does the FORTRAN compiler; on the other hand, a page of PL/I program can express considerably more than a page of FORTRAN program. For these reasons, as well as the anticipated wide availability of PL/I on other computer systems, it is the recommended language for sub-system implementers and general research users needing an expressive language.

Also available is a translator and editor subsystem for the BASIC language, developed at Dartmouth college. A "Limited Multics" service will soon be available which restricts the user to just this subsystem, if desired. The BASIC subsystem is also available to regular Multics users.

An implementation of the APL language, a powerful and popular interpretive language developed by Kenneth Iverson is under development. It is expected to be available late in 1970 or early in 1971.

A few other languages are available in the "Author-Maintained Public Library" and are not considered part of Multics; their maintenance responsibility rests with the authors (or enthusiastic users) of the translator programs. These include:

BCPL - "Basic Compiler Programming Language," developed by Martin Richards, and recommended as a good alternative where machine language might be indicated. (Also available on 7094 CTSS, System 360, and GE-635.)

EPLBSA - A machine language assembler for the GE-645 (not recommended for general use, it is very slow and the machine language is very difficult).

QED - A programmable editor which qualifies as a minor interpretive language. (Also available on 7094 CTSS and GE-635).

All of the above languages translate a source program which has been previously stored in a file. Input and editing of source files is done with one of the available text editors, EDM (a close relative of CTSS TYPSET and EDL) or QED. Although interactive, line-by-line syntax checking languages are easily implemented in the Multics environment, none are yet available.

7. Reliability and Performance

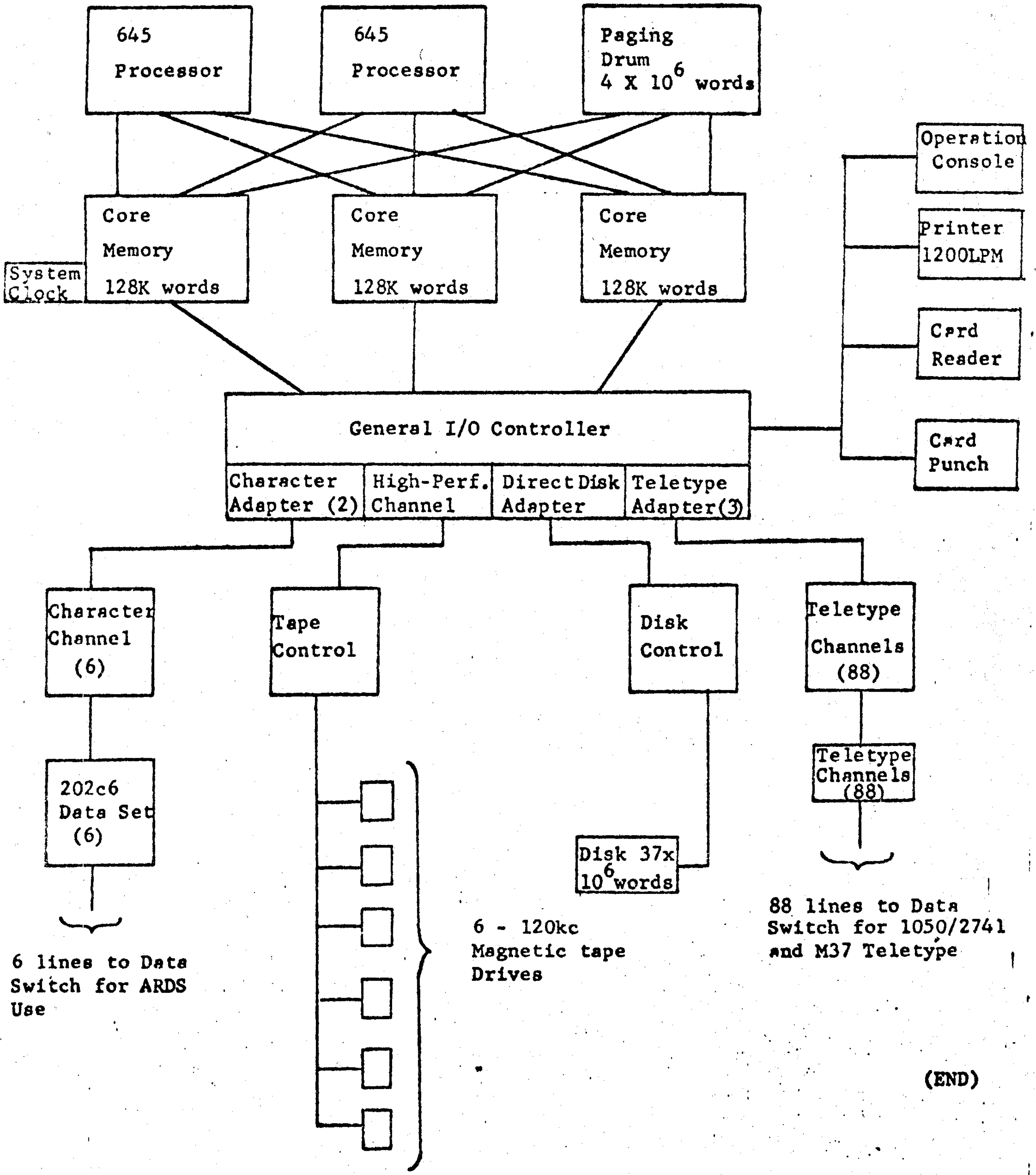
An initial version of Multics began operating on a scheduled daily basis for system programming use in September, 1968. It has been scheduled to run on a 24-hour-a-day basis since May 1, 1969. Since that time, over a year of operational experience has been obtained. During this time, reliability, functional capabilities, and performance have been brought to the point that, as of April 1, 1970, a one-processor system serves 35 to 40 users, one restricted batch stream, and two intermittent batch streams, simultaneously.

The full configuration shown in Figure 1 is expected ultimately to handle about 90 CTSS-class users, at a price per user about half that of CTSS, when doing comparable jobs. Both smaller and larger users are also runnable on the system in increased and reduced numbers, respectively.

Available hardware improvements are expected to someday provide as much as another factor of two in cost/performance. This benefit would be realized in terms of either more or bigger users accommodatable on the same hardware configuration.

It is, of course, hazardous to discuss 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 must 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 cost of false starts, ease of training a new helper, and so on.

GE645 Configuration at M.I.T.



(END)