CHAPTER 1

HIGHLIGHTS OF THE MULTICS SYSTEM

February 8, 1972

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 Honeywell Information Systems (formerly the General Electric Company computer department) 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 Honeywell 645 computer system, an enhanced relative of the Honeywell 635 computer.

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 otherwise be untried.

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 on-line at a typewriter-like terminal. 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. This goal led to the development of the Compatible Time-Sharing System (CTSS) at M.I.T. Project MAC. 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 programs 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 CTSS has had on the M.I.T. community is for other the way problems are attacked, but has caused important research to be undertaken that otherwise would not have been done. 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 ten 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 traditional needs are viewed as a subset of the broader, more demanding, new requirements.

To meet the above objectives, issues such as response time, convenience of manipulating data and programs, ease of controlling processes during execution, and, above all, protection of private information 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. 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 printing of output. In fact, they encompass the goals of many people. 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 elaborate programming systems, compilers, and features which make a computer "powerful". This results partly because it is more difficult to prepare system programs for smaller computers when limited by speed or memory size, and partly because large systems involve more persons and, hence, permit more attention to be given to 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 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 than other types of computing systems. 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 are 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 not in a position to do 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 of a computing system. 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 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; second, in an age of increasing complexity, there 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.

The Hardware System

The Honeywell 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 Honeywell 635, the Univac 1108, and the IBM Systems 360/65 and 67.

The configuration at M.I.T., shown in Figure 1-1, currently contains 384K (K = 1024) 36 bit words of core memory (1 microsecond access to 36 bits or 1.3 microseconds access to 72 bits), two central processors (330,000 instructions per second

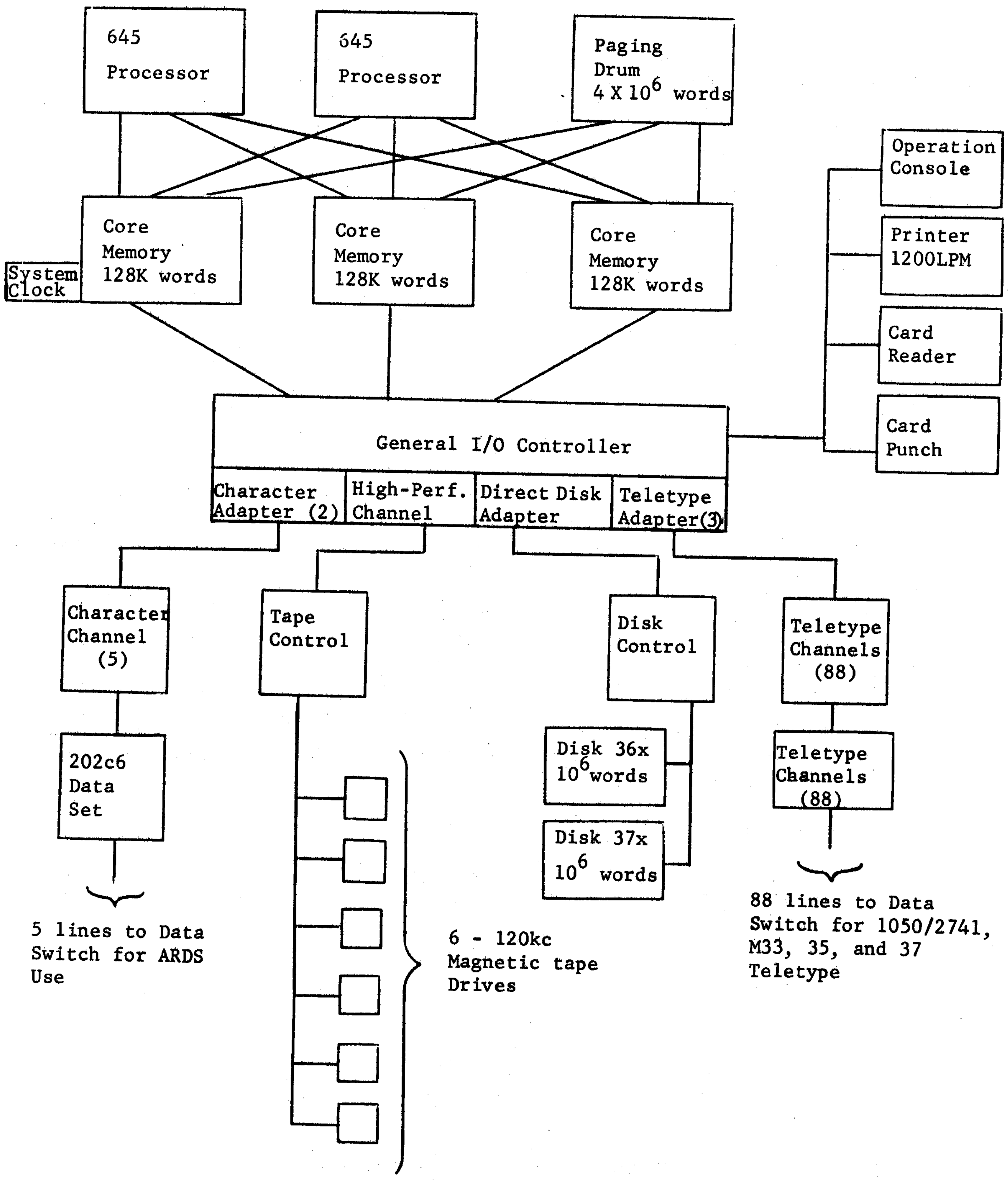

Figure 1-1: Honeywell 645 Configuration at M.I.T.

when running Multics), a high performance paging drum (it moves 1024 words in 2 milliseconds, 16 milliseconds average latency with a queue-driven channel controller), 78 million words of disk storage, and a Generalized I/O Controller which handles magnetic tapes, card equipment, and high-speed full ASCII printers, as well as all telecommunication channels. The central processor is built on the Honeywell 635 instruction set with augmentation to permit control of paging and segmentation hardware.

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 underlying consideration throughout the Multics design has been that the simple user should not pay a noticeable extra price for a system which also accomodates the sophisticated user. 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 equivalent to the limited work done. If all users are small, then, of course, the number of users can be increased in proportion to their smallness. 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 of information.

   There are a variety of applications for 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:

   - Links to other users' programs and data.
   - Ability to move one's base of operation into another user's directory (with his permission).
   - Direct access with uniform conventions to any information stored in the system.
   - Ability for two or more users to share a single copy of a program or 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 subdirectory are available.

. Ability to revoke access at any time.

. 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 the opposite direction of the little user is the person with a difficult research problem requiring a very large addressable memory. The Multics storage system, with the aid of a high-performance paging drum, provides this facility in what is often called a virtual memory of an extent limited only by the total of secondary storage devices (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 information stored in the storage system is 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. The Multics storage system takes on the responsibility for safekeeping of all information placed there by the user. It therefore automatically maintains tape copies of all information which has remained in the system for more than an hour. These tapes can be used to reload any user information lost or damaged as a result of hardware or software failures, and may also be used to retrieve individual items damaged by a user's own 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 computer 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 if and when 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 terminals 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 adapts to changes in the number of processor or memory boxes dynamically, that is, while users are logged in. Most other configuration changes (e.g., the addition of disk storage units) require that the system be reinitialized, an operation which takes a few minutes.

6. The human interface.

Experience 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 in mind. Features such as universal use of a character set with both upper and lower case letters in it, and allowing names of objects to be 32 characters long, are examples of the little things which allow the nonspecialist to feel that he does not have to discover a secret in order to be an effective user of the system. In a similar vein, a hierarchial storage system provides a very useful organization and bookkeeping aid, so that a user need keep immediately at hand only those things he is working with at the moment. Such a facility is of great assistance when attacking complicated or intricately structured problems.

Languages

Multics provides two primary user languags: FORTRAN IV and PL/I. The FORTRAN compiler is fairly standard. It is supported by the usual library of math routines and formatted input/output facilities. Its primary use is for translation of already written programs which have been imported from other computer systems.

The Multics PL/I compiler is quite interesting because it offers a very full selection of language facilities, over 300 helpful error diagnostics, and the 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 subsystem implementers and general research users needing an expressive language.

Other languages are:

BASIC - A translator and editor subsystem for the BASIC language, developed at Dartmouth College. A limited Multics service is available which restricts the user to just this subsystem, if desired. The BASIC subsystem is also available to regular Multics users.

APL - A powerful and popular interpretive language developed by Kenneth Iverson.

LISP - List Processing language, version 1.5. Both an interpreter and a compiler for this popular language for "artificial intelligence" problems are available. An interesting feature of the Multics implementation is the very large structure space provided by the virtual memory.

ALM - A machine language assembler for the Honeywell 645 computer. (It is not recommended for general use; it is slow and the machine language is very difficult.)

QEDX - A programmable editor which qualifies as a minor interpretive language.

All of the above languages translate a source program which has been previously placed in the storage system. Input and editing of source text is done with one of the available text editors, edm or qedx. Although interactive, line-by-line syntax checking languages are easily implemented in the Multics environment, none are yet available.

A source language debugging system, named debug, provides the ability to inspect variables and set break points in terms of the PL/I program being debugged. It also has a variety of features to allow inspection of all aspects of the Multics execution environment.

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, almost three years of operational experience has been obtained. During this time, reliability, functional capabilities, and performance have been brought to the point that, as of January 1, 1972, a two processor system serves 55 simultaneous users, with good interactive response.

As an offering of the M.I.T. Information Processing Center, Multics has attracted a community of about 600 registered users, and an equal number of unregistered student users. These users are organized around approximately 100 projects, thus making Multics the primary source of time-sharing services at M.I.T. (As with all I.P.C. computer systems, the use of Multics is charged to its users at rates adjusted to return full hardware and running costs when the system is operating at about two-thirds capacity.)

A Multics Bibliography

A. Manuals which are Generally Available

1. Multics Programmers' Manual. An updateable reference manual giving calling sequences and reference information for all user-callable subroutines and commands. Includes an introduction to the Multics programming environment and a guide to typical ways of using the system. approx. 800 pages.

2. The Multics System: An Examination of its Structure, by E. I. Organick. A hard cover book describing in some detail how Multics works. The description is from the point of view of a programmer developing a large program or subsystem, who wishes to gain the extra insight to help him intelligently choose among available alternatives of his implementation. M.I.T. Press, Cambridge, Mass., 1972.

3. A User's Guide to the Multics FORTRAN Implementation, by R. A. Freiburghouse. A document which provides the prospective Multics FORTRAN user with sufficient information to enable him to create and execute FORTRAN programs on Multics. It contains a complete definition of the Multics FORTRAN language as well as a description of the FORTRAN command and error messages. It also describes how to communicate with non-FORTRAN programs, and discusses some of the fundamental characteristics of Multics which affect the FORTRAN user. 68 pages.

4. Multics PL/I Language Specification. A reference manual which specifies precisely the PL/I language used on Multics. 174 pages.

5. User's Guide to the Multics PL/I Implementation, by R. A. Freiburghouse, et al. Provides detailed information about how the PL/I language is embedded in the Multics programming environment. 53 pages.

6. Graphic Users' Supplement to the Multics Programmers' Manual. In the same format as the Multics Programmers' Manual, this supplement gathers in one place descriptions of the Multics Graphics System, and the commands and subroutines needed to use it. approx. 55 pages, illustrated.

B. Manuals which may be examined in the Project MAC or Information Processing Center Document Rooms

1. Multics System Programmers' Manual. In principle, a complete reference manual describing how the system works inside. In fact, this document contains many sections which are inconsistent, inaccurate, or obsolete; it is in need of much upgrading. However, its overview sections are generally accurate and valuable if insight into the internal organization is desired. approx. 3,500 pages.

2. System Programmers' Supplement to the Multics Programmers' Manual. This updateable reference manual, in the same format as the Multics Programmers' Manual, provides calling sequences of every system module. approx. 850 pages.

3. EPLBSA Programmer's Reference Handbook, by D. J. Riesenberg. A manual describing the assembly (machine) language for the Honeywell 645 computer. The language has been renamed ALM since the publication of this manual. (Needed only by programmers with some special reason to use 645 machine language.) 85 pages.

4. Honeywell 645 Processor Manual. A hardware description, including opcodes, addressing modifiers, etc. Of interest only to dedicated machine language programmers. 175 pages.

5. Subsystem Writers' Supplement to the Multics Programmers' Manual. A manual giving calling sequences of internal interfaces of the system which are user-accessible. For the sophisticated subsystem writer who feels that it is important to bypass some standard Multics facility, this manual provides some help in using interfaces one level deeper into the system. This manual is definitely not for the casual user. approx. 50 pages.

C. Technical Papers About Multics

1. Corbató, F. J., and Vyssotsky, V. A., "Introduction and Overview of the Multics System", AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 185-196.

2. Glaser, E. L., et al., "System Design of a Computer for Time-Sharing Application", AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 197-202.

3. Vyssotsky, V. A., et al., "Structure of the Multics Supervisor", *AFIPS Conf. Proc. 27* (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 203-212.

4. Daley, R. C., and Newmann, P. G., "A General-Purpose File System for Secondary Storage", *AFIPS Conf. Proc. 27* (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 213-229.

5. Ossanna, J. F., et al., "Communication and Input/Output Switching in a Multiplex Computing System", *AFIPS Conf. Proc. 27* (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 231-241.

6. David, E. E., Jr., and Fano, R. M., "Some Thoughts About the Social Implications of Accessible Computing", *AFIPS Conf. Proc. 27* (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 243-247.

7. Bensoussan, A., Clingen, C. T., and Daley, R. C., "The Multics Virtual Memory", *ACM Second Symposium on Operating Systems Principles* (October 20-22, 1969), Princeton University, pp. 30-42.

8. Clingen, C. T., "Program Naming Problems in a Shared Tree-Structured Hierarchy", *NATO Science Committee Conference on Techniques in Software Engineering, 1* (October 27-31, 1969), Rome, Italy.

9. Graham, R. M., "Protection in an Information Processing Utility", *Comm. ACM 11*, 5 (May, 1968), pp. 365-369.

10. Daley, R. C., and Dennis, J. B., "Virtual Memory, Processes, and Sharing in MULTICS", *Comm. ACM 11*, 5 (May, 1968), pp. 306-312.

11. Corbató, F. J., and Saltzer, J. H., "Some Considerations of Supervisor Program Design for Multiplexed Computer Systems", *Proc. IFIP Conf. 1968 Invited Papers*, pp. 66-72.

12. Corbató, F. J., "PL/I as a Tool for System Programming", *Datamation 15*, 6 (May, 1969), pp. 68-76.

13. Corbató, F. J., "A Paging Experiment with the Multics System", *In Honor of P. M. Morse*, M.I.T. Press, Cambridge, Massachusetts, 1969, pp. 217-228.

14. Saltzer, J. H., and Gintell, J. W., "The Instrumentation of Multics", *ACM Second Symposium on Operating System Principles* (October 20-22, 1969), Princeton University, pp. 167-174. Also in *Comm. ACM 13*, 8 (August, 1970), pp. 495-500.

15. Spier, M. J., and Organick, E. I., "The Multics Inter-Process Communication Facility", _ACM Second Symposium on Operating System Principles_ (October 20-22, 1969), Princeton University, pp. 83-91.

16. Freiburghouse, R. A., "The Multics PL/I Compiler", _AFIPS Conf. Proc. 35_ (1969), AFIPS Press, 1969, pp. 187-199.

17. Grochow, J. M., "Real-Time Graphic Display of Time-Sharing System Operating Characteristics", _AFIPS Conf. Proc. 35_ (1969 FJCC), AFIPS Press, 1969, pp. 379-385.

18. Saltzer, J. H., and Ossanna J. F., "Remote Terminal Character Stream Processing in Multics", _AFIPS Conf. Proc. 36_ (1970 SJCC), AFIPS Press, 1970, pp. 621-627.

19. Ossanna, J. F., and Saltzer, J. H., "Technical and Human Engineering Problems in Connecting Terminals to a Time-Sharing System", _AFIPS Conf. Proc. 37_ (1970 FJCC), AFIPS Press, 1970, pp. 355-362.

20. Clark, D. D., Graham, R. M., Saltzer, J. H., and Schroeder, M. D., "Classroom Information and Computing Service", M.I.T. Project MAC Technical Report TR-80, (January 11, 1971).

21. Schroeder, M. D., "Performance of the GE-645 Associative Memory While Multics is in Operation", _ACM Workshop on System Performance Evaluation_ (April, 1971), pp. 227-245.

22. Schroeder, M. D., and Saltzer, J. H., "A Hardware Architecture for Implementing Protection Rings", _ACM Third Symposium on Operating Systems Principles_ (October 18-20, 1971), Palo Alto, California.

23. Feiertag, R. J., and Organick, E. I., "The Multics Input/Output System", _ACM Third Symposium on Operating Systems Principles_ (October 18-20, 1971), Palo Alto, California.

24. Sekino, A., "Response Time Distribution of Multiprogrammed Time-Shared Computer Systems", Sixth Annual Princeton Conference on Information Sciences and Systems, March 23-24, 1972, Princeton, N.J.

25. Corbató, F. J., Saltzer, J. H., and Clingen, C. T., "Multics--The First Seven Years", _AFIPS Conf. Proc. 40_ (1972 SJCC) AFIPS Press, 1972.

D. M.I.T. Theses Related to Multics

1. Saltzer, J. H., "Traffic Control in a Multiplexed Computer System", Sc.D., 1966. (MAC-TR-30)

2. Rappaport, R., "Implementing Multi-Process Primitives in a Multiplexed Computer System", S.M., 1968. (MAC-TR-55)

3. Deitel, H., "Absentee Computations in a Multiple-Access Computer System", S.M., 1968. (MAC-TR-52)

4. Greenbaum, J., "A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System", S.M., 1968. (MAC-TR-58)

5. Grochow, J. M., "The Graphic Display as an Aid in the Monitoring of a Time-Shared Computer System", S.M., 1968. (MAC-TR-54)

6. Ancona, R. I., "A Compiler for MAD-Based Language on Multics", S.M., 1968.

7. Clark, D., "A Reduction Analysis System for Parsing PL/I", S.M., 1968.

8. Schroeder, M. D., "Classroom Model of an Information and Computing Service", S.M., 1969.

9. Vogt, C. M., "Suspension of Processes in a Multiprocessing Computer System", S.M., February, 1970.

10. Frankston, R., "A Limited Service System on Multics", S.B., June, 1970.

11. Schell, R. R., "Dynamic Reconfiguration in a Modular Computer System", Ph.D., June, 1971.

(END)*