BA.5

Second Draft
C. T. Clingen

Initial MULTICS

Design Description

and

Performance Specification

3/18/68

TABLE OF CONTENTS

	Page
I. INTRODUCTION AND OVERVIEW	1-1
A. Scope of Specification	1-1
B. MULTICS Objectives	1-1
C. Initial MULTICS Objective and Overview	1-3
1. Objective	1-3
2. Overview	1-3
II. HARDWARE CONFIGURATIONS FOR INITIAL MULTICS	2-1
III. PRINCIPAL CONCEPTS	3-1
A. The MULTICS Process	3-1
B. Segmentation	3-2
1. Segments	3-2
2. Segment Attributes	3-2
a. Definition	3-2
b. Common Attributes	3-2
c. "Private" Attributes	3-3
3. Segment Sharing Among Processers	3-4
a. Data Sharing	3-4
b. Procedure Sharing	3-4
C. Intersegment Linkage Within a Process	3-5
1. Address Transformation	3-5
2. Dynamic Linking	3-6

TABLE OF CONTENTS (continued)

	Page
D. Process Augmentation	3-8
1. Directories and the Directory Hierarchy	3-8
a. The directory hierarchy	3-8
b. Directory contents	3-9
2. Making a Segment Known	3-9
3. Attribute Extraction and Enforcement	3-10
4. Segment Creation	3-11
E. Interprocess Communication	3-11
F. Process Creation	3-12
1. Creation	3-12
2. Process Specialization	3-13
G. Process Partitioning	3-13
1. Access Domains	3-13
2. Access Domain Enforcement	3-15
3. Supervisor Protection	3-16
H. Processor Management	3-18
1. Scheduling	3-18
2. Event distribution	3-19
I. Core Management	3-20
J. Page and Segment Management	3-21
1. Segment addressing	3-22
2. Page and Segment states	3-23
3. Page and Segment Management by a Process	3-23
4. Additional Process States	3-24
K. Secondary Storage Management	3-25
1. Device Overflow and Segment Migration	3-25
L. Summary	3-27

TABLE OF CONTENTS (continued)

	Page
IV. THE MULTICS SYSTEM	4-1
A. Process Groups	4-2
1. Need for Process Groups	4-2
2. The MULTICS Process Group	4-2
3. Stopping a Process Group	4-3
B. The Initialized System	4-4
1. Initial Process Creation Sequence	4-4
2. System Control Process Group	4-5
a. System Control Process	4-5
b. Answering Service Process	4-5
3. Universal Device Manager Processes -- the I/O System	4-6
4. The System Skeleton	4-8
C. The Dialup-Login Sequence	4-8
1. Sequence of Process Creation after Dialup	4-9
2. User Control Process	4-9
3. User Process Groups	4-10
a. Overseer Process	4-11
b. Working Process	4-11
D. The Quit-Restart Sequence	4-12
E. MULTICS "Daemons"	4-14
F. Summary	4-15
V. USER INTERFACES FOR INITIAL MULTICS	5-1
A. Commands	5-1
1. The Shell	5-1
2. General User Commands	5-2
3. Operator Commands	5-4

TABLE OF CONTENTS (continued)

	Page
V. (continued)	
B. Languages	5-5
C. Supervisor Entries and Utility Procedures	5-5
1. Segment Management Module Entries	5-6
2. Directory Supervisor Entry Points	5-7
3. Segment Control Entry Points	5-8
4. I/O System Entry Points	5-8
5. Interprocess Communication Entry Points	5-9
6. Utility Procedures	5-10
a. Basic File System-associated procedures	5-10
b. I/O-associated procedures	5-11
c. Miscellaneous procedures	5-11
VI. PERFORMANCE	
A. Introduction	6-1
B. Overall Requirements	6-2
C. Response Requirements	6-4

I. INTRODUCTION AND OVERVIEW

A. Scope of Specification

The purpose of this specification is to describe a subset of MULTICS called Initial MULTICS in terms of functional capability and targeted performance characteristics. Important system concepts and the interaction of these concepts are stressed with relatively little effort being devoted to design details. The detailed design of MULTICS is described in the MULTICS System-Programmers' Manual (MSPM).

Particular attention is paid to specifying the differences between MULTICS as specified in the MSPM and Initial MULTICS as targeted for implementation by mid-1968.

B. MULTICS Objectives

The objectives of the MULTICS project have undergone surprisingly little modification since being published in the 1965 Proceedings of the Fall Joint Computer Conference.

Rephrasing these objectives somewhat, they may be summarized as follows:

- To develop a system which provides an environment for solving a wide class of problems, which is reliable, allowing for a type of service similar to that of a utility, and which is extendable or evolvable, permitting the rapid inclusion of the inevitable series of hardware and software additions and modifications which become desirable as the system matures.

B. (Continued)

- To provide each user with a "virtual computer" of extreme flexibility and possessing few limitations or restrictions so that he may solve his problems, small or large, interactive or non-interactive in a straightforward manner with no regard for the actual hardware involved. Problems solvable in the MULTICS environment represent a broad spectrum of computer application development, ranging all the way from program or application system writing, compilation and debugging to production runs by the ultimate users.

- To permit controlled interaction among all users of the system ranging from the broad authority of the MULTICS system-programmer imposing system modifications upon all other users to the "small-problem solver" running a prepackaged application program or editing a private file; or from the on-line distribution and sharing of a procedure by a programmer with his collegues to the cooperative processing, by many concurrent non-programmer users, of a large applications system of a shared volatile data base.

The MULTICS design has remained faithful to these goals in spite of the pressures of unanticipated schedule slippages of disappointing magnitude. It is gratifying, however, to note that although the MULTICS project has suffered drawbacks, it still represents the only serious effort to implement those unifying concepts which seem to point the way for all large centralized computer systems of the future. In fact, many of these concepts are applicable to any computing system of the future. It is of overriding importance to note also that MULTICS has greater potential for ultimate growth than any other system under development today.

C. Initial MULTICS Objective and Overview

1. Objective

The objective of the Initial MULTICS development effort is to provide a basic framework which:

- demonstrates the technical feasibility and practicality of the unique combination of basic concepts embodied in the MULTICS design;

- is extensible in an orderly fashion so that full MULTICS may be produced from this framework; and which

- serves as a tool for system programmers engaged in the continuing development of MULTICS.

2. Overview

In order for this three-part objective to be met, Initial MULTICS must demonstrate basic command, file, I/O and control capabilities.

Briefly, each remote user must be able to:

- log onto the system, verifying his identity by typing his password;

- create named files;

- attach file access attributes for each possible user of each of his files;

- input information to his files, editing and listing this information as desired. (In this way EPL and EPLBSA source files will be created);

2. (Continued)

- <u>compile</u> EPL procedures and <u>assemble</u> EPLBSA procedures;

- <u>execute</u> his own procedures which may in turn <u>call</u> and <u>link</u> to many other procedures, and;

- <u>log off</u> the system to terminate his console session.

In addition, an Initial MULTICS console user will be able to <u>quit</u> his running program, even while output is occurring on his terminal, and then either restart it or begin a new computation.

Procedures written and executed in the Initial MULTICS environment will be able to:

- read input from and write output to the user's remote terminal;

- communicate with programs running on the behalf of other remote terminal users, allowing for the cooperative processing of shared data bases;

- perform any of the commands (except logging in) that the user can perform at his console; and

- transfer control to (i.e., <u>call</u>) any procedure in the system to which the user has proper access privileges.

Therefore, as early as Initial MULTICS, users will have the ability to create <u>very</u> large programs comprised of a large number of procedures and data bases and possessing the capabilities provided by EPL and EPLBSA as well as the supervisory capabilities provided by calls to the system.

2. (Continued)

The System Operator in charge of the running of an Initial MULTICS configuration will have the ability to:

- <u>start</u> <u>up</u> the system;

- <u>enable</u> communications lines so users can log in;

- <u>disable</u> lines so when users log out no more can log in.

- <u>dump</u> the contents of the mass storage hierarchy onto tape; and

- <u>shut</u> <u>down</u> the system.

There are many functional capabilities which will be provided by MULTICS but which are <u>not</u> essential to meeting the Initial MULTICS objective. Therefore, for the achievement of <u>Initial</u> MULTICS, these features need not be implemented. For example:

- the "hardcore" portion of Initial MULTICS will not be capable of modification or extension without first being shut down. System compatibility will not be guaranteed from one start up to the next as extensive supervisor changes are to be expected during Initial MULTICS usage.

- A user's procedure may be denied access to all peripheral I/O devices other than the model 37 teletype. All data accessing will be handled via the File System using segment addressing.

- There will be no absentee users.

2. (Continued)

- There will be no batch processing.

- There need be no on-line media conversion such as bulk tape-to-printer or file system-to-printer information transfers.

- There will be no user access to the alarm clock feature of the system clock.

- There will be no provisions for user "traps" upon access to File System files.

- The mechanism for assigning a procedure to a particular ring will not be completely implemented via commands resulting in the need for program editing in order to do so.

- There will be no accounting facilities.

- There will be no automatic load-balancing facilities provided to optimize the use of secondary storage hierarchy devices, optimize the number of users at a given time, etc.

- There need be no on-line administrative capabilities. For example, user identifications and passwords may be pre-assembled prior to system initialization and loaded when the system is loaded.

- There will be no incremental backup; instead file dumping will occur after all users have been logged out of the system immediately prior to system shut down.

2. (Continued)

- There will be no capability to <u>save</u> a program -- i.e., suspend the execution of a running program indefinitely -- and <u>resume</u> its execution at a later date.

System performance in terms of ability to handle large numbers of users will not be a principal concern at first. However, the system is to accommodate a small number of remote users initially, that number increasing at a targeted rate as scheduled in the last part of this specification.

With the possible exception of the first item, the removal of all the above restrictions is planned to begin upon the accomplishment of the Initial MULTICS objectives.

II. HARDWARE CONFIGURATIONS FOR INITIAL MULTICS

The GE 645 hardware configurations required to support the initialization, running and dumping of an Initial MULTICS system consist of the following equipment:

Description	Quantity[1]			Quantity[2]
645 Central Processor Unit	1			2
645 System Controllers	2	4	or	3
Amount of core per controller (words)	128K	64K		128K
System Clock	1			2
EMU-302 Drum System (4×10^6 words)	1			1
DS-10 Disc Storage Unit	1			2
Generalized Input/Output Controller	1			2
Magnetic Tape Controller (with 4 Tape Drives)	1			2
Extended Character Printer	1			2
Card Reader				

1) Minimum configuration recommended for specified performance -- see Part VI.

2) Minimum configuration recommended for "continuous operation" with only brief interruptions for reconfiguration in case of trouble or preventive maintenance. Not required for Initial Multics.

Remote consoles may be limited initially to Model 37 Teletypes, with the possible additional availability of 1050 terminals. See Part VI for number of consoles to be accommodated by Initial MULTICS and its immediate successors.

III. PRINCIPAL CONCEPTS

Incorporated into Multics is a set of key concepts. Although these features are not necessarily unique to Multics, their integration into a single system is unique. An understanding of these underlying concepts is necessary to an appreciation of Multics.

The principals and concepts embodied in Multics can be divided into two classes: intrinsic, that is dealing with the facilities provided somewhat independently of the hardware limitations imposed by any realistic hardware environment; and technological, having to do with the management of actual hardware and software resources in order to provide the desired intrinsic features.

Concepts intrinsic to Multics include the following and assume the existence of a virtual machine of essentially unlimited capacity for the execution of each program.

A. The Multics Process

Definition: A Multics process is the execution of a time-varying collection of all the procedures and data bases needed to perform a task.

On the GE-645, a special segment called a descriptor segment is associated with each process. A descriptor segment defines the span or address space of a process by listing all the procedures and data currently comprising the process. The contents of a descriptor segment may be changed as the process progresses.

A. (Continued)

No more than one processor can execute on behalf of a process at any one time.

B. <u>Segmentation</u>

1. Segments

Definition: A <u>segment</u> is a directly addressable collection of data and/or instructions, which collection preserves its identity even upon becoming incorporated into a process collection. All procedures and data bases in the Multics environment are contained in segments.

This is in contrast to the more common approach of binding <u>copies</u> of data files and procedure files into a program, whereupon such copies can no longer be identified nor their attributes respected.

2. Segment Attributes

a. Definition: <u>Segment attributes</u> are rules of usage or quantities associated with a segment which may be enforced or observed while a process is using the segment. Although many kinds of attributes may be associated with a segment, the most important attributes in a shared environment have to do with the protection of segments from unauthorized usage.

b. Common attributes

Some segment attributes are identical for all users of a segment. In Initial Multics, these include:

- segment size -- a segment may contain as many as 256K words

b. (Continued)

- segment location, either in core or on secondary storage
- time segment was last accessed
- time segment was last modified

c. "Private" attributes

Other of the attributes associated with a segment -- the access attributes -- can be different for each user of the segment. In Initial Multics, permissible access attributes are:

- read
- read and write
- read, write and append
- execute only
- execute and read
- execute, read and write
- execute, read, write and append

In most cases, the read and write attributes are used for segments being treated as data and the execute attribute for segments containing procedures.

Multics access attributes <u>not</u> to be implemented in Initial Multics are:

- write only -- write but do not read
- append but no read and/or write -- add to the end of a segment but no updating or reading of data interior to the segment text
- trap -- transfer to a specified routine, usually to compute a new access attribute.

B. (Continued)

3. Segment Sharing Among Processers

Because segments do not lose their identity when incorporated into a process collection, the same segment may be shared by more than one process at the same time. This has implications fundamental to the understanding of Multics.

a. Data sharing

Since segments can contain information considered to be data by the incorporating processes, the same version of a data base can be provided simultaneously to more than one process, permitting data sharing. This leads to a notion of cooperative processing whereby a community of processes can extract information from and modify a single data base.

b. Procedure sharing

Since segments can also contain procedures, the notion of more than one process executing in a single copy of a procedure is natural. To be successful, however, no one process must be allowed to change any instructions in the procedure while it is being shared. This is enforced in Multics by a convention requiring that all shared procedures be invariant or pure, i.e., contain no modifiable data or instructions and no process-dependent addresses.

Pure procedure introduces two more Multics concepts: linkage sections and stacks.

Each Multics procedure makes references to other segments indirectly through a linkage section associated with the

b. (Continued)

procedure. A copy of the linkage section is prepared for each process using the procedure and maintained in a separate data segment private to each process. All process-dependent addresses are placed in the linkage section, thereby permitting the procedure to remain free of process-dependent addresses. Also, as a result of procedure invariance or purity, each Multics process is provided with a call stack which is a data segment used by all Multics procedures in passing control from one procedure to the next. Each procedure appends to the stack a frame of information including volatile register values at the time of the call, temporary working storage, and argument information for the target procedure. Because of this method of implementation, all Multics procedures are recursive, i.e., they may call themselves directly or indirectly before returning control to their callers.

C. Intersegment Linkage Within a Process

Segments referenced by a Multics process collection are referenced directly, not "read" or "written" as are conventional files. When a procedure in one segment calls a procedure in another segment, a direct transfer (plus the required stack frame manipulation) occurs. When a procedure references data within a data segment, a single "load" or "store" instruction is executed by the processor. However, the preparation of machine-interpretable addresses is necessary if such intersegment references are to occur.

1. Address transformation

When a procedure is written and compiled or assembled, machine-interpretable addresses for called procedures or referenced

1. (Continued)

data in other segments are not yet known. In fact, such information does not become known until the target segments are incorporated into the process collection of the referencing procedure and it is different in general for each process. So instead of assigning addresses at compile time, intersegment references are made indirectly through the linkage section which is prepared to contain fault indicators, called <u>linkage faults</u>, initially and may be thought of as being empty.

In addition, each linkage section contains the symbolic name or <u>call name</u> used by the programmer to name the target object. For example, "sin" might be the call name of a sine routine (and its corresponding procedure segment) referenced from within a procedure. Thus, the linkage section consists of a set of "empty" locations reserved for machine addresses and a corresponding set of symbolic names by which the programmer knows the referenced objects.

The problem is to insert into the linkage section the addresses corresponding to their respective symbolic names.

2. Dynamic Linking

In Multics, address insertion occurs dynamically and only upon initial reference to the target object. In this way the work associated with linking is performed only for those items actually referenced in a particular process, other references remaining unlinked.

To accomplish dynamic linking, each Multics process is provided with a private data base called the Segment Name Table (SNT)

2. (Continued)

and a set of procedures, one of which is the Linker.

The SNT contains entries each of which is a symbol pair consisting of a segment call name and its corresponding machine-interpretable address, which on the GE-645 is a segment number. Further refinements to the linkage section permit the translation of names of items within a segment to machine addresses. (Since the segment number for a given segment in general differs in different processes, linkage sections containing references to a given segment will differ by process and hence cannot be shared.)

Upon initially attempting an intersegment reference, the processor encounters the linkage fault in the linkage section and is trapped to the Linker. The Linker and its associated procedures, running as a part of the process, check the linkage section in which the fault occurred and finds the call name corresponding to the fault, search the SNT for that name discovering the corresponding segment number, and (simplifying somewhat) replace the fault indicator with the machine-interpretable segment number. Then processing resumes at the point of the fault.

The link is thus completed and subsequent references occur without the occurrence of a linkage fault.

For the sake of accuracy it is to be noted that the above simplified description of intersegment linkage and dynamic linking has ommitted a discussion of the Initial MULTICS capability link to symbolically named procedure entry points or

2. (Continued)

data items within the target segment, representing a further refinement of the ideas presented here.

D. Process Augmentation

Thus far it has been assumed that all segments required by a process are somehow made available to the process when it begins. This is not the case; a Multics process has the capability of augmenting (or reducing) the number of segments comprising its own process collection while the process runs. That is, segments contained in the system may be known or unknown to a particular process and may change from one state to the other. A known segment has been assigned a segment number within that particular process; an unknown segment has not.

Thus the notion of a dynamic process address space presents itself.

1. Directories and the Directory Hierarchy

   Since segments need not be known to any process at all times, a convenient structure for locating them and recording information about them must be provided. The Multics directory and directory hierarchy has been provided for this purpose.

   a. The directory hierarchy

      The hierarchy can be thought of as a tree structure, each non-terminal node of which is a directory. The branches leaving a node can lead to inferior directories or, in the case of terminal nodes, to data or procedure segments. The tree-like structure makes it easy to uniquely name and locate large numbers of segments and to group them into libraries of various kinds.

1. (Continued)

   b. Directory contents

      Directories are segments. Their purpose is to contain all the segment attributes for each directly inferior segment, be it a directory or a non-directory segment.

      Each directory has one or more owners. An owner can modify the contents of the directory thereby altering the attributes of the directly inferior segments described in the directory.

      In addition to the common attributes, such as length, described in a directory, there is associated with each inferior segment a list of entries -- one for each user who may access the segment. Each entry describes the exact access privileges to be permitted the corresponding user.

2. Making a Segment Known

   A major Multics objective is that of allowing a procedure to successfully reference a segment which is unknown to the procedure's process at the time of the reference. This implies that the segment must be made known dynamically so that dynamic linking can then take place.

   Normally a segment is made known as an indirect result of a linkage fault. When a linkage fault occurs for an unknown segment, the dynamic linking mechanism discovers no segment number for the call name responsible for the linkage fault. As a result, a set of supervisory modules are invoked (as a part of the process) to search the directory hierarchy for the segment with the desired call name.

2. (Continued)

The search is directed using a set of <u>search</u> <u>rules</u> (which in Initial Multics will be the same for each process and will in general refer to standard libraries of segments) by a set of modules, the most important of which are the Segment Management Module, Directory Control and Segment Control. These modules jointly cooperate to make known to the process during the course of the search those directories specified in the search rules.

3. Attribute Extraction and Enforcement

Once the desired segment has been located, another supervisory module, Access Control, checks to insure that the desired segment is accessible by the owner of the process requesting it. If so, that segment attribute information which is applicable to the requesting user is extracted from the directory and placed in data bases private to the process for rapid access.

A segment number is next assigned to the segment, which then becomes known, and access attributes extracted from the per-process data bases are prepared for the appropriate word in the Descriptor Segment so that <u>each</u> attempted access of the segment by the processor running on behalf of this process will be monitored, insuring that only that type of access specified by the segment owner is permitted. In this way attribute enforcement is hardware assisted and consequently relatively efficient.

Finally, the Segment Name Table is updated and dynamic linking can begin.

D. (Continued)

4. Segment Creation

There is another extremely important aspect of process augmentation -- namely that of segment creation. Each process has the ability to create a segment of required size (from 0 to 256K words) and to give that segment all the desired segment attributes by naming it and describing it in an appropriate directory.

Thus it is possible for a single procedure contained in a process to initiate a chain of calls to initially unknown procedure segments resulting in the ultimate inclusion into the process of all the procedures and data bases required for the task implied by the first procedure.

E. Interprocess Communication

Definition: Interprocess communication is the act of one process informing one or more other processes of the occurrence of some awaited event. This is the method used to synchronize cooperating processes so that orderly sharing of data and sequencing of computations can occur.

The single requisite for interprocess communication is the ability for more than one process to share a single mutually known data segment. However, conventions must be established among all communicating processes if the business of informing is to go smoothly.

Multics formalizes a set of interprocess communication conventions by providing each process with a set of supervisory modules to create system-standard shared communication areas called event channels. In addition, a set of supervisory procedures are provided to each

E. (Continued)

process permitting signals to be "sent" over these channels by setting events and giving "receiving" processes the ability to easily determine the existence of and nature of such signals once they have been sent by testing events. Conceptually, the receiving process can be thought of as being in a loop waiting for the occurrence of an event until it occurs before continuing its task. Supervisory procedures performing the functions of notify and wait are provided to manage the use of event channels. Wait causes the receiving process to wait for the occurrence of an event subsequently signalled by notify in the informing process.

An additional facility making use of the Interprocess Communication module is the Locker, designed to aid processes in the locking and unlocking of shared data bases so that orderly updating of the data is possible.

F. Process Creation

In the Multics environment, processes are made and destroyed as the need arises. For example, when a user is granted access to the system he is supplied with several processes to perform his tasks. This is done by allowing a process to create and destroy one or more additional processes.

In Initial Multics, only certain privileged processes are granted the right to create a process -- the user will not have this capability under his control.

1. Creation

Conceptually, process creation is rather simple. First, it consists of creating several segments on behalf of the new process.

1. (Continued)

Among these segments are the Segment Name Table and other per-process supervisory data segments so that the collection can survive on its own in the Multics environment. Secondly, it requires the recording of certain information in system-wide data bases so that the existence of the new process is properly acknowledged.

2. Process Specialization

It is necessary to provide a means by which a newly created process can be specialized -- not all MULTICS processes are identical. This is accomplished in two ways.

First, when each process is created it is assigned to a "user"; either a person logged into the system or to the system itself as a special process. Since different users have different segment access privileges, the future makeup of processes owned by different users is effectively controlled by the privileges of its owners.

Second, the process creator passes to each created process, by means of a segment called the Process Initialization Table, a procedure name to be called by the created process. When the created process begins to run, it first calls this procedure which in turn by a <u>chain</u> of calls, linkage faults, and making segments known can cause the new process to take on the desired structure.

G. <u>Process Partitioning</u>

1. Access Domains

All data and procedures referenced by a Multics process become known to the process; as such they could be indiscriminately

1. (Continued)

referenced by each procedure in the process unless somehow restricted in availability.

Perhaps the most compelling reason for restricting access to segments known to a process is the need to permit some procedures to read and write information in a <u>known</u> data segment while at the same time denying uncontrolled access to the data by other procedures (in the same process) which might violate either the conventions assumed by the data therein or the intended use of the data. Similarly, indiscriminate access to procedures operating hardware on behalf of the process can lead to chaos.

Therefore, the notion of access domains ordered by degree of privilege are an essential feature of Multics. Because of the ordering, access domains are envisioned as 64 concentric annuli, called <u>rings</u>, numbered from the center outward as 0 to 63 with the most privileged "ring" being the "bullseye" or ring zero. The interfaces separating rings are called <u>walls</u>.

Since a process collection consists of segments, process partitioning into rings implies that each segment possesses an access domain attribute or ring number describing the privileges to be granted to it. This is so in Multics, with an additional segment attribute called the <u>access bracket</u> being associated with each segment. The access bracket, which describes the rings from which a segment may be accessed, is stored in the directory describing each segment and, as is the case for the other access attributes, may differ for each user permitted to attach the segment to his process.

G. (Continued)

2. Access Domain Enforcement

The general strategy to be enforced is as follows:

- No procedure is allowed to access data residing in a more privileged (inward) ring.
- Procedure calls to other rings must be intercepted.

The implementation of ring-structured processes is complex. However, a gross understanding can be achieved from the following greatly over-simplified overview:

For _each_ ring within which a process runs, imagine for the process a collection of segments for ring n (n = 0,1,...,63) consisting of:

- A descriptor segment describing all segments known to the process but permitting access only to those residing in ring n. This is a "fragment" of "the" descriptor segment.
- A stack to be used for interprocedure calls in ring n. This is a "fragment" of the call stack.
- A linkage segment containing only those linkage sections for ring n data and procedure segments.

Each descriptor segment is initialized such that an outward data reference to a less privileged segment is permitted _provided the other access attributes allow the attempted reference_ but such that an inward data reference to a more privileged segment is prohibited.

In addition, each descriptor segment is initialized so that _outward calls_ from one procedure to a less privileged procedure which may be accessed or _inward calls_ from less privileged to

2. (Continued)

more privileged procedures which may be accessed from outer rings cause a <u>wall-crossing fault</u>. Illegal inward call attempts are prohibited upon initial reference by placing an "illegal access" fault in the appropriate ring n descriptor segment entry when it is prepared as a result of checking the access bracket for the target procedure.

When a wall-crossing fault occurs, the process traps to a set of supervisory procedures which:

- On inward calls, check the validity of using a particular segment "entry point" to insure that the location being referenced corresponds to a valid procedure entry.
- On inward calls, copy argument pointers into the inner ring and check the validity of arguments being passed.
- On inward returns, check the validity of using a particular segment "return point" to insure that the location being referenced corresponds to a valid procedure return.
- Switch the process to the target ring so that it uses the appropriate descriptor segment, stack and linkage information.

Initial Multics will not allow arguments on outward calls.

3. Supervisor Protection

In Initial Multics, the main application of process partitioning will be that of supervisor protection. That is, each process will be partitioned such that the sensitive procedures and data bases essential to the proper running of a process will reside in highly privileged rings so that only a few entry points are accessible to less privileged user-provided procedures known to the process.

G. 3. (Continued)

In Initial Multics most supervisory procedures and data will reside in ring 0 and ring 1 with user segments restricted to ring 32.

So far, the concepts introduced describe those aspects of a Multics process which are independent of hardware contraints such as the number of available central processors or the amount of core storage. They represent those facets of a Multics process which are to be presented to the user of a process. In reality, technological constraints are such that it is impossible to have a dedicated processor running for each process and enough core memory to contain all the segments in the directory hierarchy.

As a result, a set of concepts have been integrated into each Multics process giving each process the capability of "simulating" the ideal environment by <u>sharing</u> the actual hardware with all other processes.

Each process manages its own <u>facilities</u> in a manner compatible with all other processes by accessing shared system-wide data bases describing the exact state of each facility. In this context, the term facility refers not only to hardware resources, but also other "objects" which must be managed by a process such as segments and pages. This represents an excellent example of the <u>cooperative processing</u> made possible in Multics by segment sharing.

The following concepts can thus be thought of as a means by which each process implements a <u>virtual machine</u> with a dedicated <u>pseudoprocessor</u> for each process and essentially unlimited <u>virtual memory</u> directly accessible by the pseudoprocessor. It is on this virtual machine that the above intrinsic capabilities of intersegment linking, process augmentation, interprocess communication, process creation and process partitioning have been implemented.

H. Processor Management

The fundamental problem of processor management is that of providing each process with its own pseudoprocessor, each slower on the average than the real processor, given a small number of real processors. The Multics solution is to time share or multiplex all available processors among the eligible processes as follows.

1. Scheduling

Each process is provided with a supervisory module, called a scheduler, with which to schedule itself for future use of a processor. Also, each process shares a supervisory module called the Process Exchange which allows the process, when eligible, to receive a processor from a retiring process or, when a process's time quantum has expired, to pass the processor to the next eligible process. These functions are performed with the aid of a shared, system-wide data base containing the ready list, a list of processes each awaiting the services of a processor for some limited time quantum.

When a process has been using a processor for a duration equal its time quantum, it is interrupted by a hardware timer. This causes the process to trap to its scheduler. The scheduler makes an entry for the process at some appropriate location in the ready list according to the rules defined by the scheduling algorithm; Initial Multics will use a "round robin" rule at first. Then the Process Exchange module will select the process at the top of the ready list and yield the processor to that process.

In addition, a third state exists -- the blocked state. When a process is blocked it is unable to proceed in its computation

H. 1. (Continued)

until the occurrence of some <u>event</u>. A blocked process does not appear in the ready list.

2. Event distribution

Since it is not economical for a process to loop, waiting for the occurrence of an event for long periods of time, the waiting process blocks itself. Therefore, a means must be devised for a sending process to remove the intended receiver of an event from the blocked state and return it to the ready list. This is accomplished by allowing the Interprocess Communication functions "wait" and "notify" to interact with the Process Exchange directly; when a process wishes to wait, instead of testing the awaited event variable change by looping, it blocks itself by removing itself from the ready list of processes eligible to share a processor. When the event variable is changed by the sending process it then, by programming convention, must notify the receiving process by calling the Process Exchange and restoring the blocked process to the ready list, making it again eligible for processor time.

Thus, Interprocess Communication and the Process Exchange combine to provide a means for distributing event signals to processes which have relinquished a processor indefinitely so as to improve processor multiplexing efficiency.

It should be noted that the processing of hardware interrupts such as I/O interrupts -- which in general interrupt a process other than the interested process -- are treated as events and processed accordingly.

1. <u>Core Management</u>

Although more than one process can have portions of its segments in core at any one time, there may not be enough core for even one entire process. In general, the size of the address space for any one process may exceed the size of core memory available on any technically feasible hardware system. Since segment addressing requires that the portion of a segment being accessed reside in core memory before the processor can access it directly, it is clear that there exists a problem of managing the available core and distributing it to processes as required so that the required information can be placed in core.

Each Multics process does its own core space allocation. This is accomplished by a set of supervisory procedures comprising <u>Core Control</u> known to each process and a shared system-wide data base called the <u>core map</u>.

Core is allocated in fragments called <u>groups</u>. Groups are comprised of integral numbers of physically contiguous 64 word blocks of core and may be of various sizes. The core map describes the exact status of each group of core in the entire system and, greatly oversimplifying, each group of core is either available for use -- <u>free</u> -- or it contains information for some process -- <u>used</u>.

When a process needs core, Core Control acting on behalf of the process, locates the appropriate group of free core and reserves it for this process by marking it "used". That group of core may then be filled with information for the process.

When the number of free groups falls below a threshold value, free core must be <u>replenished</u>. The only way this can be done is to move

I. (Continued)

information in used groups to secondary storage and then marking these groups free again. In the case of information which has not been altered, such as pure procedures or read-only data, a valid copy of the information already exists on secondary storage so it need not be written out before the group is freed. In either case, the previous contents of the group are overwritten or zeroed when the group is assigned to contain new information so that the previous contents will not be unintentionally revealed to subsequent users of the group.

Selection of which groups to free is determined by a replacement algorithm which maintains a record of the recency of use of all used groups by utilizing a GE 645 hardware feature which indicates whether or not a group has been referenced. The strategy then used is, using this recency information, to free those used groups which have been referenced least recently.

J. Page and Segment Management

On the GE 645, all segments are subdivided into units of equal size called pages. The page is an allocation unit of logically contiguous information as opposed to the group, which is an allocation unit of core space used to contain a page (or hyperpage). On the GE 645 a page can have a length of either 64 words or 1024 words. In Multics, each segment is divided into hyperpages of equal size, where a hyperpage is a set of adjacent pages considered to be a single unit. For simplicity, however, it is adequate to discuss the management of pages by a Multics process, hyperpage management being a generalization of this.

J. (Continued)

1. Segment addressing

A quick sketch of segment addressing on the GE 645 is an appropriate introduction to page and segment management in a Multics process.

Segment referencing is done via a <u>segment address pair</u>; the first element of the pair is a <u>segment number</u>, the second an <u>offset</u>. The segment number is the machine interpretable address for a segment known to a particular process and is taken by the processor to be an index to a <u>segment descriptor word</u> in the descriptor segment of that process. Different processes may assign different segment numbers to the same shared segment. The segment descriptor word, besides containing access information for the corresponding segment also contains the address of the <u>page table</u> for the segment.

Multics segments may be <u>fragmented</u> in the sense that pages comprising a segment need not be in adjacent groups of core and not all pages need be in core. A segment <u>page table</u> is simply a table of machine addresses, the first address being that of the first page, the second that of the second page, etc. The <u>offset</u>, which is the second part of a segment address pair, is the location of the desired word relative to the beginning of the segment and is used by the processor in determining in which page to find the desired word. If a segment is shared by more than one process, the page table is also shared.

Thus the processor locates the referenced word by using the segment number to locate the segment's page table and the offset to select the proper page.

J. (Continued)

2. Page and Segment states

In Multics, a page is either in core or _missing_. If missing, a _missing page fault_ is placed in the corresponding page table word. If not missing, the corresponding page table word contains the address of its page.

A segment is either _loaded_ or _unloaded_. If loaded, there exists a page table for the segment and the segment descriptor words in each process to which the segment is known contain the address of the page table. If unloaded, there is no page table for the segment -- typically it has been destroyed to make room for more actively referenced information -- and a _missing segment fault_ is placed in each segment descriptor word referencing the segment.

3. Page and Segment Management by a Process

Each Multics process contains a set of supervisory procedures called _Page Control_ and _Segment Control_ by which it manages its own page and segment requirements. Since segments and their associated pages can be shared by several processes, a shared system-wide data base called the _Active Segment Table_ is maintained by Page Control and Segment Control to permit page and segment management.

The Active Segment Table contains an entry for each segment being accessed by one or more processes in the system. Each entry contains information in the form of a _file map_ describing the location of each page on secondary storage for the associated segment. Also, each entry contains a list of all the processes using the segment and the segment number by which the segment is known to each process.

J. 3. (Continued)

When a process references a page which is in core, the reference occurs immediately. If the page containing the desired word is not in core, however, the processor encounters a missing page fault and traps to Page Control in the address space of the running process. In an unused portion of the page table word containing the fault, Page Control finds an index which locates the proper segment entry in the Active Segment Table. After acquiring from Core Control a group of the proper size to contain the page, Page Control then uses the file map indicated by the System Segment Table index to locate and bring into core the desired page. The page table word is then updated to contain the address of the newly fetched page and processing continues from the point of the fault.

If a process references a segment for which no page table yet exists in core, a missing segment fault is taken, when the processor references the associated segment descriptor word, invoking Segment Control. The action taken is similar to that taken by Page Control except in this case the additional effort of constructing a page table must be undertaken.

Note that when Core Control _frees_ core groups, it first calls Page Control and Segment Control to _remove_ pages from those groups and, if the last page of a segment has been removed, to discard page tables, filling the appropriate page table words and segment descriptor words with the appropriate faults and Active Segment Table indices.

4. Additional Process States

In order for a process to run, certain per-process segments must be loaded. When these segments are loaded, the _process_ is said to be <u>loaded</u>. In Initial Multics, when a process is created, it will be loaded and may remain loaded until destroyed. This implies that there may exist a relatively small upper limit to the number of processes which can exist under Initial Multics at any given instant as determined by the amount of core memory available to the system. When a mechanism is provided to <u>unload</u> dormant processes, reloading them when their services are again requested, the core storage restriction on the number of processes which can exist in the system at a given time is removed.

The decision to include process unloading and reloading in Initial Multics or to delay it for a subsequent version has not yet been made as it depends upon performance issues not yet resolved.

K. <u>Secondary Storage Management</u>

Since all required segments can not fit into core, a <u>hierarchy</u> of on-line secondary storage devices is available to the system. Each Multics process cooperates in managing the space on these devices and in moving information among them. For Initial Multics, the secondary storage hierarchy will be limited at first to one drum and one disc.

1. Device Overflow and Segment Migration

As segments and processes are created during the normal course of operation of Multics and as more and more information is entered into the system, the need for space to store segments grows. As a secondary storage device fills up, it becomes necessary to remove segments from it and move them to another device.

K. 1. (Continued)

Eventually, Multics will have an open-ended storage hierarchy with removeable magnetic media permitting an unlimited amount of overflow to off-line storage. In Initial Multics, however, the amount of storage will have a fixed upper limit. Also, Multics will eventually have a storage balancing capability which will adjust the resident location of a segment according to the usage characteristics required of the segment and the retrieval characteristics of the available secondary storage devices in the hierarchy. In Initial Multics, this balancing capability is not fully implemented.

Initial Multics will, however, have the ability to cause a segment to migrate from one device to another as follows. When a process takes a missing segment fault while referencing a segment, and further discovers that no Active System Table entry exists for the segment, it checks to see if the device upon which the segment resides is overloaded or if the segment should be moved to a device whose access characteristics better match the access requirements of the segment. If so, it locates a device which is not overloaded and prepares to allocate space on the new device to accommodate this segment. This space is called a <u>move file</u>. Then an Active Segment Table entry for this segment is created to contain not only the file map but also a <u>move file map</u>. Processing then continues as usual, with the exception that whenever a page for the migrating segment is removed from core it goes to the new device as indicated by the move file map. When migration has completed, the old file map is discarded and the move file map becomes the new file map.

III. (Continued)

L. Summary

This section has described the following concepts key to determining the nature of a generic Multics process.

- segmentation
- intersegment linking
- process augmentation
- interprocess communication
- process creation
- process partitioning
- processor management
- core management
- page and segment management
- secondary storage management

All of these concepts are supervisory in nature and fall into two classess -- intrinsic, dealing with those features to be provided to the user independent of the machine used; technological, dealing with the simulation of a virtual machine upon which to implement the intrinsic concepts.

It is important to understand that these capabilities are included in each process running in the Multics environment. This sharing of supervisory responsibility and the segments implementing these functions by all processes has led to the use of the term distributed supervisor to contrast the Multics implementation of supervisory tasks with the usual method in which the supervisor is treated as separate from user programs.

L. (Continued)

Given the above capabilities of a single generic process, it is now possible to describe the way in which processes are specialized and interact to form a coherent system of mutually cooperating processes.

IV. THE MULTICS SYSTEM

The generic Multics process, described in terms of its significant concepts in Part III, can be considered to be the basic building block of the Multics system. Although the attributes of a process determine in large measure the capabilities provided to each Multics user, the actual assignment of one or more processes to a user requires the intervention of a set of specialized system processes. These special-purpose processes together with the user processes belonging to Multics users, jointly comprise the Multics System.

In some cases, special-purpose processes are nothing more than generic processes performing relatively restricted control functions which are nevertheless best implemented as separate processes; the control processes and overseer processes described below are cases in point. Another class of specialized processes deals directly with the exclusive management of system resources, and as such represents a partial retraction of the concept of the distributed supervisor; the Universal Device Manager Process Group members are examples of such processes.

It is useful to regard the Multics System as the first applications system to be implemented in the Multics environment, although its implementors are Multics system-programmers with more privileges than the usual system programmer will have. Thus, most subsequent system applications will be superimposed upon the Multics System structure. The design is sufficiently modular, however, that it can be significantly modified for special applications or even allow the concurrent residence of more than one Multics-like System on the same hardware, should the need arise, by combining the building blocks in different ways.

IV. (Continued)

A. Process Groups

1. Need for Process Groups

A Multics process cannot run on more than one processor at a time; therefore, it is not possible to implement asynchronous tasking logic using a single process.

The ability to implement asynchronous tasking logic is important for two reasons. First, it allows for the possibility of simultaneous processing of two or more sub-tasks comprising a larger task or job. Secondly, and more importantly in an environment with only a few real (as opposed to virtual) processors, it permits a more natural description of sets of non-sequential but interrelated tasks than is possible within the constraints of the sequential logic of a single process. For example, the implementation of the control logic embodied in the Multics System is best done as a set of asynchronous but mutually dependent tasks.

2. The Multics Process Group

Asynchronous tasking is provided in Multics by the process group. A process group is a set of processes which cooperate towards the accomplishment of some task. Each process group has a unique identification and each member process shares certain information and segments with all other processes in the group. There is a system-wide data base called the Process Group Directory Directory (see Section A.4 below) containing an entry for each process group in the system. Individual process groups each possess a Process Group Directory describing the group and certain segments shared by all processes in the group.

A. 2. (Continued)

The wait and notify functions of Interprocess Communication are the means by which processes coordinate their processing with other processes in their group, much analogous to the call and return by which procedures within a single process communicate.

In Multics, access to segments, resource allocation and, in general, I/O are on a per process group basis.

3. Stopping a Process Group

Any well-designed system must make provisions for the fickleness of its users -- in the case of humans, they make mistakes and they change their minds. Therefore, the ability to stop the running of a process group has been included in the Multics System.

A special process interrupt, analogous to a hardware interrupt, has been implemented in Multics. It is called the quit interrupt and is a means by which one process can stop another process by causing it to be interrupted in the midst of whatever instruction sequence it is executing.

It is not sufficient to stop just a single process, however, as it may be a member of a process group. For this reason, each process group which is stoppable is provided with a special-purpose process called an overseer process. The remaining processes in the group are called working processes. A system convention has been established whereby, whenever a process group is to be stopped, its overseer is signalled, whereupon the overseer sends quit interrupts to each of its working processes stopping them all in an orderly fashion.

IV. (Continued)

B. The Initialized System

The structure of the Multics System is best understood by observing its behavior as a function of time. Once the sequence of process creation has been outlined, the individual processes and process groups will be described in more detail.

1. Initial Process Creation Sequence

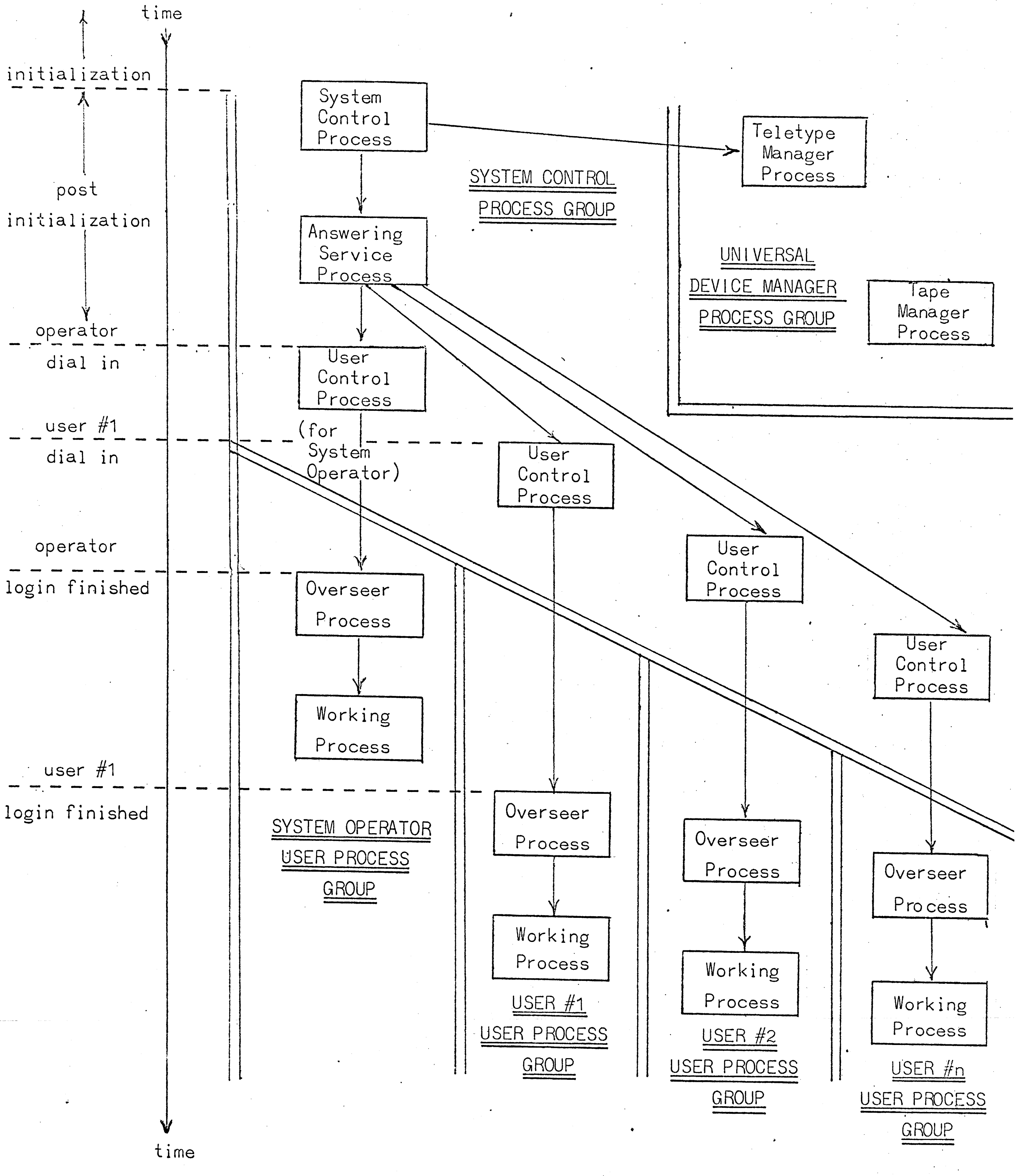
The problem is, given a "cold" machine, to bring it to a level of which it can accept input from external users.

This is accomplished by loading a small bootload program into the machine, which program in turn begins to input system tapes containing Multics modules. The first of these modules are a set of initialization procedures which construct an environment in which a Multics process, with all its "intrinsic" and "technological" functions, can run -- for example, a stack segment is provided so that interprocedure calls can occur. Once this environment is initialized, the initialization program creates a System Control Process Group initially containing one process, the System Control Process, and "retires" by calling wait, waiting for the system to shut down.

The System Control Process then creates (see Figure 1.) a Universal Device Manager Process Group containing, for Initial Multics, only a single process, the Typewriter Manager Process. Next, the System Control Process creates another process in the System Control Process Group -- the Answering Service -- and retires, waiting for system shutdown or system operator messages specifying hardware or software configurations.

Figure 1 INITIAL MULTICS SYSTEM DEFINITION --

PROCESS CREATION SEQUENCE



B. 1. (Continued)

Eventually a <u>Load Control Process</u> to optimize the potential number of users on the system at any one time will be created by the System Control Process as a part of its group; however, this will not be implemented in Initial Multics.

2. System Control Process Group

Initially, only two processes exist in the System Control Process Group: System Control and Answering Service.

a. System Control Process

The System Control Process serves two functions. First, it creates those processes necessary to permit the System Operator to login -- the Answering Service and the Teletype Manager Process. Secondly, it serves as a kind of "manager" for system operator requests regarding desired optional system configurations. In Initial Multics, however, these requests will be limited to a minimal set of capabilities, basically allowing the System Operator to enable and disable telephone lines and to create a Dump Process and Tape Manager process when a File System dump is required.

b. Answering Service Process

The Answering Service Process manages all eligible, but as yet unused communication lines. Immediately subsequent to initialization, the Answering Service Process will respond to a dial-up signal upon only <u>one</u> line -- the System Operator's line. After the System Operator authorizes general remote usage of the system, the Answering Service then "arms" a set of lines as specified by the System Operator, permitting incoming calls to be serviced.

B. 2. b. (Continued)

The Answering Service has responsibility for initiating the the chain of process creation which results from each dial-up to the system. As soon as the appropriate processes exist, the Answering Service relinquishes logical control of the communication line, regaining it again only when the user hangs up. Thus, the Answering Service can be regarded as a "switchboard", connecting lines to user processes as users log into the system.

3. Universal Device Manager Processes -- the I/O System

The Multics I/O System employs a special convention for managing peripheral devices other than on-line storage devices; this is to devote a specialized process -- called a Device Manager Process -- to the servicing of all members of a particular class of peripheral device. All other processes desiring the services of a peripheral must make requests of the appropriate Device Manager Process by means of Interproces Communication conventions.

A process invokes the services of the Multics I/O System just as it invokes any other supervisory functions -- by calls to ring 0 or ring 1 supervisory entry points provided for user access. The actual mechanics of processing I/O requests differ radically from those of most other supervisory requests, however. Most supervisory requests are satisfied by ring 0 and ring 1 segments "known" to the requesting process. In the case of I/O, this is not the case. After some of the I/O processing is done in the initiating process, it places the necessary information, e.g., a line of output text, in a segment shared with a Device Manager Process and then notifies

B. 3. (Continued)

the Device Manager Process that a task is ready for it. Since there is only one Device Manager Process for each class of peripheral device, many processes may use the same Device Manager Process. The initiating process may then <u>wait</u> or continue its working depending upon the amount of <u>synchronization</u> desired between the logical I/O and physical I/O.

When a hardware interrupt signalling the completion of an I/O action occurs, the interrupted process restores the appropriate Device Manager Process, if it is blocked, to the ready list; as soon as a processor becomes available to the Device Manager Process, it processes the interrupt and, if appropriate, <u>notifies</u> the process on behalf of which the I/O is being performed.

This splitting of I/O functions across two processes has two principal advantages. First, it provides for the possibility of asynchronous I/O, with the Device Manager Process accepting I/O requests from another process which continues on without waiting for the physical I/O to begin. Secondly, it permits an orderly processing of special interrupts, the console quit interrupt being an important example. (See D below)

In Initial Multics, because of the limited peripheral complement to be first included, the Universal Device Manager Process Group may contain only two processes -- the Teletype Manager Process for Model 37 Teletypes and the Tape Manager Process used in obtaining File System dumps.

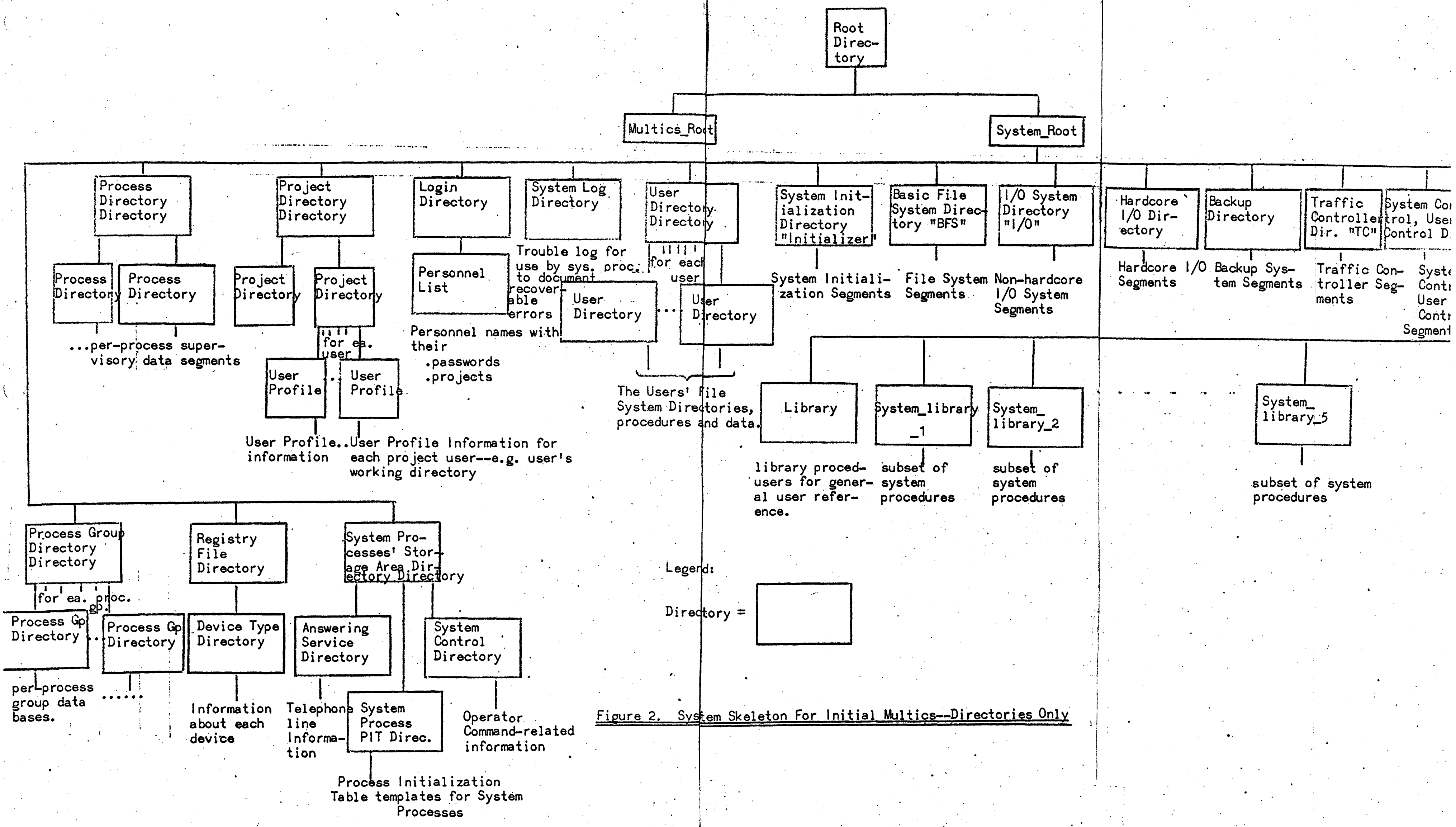

Root Directory
Multics_Root
System_Root
Process Directory Directory
Process Directory
Process Directory
...per-process supervisory data segments
Project Directory Directory
Project Directory
Project Directory
for ea. user
User Profile
User Profile
User Profile.. information
User Profile Information for each project user--e.g. user's working directory
Login Directory
Personnel List
Personnel names with their .passwords .projects
System Log Directory
Trouble log for use by sys. proc. to document recoverable errors
User Directory Directory
for each user
User Directory
User Directory
The Users' File System Directories, procedures and data.
System Initialization Directory "Initializer"
System Initialization Segments
Basic File System Directory "BFS"
File System Segments
I/O System Directory "I/O"
Non-hardcore I/O System Segments
Hardcore I/O Directory
Hardcore I/O Segments
Backup Directory
Backup System Segments
Traffic Controller Dir. "TC"
Traffic Controller Segments
System Control, User Control D
System Control User Control Segment
Library
library procedures for general user reference.
System_library_1
subset of system procedures
System_library_2
subset of system procedures
System_library_5
subset of system procedures
Process Group Directory Directory
for ea. proc. gp.
Process Gp Directory
Process Gp Directory
per-process group data bases.
Registry File Directory
Device Type Directory
Information about each device
System Processes' Storage Area Directory
Answering Service Directory
Telephone line Information
System Process PIT Direc.
Process Initialization Table templates for System Processes
System Control Directory
Operator Command-related information
Legend:
Directory =

Figure 2. System Skeleton For Initial Multics--Directories Only

B. (Continued)

4. The System Skeleton

Upon completion of initialization, a system-wide set of directories called the <u>System Skeleton</u> has been established. The System Skeleton for Initial Multics is shown in Figure 2.

Rectangles in Figure 2 represent directories. Note that Figure 2 gives an overview of the <u>entire</u> Directory Hierarchy within the Multics File System; <u>all</u> procedures and data segments -- both user-oriented and system-supplied -- fit within this structure. Procedure and data segments inferior to terminal directories are not documented here.

In Initial Multics, all segments will be inferior to the System_Root as shown in Figure 2. System-Library_1 through System-Library_5 serve as repositories for the numerous system segments loaded from tape as the system is initialized.

In subsequent versions of Multics, segments inferior to the System_Root will be those segments involved in the initialization of the System. Segments inferior to the Multics_Root for the most part will serve as special data bases for the Multics System processes described above. However, the User Directory Directory and the inferior User Directories -- one for each user -- will be an important exception, these directories being used as <u>root</u> directories for all user-supplied segments.

C. <u>The Dialup-Login Sequence</u>

Perhaps the single most important function of the Multics System is to provide users with appropriate process groups once they have dialed up and have been logged in. This sequence is described below:

C. (Continued)

1. Sequence of Process Creation after Dialup

When the Answering Service detects a ringing data set it "picks up the phone", creates, in the System Control Process Group, a special process called a <u>User Control Process</u> (see Figure 1) and "attaches" the line to the newly created process. The User Control Process logs the user in by checking the password he gives against his user identification as stated in the login command, and then creates a <u>User Process Group</u> containing an <u>Overseer Process</u>, passing the communication line from the System Control Process Group to the newly created User Process Group. The Overseer Process then creates a (single, in the case of Initial Multics) <u>Working Process</u> which is able to accept commands from the user console.

2. User Control Process

Each potential user of the system is given a User Control Process. The purpose of a User Control Process is to login one user. This is accomplished by requesting the user's password and comparing it against the password saved in a <u>Password File</u> for the user's stated identity. Until this validation occurs, the user is assumed unknown. Because a User Control Process runs for a potentially invalid user and because its access must be carefully restricted to special segments, including the Password File, each User Process is "owned" by the System Control Process Group.

The first "user" of the system, immediately after initialization, is the System Operator who will dial in on a special line and log in as a System Operator. An ordinary User Control Process is

C. 2. (Continued)

created for the System Operator, but once his identity is validated, all subsequent processes created on his behalf will have sufficiently privileged access rights that the procedure and data segments required to bring the system up can be dynamically attached to his processes as needed.

All subsequent users will be "normal" users with access privileges appropriate to their identities as verified by the User Control Process assigned to each at login time.

Thus, it is the responsibility of a User Control Process to validate the identity of any user who attempts to log in, and having done so to propagate his identity to all subsequent processes created on his behalf.

In versions of Multics subsequent to Initial Multics, the User Control Process will also have accounting responsibilities, insuring, for example, that an otherwise valid system user is not allowed system access if his account is out of funds.

3. User Process Groups

Once logged in, each user is assigned his own User Process Group. This is done in two steps: first, his User Control Process creates a User Process Group with an Overseer Process. Next, the Overseer Process creates a Working Process for the user. In Initial Multics, only one Working Process will be allowed to each user -- a restriction which may be lifted in later versions.

C. 3. (Continued)

a. Overseer Process

As explained in A-3 above, the principal function of the Overseer Process is to control the orderly stopping of a process group. In Initial Multics the stimuli to the Overseer Process are the console quit interrupt and the hangup -- received indirectly through the TTY Manager Process. Most of the life of an Overseer Process is spent in the blocked state, waiting for a signal requesting that the Working Process be stopped.

In later versions of Multics, it is planned that the scope of the Overseer Process be widened to accept other signals, e.g., automatic logout requests from the Load Control Process, and to permit the stopping of more than one Working Process.

b. Working Process

The Working Process is the most important process in the Multics System since it is the one which does all the direct work for the user.

When a Working Process is created by its Overseer, its Process Initialization Table contains the name of a special procedure called the listener, which thus becomes the first procedure executed in the process. This procedure "listens" to the teletype, waiting for a command to be issued. When a command is detected by the listener, it invokes the machinery necessary to properly interpret the command and its arguments, thereby initiating a "chain" of calls eventually leading to the accomplishment of the user's task.

C. 3. b. (Continued)

Thus, it is the Working Process which interprets all user commands (other than login) and performs or initiates the work implied by these commands.

D. <u>The Quit-Restart Sequence</u>

One of the most important control functions provided by the Multics System is that of permitting a console user to stop a set of processes running on his behalf and subsequently either restart them or start a new set of processes instead. This capability is essential to any well-implemented interactive computer system. .

In Initial Multics, this capability is implemented by means of the Typewriter Manager Process and the Overseer Process of a Working Process Group.

When a console user decides that his computations are not proceeding satisfactorily, he may push the quit button on his console. This button causes an immediate interrupt called a <u>console quit interrupt</u> at the processor even if the user's console is currently outputting information.

Whichever process happens to be executing at the time will trap to its interrupt interceptor, discover that the interrupt is intended for the Teletype Manager Process, <u>notify</u> the Teletype Manager Process of the event and then continue from the point of the interrupt.

When the Teletype Manager Process acquires a processor, it interprets the nature of this particular interrupt, discovers it to be a console quit interrupt and <u>notifies</u> the Overseer Process of the target process group by sending it a <u>stop</u> event.

D. (Continued)

When the notified Overseer Process, which is usually blocked waiting for a stop event, acquires a processor, it is able to quit all processes in the Working Process Group, regardless of their current activity. (In Initial Multics, this entails only one Working Process). When the Overseer receives a stop event, it issues a special process interrupt, the quit interrupt, to the Working Process. This interrupt causes the Working Process to block itself even if it happens to be executing on another processor at the time and to ignore all subsequent attempts to awaken it by processes other than its Overseer.

In this way, the console user is able to stop his Working Process regardless of its current activity.

However, in order for the user to be able to proceed after quitting his Working Process, he must have another process which can interpret his subsequent commands.

Therefore, immediately upon quitting the Working Process, the Overseer Process creates a new Working Process for the user and it becomes the effective Working Process.

The user now has two choices in Initial Multics. He can either restart his original Working Process or he can forget about it and use the newly created process as his new Working Process. To restart, the user merely types "start". This causes the new Working Process to signal the Overseer that a restart is desired. The Overseer then restarts the old Working Process at the point which the quit interrupt occurred and destroys the new Working Process as it has already served its use -- namely to interpret the command immediately following the console quit.

D. (Continued)

To destroy the old Working Process, the user merely types any command other than "start", whereupon the new Working Process assumes the role of the Working Process and the old Working Process is discarded.

E. MULTICS "Daemons"

The Multics System employs many processes to regulate the use of its facilities. Many of these processes are invisible to the user, but nevertheless are running concurrently with his processes. Processes which are assigned to no particular user, and do not perform direct work for the user, but instead "belong" to the system, are called "daemon" processes. Typically, they perform load balancing and resource management functions and are required in order to keep the System well balanced.

Initial Multics will have relatively few daemons; they include at least the following:

- An "idle" process. This process runs when no other process in the system needs a processor.
- The File System Device Monitor Process. This process is notified whenever an interrupt occurs for the drum or disc being used for the File System. It is the only process which can interpret such interrupts.
- The Teletype Manager Process
- The Tape Manager Process
- The Initial Multics File System Backup Daemon ("Brute Force" Dumper Process)
- The Answering Service Process.

E. (Continued)

In <u>subsequent</u> versions of Multics, additional daemons are planned for implementation.

- The Multilevel Monitor Process. This daemon will be responsible for maintaining a proper level of occupancy on all on-line storage devices comprising the File System device hierarchy and for matching the access speed requirements of each segment residing in the hierarchy to the responsiveness of available storage devices.
- The Storage Backup and Storage Reload Process <u>Groups</u>. These daemons will control the copying of segments onto removeable storage for subsequent reloading in the event of a storage hierarchy failure, in which case it will reload the system.
- The Load Control Process. This daemon will limit the number of concurrent users on the system in order to maintain an acceptable level of response and throughput.
- The Absentee Monitor Process. This daemon will run "background" jobs in the Multics environment.
- The Clock Manager Process. This daemon will be responsible for the calendar alarm clock.
- The Administrative Process Group. This daemon will perform periodic auditing and accounting fuctions.
- Device Managers for other peripherals.

F. <u>Summary</u>

This section has described the administrative logic in Multics as embodied in the Multics System. The building blocks used to build this governing body of logic are the process and the process group. The Multics System may be viewed as the first applications system to be implemented using the Multics process. The purpose of this

F. (Continued)

"applications system" being to provide each user with his own working process and a means by which to communicate with it and control it.

V. <u>USER INTERFACES FOR INITIAL MULTICS</u>

The user's view of a system is influenced primarily by the commands by which he may communicate with the system, the languages with which he can implement applications on the system and the services which his programs can request of the supervisor. These features are described below for Initial Multics.

A. <u>Commands</u>

All personnel communicating with the Multics System are considered to be "users", where the term applies to non-programmers conversing with pre-packaged applications, Multics system programmers and even the Media Operators and System Operator in charge of the system. The interface between all these users and the commands available in Multics is a software module, <u>known</u> to each interactive Working Process, called the <u>Shell</u>.

1. The Shell

The Initial Multics Shell serves several functions. The primary function served is as an interpretive interface between the user console and the command procedures in the Library Directory:

When a character string comprising a command and its arguments are typed, the Shell intercepts the character string and removes the leading substring which it interprets as a command procedure name. The Shell then calls the Segment Management Module, known to each process, requesting that the named command be located in the directory hierarchy using standard search rules and made known

1. (Continued)

to the process. Finally, once the command has been made known, the Shell calls the named command procedure, passing the remaining substrings in the input character string as arguments to the command procedure. The command then commences execution like any other procedure. Upon completion of the command action, the command returns to the Shell which may then request the next command.

In addition to this primary function, the Shell also recognizes a few punctuation marks forming a "meta-syntax" which can be used to combine more than one command in a single request from the console. For example, the user can specify a sequence of commands to be issued and the Shell will forward them one at a time until the last one has been completed. Also, the Shell syntax permits the notion of an <u>immediate value command</u> -- any valid command can assume this role by its being enclosed between the proper delimeters -- which permits the user to <u>nest</u> commands so that, for example, an immediate value command can be used as an argument for another command, the occurrence of the immediate value command being replaced by its value which is in turn used as an argument to the enclosing command.

2. General User Commands

Commands available to the general user of Initial Multics are abstracted below:

login	- permits the user to identify himself and verify his identification by means of a password. Not a generalized command in the sense that it must be the first command issued after dialing in.

2. (Continued)

list	- displays the contents of a directory file, listing file names, types, access modes, lengths, and dates created and modified.
rename	- changes the name of a file.
delete	- deletes a file.
listacl	- lists the access control information for a file.
setacl	- sets the access control information for a given file with respect to a given user.
delacl	- deletes the access control information for a given file with respect to a given user.
link	- creates a link -- a "pointer" -- to a file in the file hierarchy and places it in a specified directory.
edit	- permits context editing of an ascii file, including selective printouts and selective deletion, replacement, and insertion of text.
epl	- compiles a file using the EPL compiler.
eplbsa	- assembles a file using the EPLBSA assembler.
logout	- causes the systematic termination of the user's processes.

In addition to these commands, the user will be able to write his own commands so that his own procedures can be executed.

After a process has been stopped by the console quit interrupt, the following command convention will be followed by Initial Multics.

start	- causes the quit process to resume at the point of interruption.

2. (Continued)

any other command	- causes the quit process to be discarded and a new process substituted in its place.

3. Operator Commands

Multics provides for the recognition of Media Operators to manage tape reels and unit record devices and one System Operator who has the authority to control the entire system. In Initial Multics these two functions will be combined.

The Operator, in addition to the general user commands will be able to issue the following special commands:

op_here	- typed after the operator logs in or quits to permit him to type any of the following special operator commands.
get_line	- reports the status of communication lines attached to the Generalized Input/Output Controller.
set_line	- enables or disables the specified number of communication lines, allowing the operator to permit or augment remote access to the system or prepare to shut it down.
startup	- causes the creation of the specified System Process Group. In Initial Multics, this will be used to initiate the dumping activity.
media	- lists any messages for the Media Operator. In Initial Multics, will be used to receive messages from the dumper regarding tape mounting requirements.
shutdown	- shuts down the entire system.

B. Languages

The languages to be provided for Initial Multics are:

- EPL (compiler)
- EPLBSA (assembler)

All features described in the specification for these languages will be provided under Initial Multics, including the signalling and "non-local go-to" facilities of EPL in the ring-structured environment of Initial Multics.

The EPL language is of particular significance in Initial Multics since the system is implemented almost exclusively in EPL. Therefore, all supervisory entry points are described in EPL and obey the EPL conventions regarding data declarations and names.

Other candidates for inclusion into Initial Multics and subsequent versions of Multics include BCPL, SNOBOL 3, PAL, and FORTRAN IV.

C. Supervisor Entries and Utility Procedures

There are a large number of supervisory entry points in Initial Multics which may be called by the user from ring 32. In addition, there are some utility procedures, some of which are designed to intervene between the user and certain supervisory entry points so as to simplify the user/supervisor interface. Many of these entry points and utility procedures allow the user to write procedures which can perform the same functions which the Shell invokes as the result of user commands from the console, thereby permitting commands to be invoked not only by a user from his console but also indirectly by procedures he has written and attached to his process.

Initial Multics will include but need not be limited to the following Supervisor entries and utility procedures.

C. (Continued)

1. Segment Management Module Entries

smm$initiate	- makes an entry in the Segment Name Table associating a given *call name* (i.e., the name by which a procedure references a segment) with a segment located in the file hierarchy by a given *path name* (i.e., the list of all directories between the segment and the "root" of the hierarchy), and returns the segment number. The segment is made known if necessary.
smm$get_segment	- returns two segment numbers; one for the segment whose call name is supplied and the other for a *related* segment, such as the named segment's linkage segment. The segments are searched for and made known if necessary. An entry is made in the Segment Name Table. This call is tailored primarily for the Linker.
smm$get_set_ptr	- returns a segment number for a segment, given the segment call name, provided an entry for the call name already exists in the Segment Name Table.
smm$get_path_name	- returns the path name of a segment given its segment number.
smm$set_name_status	- a general-purpose call combining many of the above capabilities. Permits, for example, the location of, creation of, or copying of a segment.

1. (Continued)

smm$terminate	- "disassociates" a call name from a segment number, removing an entry from the Segment Name Table. Is the converse to smm$initiate.

2. Directory Supervisor Entry Points

list_dir	- itemizes the contents of a directory.
status	- itemizes the contents of a single specific entry in a directory.
chname	- changes name of an entry in a directory.
delentry	- deletes a specified entry in a directory, first deleting the segment to which the entry points.
readacl	- returns the Access Control List of a non-directory or the Common Access Control List of a directory.
writeacl	- replaces the entire (Common) Access Control List for a specified directory or non-directory.
set$copysw	- change the setting of the <u>copy</u> switch for a specified segment. If on, each user of this segment will get his own private copy.
appendb	- creates a new <u>branch</u>, i.e., set of segment attributes, in the file hierarchy by appending it to a specified directory. This is how a segment is <u>created</u>.
appendl	- creates a new <u>link</u> in a specified directory. A link is a "pointer" to a segment described elsewhere in the file hierarchy and can be used to reference the segment by pointing to its <u>branch</u>.

2. (Continued)

setml	- changes the maximum length of the segment specified.
movefile	- moves a segment from one section of the hierarchy to another.

3. Segment Control Entry Points

uim$truncate_seg	- reduces length of a segment by discarding information at the end of the segment.
check_ring	- checks whether a given set of segments is accessible from a specified ring. Important for validating arguments passed on inter_ring calls.

4. I/O System Entry Points

attach	- associates an <u>ioname</u> to a device (or another ioname) within the framework of a process group; defines the <u>type</u> and the <u>mode</u> associated with the attachment. An <u>ioname</u> is a symbolic name which identifies a device or a <u>frame</u> (a frame is a data item which may be read from or written into as if it were a device). The <u>type</u> of the attachment specifies the nature of the object associated with the ioname (tape, printer, file, etc.). The <u>mode</u> describes certain characteristics related to the attachment (e.g., readable, writeable, appendable, random or sequential access, etc.)
detach	- removes the association established by an attach call.

4. (Continued)

localattach	- identical to attach but the scope of attachment is specific to the process rather than to the group.
divert	- suspends any current I/O on the specified attached device and allows immediate initiation of new I/O on the specified new ioname.
revert	- reinstates the original attachment suspended by the previous divert call.
resetread	- deletes unused read-ahead data associated with the specified ioname.
resetwrite	- deletes unused write behind data associated with the specified ioname.
abort	- cancels any physically incomplete previous read and write calls with the specified ioname.
read	- reads into the specified workspace, the requested number of elements from the frame specified by the given ioname.
write	- writes from the specified workspace the requested number of elements onto the frame specified by the given ioname.

5. Interprocess Communication Entry Points

ecm$create_ev_chn ecm$delete_ev_chn	- creates and deletes an event channel
ecm$decl_ev_call_chn ecm$decl_ev_wait_chn	- declares an event channel to be of type "event call" or "event wait".

5. (Continued)

ecm$set_call_priority ecm$set_wait_priority	- assigns priority to event_call channels or to event_wait channels in the process.
ecm$give_access	- gives access of an event_channel to a list of process groups.
ecm$set_event	- records the occurrence of an event in the appropriate event_channel and wakes up the corresponding process.
wc$wait	- causes a process to wait for an event to occur.
wc$test_event	- tests if an event has occurred.

6. Utility Procedures

Only the more important utility procedures are listed here.

a. Basic File System-associated procedures

These procedures serve as simplified interfaces with many of the Directory Supervisor entry points described above.

change_name	- removes an old name from a directory and adds a new name.
delete_entry	- deletes an entry (and the associated segment if it is a branch) from a directory.
append_branch	- adds a branch to a directory.
append_link	- adds a link to a directory.
set_max_length	- sets the maximum length of the segment associated with a branch.
move_file	- moves information associated with one entry to another in a directory.
set_copy	- sets the copy switch for an entry so that each user of the segment gets his own copy of the segment.

a. (Continued)

set_retention_date	- sets the retention date for an entry.
truncate_seg	- truncates a segment.
check_access	- obtains the effective mode of a segment.

b. I/O-associated procedures

read_cs	- obtains a string of characters from a segment.
write_cs	- puts a string of characters into a segment.
read_in	- reads a string of characters from the user console.
write_out	- writes a string of characters on the user console.

c. Miscellaneous procedures

get_calendar	- obtains the current calendar clock reading in microseconds and converts it to a readable data-time form.

VI. PERFORMANCE

A. Introduction

The purpose of the following set of performance specifications is threefold:

1. It is meant to define consistent sets of capabilities and performance goals representing the achievement of useful systems at specified times.

2. It is meant to help focus attention on those Multics functions the performance of which is critical to the usefulness of the system.

3. It is meant to serve as criteria in balancing performance vs. function as the system is modified and augmented throughout its preliminary development stages.

The following specifications are not intended to represent a set of rigid performance and schedule requirements by which to judge the failure or success of the Multics System; clearly many acceptable trade-offs are possible. Instead they are intended to serve as a commonly agreed-upon reference about which to define further refinements and extensions as actual system performance becomes more sharply defined.

VI. (Continued)

B. Overall Requirements

Before stating the performance goals, it should be noted that the course of action for the Multics project is to continually integrate modules into the system so as to add new capabilities while continually tuning various system components so as to improve performance. Hence, the system will have both increasing capability and performance as time progresses.

The short range goal is to produce what is specified within this document as Initial Multics, with a small number of users utilizing the system as a Multics development tool and obtaining reasonable response time. The medium range goal is to produce what is often called Prototype Multics which is a system capable of replacing CTSS with some thirty users obtaining "reasonable" response time.

For the purpose of this document, "reasonable" response time is defined as being roughly equivalent to that obtainable on CTSS. The maximum number of users who can use the system is defined as that number which allows each user to obtain reasonable response irrespective of what the other users are doing. Of course it is assumed that the users are not cooperatively trying to overload the system; therefore, there will be a "normal" distribution of various types of work being done by the users.

No attempt will be made to define and measure the cost of using Multics beyond stating the minimum acceptable number of concurrent users at various stages of development given the hardware configuration outlined in Section II.

B. (Continued)

A graph of targeted performance and capability goals is given in Figure 3.

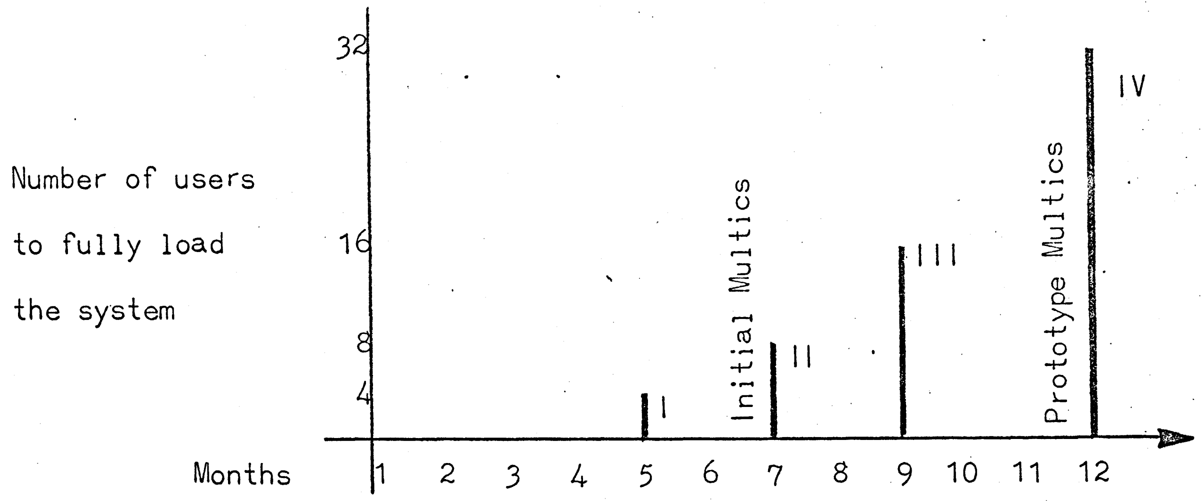


Figure 3.
Targeted Performance and Capability Goals for Multics in 1968

- The system numbered I contains Initial Multics without languages, quit-start capability or Backup Dumper.
- The system numbered II is Initial Multics.
- The system numbered III is Initial Multics with improved capability for the Backup-Multilevel system and various new commands.
- The system numbered IV is III with an accounting system and various new commands for general usage.

Although this specification deals primarily with Initial Multics (System II in Fig. 3), the performance of system I on May 1 is of interest as it can be used to get a reasonable judgment on the likelihood of Initial Multics attainment by July.

VI. (Continued)

C. Response Requirements

Listed here is a set of possible "transactions" to be performed with the value of maximum allowable mean response time. Each transaction is specified by its beginning and ending point. The times given are goals for the fully loaded systems defined by Fig. 3 and the single processor configuration of Section II.

	Transaction	Beginning Pt.	Ending Pt.	Time	
1	dial-up	Comp. of dialup	response from system that dialup is rec'd	.10 sec	✓
2	dial-up and login	Comp. of dial-up	response from system that user is logged in and may type a command	1 min	✓
3	command acknowledgment	depression of carriage return after command type-in	"wait" typeout signifying command has been received	3 sec	✓
4	trivial commands: commencement of result delivery (echo)	depression of carriage return after type-in	commencement of typeout by command	10 sec	✓
5	5 invocations of trivial (echo) command requiring minimal typing	commencement of type-in of command	"ready" print-out after completion of 5th echo	70 sec	✓
6	typeout of 30 lines by edit	start of typeout	finish of typeout	1.2 x typewriter speed	✓
7	type-in of 30 lines to edit	start of type-in	end of type-in	40 words/min	✓
8	trivial edit; e.g., modify a line	depression of carriage return after type-in	completion signal at console	10 sec	✓
9	epl compilation of "end statement" program	type-in of epl command	ready typeout at command completion	2 min	✓

C. (Continued)

	Transaction	Beginning Pt.	Ending Pt.	Time	
10	eplbsa assembly of "end statement"	type-in of command	ready typeout at command comp.	1 min	✓
11	eplbsa assembly of 2-3 source page program	type-in of command	ready typeout of command completion	5 min	✓
12	epl compilation of "typical" program of 2-3 source pages	type-in of command	ready typeout at command completion	30 min	✓
13	sequence of links, lists, edits	beginning of command sequence	end of command sequence	time to perform similar runs on CTSS	✓
14	quitting	depression quit button	response by quit responder	10 secs	✓
15	quitting and starting	depression of quit button	continuing of operation	30 secs	✓
16	interactive resp. within some procedure	depression of carriage return	beginning of typeout	10 secc	✓
17	tight computation loop which would take 10 secs if wired down with only one user on system.	start of computation by clock read	end of computation by clock read	1-2 mins	✓

Times should be considerably better for a lightly loaded system or a system with more resources; e.g., 2 processors.

It is suggested that system performance be measured by having the required number of users perform a standard "script" comprised of some or all of the above transactions, the response to each of which would be measured and averaged.