From: W. Olin Sibert

To: MTB Distribution

Date: December 1, 1980

Subject: Multics Initialization Redesigned: A Replacement for BOS

This MTB proposes a mechanism, called "Bootload Multics", that replaces the functions presently provided by BOS. The major benefit of this proposal is that the almost all the programs will be in pl1, and therefore have the inherent maintainability provided by the standard Multics programming environment. This contrasts with BOS, which is coded entirely in ALM, and is difficult and obscure even as ALM goes.

It will also drastically reduce the effort to support new hardware, ORION in particular. This is partially because BOS is a large body of code which would have needed modifications, and is eliminated, but mostly because the modifications would have been much more difficult due to the ALM environment in which it runs.

Finally, it will make it much easier to make a simple, consistent operator interface for the functions BOS provides today: it will provide mnemonic names, intelligible error messages, and more tolerant syntax.

This concept owes most of its basic structure to Charlie Hornig of CISL, without whom it would never have taken shape. I am also indebted to Bernard Greenberg and Benson Margulies, also of CISL, for much useful technical commentary and support.

Please send any comments, questions, etc., to the author:

By Multics Mail, at either MIT-Multics or System-M:

Sibert.Multics

By Honeywell Express Delivery Interoffice Mail:

W. Olin Sibert
Cambridge Information Systems Laboratory
HED MA22

(or) 575 Technology Square
Cambridge, Massachusetts 02139

Introduction:

Bootload Multics is a mechanism for implementation of low-level system test and support features, such as are currently provided by BOS. Ultimately, it is intended to replace BOS. It will initially provide only the debugging/test features from BOS, but other BOS functions which are not in the critical path, such as SAVE and RESTOR, will be implemented before the product is finished.

Although this MTB discusses various stages in the development of Bootload Multics, and the functions available at each stage, these stages are only visible internally. The only version ever shipped to customers, with the possible exception of System-M and MIT, will include all the functions described herein. This is a requirement for the ORION, of course, since Level 68 BOS will not run on the ORION at all. For Level 68 and DPS-8 systems, the BOS interface will remain in the shipped system, for a time at least, though it will never be used except in highly unusual circumstances.

The major gain from this proposal will be the ability to code debugging utilities in pl1, in a Multics pl1 environment, rather than in ALM in the BOS environment. This is very desirable from a maintenance standpoint, as well as making it much simpler to create or modify such utilities -- an important improvement easing system checkout. If for no other reason than that there are ten or more programmers competent in pl1 for every one who knows ALM, this is a good thing.

Bootload Multics provides a shorter path to making Multics run on the ORION hardware, and also provides several other desirable features: reliability improvements in auto-reboot; simplified operator interface for bootloading Multics; and Multics bootload from disk, improving reliability and performance. These are described in more detail in the Justification section.

The MTB contains the following sections:

Introduction (you're reading it)
List of forthcoming MTBs
Justification: Benefits of Bootload Multics
Functional Overview
Partition Organization
Scenarios for Using Bootload Multics
Functional Correspondence between BOS and Bootload Multics
How Bootload Multics Works
Communications Between Bootload Multics, Real Multics, and BOS
Basic Implementation Plan

JUSTIFICATIONS; THE BENEFITS OF BOOTLOAD MULTICS:

This project answers a variety of needs, both externally from customers and internally from the marketing and development organizations. The major benefits of the proposal are listed here, and described briefly. Details of the individual points can be found in the appropriate sections later in the MTB.

1) First, and most important: this is BOS for ORION. Rather than converting both present-day BOS and Multics to run on ORION, this will completely eliminate the work of converting BOS. Much of the work which would be required to convert BOS for ORION must also be performed to convert Multics. Rather than doing it twice, in different environments, conversion will only be done once, and in a hospitable programming environment.

2) An important reliability improvement will be provided. The ability to boot from disk is something which many sites have requested. No longer will auto-reboot be subject to the vagaries of operators and tape drives.

3) Bootload time will be somewhat reduced. Additionally, the new environment will make it easier to implement "hot" bootload, which could provide an even larger reduction in bootload time.

4) The necessity for operator intervention at bootload time will be reduced. A normal bootload will be initiated will a single command, or by mounting a tape and pressing Initialize and Bootload three times. No other intervention will be necessary, unless the default configuration information is incorrect or inappropriate. In that case, the operator will be faced with an initial bootload dialogue, much the same as today. Even if "automatic" bootload is in use, therefore, one is not limited to the default configuration description.

5) The operator interface for all functions provided by Bootload Multics (that is, the BOS functions and anything new) will be much improved. It will provide readily understandable error diagnostics and tolerant command syntax, rather than the difficult operator interface which BOS presents today.

6) The BOS environment is difficult to program in. Because BOS is written entirely in ALM, it is difficult to understand and modify. The conventions used in BOS programming are unlike those used in Multics ALM programming, and are also not even standardized within BOS. By converting all the BOS functions to the pl1 environment, the system will become easier to maintain and upgrade.

7) It will be practical to write debugging utilities in the Bootload Multics environment which would have been very difficult in BOS. This is very important for the ORION -- in order to meet our schedule, we must have the most powerful debugging tools possible.

8) It will be much easier to implement sophisticated automatic crash recovery strategies. For instance, if we had hardware capable of providing a several second ride-through, we could easily implement power-fail auto restart. It will also be easier to implement the complex crash recovery strategies discussed in VanVleck's (unpublished) recovery MTB.

9) When SAVE and RESTOR are implemented in the native Bootload Multics environment, they will be implemented with the ability to do multiple SAVEs at once, thereby reducing downtime considerably. They will also be able to automatically take advantage of multiple IOM and MPC paths, unlike BOS, which knows of only one path per subsystem. Since SAVE/RESTOR is completely I/O bound, it should be possible to run at least three simultaneous SAVEs, assuming sufficient channels are available.

10) It will eliminate the necessity of supporting new peripherals in BOS. This reduces the number of programs that must be modified in order to support any particular new peripheral, and it also eliminates the need to support them in a difficult programming environment.

11) It combines all peripheral support into one area; the most important benefit of this is that I/O errors will be handled in only one way, produce only one type of error message, and need only be documented once. This will dramatically improve the quality of I/O error messages compared to BOS, since the status information are interpreted, rather than just being printed in octal.

This proposal is not without its costs, of course. The entire operator interface, though improved, will be different, requiring that operators be retrained. It requires approximately 2000 additional records of disk dedicated to the bootloading function (though they need not be on the RPV). In order to utilize the no-frills bootload mechanism of (4) above, it will be necessary for each site to make a site-specific boot tape. Of course, the general release boot tape will work everywhere. There is some work involved in installing this mechanism at an existing site, primarily disk rebuilds to allocate new partitions. The conversion procedures will be detailed in a future MTB.

While implementation of Bootload Multics will be more expensive than implementation of any of the specific features mentioned above, as a whole it is cheaper. Many of the benefits would be difficult to derive from present-day BOS, because its programming environment is inadequate. In particular, it will shorten the critical path to ORION support, by eliminating BOS and making the Multics development somewhat longer -- but not by as much as was saved by eliminating BOS. Bootload Multics will bring Multics initialization and recovery out of the dark ages, and make practical the implementation of innovative recovery strategies.

FUNCTIONAL OVERVIEW OF BOOTLOAD MULTICS:

The key to this proposal is the creation of a "Bootload Multics" system, which will serve (for debugging and crash recovery) the same purposes that BOS does today. While it is being developed, it will coexist with BOS, though it will be a complete replacement for BOS when it is finished. Until then, BOS can continue to be used (on non-ORION systems) for functions which are not yet implemented in Bootload Multics.

A Multics system using Bootload Multics will operate in two different modes: Bootload Multics, and Serice Multics. Service Multics is the same as Multics is today: it runs user processes, in the user ring. Bootload Multics is used for system initialization, crash recovery, debugging, testing, and all the other things BOS does today. It runs a single process, entirely in ring zero, without a file system. This environment is very robust; it is "guaranteed" to work no matter wht state the system is in (as long as enough hardware works), and will function in the face of difficulties that would (and perhaps DID) stop Service Multics in its tracks.

The Bootload Multics will provide all the features necessary for normal system operation: bootloading, shutting down, and recovery from crashes where ESD works. The Bootload Multics will be used for the following purposes:

1) Bootloading the Service Multics system. This function is equivalent to the BOS BOOT command. It will always be performed from the (new) BOOT partition.

2) Creation and loading of the BOOT partition from tape. There is no counterpart to this in BOS.

3) Setting the clock and loading (some) firmware, if the system is being brought up after a power shutdown.

4) Creation and editing the config deck. The standard config deck will be kept in the (new) DIR partition (equivalent to the BOS directory); it can be edited and saved, or edited and used temporarily. This provides the same features in today's BOS config command. Additionally, this will include the code which, for ORION systems, processes the configuration information in Reserved Memory and generates a config deck from it.

5) Crash interception; the same function which "return to BOS" provides today. It can also perform FDUMPs, ESD, and auto reboot.

6) Analysis and debugging facilities, such as are provided today by BOS DUMP, PATCH, and ABS commands. It will also be used to provide any new debugging facilities needed, at a much lower programming cost than today. As desired, it would also be straightforward to implement pl1 versions of other, more esoteric BOS analysis tools such as MPCD and TSTCHN.

7) Bootload Multics will provide a mechanism for executing the equivalent of SAVE and RESTOR. This will be done initially by transferring to BOS and executing BOS commands; ultimately, we will replimplement these functions in pl1.

8) There will be a mechanism similar in intent to BOS RUNCOM, which will be used for setting up various automatic mechanisms and macro commands. The syntax will be more like exec_com, and flow of control will use the call/return model, rather than the goto model used by BOS.

The reason this mechanism is called the "Bootload Multics" is that it IS, actually, a whole Multics supervisor. The system it runs will be all of collection 1, and most of collection 2. The entire pl1 runtime environment will exist. Standard supervisor prelinking will have been done. Page control will be active, but segments will not exist, and the entire file system will be unknown to it. It nonetheless provides a powerful and familiar programming environment.

The Bootload Multics requires that only slightly more hardware than is necessary to run BOS today. All that is required is a CPU with an operating appending unit, 256K of low order contiguous memory, the disk drive(s) with the partitions, and the associated I/O hardware. The only significant difference between the hardware requirements for Bootload Multics and BOS is that 256K of memory rather than 64K is required, and that the CPU must be able to access through PTWs; BOS uses only SDWs, and all its segments are unpaged. This does NOT mean the ability to take page faults and restart in the middle of instructions; all PTWs used at this stage must describe actual memory frames. Although the 256K of memory required for Bootload Multics is much larger than that required for BOS today, there is no longer any Multics system anywhere which would have difficulty fulfilling that requirement.

The "basic" Bootload Multics environment does not utilize page faults, connects (received, CPU-CPU connects; I/O, of course, is done by sending connects to the IOM), or timer runouts; this insures that it will function, if it possible to function at all, in the environment used for crash recovery and bootloading. Interrupts are only used in a very primitive fashion. For the most part, the system runs masked, and only unmasks when actually waiting for an interrupt. This is the environment used for low-level functions, such as fdump, dump, boot, and the config editor; the functions necessary for bootload, crashing or shutdown. The toehold which performs the memory swapping operates at an even lower level; it loops awaiting status, and does not use interrupts at all.

For more complicated applications, the Bootload Multics environment will have the ability to take page faults, and operate in a more Multics-like I/O environment. These features might be usable for a reimplementation of SAVE/RESTOR or other non-critical BOS utilities, since there is no need for them function in the face of hardware difficulties.

The environment provided by Bootload Multics will be essentially the same as that existing just before accept_fs_disk runs to accept the RPV. All paging (done only in places where the environment supports it, of course) will be done from a special disk partition belonging to Bootload Multics. There will be some differences; various tables (the SST, tc_data) will have sizes appropriate to the single-process, non-filesystem environment. The syserr logging mechanism will not exist as such; there will be no running syserr daemon hproc, and all syserr messages will be printed on the console. This will have to change to support ELAN properly, since all messages from both systems will have to be logged in a place accessable to ELAN.

A future enhancement would be to have Bootload Multics perform syserr logging, using the same LOG partition that Service Multics uses; this would make it possible to keep track of disk errors for analysis purposes, which is impossible today with BOS. In general, however, Bootload Multics should operate with as little communication between it and Service Multics as possible, so that interdependencies are kept to a minimum.

For accessing the Service Multics address space, Bootload Multics has an appending mechanism, similar in intent to BOS APND, but with different implementation. It operates by causing simulated page faults, using PTWs from AST entries in the Bootload Multics SST. Segments in the Service Multics address space are accessed by grabbing an ASTE of the appropriate size and calling a subroutine to copy the ASTE from the Service Multics SST into the Bootload Multics SST, which will perform appropriate translations for all the PTWs.

A new type of PTW address type will be defined, specifying that "This page was in core, but is now in the saved memory image; it was swapped out when we switched from Service to Bootload Multics, so you must do I/O to get it". When an ASTE is copied, PTWs for all pages that were in core in the low 256K will be translated to specify this address type; all other PTWs will be copied as is, since the rest of memory is still intact, and the disk PTWs describe the correct disk pages. The CMEs for in-core pages will also be copied and merged appropriately.

Since this mechanism will use page_fault (by a side door that omits the actual fault processing, but does everything else) it can even be used to take page faults on segments in the Service Multics address space, since all the databases are set up so that will work. While this will not be used by the low-level functions, and probably not by the initial implementation at all, it may be used as the basis of sophisticated crash recovery or analysis tools for the Service Multics.

The operator interface to Bootload Multics will be as much like Multics command level as possible. This primarily means commands with Multics-like arguments and control arguments and the standard quoting conventions, and may also include active functions and iteration sets if that proves desirable. The operator interface will go through the operator console, or, for ORION, the SSF console emulator. Bootload Multics will not use the SSF Session Control interface on ORION systems unless it proves necessary for communication about configuration information; this will not be known until the design of reconfiguration for ORION is complete. It should also be possible to use the LCC remote access facility to access Bootload Multics remotely. I/O will be completely synchronous. The REQUEST button will interrupt all normally functioning operations; if it fails, an Execute fault can be used to cause Bootload Multics to reinitialize itself and reboot.

Bootload Multics commands will take arguments and control arguments like standard Multics commands. Some commands will have request loops, which will accept input (unprompted, except for the LCC) much the same way BOS does today for things like PATCH requests and volume extents in SAVE, but with superior error diagnostics. An important feature of the Bootload Multics programming environment is that it will be much simpler to provide informative error diagnostics -- no more of this "?" nonsense that BOS gives you today. Code to generate informative error messages is easier to write, and the messages can be documented with the standard Error Message Documentation system. Additionally, all error messges will be documented, in the source of the programs from which they emanate, using the standard Supervisor Error Message Documentation system.

The Bootload Multics will communicate with the rest of the system, and keep its "permanent state", in a toehold of its very own, located at some known location in absolute memory near where the BOS toehold is today. The Bootload Multics toehold and the BOS toehold will coexist; all three systems will know about both of them. For details on the communication between systems, see the later section on this subject.

Bootload Multics is designed so that it "cannot fail". If it encounters a serious problem, it will perform whatever recovery it can. An unexpected fault will be caught by the condition handler around the command listener, which will cause the current command to be aborted. Individual procedures may establish condition handlers of their own also, to catch miscellaneous "expectable" errors.

An execute fault, a serious I/O related error, or something like a trouble fault, will cause it to completely reinitialize itself and reboot by executing the relevant code in the Multics toehold. The same will be done in response to an Execute Switches; this is similar to what BOS does today. The major difference is that the Bootload Multics will completely reinitialize itself under such circumstances, which will take a little longer than reinitializing BOS. This is being done because it is the simplest approach, and it will work at least as reliably than other approaches.

List of Forthcoming MTBs:

The following is a tentative list of some MTBs which I expect to be written for this project. Most of them are simply internal documentation of the design, though some are intended as user/operator interface documentation. This list is very tentative; some of the MTBs will probably never be written, and others not listed here probably will. Some of them are simply more complete documentation for things described in this MTB, but most of them will concern topics which are entirely omitted from this document, because their design is only very nebulously conceived now, if at all. Some of these descriptions include a brief overview of technical details to be expounded on at length in the MTB; this detail is provided here for early review.

Management of the Config Deck:

A document describing the way in which the config deck is created, modified, and maintained so that it can be used by both systems. It will also describe the subroutine interfaces for performing these operations.

The Bootload Multics File System:

A description of the internal interfaces used to manipulate files in the DIR partition; also describes the user commands used to manipulate those files from Service Multics.

The Bootload Multics Programming Environment:

Documentation of how to write programs to run in Bootload Multics; this is largely concerned with writing commands, and the subroutines, conventions, temporary space management mechanisms, etc. used to do so. The command environment is very undefined as yet, but I expect it will include a reasonably large stack, cu_, condition handling, some sort of area for temporary storage, and a variety of subroutines for interfacing with specific Bootload Multics facilities. Commands which require large amounts of temporary storage will be best suited by reserving segments (init-segs) for themselves in the system tape header.

The Bootload Multics Operator Interface:

This will probably be several documents, which describe all the operator commands in Bootload Multics. They will include detailed descriptionr of syntax, function, and arguments for each, comparisons (where relevent) with comprable BOS facilities, and the reasoning behind various design decisions about the interface. The design of the operator interface is expected to take a long time, much exposure, and several design reviews; there will be many revisions of documentation as well.

The Bootload Multics Command Environment and Runcom Mechanism:

A discussion of the command environment (as opposed to the command programming environment) and how runcoms (or whatever we choose to call them) will be used. The command environment will be very similar to Multics command level, although many facilities will be missing. Topics will include error handling, input and output stream management, and argument conventions. The section on runcoms will describe flow of control constructs, call/return for runcoms, recursion, and similar exec_com issues.

The Dialogue for Cold Bootload:

A description of the dialogue at cold boot time; what it means, and how to use it properly. This will also describe the mechanism (bootload_info) used to generate a tape with present answers for the cold boot dialogue, so that operator intervention can be minized. This document, in combination with the operator interface and command environment documentation, will serve as the base for the Multics Operators Handbook sections describing Bootload Multics.

Installation of or Conversion to Bootload Multics:

A description of the procedures used to install a system based on Bootload Multics: both for new sites, and for existing sites whose systems must be converted. Discusses requirements for disk formatting and rebuild, making system tapes, setting up boot-from-disk, etc.

Automatic Operation with Bootload Multics:

A description of the facilities for automatic crash recovery and reboot, as well as the mechanisms for booting new supervisors without operator intervention. Primarily a description of how the runcoms for automatic operation are set up.

Multics Initialization PLM:

Basically, a complete rewrite of the Multics System Initialization PLM, which will be updated to describe all the facilities used by Bootload Multics, as well as all the other changes which have occurred since it was originally written. This is a low priority item, and will probably not appear until long after the other documents.

PARTITION ORGANIZATION:

Because of the additional functionality provided by this mechanism, several new disk partitions will be required. They may be placed on whatever disks are desired. The partitions are required for operation in the sense that they are "files outside the file system". Since the Bootload Multics cannot assume anything about the integrity of the Service Multics file system, and will have none of its own, it must use partitions for any permanent databases it has.

The names shown here for these partitions may be changed, if desired. They are used as defaults for normal system operation. It should only be necessary to change them during the course of intensive Multics hardcore development. The partitions will be found, as they are today, from cards in the config deck. When doing a cold boot (that is, from tape), the locations of the partitions necessary for bootload will be specified by the operator, much as they are today by the BOS WARM and COLD commands. The new partitions occupy approximately 2000 disk records (700 + 700 + 257 + 257 + 30).

The following partitions are used today by Multics:

BOS This partition is used to hold all the BOS programs, the BOS file directory, and the low core image of Multics, saved when BOS is entered. Its location is specified by the WARM or COLD command. It is generally 200. records, and is used only by BOS.

LOG This partition is used by the syserr logging mechanism as a buffer for messages. Its location is specified by a PART card. It is generally 256. records, and is used only by Multics.

DUMP This partition is used by the BOS FDUMP command to save an image of the Multics supervisor after a crash. It is accessed by Multics later, when this image is copied into the hierarchy. It is generally 2000. or more records, and is used jointly by BOS and Multics.

HC This partition (of which there may be as many as there are RLV disk drives) is used by Multics to hold the Multics hardcore supervisor. It is from here, rather than from the storage system disk area, that the pages of pageable hardcore programs and databases are drawn. There may be several, on different drives, but each must be large enough to hold the entire supervisor. Their locations are specified by the ROOT card. An HC partition is generally 1024. records, and is used only by Multics.

The Bootload Multics facility will require additional partitions. The names of these partitions may be changed as desired, in order to provide additional flexibility; the names listed are the defaults. This is a complete list of partitions required to operate BOS, Bootload Multics, and Service Multics. The sizes listed here for the new partitions are only approximate, as they are dependent on the size of the Bootload Multics supervisor, which has not yet been determined. All partitions, except the CONF partition, are located by means of PART cards in the config deck. The PART cards are may also be used to specify that a different name for the actual disk partition used for any particular function.

CONF The CONF partition (which, alone among the Bootload Multics partitions, has a specific name by which it must be known) is used to contain the config deck for all instances of the system. This information is carried from bootload to bootload, and is considered "permanent"; it can be changed, or replaced with a completely new version, but it does not change without explicit action. During Service Multics operation, the segment config_deck is dynamically paged from this partition. During initialization of either form of the system, its contents are read and placed in a known region in low memory. As today, the config deck is updated by the dynamic reconfiguration software. The CONF partition is 4. records long.

LOG This partition is used by the syserr logging mechanism as a buffer for messages. Its location is specified by a PART card. It is generally 256. records, and is used only by Service Multics. Bootload Multics does not use syserr logging in this fashion.

DUMP This partition is used by the Bootload Multics fdump command to hold a saved image of the Service Multics supervisor after a crash. It is accessed later by Service Multics, when this image is copied into the hierarchy. It is generally 2000. or more records, and is used jointly by Bootload Multics and Service Multics. Multiple DUMP partitions may be used for multiple dump images, by giving them different names. A DUMP partition is not strictly necessary; if one is not present, however, crash dumps cannot be handled automatically. If the DUMP partition is absent, or unusable for some reason, a dump image can be written to tape, so this is not an unworkable situation.

HC This partition (of which there may be as many as there are RLV disk drives) is used by Service Multics, for the same purposes as today. The size(s) and location(s) of the HC partitions are the same as they are today. An HC partition is used only by Service Multics.

MCI This new partition will contain the lower 256K of the saved Service Multics Core Image. This core image will be saved whenever Bootload Multics is entered, as it will then run itself in the lower 256K. The saved image is swapped back in when (if) Service Multics is restarted.

The MCI partition must be at least 257. records. If it is larger (large enough to save all of main memory), it can also be used by the Bootload Multics to save main memory, as the equivalent of the BOS CORE SAVE command. As much MCI partition as is available will be used; therefore, if it is large enough, all of main memory will be saved and restored, so that the system may be powered off and its state later completely restored. If the MCI partition is not large enough for this, a core save to tape may be used, but this sacrifices the automatic operation provided by the partition.

BOOT This new partition is used to hold the Multics System Tape image from which Multics (both Service AND Bootload) is bootloaded. It may be created either online, by a privileged program which writes directly into it, or by reading a Multics System Tape with Bootload Multics. It is probably about 700. records. It is read by both Bootload Multics and Service Multics, and can also be written (to create a new bootload image) by both systems.

BHC This new partition is used to hold the hardcore supervisor and databases for the Bootload Multics. It is roughly equivalent to the HC partition used by the Service Multics, except that exactly one such partition is required for Bootload Multics, rather than the several that regular Multics can use. It is probably about 1000. records. It is used only by Bootload Multics.

BCI This new partition is used to hold the Bootload Multics core image when Service Multics is entered. Its function is very similar to that of the MCI partition, except that it need only ever be 257. records, since Bootload Multics will never use more than that.

DIR This new partition takes the place of the BOS directory as a repository for the config deck sources and various ASCII files, most of which are will be the equivalent of runcoms. It is probably about 30. records. It is used primarily by Bootload Multics, although there is an interface for getting files from it or putting files into it from Service Multics.

SCENARIOS FOR BOOTLOAD:

This section lists a number of scenarios for use of Bootload Multics. Because the operational details of Bootload Multics are not fully defined yet, the scenarios can not be described in detail. They are intended to give an idea of what is possible.

The basic power-on bootload scenario involves the automatic bootload described earlier. A Multics tape is mounted on drive 1, and the Initialize and Bootload buttons are pressed three times in succession. Bootload Multics reads the first part of itself in, and, assuming the necessary information is in bootload_info, reads in the rest of the system from the disk partition. It loads disk, tape, and URC firmware for all controllers described in the config deck. At this point, it pauses, at Bootload Multics command level. The bootload dialogue, and the bootload_info mechanism used to supply answers for it, will be described in a subsequent MTB.

Somewhere in this path, the time may have to be set. When and how this will be done is not yet specified.

Here, the BOOT command may be issued to start Service Multics, or other operations may be performed, such as editing of the config deck.

When Service Multics crashes or shuts down, Bootload Multics command level receives control again. In the case of a crash, it can try to do an ESD or other crash recovery actions.

If the configuration has changed in such a way that the bootload_info for Bootload Multics is not correct, a special pattern may be placed in the processor switches to force it to go through the bootload dialogue. Thus, would never be necessary to have another tape which has a bootload_info that does not specify the dialogue information, since the dialogue can always be forced.

If the system has never been booted before, or if the disks have all been wiped out, the dialogue should be forced, and the appropriate response given to indicate a cold bootload. This will initialize the label of the bootload disk drive, optionally formatting it first, and initialize the various partitions. The dialogue for cold bootload is considerably more complex than for warm bootload, and is not yet completely defined. The formatting of disk packs is an area which is presently in flux; we are supposed to be using MTAR for it, but this seems inappropriate in that MTAR can only be used under GCOS. In the meantime, Bootload Multics will avoid addressing the problem.

If the configuration seems incorrect, Bootload Multics will attempt to describe why it holds this opinion, and after doing so will either crash or enter Bootload Multics command level. If it crashes, it must be rebooted with the switches set to force the dialogue. If it pases at Bootload Multics command level, the config editor can be used to change the config deck.

A distinction is made between "early" command level and "real" command level. At early command level, only those functions are available that are necessary to get the system running are available. At real command level, all Bootload Multics commands are available. The exact set of commands, as well as the precise point in initialization at which each of the two command levels occurs, has not yet been defined.

SCENARIO: Booting Multics from tape for the first time, on completely new hardware and virgin disk packs, or on working hardware and dead disk packs.

1) The switches are set to the "force dialogue" pattern.

2) The tape is mounted on drive 1, and the Initialize/Bootload buttons are pressed three times.

3) Responses are given to the dialogue to indicate the drive to be used as bootload disk drive, and whether or not it must be formatted. The time is also set as part of this dialogue.

4) The disk pack is initialized, perhaps having been formatted first. The DIR partition is initialized.

5) The system waits at Bootload Multics early command level, and a config deck is typed in. It is saved, and the system is instructed to reinitialize with that config deck.

6) At this point, the system has come up to Bootload Multics real command level, and operations may proceed as in the next scenario. If necessary, disks can be RESTORed; note that this can even be done to the drive in use as the Bootload Multics disk, as long as the three partitions (DIR, BHC, and BOOT) which Bootload Multics expects to remain intact are not reloaded.

SCENARIO: Booting Multics from tape with intact disks after powering up the system.

1) The tape is mounted on drive 1, and the Initialize/Bootload buttons are pressed three times.

2) The system boots according to the parameters for automatic boot. The time may have to be set at this stage. It finishes booting, and waits at Bootload Multics real command level.

3) At this point, the config can be altered, various maintenance operations can be performed, or the Service Multics system can be booted. If automatic bootload is being used, this is the only interaction which will occur before Answering Service initialization in Service Multics.

SCENARIO: Booting Multics from disk, after a crash or shutdown.

1) After a crash or shutdown or Service Multics, the system returns to Bootload Multics real command level and waits for a command. Any Bootload Multics command may be issued; for instance, a SAVE could be done before booting Service Multics again. Crash recovery could also be performed at this time - any kind from ESD to disk label patching.

SCENARIO: Booting Bootload Multics for purposes of editing the config deck, and then booting Service Multics.

1) This is essentially equivalent to a simple boot from tape or disk, except that after the config is altered, Bootload Multics must be instructed to reinitialize itself. It does so by rebooting, and pausing once again at Bootload Multics real command level.

SCENARIO: Entering Bootload Multics for debugging purposes, and restarting Service Multics.

1) The Initializer "rtb" command is issued to return to Bootload Multics. Bootload Multics waits at real command level, and any commands may be issued. Once this is finished, the GO command is issued, and it returns to Service Multice.

SCENARIO: Making a new Multics Supervisor, shutting down, and rebooting, without ever using tape.

1) Assume Service Multics is up and running. A Multics System Tape file is generated using generate_mst -file, and the write_mst_partition command is used to put it in a partition.

2) The shutdown.ec file in the DIR partition is edited, using the Service interface to the Bootload Multics file system, to indicate that the system should immediately reboot from the partition which was just written, and that it should take a dump and reboot the old system from the standard boot partition in the event of a crash in the new system.

3) Service Multics is shut down. Upon return to Bootload Multics, Bootload Multics executes the shutdown.ec file, and reboots from the specified partition.

4) If the new system runs, all is well. The user logs in again, and uses it. Otherwise it crashes, Bootload Multics reboots the OLD system, and our developer logs in and debugs the dump. All this is done without tape, and without human presence at the machine. If the new system loops, there is no planned recourse. A pity.

FUNCTIONAL CORRESPONDENCE BETWEEN BOS AND BOOTLOAD MULTICS:

The following table summarizes all the present BOS commands, and gives their equivalents in the Bootload Multics system.

Any Bootload Multics function which is not in the critical path for development is identified by the comment "(NC)"; these are any functions which are basically useless or can continue to be provided easily by present-day BOS while development is progressing. All the functions must be available by the time a system is first shipped, of course.

BOS Command	Function in BOS	Bootload Multics Equivalent
ABS	Displays main memory by absolute address, and analyzes cache contents. Outputs to either console or printer.	This function will be subsumed into the general interactive dump/patch facility. (NC)
BLAST	Sends a message to all terminals dialed into an FNP. Can be used for announcing crashes and the like. Today, it hardly ever works.	Will be re-coded in pl1, and will use the Multics FNP software, rather than an entirely separate mechanism as BOS does today. This should improve reliability considerably; the code will also be much simpler. Because the MCM databases belonging to Service Multics are available (using the segment accessing mechanism described earlier), it will be possible to make BLAST work for multiplexed channels, as well as for ordinary FNP channels. (NC)
BOOT	Bootloads Multics from tape. Reads part of the Multics System Tape, and transfers control to the program it has read in, which continues the process of bootload.	There will be commands to bootload Multics (Service or Bootload) from disk partitions or tape. Usually, though, this will be done without explicit intervention. The operator interface will be completely different.
BOSTAP	Writes a new BOS System Tape, from the contents of the BOS directory.	Obsoleted; no equivalent. To save the DIR partition, which is essentially the equivalent of what BOSTAP does for BOS, SAVE can be used.

CONFIG	Provides a simple interface for manipulating the current config deck. Uses files in the BOS directory. The editor is primitive and baroque.	A direct equivalent will exist, with an improved interface.
CONTIN	Restores the Multics machine and memory image, and restarts Multics.	A direct equivalent will exist.
CORE	Saves and restores the Multics core image from tape. This exists primarily because the BOS partition is not large enough to contain a saved image of all the core that BOS uses while executing some commands. It cannot save core to disk.	This is largely automatic in Bootload Multics. All core that will be used is saved automatically by the toehold. A command will be provided to save core to a specified disk partition or to tape. (NC)
DELETE	Deletes a specified file or program from the BOS directory.	A direct equivalent will exist for files in the DIR partition.
DIE	Destroys the BOS partition and main memory image.	Not necessary; no equivalent.
DMP355	Dumps FNP memory to tape or printer.	An equivalent could be codedif desirable. (NC)
DUMP	Writes a dump of Multics to a tape or online printer. The segments dumped can be selected in various ways.	A general purpose dump/patch facility will exist for performing both interactive dump/patch operations and dumps to tape or the printer. See below for details.
EDIT	Edits a BCD file from the BOS directory.	An edm-like editor (derived from user ring edm) will be provided.
ESD	Restarts Multics for the purpose of performing an emergency shutdown.	A direct equivalent will exist.
FD355	Creates an FDUMP of an FNP in the DUMP partition.	An direct equivalent can be coded easily using the Multics FNP software. This will be done as necessary. (NC)
FDUMP	Creates an FDUMP of Multics in the DUMP partition.	A direct equivalent will exist.

FLAG	Sets toehold flags in the BOS flagbox.	Some equivalent will be provided, though the interface will be completely different because of the different toehold management.
FMT	Formats disk packs for use by Multics. Today, it cannot deal with all the different types of disks Multics supports.	The format_disk_pack command can be used for all cases except cold boot, for which an adequate formatting program will exist. It will be necessary to modify format_disk_pack to support disk types not presently supported. (NC)
FWLOAD	Loads MPC firmware.	A direct equivalent will exist, though most firmware will be loaded automatically in the bootload process according to the config deck.
GO	Restarts Multics. Exactly the same as CONTIN.	See CONTIN, above.
IF	Performs testing and conditional execution of BOS commands for use in RUNCOMS.	Some equivalent will exist in the RUNCOM replacement.
LABEL	Prints the label from a tape.	Not used -- no equivalent.
LIST	Lists the BOS directory.	A direct equivalent will exist for files in the DIR partition.
LOADDM	Loads additional or replacement commands from tape into the BOS directory.	Not used -- no equivalent. Files can be placed in the DIR partition from Service Multics, or the entire partition may be RESTORed.
MPCD	Dumps MPC memory or the MPC trace table (to the printer, I assume). This is a little used feature, and is only interesting to Field Engineers, who have better ways of doing it.	No equivalent is planned. The function is available from Service Multics. MPCD doesn't work in BOS today, anyway.

PATCH	An interactive dump/patch utility for disk, bulk store, absolute memory, or Multics virtual address space. It uses the operators console, though it can also use the card punch.	A general purpose dump/patch facility will exist for performing both interactive dump/patch operations and dumps to tape or the printer. See below for details.
PRINT	Prints a tape produced by the ABS or DUMP command on the printer.	Not used -- no equivalent.
RENAME	Renames a file in the BOS directory.	An equivalent will exist for files in the DIR partition. (NC)
RESTOR	Restores the contents of secondary storage devices from SAVE tapes (or disks) produced by the SAVE command.	An equivalent will exist in the final implementation. (NC)
RUNCOM	Executes a file of BOS commands (a "RUNCOM").	A replacement for the RUNCOM package will be exist, to provide much the same exec_com-like functionality. The actual interface details have not yet been worked out; it will probably be considerably different from BOS RUNCOM in flow of control.
SAVE	Saves the contents of secondary storage devices on SAVE tapes (or disks) for use by the RESTOR command.	An equivalent will exist in the final implementation. (NC)
SSTN	Fills the SST Name Table, a database used by FDUMP and the dump/patch utilities.	A direct equivalent will exist.
TAPED	Produces an octal dump (on the printer) of the contents of a tape.	Not used -- no equivalent.
TEST	Tests disks and Bulk Store; can read or write and check patterns. It is primarily used today for zeroing portions of disk, such as the LOG partition.	Some equivalent will eventually be provided. (NC)
TIME	Sets and reads the calendar clock in the SCU.	A direct equivalent will exist. It is needed only for non-ORION systems.

TSTCHN	Allows testing and exploration of ICM channels from BOS command level. Today, this sort of thing is usually done with T&D programs.	Not used for testing channels; many sites use it to rewind tapes after crashes, however. A command to rewind tapes will be provided.
WRITE	Writes a message on the BOS console.	A direct equivalent will exist. (NC)

HOW BOOTLOAD MULTICS WORKS:

The config deck is a database that will be shared by Bootload Multics and Service Multics. It will describe the hardware configuration of the system. It will contain some information which is specific to each of Bootload and Service Multics, but most of the information will be common to both. It is initially read in from the CONF partition on the bootload disk drive at cold boot time, and is continuously updated as Bootload or Service Multics operates. The major difference between its function today and the new features is that it will be "permanent"; that is, it can be considered as being written once, when the system is installed, and merely modified thereafter. The config deck will specify the location (disk drive, and, optionally, alternate name) of all the partitions, along with all the same information it contains today.

At cold boot (boot from tape) time, since no config deck is available, the operator must go through a cold boot dialogue and specify much the same things as are specified now at BOS boot time: the tape controller type; the IOM, channel, disk type, and drive number of disk drive containing the config deck (CONF partition); and the firmware for the disk controller. This dialogue is intended to acquire the bare minimum of information for getting Bootload Multics started; it does not require additional firmware loading as BOS does today, or other extraneous dialogue, to start the booting process. Bootload Multics does require more partitions to really get started, but since these are all obtained from the config deck, there is no additional complication. The cold boot dialogue can be suppressed by proper use of the bootload_info mechanism to supply preset answers to it; this is described in a forthcoming MTB.

The dialogue is dependent on the information contained in the module bootload_info, which is part of the first program read in from tape. This is a cds program which, at compile time, asks the user the same questions which would be asked at bootload time; if satisfactory answers are provided, they will be used as the bootload parameters rather than querying the operator at bootload time. It is not necessary that any of this information be specified; if it is not, the dialogue will be entered to ascertain any information not supplied.

If bootload is being done from tape, then Bootload Multics will pause at an early stage to allow the clock to be set. Prior to the setting of the clock, no clock readings will be trusted for absolute time, although the clock is assumed to be running. The config deck may also be altered at this time, or a different config deck loaded. Also, if this is a cold boot, firmware will be automatically loaded into all the MPCs listed in the config deck (once it exists). This firmware is expected to be on the tape, in collection 1. As part of the MPC loading process, or instead of it if a warm bootload is happening, the firmware version and revision running in each configured MPC will be logged in the syserr log.

Normally, both Bootload and Service Multics will be bootloaded from the Multics System Tape image in the BOOT partition after the initial dialogue which is read in from the tape. If the BOOT partition must be reinitialized from the tape, a special control argument can be given to the initial dialogue which will instead cause Bootload Multics to be bootloaded from the tape; it will then rewind the tape and write its contents into the BOOT partition before proceeding to boot Service Multics.

The precise flow of initialization is not yet determined. It presently looks like it will be desirable to split what is presently in collection 1 into two collections, one of which is read from tape and the other of which is read from the BOOT partition. It is also not clear how it will be possible to alter the config deck at bootload time if automatic (no dialogue) bootload is being performed and the changes must be made in order for bootload to continue. These difficulties present no fundamental problems, and will be ironed out as implementation progresses.

COMMUNICATIONS BETWEEN BOOTLOAD MULTICS, REAL MULTICS, AND BOS:

During the implementation effort, it will be necessary for Bootload Multics, Service Multics, and BOS to coexist. This will be accomplished by a dual-toehold mechanism, in which both the BOS and Multics toeholds exist. The BOS toehold remains where it is today, at absolute location 10000. The Multics toehold is located at location 14000, for 4000 words (2 pages). The first program read in during a Multics (either kind) bootload is loaded at location 20000 (bootstrap1 is loaded at 14000 today, by the BOS BOOT command) by the self-relocating program started by the IOM Initialize/Bootload sequence. No modifications to BOS are required to use it to debug Bootload Multics.

Until the Bootload Multics/Service Multics switching mechanism is operational, Bootload Multics will be debugged as follows: Service Multics (of today -- none of the new bootload mechanisms are operational in this system) is brought up, using the BOS BOOT command. A Bootload Multics tape is generated, and Service Multics is returned to BOS with the Initializer bos command. A CORE SAVE is done from BOS, and the Bootload Multics tape is booted with the Initialize and Bootload buttons. It does whatever it's gonna do, and crashes or returns to BOS. BOS is the used to take an FDUMP, a CORE RESTOR GO is done, and Service Multics is restarted.

This is a very efficient way to test new systems. The total overhead for generating a tape, taking the system down, saving, restoring and restarting amounts to about five minutes. It is quite practical to test a new tape every 20 minutes this way, assuming that the problem can be analyzed and fixed quickly, as many problems can. Since there is a Service Multics "running" (but suspended), it is necessary for Bootload Multics to use different volumes for its RPV, etc., but this is easily accomplished. For the same reason, since BOS "knows" that Multics has returned to it, it is necessary for Bootload Multics to patch the location in the BOS toehold that indicates it is to save the core image when returned to; this permits FDUMP to work.

In order for Bootload Multics to be able to get a Service Multics started, and switch back and forth between itself and Service, it must have a toehold of its own, used for the same sorts of things that the BOS toehold is: swapping memory, saving the core image and machine state, etc. It will still perform minimal communication with BOS to set the flagbox up so that the current Multics (Service or Bootload) can be dumped when BOS is returned to, but its dependence on BOS will steadily lessen. One of the first things done will be to change sys_trouble to return to (or restart) Bootload Multics, rather than BOS; then the only way to get into BOS will be by an Execute Switches.

Once Bootload Multics is fully operational, the BOS toehold will be just a vestigial remnant of earlier days; something like vermiform appendix. Still, for the Level 68, there is no real reason to get rid of it entirely -- it is just a page, and it may be useful for debugging now and then if BOS is booted beforehand. Customers will presumably never use it, since they will not have the necessary BOS partition, and will not have booted BOS on the machine to set it up anyway. For ORION systems, it will be totally useless. There will be a mechanism for discarding the toehold, and reclaiming the page it occupies, when it cannot be used.

The Multics toehold will be "entered" from Service Multics the same way the BOS toehold is today: by an appropriate Execute Switches (which has to change again, to XED 14000, *sigh*), by an Execute Fault, by sys_trouble when syserr crashes the system, or by a deliberate call to Bootload Multics. If one of those things happens to Bootload Multics, it will reinitialize itself, and restart. A go command from Bootload Multics will start Service Multics again.

IMPLEMENTATION PLAN FOR BOOTLOAD MULTICS:

The following table is a brief summary of the implementation plan for Bootload Multics. For more detail, see the associated Multics Task Report on this subject. The general picture should be correct, although individual items may be added or removed as the implementation progresses. The earlier time estimates are relatively firm, but they become less so for the later steps.

This plan does not include any changes which are specific to the ORION, although many ORION changes will be implemented as part of the work. Also never mentioned explicitly in this plan is any time for study of existing code, or of methods for implementation of the new features.

This plan also does not include any time specifically allotted for writing other MTBs or user/operator documentation, or human engineering any of the user/operator interfaces. These topics are not covered for two reasons: one is that most of the internal design MTBs are simply treated as part of the implementation effort for any particular phase, and the other reason is that the finalizing of the user/operator interfaces is expected to take place over a long period, and will require little concentrated effort. Those interfaces have not even been designed, and they will require considerable exposure to insure that they are acceptable; since the exposure time will occur entirely in parallel with other projects, it is not relevant to budget specific times for it.

Implementation plan:

1) Prepare tools for accessing disk partitions and writing Multics System Tape images into partitions. 2.5 weeks.

2) Prepare a Multics system which can be booted without the aid of BOS. It will be the prototype Bootload Multics, although it will not be able to start Service Multics. 2.5 weeks.

3) Prepare a Bootload Multics which can operate in a different hardcore partition, set up the Multics Toehold, and swap its core image in and out. Create a primitive command level for Bootload Multics, with some commands to manipulate the toehold. Make a mechanism to bootload from a disk partition, and interface this to the primitive command interpreter. This is the first real appearance of Bootload Multics. 5 weeks.

4) Create commands for the Bootload Multics environment: DUMP, FDUMP, PATCH, etc. Create the appending simulation mechanism by appropriately modifying page control. Modify Service Multics interfaces to FDUMPs (copy_dump, ifd) to use the new information. Create the Bootload Multics file system (DIR partition) and associated commands. 6 weeks.

5) Create a more powerful command listener/interpreter. Add error handling facilities. 1 week.

6) Implement the replacement for RUNCOMs and associated features. Modify the toehold to properly support automatic reboot. 2 weeks.

7) Create the file editor, by converting edm to run in the Bootload Multics environment. 3 weeks.

8) Implement replacements for BOS SAVE and RESTOR. This includes creation of a Bootload Multics tape I/O, probably by converting tape_mult_. 6 weeks.

9) Write replacements for the FNP commands, such as BLAST and FD355. 3 weeks.

10) Write replacements for any remaining BOS commands, such as CORE, FWLOAD, LABEL, etc. These should require relatively little effort, since all the underlying mechanisms will now exist. 3 weeks.

The Bootload Multics System is essentially complete and shippable at this point. Much can still be done to add features and functions, of course.