TO: Distribution

FROM: R. A. Tilden

DATE: July 25, 1972

SUBJECT: Updating the Hardcore Libraries

This MSB obsoletes Multics Staff Bulletin #31.

TABLE OF CONTENTS

INTRODUCTION

This document outlines the procedures required to create a new Multics Hardcore System from an existing one. After two sections defining the data requirements for the system, the remaining sections outline the procedures currently in use to prepare the specified data and document the contents of the system.

Throughout this MSB reference is made to a series of exec_coms and abbreviations which have been created to facilitate the preparation of

a Multics Hardcore System. In each case, an attempt is made to provide sufficient detail so that the operation may be performed using only standard Multics facilities, then instructions are provided for effecting it by use of these tools. The novice user may wish to confine himself to the tools provided until such time as familiarity with the procedures makes him wish to change them, which he is of course free to do. The tools are merely aids the author found convenient for his own use. Further details of the tools may be found in the appendix.

It should be pointed out, however, that although the tools and procedures defined here are no more than suggestions, their results are not: the various data bases and their contents are expected by others to be in the states defined here.

The reader is assumed to be familiar with the basic Multics command repertoire and the nature of the Multics file system hierarchy, plus the use of the following commands which may be considered "more sophisicated": qedx, archive, exec_com, and abbrev. The SPS writeups of generate_mst, check_mst, and updater will probably be useful for details beyond what is mentioned here, but are not prerequisites.

1. Overview of the Organization of the Hardcore Libraries

The hardcore libraries are set up in a series of directories off of >library_dir_dir. Schematically, directories of interest include (alternate names are given after primary ones):

```
>library_dir_dir, ldd
      >hardcore, hard, h
            >source, s
            >object, o
            >bound_components, bndc, bc
            >info
                  >hardcore_system_book, hsb
                  >short_system_book, shsb
            >"system designation"
            >"system designation"hold
      >include
      >smag
            >multics_checkout_bulletins, mcb
            >updater_test
      >listings, list, l
            >"system designation"
```

In the foregoing schematic the term "system designation" appeared. This represents the designation of the new hardcore system and consists of two numbers separated by a hyphen, e.g. 17-1. In the remainder of this document, "system designation" will be abbreviated to SYS. Also, when reference is made to these directories, the shortest forms of their names will be used.

A. The Current Installed Hardcore System

The current hardcore system is to be found in four directories off of >ldd>h, with a fifth directory, >ldd>include, containing all the system include files. All of the source segments of the hardcore are archived in the directory >ldd>h>s; all of the hardcore object segments are located in the directory >ldd>h>o as bound or unbound object segments; all bound segment archive components of the hardcore are in the directory >ldd>h>bc. The headers, checkers, msl, and the map, bindfile and linkinfo archives (all defined in section II.) are kept in the directory >ldd>h>info.

B. The Hold Directory

When a hardcore system is changed a directory known as the save directory is created off of >ldd>h, called SYShold. The current system source archives and header are copied into this directory before updating to the new system is begun. This is a backup measure to facilitate recovery if the old system must be restored. Some time after the new system has been updated and is running satisfactorily, this hold directory is deleted.

C. New System Directories

There are two essential directories to create in which to put together a new hardcore system. One directory, created off of >ldd>l, is named SYS and will contain the listings for the new system; the other is created off of >ldd>h and also named SYS. This is the directory upon which the updater works (See section II.).

D. Other Hardcore Directories

Another important hardcore directory off of >ldd is named smag. In this directory are many tool programs with which to put together new systems, as well as certain documentation segments to be mentioned later.

Off of >ldd>smag is the directory named mcb in which the hardcore installation MCB is put together. And finally, also off of >ldd>smag, is a directory named updater_test which is used as an environment in which to test new versions of the hardcore updater.

E. Tapes

The former hardcore system tapes are kept for as far back as eight or ten systems. Also, each time a new system is installed, the listings of that system are dumped to tape. The list of systems and corresponding tape numbers can be found in a file called tape.list in the directory >udd>smag, and are maintained on a wall chart in NE43-505. All tapes are kept in the machine room on the

ninth floor of NE43.

II. Organization of the Updating Directory

The procedures for installing a new system are largely built around the requirements of a program called the updater. The updater installs the contents of an online directory into the hardcore libraries. The directory with which the updater works is known as the updating directory and the rules governing its contents are given below.

"Building a system" consists essentially in creating this directory, placing the required segments in it, and running various programs against it. It should contain the entities listed below, and nothing else.

A. Source Archive(s)

The updater requires that all source modules to be updated be in an archive or archives. There should be no include files in these archives. Include files remain unarchived in the updating directory. The source archives must have the name SYSsource.archive. If there is more than one source archive, successive archives are named SYS%source.archive with "%" any single letter or digit except s.

When a group of source modules are submitted together, often it is convenient to put them in an archive by themselves; also it is convenient to keep the source archive sizes down to around 20-30 records. These need not, however, be criteria for numbering source archives. If a system is small, the source archive need not be numbered.

B. Include Files

All include files must be unarchived (loose) in the updating directory and must have only their primary segment name.

C. Object Segments

All non-bound object segments must have only their primary segment name, and must be loose in the updating directory.

There is one known exception to this rule, the segment named "pds". See Special Cases, in section IV, part L.

D. Bound Segment Archives

All bound segment archives must have only their primary segment name.

E. Bindfile Archive

The Multics binder uses as control input a segment known as a bindfile, identified by the suffix ".bind", which it expects to find in the archive it is binding. All bindfiles which have been changed must be in an archive named SYSbindfiles.archive, as well as in the bound segment archive to which they pertain.

F. Bound Object Segments

These are generated by the use of the Multics Binder on the archives in D. All bound object segments must have only their primary segment name.

G. Map Archive

Bind maps as generated by the "-map" option to the binder must be contained in an archive named SYSmaps.archive. There should be a map for each bound segment involved in the new system.

H. Linkinfo Archive

The results of a print_link_info command for each bound object segment in the updating directory must be in an archive named SYSlinkinfos.archive.

I. Headers

A runnable Multics hardcore system is written on a Multics System Tape by use of the program "generate_mst". Control input to gm is a segment called a header. This header contains instructions to gm as to the names, position, and characteristics of each record it is to write on the tape. The header for the current system is kept in >ldd>h>info; if the new system contains no changes affecting the header, the updating directory may contain a link to the current system header, otherwise the new header must appear in the updating directory.

Now that the Multics development machine no longer uses a GIOC, but instead has the DataNet 355/IOM as replacement, special systems must be generated for use on the development machine. Great care has been taken to establish and to maintain parallel names between IOM oriented and GIOC oriented segments, bound segments, and include files in the hardcore libraries. All segments relevant to one type of hardware only name it: e.g., bound_gioc_page_control and bound_iom_page_control. Refer to MCB 883, "IOM Development Machine Usage", for tables detailing differences in the two flavors of systems.

Since all new systems have to be tested on the development system prior to service system installation, an IOM version as well as a GIOC version must be made. For the most part, the IOM system is a

reflection of the GIOC system. One version of the header (known as the IOM header, named iom_hardcore.header) is used to generate an IOM system and another version (named simply hardcore.header) is used to generate the GIOC system.

The GIOC header and IOM header should be in the updating directory either as edited segments or as links. The GIOC header must have the name SYS.header; the IOM header has the name iom_SYS.header. Notice that the first component of the name given to the GIOC header in the updating directory, whether it is a link or a segment name, will be the same as the relative pathname of the updating directory itself; for example, system 15-30 is updated from the directory 15-30 with the associated headers 15-30.header and iom_15-30.header.

J. Other Segments

The above are the segments needed to define a new system. Three other classes of segments are ordinarily found in the updating directory as well: the Multics System Tape generation program provides a listing of its operation which is stored as SYS.list (or iom_SYS.list); the MST checker program stores its output as SYS.checker (or iom_SYS.checker); and, finally, if this set of changes includes the deletion of segments from the system, a deletion control segment will have been constructed with the name SYSdel_list (see section IV, part K.). If you choose to use the table-of-contents (toc) abbreviation to keep track of what is being done in preparation of this system, its output will also be present, named SYS.ix (see Section VI.).

K. Additional Notes:

Only segments will be processed by the updater. Links in the updating directory will not be processed. Therefore the only permissible links are to unchanged headers, which don't need updating. There must be no directories inferior to the updating directory.

III. Initial Procedures

Installation submissions to the hardcore libraries are made on a standard form (yellow) indicating the name of the module being added to, deleted from, or replaced in the libraries; the pathname of the source and object of that module; the name of the bound segment associated with it if the module is bound; any include files required that are changed; any bindfile changes; whether or not a header change is required; and any other special instructions regarding installation. This form must be approved by the person responsible for hardcore system development, currently N. I. Morris. This form, together with a listing (made with the compiler's list option) for each requested change, is submitted to NE43-505.

When enough forms (enough by volume or enough by importance) are accumulated, a new system is defined. This system is assigned the next system designation by incrementing the number to the right of the hyphen by one, unless the changes are major enough to justify cycling the lefthand side. The first system with a new lefthand side has a zero righthand side.

The exception to the naming rules just described is for the all-too-frequently needed "fix" systems. A "fix" system corrects some pressing problem, usually one with a new system just installed. Such systems are identified by a letter suffix to the designation of the system they correct. The procedures outlined in this document are applicable to fix systems as well as ordinary installations, but are often shortcut due to the emergency that may be involved.

A. The yellow change forms are to be Xeroxed at some point in the updating procedure and passed on to the MPM technical editor for MPM documentation verification. Since the actual updating may be done anywhere, it is convenient to Xerox the forms right away and work from the copies. The originals should be kept available in the top drawer of the filing cabinet in NE43-505.

B. Next, log in and create an updating directory off of >ldd>h. Set access on this directory to rewa for SysLib, SysMaint, SysDaemon, and re for everyone else.

C. Create a listings directory off of >ldd>l to contain the listing segments. Set the access on this directory as above.

D. Change your working directory to the updating directory to begin creating the new system. Get a copy of the data segment active_all_rings_data.alm using the tools command get_library_source (gls). Using an editor, change the constant at label "system_id" to reflect the new system designation (note that the value is to be padded on the right with blanks to eight characters). Assemble active_all_rings_data leaving the object program in the updating directory and the listing in the listing directory.

E. At this point it is convenient to obtain the system headers either by copying them or linking to them depending on whether it will be necessary to modify them:

```
copy >ldd>h>info>("" iom_)hardcore.header (SYS iom_SYS).header
```

F. As mentioned earlier, there is a set of abbreviations you may wish to use to facilitate the operations being described. To obtain access to these abbreviations, see the introduction to Appendix A.

All of the operations described in this section (A through E above) may be accomplished by use of the abbrev "ss". Just type:

```
ss SYS
```

- and the directories will be created, the headers copied, and active_all_rings_data will be modified and submitted for absentee assembly.

This abbrev also redefines the abbreviation "sys" to be the current system designation. If the set of Librarian abbreviations is to be used, this one must be current, since many of the others are defined in terms of it.

IV. Preparing the Updating Directory

The following paragraphs detail a method of constructing the updating directory. Procedures are outlined in the order that has seemed safest and most straightforward in the past, but there is no necessity for following this order so long as the same result is achieved.

As a matter of fact experience shows that strict following of this program is usually impossible, because invariably someone shows up with a last minute change, or an error necessitates redoing part of the operation, or something of the sort. What follows, then, assumes that no errors are made, no special requirements force you to build the system piecemeal, and that there is sufficient time to proceed in an orderly manner. An ideal system construction scenario might call for accumulating all the source and submitting absentee compilations for everything overnight (an hour or so); the next day doing all the binding and archiving and generating the system tapes and submitting the iom_system for testing (a couple of hours); and on a third day, updating the libraries and preparing the documentation (another couple of hours). Generally a day will intervene between submitting the test and updating the libraries since the success of the test is known the morning of the third day but the system cannot be installed for users until that night.

A. Copy all source, bind, and include files indicated on the change request forms into the updating directory. This should be a straightforward process but usually isn't because of shortcomings on the part of those filling out the forms. There is provision on the form for the submitter to indicate whether the source pathname he is supplying is that of a directory in which the source is stored loose or that of an archive containing the source. Often no such indication is made. Often, also, there is difficulty in reading handwriting, or the language suffix is not provided, or there is some other problem interpreting the instructions of the person who submitted the change.

Any problem with an incorrect form is sufficient cause to return the form to its author for correction. Sometimes a guess will, however, save time, and as experience is gained, the guesses get better. It should be emphasized that you guess at your own risk, and that it is safer to ask. Most personel submitting changes to the hardcore use the AML ipc_message facility (q.v.) and often you can ask them online for clarification. Or if you guess, send them a message telling them what you guessed so they can stop you if you're wrong.

B. At this point the updating directory contains all of the source modules, include files, and bind files for the new system. The source segments and the bindfiles are destined for storage in archives, and you may wish to proceed to doing this at this time. The writer delays archiving the source until after all compilations are complete, but proceeds at this point to archive without deletion the bindfiles. An abbrev exists which will store all segments with the suffix ".bind" in the archive SYSbindfiles.archive:

```
acbind
```

The bindfiles are not hereby deleted; they will be deleted in the course of updating them into the bound segment archives in step G. This abbrev also changes the ACLs of the bindfiles to "r" to reduce the chances of their being deleted inadvertently.

C. There exist abbrevs and exec_coms to accomplish the compilation and assembly of the source modules you have accumulated, but the discussion of their use will be deferred until after the following description of what to do if you are not using them.

In the updating directory at this point are source modules with suffixes ".pl1", ".alm", ".gate", ".ioc", ".et", and ".mt". These are to be compiled or assembled using various language translators with the object segments ending up in this directory and the listings ending up in the listing directory. The most economical way of doing this is by use of links, rather than moving segments from place to place.

The exec_com described below uses the following technique: change working directory to >ldd>l>SYS; make links to all include files in >ldd>h>SYS; define the object segment via a link to >ldd>h>SYS; use the command expand to expand the source if the compiler in question doesn't do it for you; issue the command to compile with the appropriate options; unlink the links and delete the expanded source, if any. In the case of pl1 and alm source programs the compilation is effected by use of the full pathname of the source program as resident in >ldd>h>SYS when the compile command is issued; in the case of gate, ioc, et, and mt source segments they are copied into >ldd>l>SYS for compilation and left there for eventual printing since those processors don't provide source

listings as part of their output. (See section L for a special treatment for ".mt" segments.)

Tools for doing Compilations

There exist two exec_coms for doing these operations, useful because they make it unnecessary to do the typing described above and unnecessary to remember what the appropriate compiler options and names are, etc. One exec_com is for absentee operation and the other for interactive; both are available in >ldd>smag. To submit a group of segments for absentee compilation:

```
exec_com submit_hardcore_compilation SYS m1 m2 ... mn
```

or:

```
schm m1 m2 ... mn
```

- the latter being an abbrev which expands into the former. "m1", "m2", etc., are the names of the segments to be compiled, without their suffixes. When using the exec_com directly, an arbitrary number of segments may be specified; when using the abbrev, there is a limit of 9. However, if you list a great many, beware of the cpu time limit on absentee jobs. As an example of the use of submit_hardcore_compilation (also known as shc), to compile error_table_.et and gate_.alm for system 18-6, type:

```
ec shc 18-6 error_table_ gate_
```

or:

```
schm error_table_ gate_
```

This exec_com submits an absentee job which when scheduled makes use of the exec_com compile_hardcore, which may also be used directly for interactive compilation:

```
exec_com compile_hardcore SYS m1 m2 ... mn
```

or:

```
ec ch SYS m1 m2 ... mn
```

or:

```
chm m1 m2 ... mn
```

D. To be acceptable for installation into the hardcore libraries, a module must compile or assemble without errors of any kind from the compiler, including "implicit or contextual declarations". If such errors show up during processing, the yellow form for that module should be returned to the programmer who submitted and the

module should not be accepted until it has been fixed to compile successfully. It will depend on circumstances whether you wait for the corrections or excise the change from the new system.

When the compilations and assemblies are completed interactively, examine the console output. This will tell you if there was any difficulty processing the source modules.

When the compilations are completed by use of the shc exec_com the absentee output file will have been dprinted and deleted. Its name is compile_hardcore.absout.n where n is an integer suffix generated to keep multiple invocations of shc separated. If the absentee job failed, as indicated by your not receiving this printout, check >ldd>smag. This file should be there (possibly needing its bit count corrected by the adjust_bit_count command to be printable) as well as the segment "chabs.absout" which contains all the login lines for recent invocations of shc.

E. At this point the source segments may be archived. A useful aid in this effort is the AML active function "segs", which allows the use of the star convention in commands which do not ordinarily permit it:

```
archive ad SYSsource [segs *.pl1] [segs *.alm] ...
```

Actually, it is dangerous to simply give the command above unless you know the total size of your source segments. If the total should exceed 64 pages, the archive command will fail. So you may wish to use a sequence such as:

```
listtotals *.(pl1 alm)
archive ad SYS1source [segs *.pl1]
archive ad SYS2source [segs *.alm]
```

F. Once any source errors are resolved and all compilations are complete, you are ready to do the binding. Now, in addition to the source archives, include files, and bind files, the updating directory will contain two other classes of things: free standing object segments and object segments to be bound into bound segments.

To effect the necessary binding it will be necessary to copy bound segment archives containing other, unchanged object segments for consolidation with these new ones. This is normally straightforward unless more than one new system is being worked on. If this is the case, be careful to check whether the other system contains any bound segments matching those in your system. If it does, and is to be installed before yours, you must use its versions of the bound segment archives, not the ones from the library.

This next section covers the procedures for binding bound

segments. If you are familiar with these procedures, or don't wish to use the tools to be described, skip over to item J.

G. Before bound components can be bound into bound segments, they must be put into bound segment archives. Looking at the change request forms, find the appropiate bound segment for each of the object modules to be bound. You may have to check the msl (multics segment list) to see if a module is bound or unbound, as the change request forms often omit this information. This is done by use of the msl_info command (see the SPS). For example, to check on the segment tape_:

```
msl_info tape_ >ldd>h>info>msl
```

You need supply the location of the msl only once per process; it can be omitted on the second and subsequent mi commands. This will result in a line of the form:

```
tape_ al 03/31/71 bound_tape_ >ldd>hard>source>t4
```

This says that the module tape_ is written in alm (it would say "pl" for pl1); that it was last changed in March of last year; that it is a component of the bound segment bound_tape_; and that the source for it is stored in the archive >ldd>h>s>t4.archive.

The hardcore bound segment archives are kept in >ldd>h>bc. Copy the appropiate archives into the updating directory, replacing the associated object segment(s) as you go. Don't forget to check that the new version of the bindfile is updated into the archive if applicable. Then bind the archive using the "bind" command and specifying its -map option.

H. Now that the everything that needs it has been bound, the maps and the linkinfos are to be archived to comply with the requirements of the updater. Place all the .map segments in SYSmaps.archive. For each bound object segment, do the commands:

```
file_output BOUNDSEGNAME.linkinfo
print_linkage_info BOUNDSEGNAME
console_output
archive ad SYSlinkinfos BOUNDSEGNAME.linkinfo
```

Be sure that ready messages are turned off to avoid having them in the .linkinfo file.

I. Several abbrev's have been created to facilitate the above binding and archiving. In the simplest case, the copying, binding, and generation of linkinfo for a bound segment may be done using the "acbd" abbreviation. If your new system contains object segments destined for two bound segments, bound_a_ and bound_b_, all processing described in G and H, above, may be done by typing

```
acbd bound_(a b)_
acmaps
```

The first command, executed twice because of the parentheses, is set up to ask you about each object segment it places in the archive:

```
archive: segment blah is protected. Do you want to delete it?
```

You must type "yes" for each one. The reason for this is to allow you to check with the forms that what you expect to happen is indeed happening. If you are not questioned about a segment which is part of your system, for example, you know that you probably forgot to compile it and can take corrective action. Also, sometimes an object segment is a member of more than one bound segment archive. In this case you simply answer "no" for archives other than the last.

The second command, convenient to issue after all binding is complete, archives all the maps and linkinfos it finds in the updating directory.

It frequently becomes necessary to rebind a bound segment archive for some reason - a component changes or something. The "acbd" abbrev will refuse to operate when the bound segment archive is already present in the updating directory. In this case the archive is to be updated by hand and then bound using

```
bpl BOUNDSEGNAME
```

This abbrev does the same as acbd, without the archive copying and replacing. Or, if all updates to the archive are to be redone, the archive may be deleted entirely and regained and reprocessed using acbd.

J. Now to deal with the headers. If the headers are not to be changed, as will be indicated on the yellow forms, just link to them in >ldd>h>info. In linking to the headers, name the links SYS.header and iom_SYS.header according to conventions. If however the headers are to be edited, copy them from >ldd>h>info and edit them according to the change request forms. (The abbrev "ss", which you may have used to initiate your procedures, will have already copied the headers for you.)

A caution is appropriate here: be very careful if the current installed system was updated after work began on this system or if two new systems are being developed simultaneously that the headers in the updating directory are current. This is a common problem area - if there is doubt that the headers are current, locate the headers that should have been worked from and use the compare_ascii command to verify that the only differences are those specified on the forms for SYS.

Following the header change specifications on the forms is generally simple if the changes are alterations to existing header entries. When a new entry is supplied, the author should specify where it should go. He may only state "add to collection n". (There are three "collections", collection n being that portion of the header preceding the line

```
collection: n;
```

and following the corresponding line for collection n - 1). If no header position is specifically stated for a new entry, convention calls for it to be placed in its alphabetical position with respect to other entries in the same collection.

K. The Deletion List is prepared next. The syntax specifications and description of entries for the deletion laret are found in the SPS writup on the updater. Briefly, a deletion list is an ascii segment named SYSdel_list. It contains lines of the form

```
blah.pl1 *
```

- to delete source and object for blah from the system. Refer to the SPS for the syntax for other forms of deletion list items (such as deleting source only) if they are required.

Include files are not deleted by anything that is part of the hardcore Librarian's responsibility: they are deleted as an online installation. Deletion requests for include files found on changeforms should be forwarded to Arlene Scherer for processing.

L. Special Cases.

In this section are listed various exceptions to the rules for normal processing, usually because of shortcomings in the tools. In time, all of these will be eliminated.

1) When the segment pds is changed, it must be installed twice, once as "pds" and once as "template_pds". This should be mentioned on the form, but often is not. To achieve this, simply add the name "template_pds" to the pds object segment after it is assembled.

2) The source library updater doesn't know about .mt segments at present. Such segments must be stored under pseudonyms - after compilation, rename *.mt *.ioc, and let it be stored that way.

V. Creating and Verifying the System Tape

When all of the preparations to the updating directory have been made, a system tape may be generated. All system tapes are labeled with two to three digits preceded by the letters "sa". The tape

checkout list is to be found in >ldd>info>sys>tape.list and on the wall of NE43-505. Check out a pair of tapes by updating both lists.

The command generate_mst (gm) (see the SPS) is the procedure used to create the Multics System Tape (MST). This command uses the header as a driving file for generating the tape. It is called with three arguments: the first component of the header's name (SYS or iom_SYS), the system tape number assigned, and optionally the search list specifier, "-dr". This last argument is necessary only if an installed but not updated system exists when you want to generate the tapes. If you need to include such a system in yours, prepare a file named SYS.search in the updating directory containing:

```
(blank line)
>ldd>h>OTHER_SYS
>ldd>h>o
```

and use the -dr option. The default rules are to search the current directory, then if a segment is not found, search >ldd>h>o, and then if it is not found, it is missing.

When the tape has been generated, there will be an ASCII segment named SYS.list in the updating directory. This segment contains a list of the contents of the newly generated tape.

If there were no tape errors incurred while making the system tape, a message "tape errors = 0" appears. If there were tape errors, call the operators, x5948, and ask them to use a different handler. Then go back to write the tape anew. Continue trying until you have the message "tape errors = 0". If changing handlers fails, have them try stripping the tape. When there are no tape errors, a checker can be run.

The checker reads back the tape just generated and compiles a cross-referenced listing of its contents. It is called with three arguments: the first two should be "3 tape"; the third the tape number. The checker produces as output a segment T.ckrout, where T is the tape number; rename this segment SYS.checker.

The above operations are easily accomplished with abbrevs. To generate and check a GIOC system, type

```
gmg T
```

(T the tape number). For an IOM system, use "gmi" instead of gmg. If you need the -dr option, supply it as the second argument. In the event that gm succeeds but checker fails, you may use the checker by itself:

```
ckrg T
```

(or, of course, "ckri").

VI. Testing and Installing the New System

There are standard procedures for testing a new system on the development configuration of the 645. In the best of conditions these are handled by Operations without any necessity for programmer presence. Fill out a system test form (get one in NE43-505) by supplying the system identification and the IOM tape number, and give it to the operator on the ninth floor. These requests will also be accepted by phone (x5948). The test will be run when time becomes available, usually overnight.

In the morning, inspect the results which will consist of console and printer output from several canned scripts, plus, possibly, a dump if your system crashed. If the system failed, obtain the assistance first of crash analysis personnel to pinpoint the trouble area and then of the author of the suspect change.

When a system fails, it is usually necessary to change it to correct the bug or to de-install a defective module. Sometimes this must be done several times before the system proves solid. In cases like this it is convenient to have a current listing of the precise contents of the directories defining the system and of the archives contained therein. Such a listing may be generated and dprinted by typing

```
toc
```

A file SYS.ix will be generated and printed.

If the problem with the system is not obvious, the crash analysis people will need to have more details of the construction of the system. To print the headers, checkers, lists, and changed maps and linkinfos, use the "nsb" option to the print_sbook exec_com described in section IX, or the abbrev:

```
pnsb
```

When the system is working, systems assurance management will decide when to install it for users. Until this is done, it may be a waste of time to complete any more of the procedures described below (you mustn't run the updater) since the system is still subject to last minute change. It may however be worthwhile to go ahead and print one copy of the listings and make up the listing books (section VIII).

The morning that the new system, say 16-10, is installed, it will be called MSS 16-10x. When the updater is finished running later that day, the "x" will be removed from the system name by the operators.

VII. Running the Hardcore Updater

The updater (see the SPS) is responsible for installing the contents of the updating directory into the hardcore libraries. Currently, only people with SysLib access may execute the updater.

There is an exec_com in >ldd>smag called update which controls the updater. In the case of the updater, it is safer always to use the exec_com, as the calling seguence for using the updater directly is rather complicated. Type

```
ec update SYS
```

or:

```
UPD
```

The updater is very careful to keep you informed of what it is doing; not many of its type-outs constitute errors. But, if they do, it is important to correct them quickly as the system libraries may be in an inconsistent state. A list of error messages and brief explanations as to their meaning can be found in the updater writeup. Among the messages are ones stating that certain segments were not recognized (were "invalid"). These are not errors for SYS.list, iom_SYS.list, SYS.search, and SYS.ix; check them for anything else.

If there is some reason that the updater is not able to finish processing the updating directory, such as a system crash, recovery procedures from incomplete processing are described in the SPS writeup on the updater. Briefly, rename the updating directory and create a new directory >ldd>h>SYS. Copy all unprocessed segments over from the former updating directory and rerun the updater. Then delete this "fix-up" updating directory and rename the complete version to its former name.

The update exec_com sends a message to the operator telling him to remove the "x" from the system identification. To suppress this message supply an extra argument, "nomsg", to the exec_com.

After the updater has run, record the time, as this will indicate when the libraries were updated, a datum needed for the hardcore installation MCB which you will be preparing shortly. Save the output from running the updater and file it under SYS in the file cabinet in NE43-505 along with the yellow forms.

VIII. Getting the New System Listings

The new system listings in >ldd>l>SYS are printed twice: once to get the initial copy of the listings to make up a listings book and once to get a copy of them to send to be photo-reduced at Xerox. Often

the listings book is made up just before the new system is updated. Index tabs for each listing are made and the book is collated and labeled and filed in room NE43-504. The listings to be Xeroxed should have the date on which the system was updated on them. It is a good idea to thumb through these listings to check for badly printed pages or missing pages.

It is convenient to issue the dprint requests in alphabetical order, and an exec_com for the purpose is available:

```
ec list_print SYS  text
```

or:

```
lp  text
```

Text, if provided, appears on the separator page; if omitted, the date is used.

It is often desirable to defer printing until off-shift:

```
ec submit_list_print SYS time  text
```

or:

```
sarlp time  text
```

The exec_com submits an absentee job at the specified time or at midnight if time is omitted.

IX. Getting the System Books

As soon as a system has been updated, the "system books" for it must be printed. There are two varieties of these books, a regular (long) hardcore_system_book, consisting of checker, maps, header, bindfiles, linkinfos, and msl; and one called short_system_book which omits the header, bindfiles and linkinfos.

These "books" are defined by multi-segment files of the names mentioned above in >ldd>h>info. The data portions of the books are links to the appropriate entries in >ldd>h>info, entries which will have been put there by the updater. The exeption to this is the Multics segment list (msl), which must be formatted using the tool msl_global_format giving the output the name "MSL_GLOBAL".

Four books are needed, one long and three short: one short copy to be left in the machine room (ninth floor NE43); the complete copy for the listings room (NE43-504A); and the other two short copies for the system assurance office (NE43-505) and for Xeroxing. All but the copy for Xerox must be assembled into thumb-indexed binders, and labeled with the system name; the Xerox copy is delivered to

NE43-504 for shipment to Xerox.

Of course, there is an exec_com to do this:

```
ec print_sbook sb SYS
```

or:

```
psb
```

will print all four books after formatting the msl. To print a long book only, without formatting the msl, use "hsb" instead of sb; for one short book, no formatting, use "ssb". Abbrevs for these two are phsb and pssb.

X. Dumping the Listings

After the listings and system books have been printed, the listings have to be dumped to tape as a backup measure. There is a set of programs in tools known as the listing tape system designed to maintain and print listing tapes. Three sets of tapes are used in rotation, each set consisting of the three reels currently needed to contain the listings. These tapes are designated as list1a, list1b, list1c; list2a, b, c; and list3a, b, c. Dictionaries of their current contents are maintained in >ldd>smag>listing_tapes.archive as list1a.dict, list2a.dict, and list3a.dict.

Each time a new system is installed, its listings are merged with the most recent tape's. This can be done using the listing_tape exec_com in >ldd>smag:

```
ec listing_tape SYS n
```

or:

```
ult n
```

The argument n is "1", "2", or "3", specifying which set of listing tapes is to be used as input. Output will be to the next higher set except that the output of 3 goes to 1. The exec_com effects the merge and archives the dictionary as well as dprinting it. Absentee versions are available (this merge takes a long time):

```
ec submit_listing_tape SYS time n
```

or:

```
sult n
```

The sult abbrev uses a time of midnight.

As successive tapes are made, eventually listings will appear on tape that are no longer wanted. To remove them, place a segment in >ldd>smag named SYS.del_listing containing their names separated by newlines and prefixed by hyphens.

XI. Updating sys.info

The "help" files which Multics makes available to users are pooled in the directory >udd>message. Among these files is system_changes.info (sys.info), which provides a quick synopsis of the most recent hardcore system installations. Each time a system is installed, sys.info has to be updated. Typically, this is done by adding a paragraph containing an overview of the new system to the top of the sys.info segment.

Write a paragraph which contains a short but comprehensive overview of the new system. This overview will be much the same as the introduction section of the hardcore installation MCB (see below).

The first line in the descriptive paragraph will be the date on which the sys.info is being updated. The next line provides the name of the new system, and the date and time of its installation. After these initial two lines follows the text describing the new system. The most important changes should come first and then the lesser changes. As the last line of the paragraph state the time the libraries were updated (the time the updater finished running). Prefix this paragraph to >udd>message>sys.info and add an ACK character ("More help?" - octal 6) to the dateline of the former first paragraph.

After printing out the addition to sys.info and seeing that it looks all right, write it out and then type "help sys" as a final check on the update.

XII. Writing the Multics Checkout Bulletin (MCB)

After a new system has been updated, the system books and listings have been printed, and the listings dumped, a Multics Checkout Bulletin (MCB) has to be written describing the newly installed system. This MCB has a definite format comprising five distinct parts: the introduction, the system change log, the segment change log, the bound segment change log, and the header change log.

A. Introduction

The introduction provides an overview of the newly installed system, mentioning all of its features, defining the date and time that the system was installed, and the date and time that the libraries were updated. Typically, the changes that are most important to the users are mentioned first in the overview, with

the less significant changes coming later.

B. System Change Log

The system change log annotates each change as described on the yellow change request. The format consists of each sequential change request form number, followed by a description of the corresponding change. To obtain the first number to use in numbering the forms, refer to the MCB for the previous system.

C. Segment Change Log

The segment change log lists, alphabetically, all of the source segments that were changed in the new system, then all of the changed include files, and then all of the changed bind files. (Of course if either include or bind files were not changed, these catagories are merely omitted in the log). Each entry in this section includes a change index followed by the change request form number. The change index reveals the nature of the change, i.e., A if the segment was added, R if replaced, or D if deleted.

D. Bound Segment Change Log

The bound segment change log lists, alphabetically, the bound segments that were changed in the new system. This list consists of the name of a bound segment, followed by groupings of added, replaced, and deleted components of that segment.

E. Header Change Log

Finally, the header change log contains a report of all changes made to the headers of the new system. Each header change is mentioned individually, citing the change request form number.

An exec_com is available for formatting this document preparatory to typing it using the runoff command. In >ldd>smag>mcb type

```
ec hcmcb SYS
```

or:

```
MCB
```

The response will be to type three lines and wait for input:

```
FROM: author
DATE: date
SUBJECT: system
```

You are in qedx and can edit in your name, today's date, etc. When these three lines are as you want them, write them out as "part.rfc" and quit qedx.

```
wpart.rfc
q
```

Response will be the line:

```
input the intro to b(intro)
```

You are in qedx and in input mode. The current buffer is (intro). Type in the introductory paragraph, beginning new sentences on new lines. Use an empty line to separate paragraphs if you have more than one. You can leave input mode to edit what you have typed, of course. When finished write "part.intro" and quit qedx. Next you will get:

```
input the change log to b(cl)
```

Again you are in qedx input mode, for buffer (cl) this time. Type in the changes one by one, using many lines of prose per change as needed. Don't supply any blank lines. Begin the first line of each change description with a number sign followed by a four digit change number followed immediately (without blanks) by the beginning of the descriptive sentence. When done, write part.cl and quit. Next comes:

```
input the segment log to b(sl)
```

As before you are in input mode, now buffer (sl). Go through the change forms one by one. For each form type a line containing only a number sign and a four-digit change number. Then type in the segments changed by this form, in any order, one per line. Begin each line with "A", "R", or "D" as appropriate, followed immediately by the segment name. If the segment is included in a bound segment, follow the segment name by a blank or a tab then the name of the bound segment minus the beginning "bound_". If a segment is contained in more than one bound segment, use one line for each occurance. A sample input might be:

```
#1234
Rmap.pl1 command_loop_
Aio.alm init_1
Dio.alm init_2
#1233
Aa.incl.pl1
Rbound_faults.bind
Rpage.pl1
```

When finished, write part.sl and quit. You'll get

```
input the header changes to b(hcl)
```

Do so, in no special format, then write part.hcl and quit. If there are no header changes, quit without writing anything. The exec_com

then says

```
  begin formatting
```

- and eventually

```
  runoff SYSpart.rfc
```

Use runoff to type out the completed MCB. Be sure you have a dark ribbon and start the left margin one or two tab positions in.

If there are corrections to be made, you can correct the segments part.xx as necessary using an editor. Note however that the exec_com has prefixed all the names with SYS. So to correct the introduction, for instance, read SYSpart.intro. Then to reformat the document,

```
  ec hcmcb SYS review_input
```

or:

```
  MCB review_input
```

When the MCB has finished printing, it is to be delivered to Cathy Doyle in NE43-512. She will assign it a number and see to its publication.

Appendix - Abbreviations and Exec_coms

A set of useful abbreviations is at >ldd>smag>Librarian.profile, and the exec_coms as mentioned in the text are in >ldd>smag>Librarian.archive.ec. To obtain the abbreviations you may type:

```
.u >ldd>smag>Librarian.profile
```

or, to add them to your own profile:

```
ec make_profile
```

Most of the abbrevs depend on the currency of the abbrev "sys", which must be defined as the system identification of the system you are building. An asterisk to the left of the description below indicates such a dependency. When the description begins with a capital letter, it is a general statement of usage; otherwise it is the precise definition of the abbreviation. When material appears in parentheses it is a reference to the appropriate explanation in the body of this MSB.

Name	Usage
CRDIR	Create directory &1>&2, set CACL.
DD	answer yes dd
MCB	*Exec_com hcmcb SYS &1. (XII.)
UPD	*Exec_com update SYS &1. (VII.)
acbd	*Copy archive &1, bind and print linkinfo. (IV.I)
acbind	*Protect all ".bind" segments, archive them. (IV.B)
accrd	Copy, replace, and delete using hardcore bound segment archive library.
acmaps	*Archive all ".map" and ".linkinfo" segments. (IV.I)
b3w	bound_355_wired
ba1	bound_active_1
bcl	bound_command_loop_
bea	bound_error_active
beh	bound_error_handlers_
bf	bound_fsim_
bfs	bound_file_system
bga	bound_gioc_active
bgew	bound_gioc_error_wired

Name	Usage
bgid	bound_gioc_imp_dim_
bgii	bound_gioc_io_init
bgis	bound_gioc_imp_status
bgpc	bound_gioc_page_control
bgt1	bound_gioc_temp_1
bgw	bound_gioc_wired
bi	bound_ipc_
bi1	bound_init_1
bi2	bound_init_2
bia	bound_iom_active
bic	bound_init_control_
biew	bound_iom_error_wired
biii	bound_iom_io_init
bipc	bound_iom_page_control
bit1	bound_iom_temp_1
biw	bound_iom_wired
bmp	bound_mseg_prim
bn	bound_network0_
bo	bound_oc_
bpc	bound_process_creation
bpl	*Bind &1, create ".linkinfo" segment. (IV.1)
br	bound_reloader_
bs	bound_salvager
bsa	bound_sss_active_
bsf	bound_system_faults
bsw	bound_sss_wired_
bt	bound_tape_
bt2	bound_temp_2
bta	bound_tty_active
btw	bound_tc_wired
chm	*Exec_com compile_hardcore SYS &1, &2, ... (IV.C)
ckr	Execute hardcore checker.
ckrg	*Execute checker for GIOC system. (V.)
ckri	*Execute checker for IOM system. (V.)
crdir	*Create listings and updating directories, set CACLs.
dli	Create "&1.linkinfo" segment.
ec compile_hardcore SYS m1...	Interactive compilation request. (IV.C)
ec hcmcb SYS	Format Multics Checkout Bulletin. (XII.)

Name	Usage
ec init_sys SYS	Initialize for system preparation.
ec list_print SYS text	Print system listings. (VIII.)
ec listing_tape SYS n	Merge system listings to tape. (X.)
ec make_profile	Add Librarian profile to existing profile.
ec print_sbook book SYS	Print system documentation book. (IX.)
ec submit_hardcore_compilation SYS	m1... Absentee compilation request. (IV.C)
ec submit_list_print SYS time text	Absentee system listing print. (VIII.)
ec submit_listing_tape SYS time n	Absentee merge system listings to tape. (X.)
ec update SYS	Update hardcore system. (VII.)
glsh	gls -sys hard
gmg	*Generate and check GIOC system. (V.)
gmi	*Generate and check IOM system. (V.)
hdcp	*Copy hardcore headers.
hdir	>ldd>hard
hdlk	*Link to hardcore headers.
ldir	>ldd>listings
lp	*Exec_com list_print SYS &1. (VIII.)
phsb	*Exec_com print_sbook hsb SYS. (IX.)
pnsb	*Exec_com print_sbook nsb SYS. (VI.)
psb	*Exec_com print_sbook sb SYS. (IX.)
pssb	*Exec_com print_sbook ssb SYS. (IX.)
sarlp	*Exec_com submit_list_print SYS &1 &2. (VIII.)

Name	Usage
schm	*Exec_com submit_hardcore_compilation SYS &1, &2 ... (IV.C)
shcm	*Exec_com submit_hardcore_compilation SYS &1, &2 ... (IV.C)
smdmj	send_message Jordan SysLib
smrar	send_message Roach SysLib
smrat	send_message Tilden SysLib
smrbr	send_message Rakip SysLib
ss	Exec_com init_sys &1. (III.F)
sult	*Exec_com submit_listing_tape SYS midnight. (X.)
sys	System designation.
sysgo	*Change wdir to updating directory.
toc	*Dprint listing of contents of updating directory. (VI.)
ult	*Exec_com listing_tape SYS &1. (X.)