To: Distribution

From: P. Kelley

Date: December 28, 1973

Subject: The Multics Online Installation System

This document is an introduction to the program logic of the Multics Online Installation System. Excluded from this discussion are topics such as the submission of an installation, and the preparation of an installation including compilation, testing, systems assurance tests and code auditing. The complete installation procedure will be detailed in a future publication, the "Library Maintenance Manual". The term "installation system" throughout this document shall refer only to the actual updating tool.

I Multics Online Installation Requirements

The primary goal of a Multics online installation system is that it must be capable of updating a group of logically related system library segments while keeping the libraries in a consistent state. This implies that:

1) segments which are to be removed from service may appear in the address space of existing processes, and hence may not be deleted but must be preserved with all attributes except names intact until all such processes have terminated.
2) the unavoidable periods of inconsistency which occur while installing modifications should be minimized;
3) the installation process should be reversible, to allow removal of a bad (or inconsistent) modification;
4) the installation process should be restartable across system or process failures, to minimize the periods of library inconsistency which can occur after these failures;
5) the installation process should detect all errors which occur while the libraries are being modified; it should diagnose the cause of these errors, and it should recover from these errors, leaving the libraries in a consistent and usable state.

Other goals for the performance of the installation system are:

6) the installation procedure should provide generalized facilities for manipulating the attributes of segments being installed (e.g. names, ACL's, ring brackets);

7) the installation procedure should be short, simple to understand, and easy to use;
8) the installation procedure should have a simple input interface so that it can be invoked quickly;
9) full and accurate automatic documentation should be produced as part of the installation process;
10) the installation procedure should be table-driven to facilitate modification or extension of the system's functional capability.

II Multics Online Installation System Definition

Confronted with the installation system goals stated in section I, the next step is to design a system which will meet these goals. First, a definition of the terms "modification" and "installation" in terms of library maintenance. A modification is a group of segments which are related, either logically or physically. A logical relationship might be composed of several bound segments of the v2pl1 compiler, while a physical relationship might be composed of a bound segment, its source components, object components, source archive, and object archive. An installation is the actual implementation of changing the system libraries. It can:

1) add the new segments in a modification to the libraries;
2) replace existing library segments with segments in the modification;
3) replace existing library segments with segments in the modification while moving the replacement segments to another library;
4) move existing library segments to another library; or
5) delete existing library segments.

The act of updating a segment can be broken down into a series of ordered steps necessary to perform the update. These distinct steps, in the case of a replace operation, might include: copying the new segment into the library; setting the ring brackets of the newly created segment; setting the access of the newly created segment; freeing the names on the old (existing) segment; adding names to the newly created segment; and, logging a description of the update in an installation log file. The set of these ordered steps is called a task, and each ordered step necessary to perform a task is called a subtask. Which means, a modification is a set of tasks, each composed of a set of subtasks, which are implemented by an installation. Therefore, an installation can be accomplished by creating a task list and executing each task in the list. A simple example of a task list generated for the replacement of the two segments segment_1 and segment_2 as a modification might look like:

```
task 1:  replace segment_1 in SSS
         subtask 1:  copy the new segment_1 into SSS
         subtask 2:  set access on newly created segment_1
         subtask 3:  free the names on the old segment_1
         subtask 4:  add names to newly created segment_1
         subtask 5:  log the installation of segment_1
task 2:  replace segment_2 in SSS
         subtask 1:  copy the new segment_2 into SSS
         subtask 2:  set access on newly created segment_2
         subtask 3:  free the names on the old segment_2
         subtask 4:  add names to newly created segment_2
         subtask 5:  log the installation of segment_2
```

The execution of this task list results in the two segments being installed, but does not attempt to minimize the time between which the first segment is installed and the second segment is installed. (1) Note that after the segment in task #1 is installed, the segment in task #2 still has to be copied, have its access set, have the names removed from the old segment, and have names added to it until it is also installed. Therefore, the time interval of library inconsistency is a function of the execution of these subtasks and the number of segments being installed.

A solution to minimize the interval of library inconsistency is to re-order the subtasks within the task list. If a sequence number is assigned to each subtask of the original task list, then a new task list is created containing the subtasks ordered by sequence number, the result would look like:

```
(task 1)  subtask 1:  copy the new segment_1 into SSS
(task 2)  subtask 1:  copy the new segment_2 into SSS
(task 1)  subtask 2:  set access on newly created segment_1
(task 2)  subtask 2:  set access on newly created segment_2
(task 1)  subtask 3:  free the names on the old segment_1
(task 2)  subtask 3:  free the names on the old segment_2
(task 1)  subtask 4:  add names to newly created segment_1
(task 2)  subtask 4:  add names to newly created segment_2
(task 1)  subtask 5:  log the installation of segment_1
(task 2)  subtask 5:  log the installation of segment_2
```

The interval of library inconsistency in this new task list has been diminished drastically by limiting the interval to be a function of the number of segments being installed and the size of their respective name lists.

What other benefits can be derived from this "task list" approach to installations? Normally, the task list is executed by starting at the first subtask in the list and ending at the last. What would happen if the subtasks were executed in reverse order, starting at the last subtask of the list? If each subtask

---

(1) A segment is defined as "installed" when its external names have been added to it, thus enabling users to access it.

were designed such that it would perform its logical inverse function when executed in "reverse" order, then a picture of the execution of the task list in "reverse" would look like:

```
(task 2)  subtask 5:  log the de-installation of segment_2
(task 1)  subtask 5:  log the de-installation of segment_1
(task 2)  subtask 4:  remove names from new segment_2
(task 1)  subtask 4:  remove names from new segment_1
(task 2)  subtask 3:  restore the names to the old segment_2
(task 1)  subtask 3:  restore the names to the old segment_1
(task 2)  subtask 2:  remove access from new segment_2 (2)
(task 1)  subtask 2:  remove access from new segment_1
(task 2)  subtask 1:  delete the new segment_2 from SSS (2)
(task 1)  subtask 1:  delete the new segment_1 from SSS
```

The "reverse" execution of this task list would, in fact, de-install the modification, restoring the libraries to their original state.

A means of accessing this task list in a different process from which it was created, would be to store this list in a segment area. In this way, it could be executed, either forward or in reverse, in any process. By further maintaining in this segment area, a means of determining which subtask is being executed at any given moment during an installation, any interruption of the execution of the installation (e.g. process termination, system crash, etc.) can be restarted, or de-installed, from where the installation was interrupted. (Provided that the Multics hierarchy remains intact.)

Similarly, if an installation error is detected during the course of execution, an automatic error recovery mechanism can be invoked to reverse the direction of execution, de-installing the modification.

Note that there is a subtask to document each of the segments being installed. This provides automatic documentation of the modification at the time of installation (or conversely, de-installation).

III Multics Online Installation System Interface

A. General Command Interface

One possible command interface to perform this online installation scheme is to provide a procedure which appends tasks to a task list and then "installs" these tasks. For each segment of the modification being updated, there would be a task in the form of a segment request specifying what the task is to perform. The segment request would contain information such as: the pathname of the new segment to be installed; the pathname of the

(2) In actual implementation, access is not removed from the newly created segment, nor is the new segment deleted.

old (existing) segment being changed; the pathname of the target for this new segment (if different from the pathname of the old segment); the ACL to be placed on the updated segment; and, the ring brackets to be placed on the updated segment.

An advantage of this scheme is that it is completely independent of any library organization. A disadvantage is that this independence assumes that the installer has a priori knowledge of the library organization and must continually utilize this knowledge by specifying all the pathnames of the library segments. Another disadvantage is that the installer must explicitly state, in the form of requests, each segment to be updated, each ACL, and each ring bracket triplet associated with every segment to be updated.

The laborious specification of ACL's and ring brackets can be alleviated by making the following assumptions: in a replace operation, the ACL's and rings to be placed on the new segment can be derived from the old (existing) segment; and, in the case of an addition of a new segment, the installer may define default ACL's and rings for the installation, to be used for new segments. Both assumptions may be explicitly overridden by the installer at the time of issuing the request.

B. Simple Command Interface

To produce a simpler installer interface, the organizational independence of this system must be compromised. However, rather than completely sacrifice the system by coding library organization into it, the creation of a driving table containing all library dependencies will maintain the integrity of the system, and, at the same time provide organizational specifications necessary for a simple installer interface. This table, called a _library descriptor_ segment, will provide information such as: structure of the libraries; structure of backup and source libraries; initial ACL's for these libraries; and, ring brackets to be placed on segments created within these libraries. To define a different library organization, it would only be necessary to build a new library descriptor segment.

The formation of the installation task list may be accomplished by one of several approaches. One is based upon the concept of an "installation directory". All new copies of segments to be updated into the libraries would reside in this directory. The command interface would scan this directory creating a task list composed of tasks to update each segment located in the directory. Information such as ACL's, ring brackets and backup libraries, would be discerned from the library descriptor segment. This interface does not lend itself to the existence of multiple libraries, nor does it facilitate the necessity of installing special case segments such as gate segments, which need non-standard ACL's and rings. However, for applications such as the hardcore updater, it would render very useful.

An alternative is to create a "source" segment comprised of a list of segments to be updated with keywords indicating the library and action (i.e. add, replace, delete, or move), perhaps also residing in an "installation directory", in which ACL's and ring brackets, if not to be derived from the library descriptor, may be explicitly stated. The installation procedure would "compile" this list into a task list which, when executed, would update each segment accordingly. This approach maintains the flexibility of the updating tool while providing a simple installer interface.

IV The Multics Online Installation System (MIS)

Sections I through III of this document have attempted to state: the goals of an online installation system; the design to implement these goals; and, possible command interfaces for this implementation. This section briefly defines the current Multics Online Installation System, its limitations with regard to the previous sections, and its future development.

The Multics Online Installation System (MIS) follows the design stated in section II. It supplies a means of installing segments which is restartable, reversible, recovers automatically from errors, minimizes the period of library inconsistency, and provides the facility of automatic documentation. The current command interface to MIS is the procedure update_seg. Its level of development is as described in section III.A of this document. Although being an extremely flexible tool, the major deficiency of the current MIS is its lack of simplicity. The following outline define areas of development being pursued at this time, in their order of importance.

A. Full and accurate documentation of library modifications.

The documentation currently produced by MIS is a list of segment modifications to the libraries. This list is published weekly as an MIB ("Online Installation Changes"). The help file news.info is manually updated after an installation. The proposal for documentation (MCR #189) will also append a brief reason for the library modification in both the installation log file and news.info during the installation process. Formatted documentation for info and include files will also be produced along with the time of actual installation.

B. SPS documentation for MIS.

1. Submit SPS documentation for the update_seg command interface.
2. Draft an MTB describing the proposal for documentating MIS as a separate section of the SPS entitled "Library Maintenance Manual". This supplement would include a section on library maintenance, program logic descriptions of MIS, and SPS-type module documentation.

C. Increase the size of online installations.

MIS is currently limited in the number of segments which can be updated in a single modification. The proposal (MCR #188) will increase the number of segments which MIS can handle so that modifications the size of the V2PL1 compiler may be installed with a single installation.

D. Initial ACL's and safety switches.

Incorporate the concept of Initial ACL's for segments being added to the libraries. Currently ACL's are determined explicitly by the installer. Also incorporate the "safety switch" facility into MIS.

E. Simplify the installer's interface to MIS.

1. By splitting the update_seg procedure into its two logical sections (i.e. argument processing and function execution), the capabilities of MIS will be made available for alternative forms of command interfaces.
2. Once update_seg is split then the concept of a library descriptor segment can be implemented by creating an appropriate command interface as described in section III.B.