To: Clingen
Corbató
Daley
Feiertag
Gintell
Morris
Roach
Saltzer
Steinberg
Voydock
Webber

From: T. H. Van Vleck

Date: 6/7/72

Subject: Tape Mounting and Labeling, and Operator Communication

Three memoranda are attached, which cover high priority tasks numbered 3 and 4 in MSB-26.

The implementation described here will provide

1. An operator communication module
2. System validation of access to tape
3. Accounting for tape usage

The design has been laid out so that we can implement the most-needed features first, with minimum effort, and add pwerful new facilities later.

Your comments will be appreciated. If no significant objections are made to this implementation plan by, say, June 26, these memos will be published as MCB's, task lists will be generated, and implementation will begin.

To: Distribution

From: T. H. Van Vleck & Dennis Capps

Date: 6/6/72

Subject: Initializer and Daemon terminal management

This memorandum describes a set of changes to the system-control programs which will accomplish the following goals:

1. Minimize machine-room terminal requirement.

2. Allow use of additional machine-room terminals for the convenience of operations with large configurations.

3. Eliminate problem of system interrupting operator typein.

4. Provide for flexible hard-copy and multiple-console message disposition.

5. Simplify system initialization sequence.

6. Provide for communication between user process and operator (e.g., tape mounting/authentication).

These goals can be realized by writing only a few programs, and modifying a few others. In particular, no rewrite of the daemons is required, and only minor changes to the answering service.

System Control Functions

Several system control functions need to communicate with the operator and with each other with varying urgency. These are:

1. Answering service (telephone answering, login, absentee scheduling)

2. Bulk media management (e.g. IO Daemon)

3. Backup

4. Tape facility

5. System administration routines

6. Accounting procedures

7. Operator commands

Currently some of these are handled in the initializer process, some by separate processes. Regardless of the organization, these functions will be present and will need to communicate. This system control process group will be provided with a message coordinator which will be event-call driven and which may operate in ring 4.

Consequences of goals

The easiest way to accomplish these desired results is to write a new special IOSIM, called the "message routing dim" (mrd_), which will direct all ouptut from the system control functions to a special "message" segment, shared between the daemons and the initializer process. The functions will also extract any input lines they need from this segment. The message coordinator will write output messages on the typewriters assigned to system control and/or in "log" files. Output from one function may appear on more than one typewriter and individual messages may appear on several typewriters simultaneously if desired. (IO conventions are possible which could allow a function to bypass the message coordinator with, for instance, a known trivial message which is merely to be logged.)

The major work needed to implement the new strategy is to modify system_control_ so that it can operate multiple terminals. The code to do this can be constructed after the pattern used in dialup_ and aswa_. Each device and each function will have a small data base and an event-call channel associated with it, and system_control_ will perform most of its operations in response to wakeups from these event-call channels.

When the system is brought up, the initializer process will discover its main terminal channel in a supervisor data base, just as it does now. For a minimum configuration, this will be the SIOC or IOM master console channel. This channel, and any others listed on the OPTY card in the BOS configuration deck, will be given to system_control_ to operate. Initializer commands may be given to system_control_ at any time to add or delete any of these terminal channels. A simple startup command list facility will be added to the initializer to permit abbreviation of the "standard" startup sequence.

A simple modification to the answering service will permit the initializer to request the automatic login of the standard daemon processes at operator command. The same startup command list feature can be used here. A final easy modification to the answering service will convince the daemons to use mrd_ instead of tw_ when attaching their i/o streams.

Most initializer commands do not interact with the operator. When called, they perform their function and produce output, but do not read additional input. The exceptions to this pattern are the reconfiguration commands, message-of-the-day editing, and initializer "admin" mode usage of the command system. These usages will be supported by temporarily detaching a typewriter channel from use by system_control_ and attaching it through the normal ring-4 typewriter dim. Only one of these functions can be performed at any time, since the ring-4 tty dim cannot support asynchronous operations.

Input-Output Protocol

We will distinguish between

Channels	device channels (or files)
Virtual Consoles	Symbolic names, attached to channels
Sources	Classes of messages, from same process
Source ionames	Ionames within a daemon process

The "routing table" will connect sources to virtual consoles. A source must be sent to at least one virtual console and may be sent to many. Figure 1 shows how the input-ouput routing graph will look.

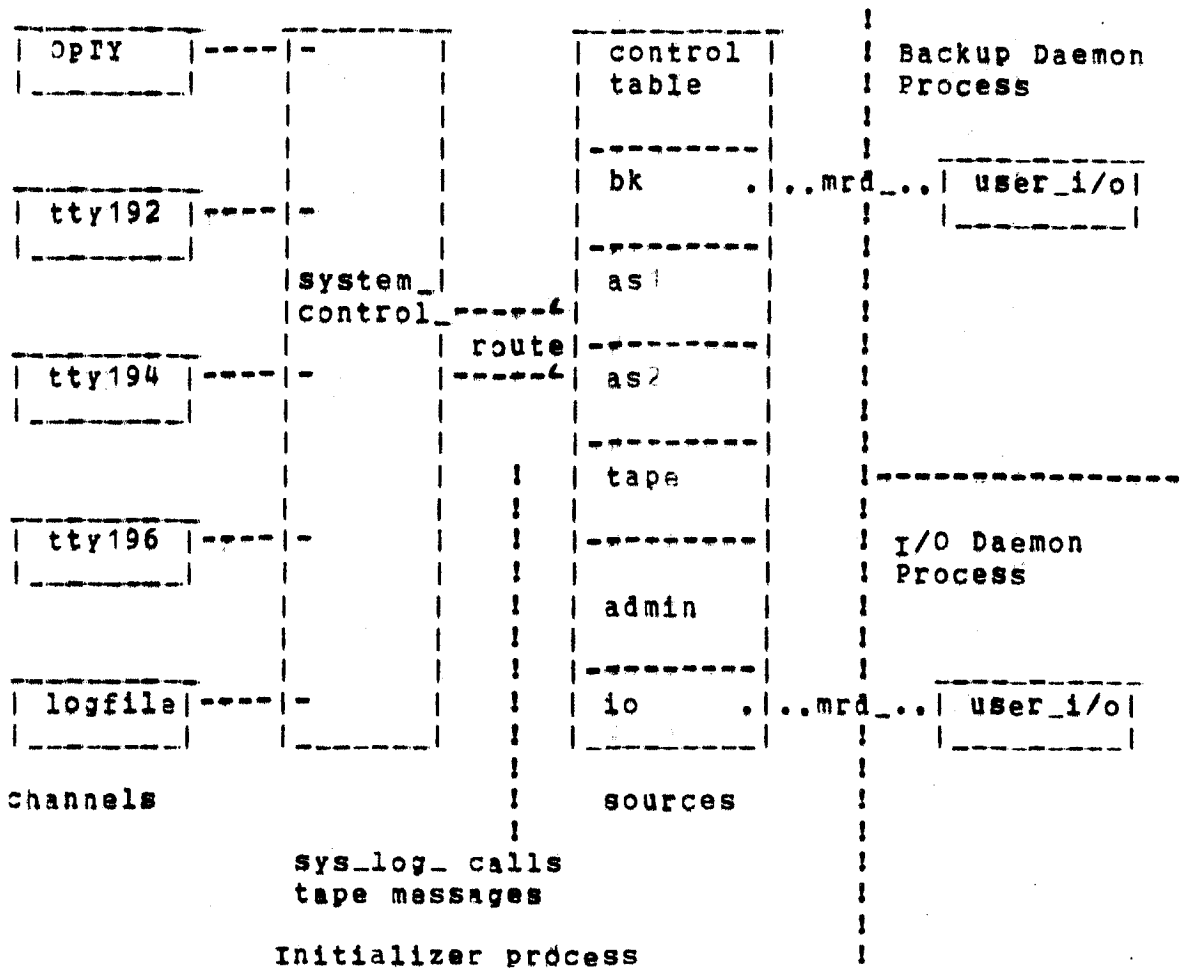

Figure 1 - Routing graph

When an operator wishes to input a line, to reply to a tape-mount request, or start up or recover a daemon process, he simply goes

to the nearest console and enters a carriage return. When the return is received, system_control_ will suspend output on that terminal (for say 2 minutes), and type

OPER:

so that the operator may enter an initializer command. The initializer command will usually be "reply", and will be used to send a line to a daemon process. (Carriage return is used because of a deficiency in the "quit" signalling mechanism -- when a quit is signalled, the initializer has no way to determine which console hit quit.)

A fragment of console output on an initializer console might look like this:

```
1021.2 as up_sysctl_: SAT installed by RHart.SysAdmin
1030.1 as LOGIN tty208 27#1 333 Smith.Proj1
1035.1 tape 49 mount reel 275 on drive 3 for Smith.Proj1
-->tape
1036.8 tape 50 mount reel 903 on drive 4 for Smith.Proj1
-->tape
1036.9 as LOGIN tty102 TTY33 none Jones.Proj2
   (operator hits return)
OPER: reply tape 50 ok bvm
1040.5 bk dumper waking up.
   (return again)
OPER: r tape 49 notape
1052.1 tape 51 mount reel 725 on drive 3 for Smith.Proj1
1052.1 tape -->
   (return again)
OPER: r tape 51 ok qzx
1112.2 tape dismount tape on drive 3 and return to slot 725
```

The sentinel "-->" is typed when input is expected from the operator. Note also that the operator does not have to reply to the mount messages in the order received. The number in the message is used as a tag which tells which request he is honoring. (There will be some natural upper bound on the number of such requests that may be pending.)

Startup sequence for the Initializer

The following is a listing of the "startup command list" file for the initializer process.

```
* MIT system startup deck
*
* define virtual consoles and their channels
*
define master tw_ opty
define as_console tw_ tty192
define io_console tw_ tty194
define bk_console tw_ tty196
define as_log file_ answering_service_log
*
* route messages to virtual consoles
*
route bk user_i/o bk_console
route io user_i/o io_console
route tape user_i/o as_console
route as3 user_i/o *all
route as2 user_i/o *as_console as_log
route as1 user_i/o as_console as_log
route as0 user_i/o as_log
route asm1 user_i/o as_console
route asm2 user_i/o *as_console
route * user_i/o master
*
* log in the daemons
*
login IO SysDaemon io
login Backup SysDaemon bk
*
*end
```

In this example, "master", "io_console", "bk_console", and "as_console" are virtual console identifiers.

The "route" commands send messages from the sources "io", and "bk" to virtual consoles. Leftover messages will be sent to the master console, if any arise. The sources "as3", "as2", "as1", etc. are the various levels of severity of answering service messages. An asterisk before a virtual console means to surround the message with stars and to ring the typewriter bell if possible.

We can gain powerful benefits of this scheme later, when we have time to modify the daemons to use many streams for different types of messages instead of just error_output and user_output. We will be able to reduce the amount of typing of "normal status" messages, by directing these messages to a log which is dprinted automatically, and at the same time insure that unusual status messages are brought to the attention of the operators and the

responsible system programmers, by sending them to a special log file and to a console.

Code to be written

1. Message routing module. (mrd_)
   This is very easy; the current AML interprocess message facility provides most of the insides.

2. Rewrite system_control_ for multiple terminals.
   a) Channels will be event-call driven, as in dialup_.
   b) Special code to detect commands which capture a console, and detach them from multi-console control until exit.

3. Other modifications to system_control_
   a) Command handling rework.
   b) Add startup command list feature
   c) Add "reply" command to send lines to daemons.
   d) Add "login" command to log in daemons
   e) Add "define" command to define virtual console
   f) Add "route" command to route source to virtual console

4. Answering service modifications to allow automatic daemon login.

5. Possible later split of answering service into separate process from initializer.

6. Possible later modifications to "dial" to allow us to "dial initializer", and have console accepted and added to configuration.

Thoughts on design of system_control_

The system_control_ program will do most of its work in response to event signals, by use of the event-call mechanism. Entry points must be added to system_control_ for console interrupt (i.e. read completion) processing, and to respond to requests for input from daemon processes.

We will need several data bases to support the multiple-console features. For each function and each virtual console a communications segment like the "con_msgs" segment of the AML send_message command will be needed for message passing. In addition, we will need a data base giving the list of channels currently controlled by the answering service, and their status and control variables; a data base listing the known sources, with process ID and event channel; and a routing data base which connects sources to virtual consoles. Perhaps someday the routing may include a content specification, somewhat like the specifications for the command which processes the answering service log.

Normally, system_control_ will block explicitly, not waiting on any console channel. When in "admin" mode or when editing the message of the day, its block point will be in the normal ring-4 tty dim. When the "motd", "admin", or reconfiguration commands are recognized, system_control_ will detach the channel which produced the request from the multiple-console control and attach it through the standard typewriter dim. A flag will be set which will cause any other request of this type to be rejected, with the message "Function busy", until the operator finishes his operation.

If a virtual console is attached to a segment rather than a typewriter channel, we can either attach the segment by use of the "file_" dim or modify the program sys_log_ to keep these segments. If we use a special program, we will not have to modify the administrative log-processing programs.

Appendix A
New Data Bases

System Control Message Segments

Each function and each virtual console will have associated with it a system control message segment similar to the "con_msgs" segments of the AML ipc_message_facility_. The following declaration was lifted from there.

The segment consists of a header followed by many message blocks, each capable of holding a message of 132 characters plus information on when the message was sent, who it's from, etc. These blocks are "allocated" only as needed and placed on a free storage list when fread. Blocks in use are chained together in a first-in-first-out queue.

```
dcl 1 syscon_msgs_dscr based;
          2 mlock bit(36),
          2 ipc_channel fixed bin(71),
          2 current_process_id bit(36),
          2 flags unaligned,
               3 unused bit(36) unal,
          2 first_message fixed bin,
          2 last_message  fixed bin,
          2 first_free_buffer fixed bin,
          2 last_assigned_buffer fixed bin,
          2 messages(1) aligned,
               3 next_message fixed bin,
               3 time_sent char(24),
               3 from_user char(22),
               3 from_proj char(9),
               3 flags unal,
                    4 continue bit(1),
                    4 unused   bit(35) unal,
               3 message_body char(132);
```

To: Distribution

From: T. H. Van Vleck

Date: 6/3/72

Subject: User Tapes

This memorandum describes an implementation of the user tape facility for Multics.

The general design of the Multics tape management is described in three memos by Mike Spier, dated October 12, 17, and 21, 1972. Several steps toward the full proposal have been made recently. This document proposes enough additional implementation to allow us to open the tape facility to all users.

Specific features of the implementation described below are:

1. The use of code in ring 1 to intercept attach and detach requests.

2. Ring 1 maintenance of a Tape Reel Description Segment (TRDS), which provides logical protection of each tape reel.

3. An orderly communication path between ring 1 of a tape user's process and the system tape operator, which allows the user to inform the system of tape mount and dismount requests.

4. Tape label checking (and/or authentication code checking), performed by ring 1 of the tape user's process.

5. Tape accounting.

Overview of Tape Management

This section gives a brief overview of the tape management facilities which will be implemented. More detailed discussions of the philosophy behind the implementation may be found in Mike Spier's memoranda.

Information Protection

Two levels of information protection must be supplied for tape data: logical protection, which defines which users of the system may use the data and how the data may be used, just like the protection provided by the Multics storage system; and physical protection, which arises from the detachable nature of tapes, and which insures that the physical reel being used is indeed the correct one to which the logical protection applies.

Logical protection will be provided primarily by an extended access control list like the standard Multics ACL for a message segment. This access will be managed in ring 1, by special tape-handling primitives. Since some uses of tapes bypass the normal Multics access control mechanisms (such as use by BOS and by the reloader), these stand-alone uses must have some simplified mechanism for access control, to insure that they do not accidentally bypass the system's logical protection mechanism. Tapes may, therefore, also have an accessibility indicator, which will be checked by the stand-alone mechanisms if the TRDS is not available. This indicator will be stored in the TRDS and written on the tape label. The values for this indicator will be:

blank - User tape. Stand-alone mechanisms will refuse.
S - System tape. For bootloading only.
R - Reload tape. Backup may write, cold boot may read.
B - BOS tape. May be read or written.

During normal Multics operation, the accessibility indicator will not be checked, although it will be written into the tape label from the TRDS. The value of the indicator in the TRDS may be changed by a privileged operation. Note that a user may read or write a tape during Multics operation, without regard to the value of the indicator, since his access is controlled by the ACL on the TRDS; note also that only labeled tapes may have a non-blank accessibility indicator.

An optional retention date will also be associated with the data on each tape reel, to specify the earliest date that the contents of the tape may be rewritten.

Physical protection will be provided by code which will be installed in ring 1. For Multics standard format tapes, this code will check the tape label of the tape and insure that the tape mounted by the operator is in fact the tape requested. For

non-standard tapes, the ring 1 code will check to make sure that the tape either has a correct label, or no label. For tapes which are not labeled, the operator will be required to specify an authentication code when he mounts the tape. He will be told the number of the tape to mount, but not the authentication code; when the tape is mounted, the operator will be required to type in the three-character code (which will be on a sticker on the tape reel). The ring 1 program will reject the tape if the authentication characters are not correct. The authentication code is a non-obvious function of the tape reel number, and can be calculated by the system.

Reel Management

Besides insuring data protection, the tape management facility must provide the facilities necessary to allow installations to manage their inventory of tape reels. This will include administrative procedures for labeling and registering tapes, for obtaining a list of all tapes, or all tapes for a given user, and for producing accounting figures which can charge a user for reel and slot rental and for tape mounts and mount time.

Internal Data Bases

Each reel of tape at a Multics installation will be associated with a ring-1 segment known as a Tape Reel Data Segment (TRDS). These segments will be kept in a special directory in the file system hierarchy. The extended access control list on each TRDS will give the logical access control for the data contained on the tape reel.

The information contained in the TRDS will include the reel name, coded as part of the path name of the segment; the authentication and label-checking switches; the accessibility indicator; and information about the density, recording mode, and number of channels. Other information may also be useful, such as error count. Most TRDS's will be zero-length segments, to conserve storage; default values for all data which might be contained in a TRDS will be chosen if its length is zero. A data base write-up for the TRDS is attached as an appendix.

It is desirable, for maximum physical protection, for most tapes at an installation to be labeled. The label must contain the tape identification, so that the system can verify that the reel which has been mounted is the same as the one which has been requested. Another memorandum will propose a new tape-labeling scheme which will conform to ANSI standards.

The hardcore I/O Assignment Table (IOAT) will continue to be used to record the association between a process and the channels or devices associated with it.

Overview of Tape Management

One additional data base will be provided, for communication between the ring-1 attach/detach programs and the operator process responsible for tape management. This data base will record all mount requests, from the time they are made until the tape is dismounted or the mount rejected, so that a list of all currently-mounted tapes can be extracted from it. The request state, the user's process ID, the tape label, and the time of the request are examples of data in this segment. A data base write-up for the communications segment is attached as an appendix.

User Interface

Users who are now using Multics standard tapes will not be required to change any of their operational procedures, if they are using system-provided tape software. The user will continue to provide, as the third argument to ios_$attach, the tape reel identifier of the tape which they want mounted. The system will look up the TRDS for the tape in the tape directory, and, if access is correct, issue the mount request. After the tape is mounted, the user may read and write tape just as he does now. When the user detaches a tape device, the system will rewind and unload the tape (usually), and will request the operator to re-file the tape.

Since the authentication code is a function of the tape reel number, unlabeled tapes can be handled in the same manner. That is, when the user requests the attachment of a tape reel which is registered as a unlabeled tape, the system will calculate the authentication code from the reel number, and include the authentication characters in the table entry for the mount request. The operator will be instructed to reply with the tape authentication characters when he mounts the tape, and the mount will be rejected if the authentication characters do not match.

In order to read in tapes from other systems, we may occasionally need to be able to handle tapes which have no TRDS. These functions will be supported by requiring a special syntax in the tape reel identifier, so that a user might request the mount of

```
unregistered,9TRACK,NL,comment="reel 37A5"
```

meaning that he wants an unregistered reel mounted, on a nine-track drive, that the tape has no label, and that the operator should look for a reel labeled 37A5. See the memo on tape identification and labeling for more information on this subject. In order to insure the integrity of the logical protection mechanism, no tape which has a TRDS may ever be mounted by this method. Special privileged intervention by the operator will be required to verify the user's permission to use the tape in this case.

Several other special tape identifiers can be constructed for the convenience of users and system operations. The identifier "scratch" will refer to any available scratch tape. The identifier "next_backup_tape" will refer to the next reel for an incremental dump, and the system will assign a tape and report the reel number used back to the daemon.

Operator Interface

The system control tape management function may execute in the initializer process, or in some other process. Its mount and dismount requests will be sent through the system control terminal management facility to whatever physical console is designated as the destination for mount and dismount messages.

The tape management function will be invoked by event call whenever a user request is made. The event-call message will contain a reference to the communication segment, where the details of the mount request will be found.

The mount request will cause a message to be typed to the operator which looks like this:

```
1035.1 tape 49 mount reel 275 on drive 3 for Smith.Proj1
--> tape
```

The number "49" in the message is an internal table index, which the operator will use in his reply. Indexing the messages in this way permits the system to have more than one mount pending at once, while retaining the ability to associate the operator's reply with the proper request. Note that, in general, we cannot have more than one request for each drive on the system pending.

The operator will use the "reply" command to tell the tape function of the success or failure of the mount request.

```
reply tape 49 ok bvn
```

This example is an affirmative response, in which the operator also typed in the tape authentication characters from the sticker on the tape. Authentication might not have been required, if the tape were a standard Multics labeled tape; the system will check the authentication if it is supplied, for additional redundancy.

The operator will be given a new command called "tape" which can list all the requests in the communication segment if the operator types

```
tape list
```

or which can print the contents of a TRDS if the operator or user types

```
tape status 300
```

This command can also be used to register a new tape reel or unregister an old reel, if the operator types

```
tape register 3701,9TRACK Doe,Multics
```

Suitable defaults will be applied to the arguments to "tape", and the star convention will be interpreted where appropriate. This command may also be used from other processes, if the user has the appropriate access to the ring-1 gate which provides the support for the command. A write-up for the operator commands which will be available is attached as an appendix.

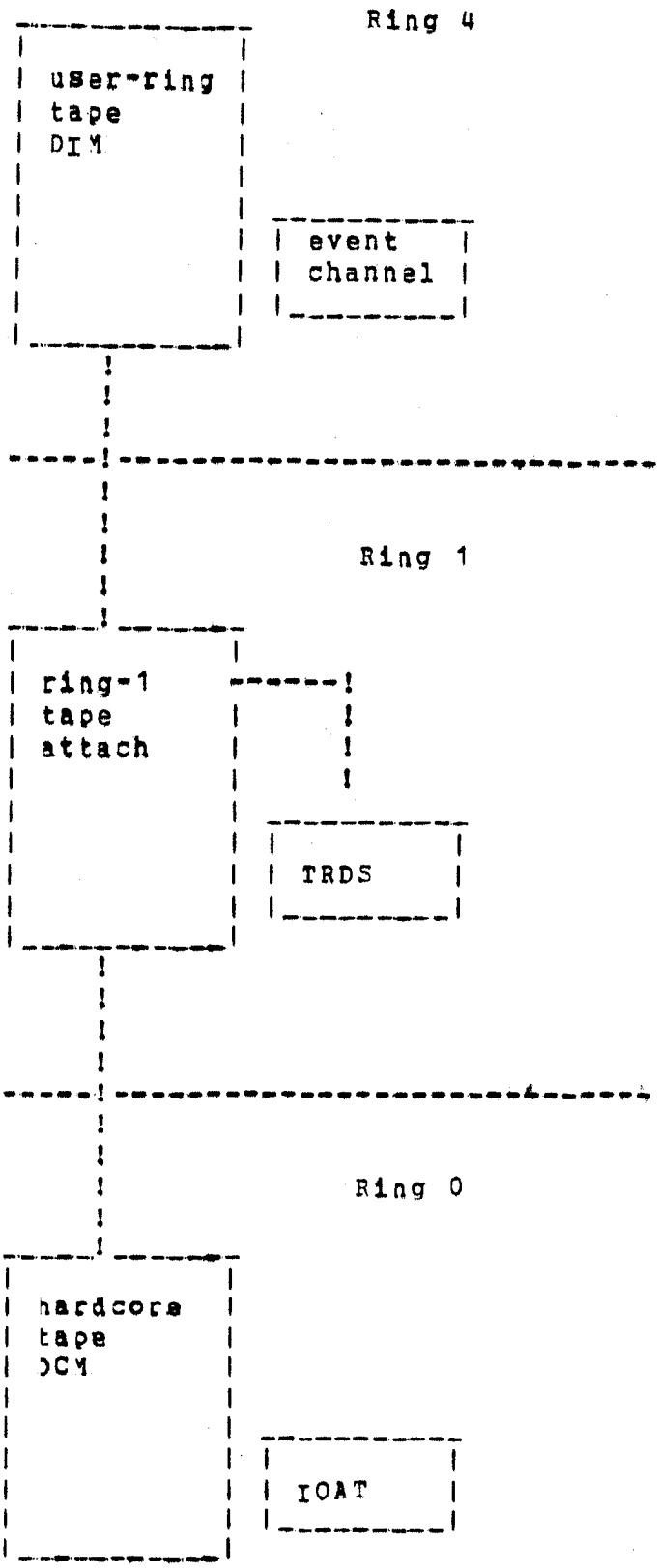

Figure 1 - User calls to attach tape

Tape Mounting Protocol

When the user makes a call to attach a tape, the user-ring tape DIM calls into a new gate segment into ring 1, which

1. Obtains and checks the tape reel identifier.

2. Interprets conventional identifiers like "scratch".

3. Looks up the TRDS for the tape requested.

4. Checks that the user has access to the TRDS.

5. Obtains all tape attributes from the TRDS or assigns default attributes if TRDS length is zero. Retention date is checked here if write attachment is specified.

6. Calls the haricore DCM to get a drive assigned. This drive cannot be used by the process from ring 4 yet.

Tape Mounting Protocol

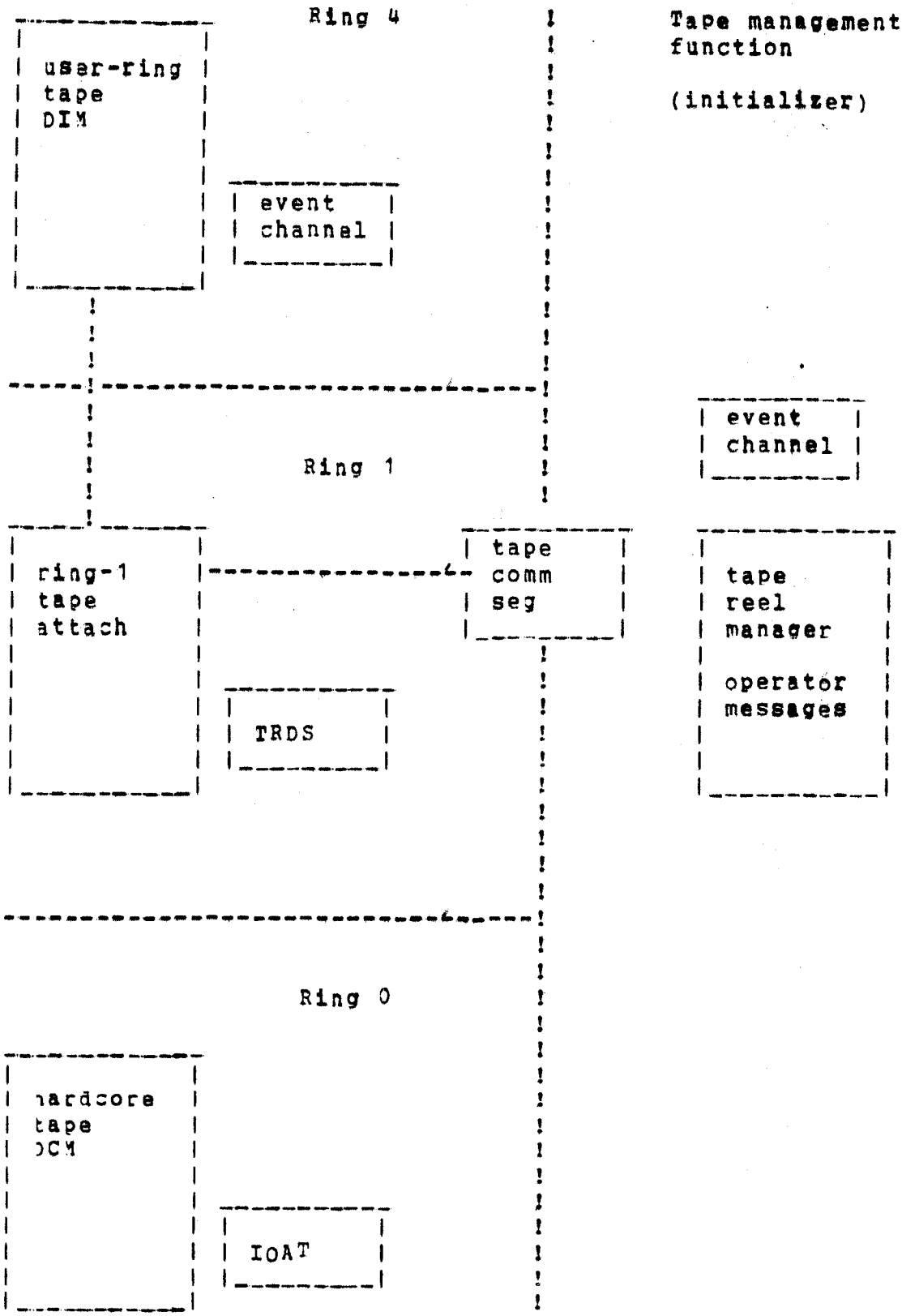

Figure 2 - User signals tape manager

Tape Mounting Protocol

If a drive is successfully assigned, the ring-1 tape attachment module obtains a free slot in the tape communications segment, and fills in the mount request information. This will include the tape identification, authentication if required, and perhaps a user message. The user's ring-4 event channel identification and process ID will also be placed in the request, so that the tape manager may reply to the user.

The user process will then signal the tape management function over a public event channel. The message sent over the channel will contain the index in the communications segment of the mount request.

The user process then returns to the user-ring tape DIM, which blocks on the user's event channel for the tape.

The tape manager, when awakened, performs any necessary validation of the mount request and then displays a message to the operator of the form

```
1324.2 tape 19 mount reel 3702 on drive 6, auth
--> tape
```

The tape manager then changes the state indicator in the tape communications segment's request block to indicate that operator action is pending, and returns.

Tape Mounting Protocol

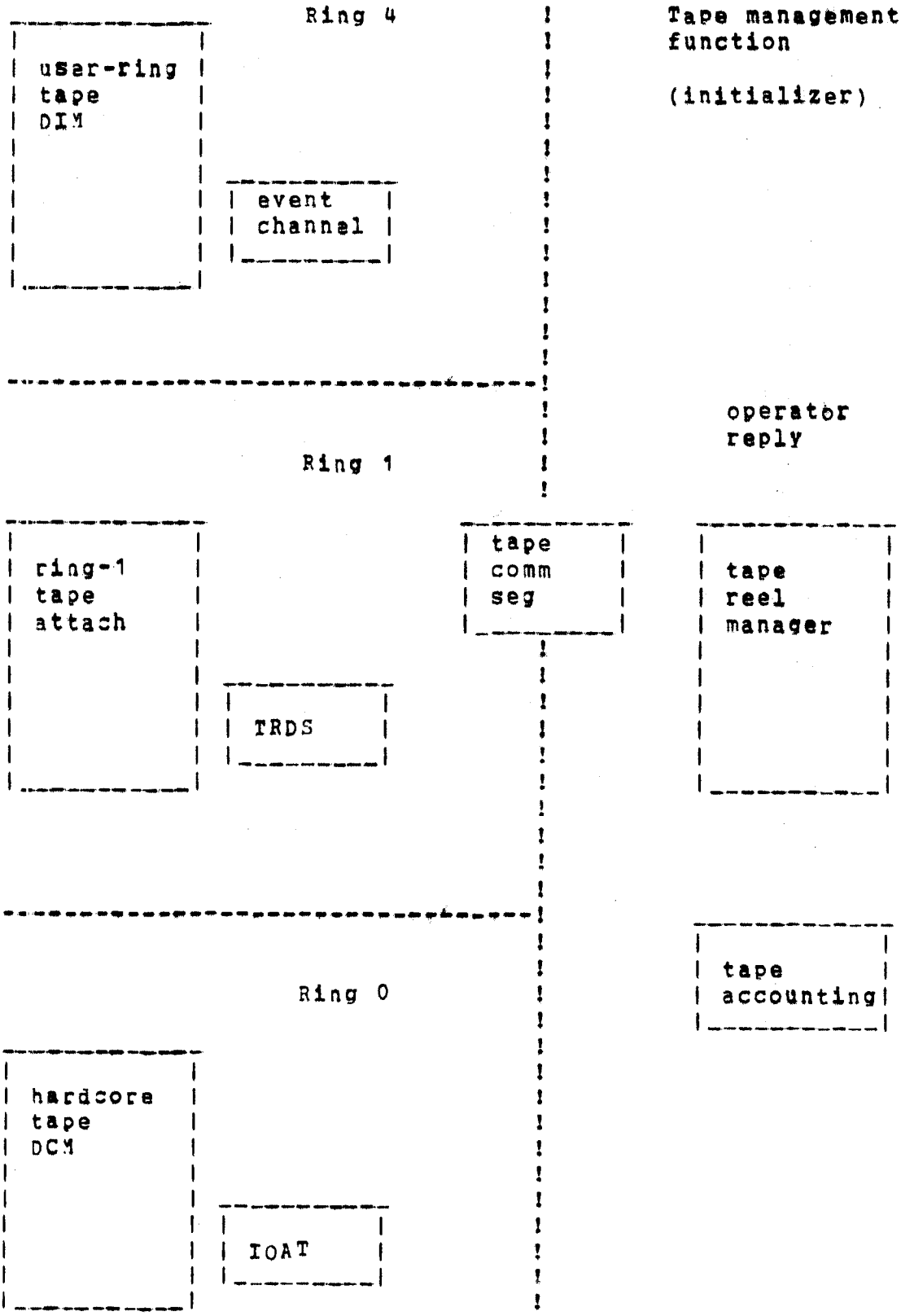

Figure 3 - operator replies to mount request

Tape Mounting Protocol

The operator locates and mounts the tape. (The current "special interrupt" provided by the tape controller when a tape is readied should be used for control purposes only, and not reported out of the hardcore.)

When the tape is mounted, the operator gives a command of the form

```
reply tape 19 ok qmn
```

where the "19" is the slot number in the tape communications segment, and "qmn" is the authentication code from the sticker on the tape. The tape manager verifies the authentication, and may reject the mount if the authentication code is invalid.

If the operator cannot locate the user's tape, or if the user is not supposed to use the tape, the operator may make a negative response, like

```
reply tape 19 notape
```

to indicate that the tape cannot be located.

The operator will be reminded of a request if he does not respond in 5 minutes. The operator will be able to request a list of all current drive assignments and all pending mount requests.

If the mount is successful, the tape manager informs the accounting package that the user is signed on to a tape.

After the operator replies, the tape manager sends a signal to the user process, over its event channel, to tell it that it may proceed.

Tape Mounting Protocol

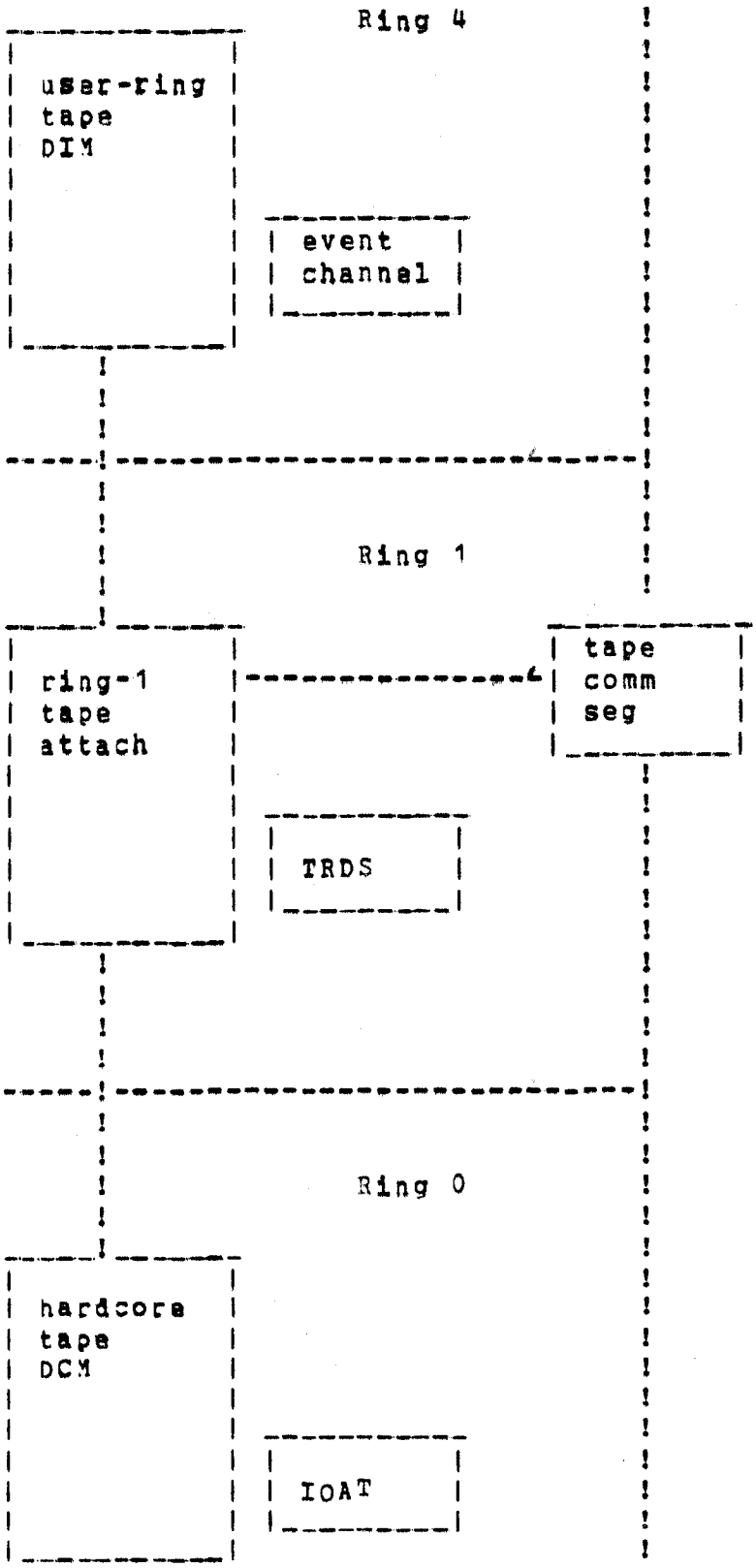

Figure 4 - User is awakened and checks mount

Tape Mounting Protocol

When the user process is awakened, it calls from the user ring into ring 1 again to check the mount.

The ring-1 program first examines the tape communication segment to make sure that the tape was successfully mounted.

If label checking is required, the ring-1 module then reads the tape label and checks that the tape is the one requested. If this test fails, the tape is unloaded and an error is returned to the user ring.

If the tape is mounted for writing, and the "not_yet_labeled" switch is on, a label in the appropriate format will be written on the tape.

If the label is correct, or if the tape is a non-standard tape with no label, the ring-1 module informs the hardcore tape DCM that the tape may be used from ring 4 directly.

Control is then returned to the user ring, which then calls directly into the hardcore for any tape reading or writing required, just as we do currently.

Miscellaneous topics

Once we have established this framework, the tape package can be extended to handle multi-file reels and multi-reel volumes in a straigntforward fashion, since the ring 1 program can discover the information necessary for such operations from the appropriate TRDS.

In addition, an "append" mode for tape files should be allowed, so that users can append to the end of a tape even if they do not have permission to rewrite data already on the tape. This ability can be implemented at some later date, after the basic mechanism is working; the key to this is the ability to instruct the DCM that backward repositioning of a tape is not permitted, and a simple addition to the label-checking routine which positions the tape at the end of the file at attach time.

The transition from the current mode of operation to the new method can be done in several steps, without affecting user or system operation. The new ring 1 programs and the tape manager can be checked out completely before we force the tape DIM's to use them. Once they are checked out, we can place the ring-1 pattern in service by simply changing the user-ring tape DIMs to call the new code instead of the old; the old DCM calls to attach and detach a tape can be removed at a later date, when we are sure that the new pattern is operational.

The initial version of the ring-1 attach code need not be delayed by the requirement that every reel in an installation have a TRDS created for it. The access-checking code can be constructed so that, if a TRDS does not exist for a tape reel, the operator is asked whether one should be created. The operator may then specify special attributes (say in the case of a backup tape), may simply let the request go through (for a normal user tape), or may make a negative response (in the case of an error).

We need not relabel all of our tapes, either, in order to get the new facility on the air. Indeed, we can even delay the production of the ring 1 label handling code if we wish, in order to get the mount-handling pattern on the air. This can be done by requiring authentication for all tapes mounted. IPC already has procedures for assigning authentication codes and numbering tape reels operating on the Batch system.

I have already coded the little program to return an authentication code given a reel identifier, just to see if it could be done.

Code to be written

The following is a short description of the programs necessary for this implementation.

1. Tape Manager
   This system control function can be invoked by a user request, to type out a message to the operator, or by an operator reply. It will manipulate the information in the tape communication segment.

2. Tape accounting
   This program will have sign-on and sign-off entry points, and may have an entry for an accounting update. It will charge the user for tape mounting, and for tape connect time.

3. Ring-1 tape attachment code.
   This program will respond to three user requests:
   a) Mount requests
   b) Mount-check requests (after operator mounts)
   c) Dismount requests
   The program accesses the TRDS segments for tapes, and makes calls to the hardcore tape DCM for channel and device assignment, label reading and writing, and channel unassignement. The program also accesses the tape manager's tape communications segment, and signals the tape manager when a mount request is made.

4. Changes to the tape DCM
   These changes will be made in several steps. Functions called by the ring-1 module will be written first. These include channel assignment; read, write, etc. for label manipulation; detachment; and an entry to indicate that the tape may be used by the user ring. Later, the tape_attach and tape_detach entries called from the user ring can be deleted. Further modifications, to insure that the user does not rewrite the tape label, can be made after the general scheme is in operation.

5. User-ring DIM changes
   These changes can be made to one user-ring DIM at a time, and the removal of the "old-fashioned" way of using tapes deferred until all of the new method has been shown to work. The changes consist of modifying the attach and detach calls to call ring 1 instead of the hardcore, and to cause the creation of an event channel and the blocking of the user process after the mount is requested.

6. Tape registration and administration
   Special programs which can generate a new TRDS and label a tape will be required. Other programs may be needed, to do things like prepare a list of all tapes registered. The 360 program which makes tape labels on the printer should also be

Code to be written

written for Multics someday.

7. BOS and bootload changes
   At some later time, we will modify BOS to examine the tape label of a tape it intended to use, to insure that the tape's accessibility indicator permits the kind of use intended.

8. Append mode
   Modifications to the tape DCM to allow an append-only mode for a tape reel may be done after the rest of the scheme is in operation. This will involve modifying the label-checking code to position the tape after the end of data instead of after the label, and will require the modifications to the DCM which prevent backward repositioning.

Two new gate segments will be added to the system to support tape operations. One will be callable by any user, and will support the user-ring DIM calls and any calls made by user utility programs. It will be called "trm_".

A privileged gate, "priv_trm_", will be required to support the operator and administrator commands.

Sample calls to each entry in the gates follow. The details of these calls will probably change during coding, but the list should give the flavor of what is intended.

1. Call made by user DIM to mount tape.

```
call trm_$mount (tape_id, evchn, message, tcsindex, ec);
```

   This call returns the index in tcs of the request block set up. The user process should block until signalled by the tape mount function.

2. Call made by user DIM to check mount.

```
call trm_$mount_check (tcsindex, ec);
```

   This call is made when the user is awakened by the mount function.

3. Call made when user DIM detaches tape.

```
call trm_$dismount (tcsindex, ec);
```

   Perhaps an index of 0 means all tapes for this process.

4. Call made to return contents of a request block in tcs.

```
call trm_$tape_status (tcsindex, p, ec);
```

   This fills in some block of words pointed to by p with the contents of the request block in tcs. Perhaps we should pass an area pointer and have the data allocated.

5. Calls pertaining to access control.

```
call trm_$acl_replace (tape_id, aclp, n, ec);
call trm_$acl_add (tape_id, aclp, n, ec);
call trm_$acl_add1 (tape_id, uname, mode, brackets, ec);
call trm_$acl_replace (tape_id, aclp, n, ec);
call trm_$acl_delete (tape_id, aclp, n, ec);
call trm_$acl_list (tape_id, aclp, n, uap, ec);
```

These calls are constructed on the pattern of the corresponding calls for segment ACL's. They should be made to look like the new access control primitives when documentation on these is published.

6. TRDS management.

```
call trm_$read_trds (tape_id, p, ec);
call trm_$write_trds (tape_id, p, ec);
```

These calls allow the user to inspect and modify tape attributes. Some information in the TRDS may not be available to the user, and some combinations of attributes may be illegal. This entry performs whatever checking is appropriate.

7. Reservation of resources.

```
call trm_$reserve (type, from, to, tcsindex, ec);
call trm_$unreserve (tcsindex, ec);
call trm_$list_reservations (tcsindex_array, n, ec);
```

These calls can be implemented later to support resource reservation.

8. Privileged calls

```
call priv_trm_$register (tape_id, owner_id, trdsp, ec);
call priv_trm_$unregister (tape_id, ec);
call priv_trm_$read_trds (tape_id, p, ec);
call priv_trm_$write_trds (tape_id, p, ec);
call priv_trm_$acl_list (tape_id, aclp, n, uap, ec);
call priv_trm_$acl_replace (tape_id, aclp, n, ec);
call priv_trm_$acl_add (tape_id, aclp, n, ec);
call priv_trm_$acl_delete (tape_id, aclp, n, ec);
call priv_trm_$list_tcs (tcsx, p, ec);
call priv_trm_$modify_tcs (tcsx, p, ec);
call priv_trm_$force_dismount (tcsx, ec);
```

The function of these calls is fairly obvious. Note that the tape mounting function does not have to be operating in ring 1, if appropriate entry points are provided in this privileged gate. The overhead involved should be investigated.

APPENDIX Operator replies

When the operator replies to the tape management function concerning a mount message, he types

```
reply tape NN KEY ARG
```

where "tape" is the identification of the system control function, used by the "reply" command to discover where to send the message; NN is the request index typed in the mount message; KEY is the keyword which indicates the success of the mount or the reason for failure; and ARG is an optional argument.

The following values are legal for KEY:

ok <auth>	mount successful, authentication is "auth"
notape	Tape reel cannot be located.
nopermit	User is not permitted to mount reel.
nosys	System or operations difficulty.
alrdy	Tape already mounted for another user.
redun	Tape already mounted for this user.

Name: trds

A Tape Reel Description Segment (trds) is created for each tape reel which is registered at an installation. The trds contains information about the attributes of the tape reel, and its access control list controls which users may access the tape reel. Each trds has a segment name of the form "reelid.trds"; all of the trds's are kept in a special system directory. The trds's are accessed by the administrative-ring tape mount and dismount programs when a user requests a tape mount.

Usage:

```
dcl 1 trds based aligned,
    2 flags,
      3 not_multics_standard bit (1) unal,
      3 no_label bit (1) unal,
      3 not_yet_labeled bit (1) unal,
      3 no_label_check bit (1) unal,
      3 no_authenticate bit (1) unal,
      3 small_reel bit (1) unal,
      3 seven_track bit (1) unal,
      3 not_high_density bit (1) unal,
      3 has_retention_date bit (1) unal,
      3 has_access_indicator bit (1) unal,

    2 version fixed bin,
    2 retention_date fixed bin (71),
    2 read_error_count fixed bin,
    2 write_error_count fixed bin,
    2 density fixed bin,
    2 accessibility_indicator char (1);
```

EXPLANATION OF VARIABLES

1. not_multics_Multics standard	is "1"b if the tape is not in Multics standard form (See MPM Reference Guide Section 5.3.)
2. no_label	is "1"b if the tape is unlabeled
3. not_yet_labeled	is "1"b if the label has not been written
4. no_label_check	is "1"b if the label is not to be checked
5. no_authenticate	is "1"b if the authentication is not required
6. small_reel	is "1"b if the reel is a small size

7. seven_track	is "1"b if the tape is seven-track
8. not_high_density	is "1"b if "density" value has meaning
9. has_retention_date	is "1"b if "retention_date" has meaning
10. has_access_indicator	is "1"b if the tape has a special value (i.e. not blank) for "accessibility_indicator" below.
11. version	is the version number of this declaration (currently 1)
12. retention_date	is the date before which the tape cannot be written
13. read_error_count	is the count of read errors on this tape
14. write_error_count	is the count of write errors on this tape
15. density	is the recording density in BPI
16. accessibility_indicator	is blank for normal user tapes. This indicator is written in the tape label, wehre it will be checked by some special programs which access tapes when the TRDS and its access control list are not available.

(END)

Name: tcs

The Tape Communication Segment (tcs) is an administrative-ring segment which is used for communication between the administrative-ring tape mount and dismount programs in a tape user's process and the tape mount and dismount request handling function in a system process. An entry is made in this table whenever a tape mount is requested. The entry is freed when the tape is dismounted.

Usage:

```
dcl 1 tcs based aligned,
    2 nents fixed bin,
    2 mount_proc bit (36),
    2 mount_chan fixed bin (71),
    2 n_mounted fixed bin,
    2 n_pending fixed bin,
    2 freep fixed bin,
    2 pad (25) fixed bin,
    2 array (100),
      3 fill (128) fixed bin;

dcl 1 tape_status_block based aligned,
    2 state fixed bin,
    2 procid bit (36),
    2 evchn fixed bin (71),
    2 uname char (24),
    2 uproj char (12),
    2 tape_reel_id char (16),
    2 drive_id fixed bin,
    2 channel_id fixed bin,
    2 devx fixed bin,
    2 authentication char (4),
    2 pdtep ptr,
    2 time_requested fixed bin (71),
    2 time_mounted fixed bin (71),
    2 time_dismount fixed bin (71),
    2 flags,
      3 not_multics_standard bit (1) unal,
      3 no_label bit (1) unal,
      3 not_yet_labeled bit (1) unal,
      3 no_label_check bit (1) unal,
      3 no_authenticate bit (1) unal,
      3 small_reel bit (1) unal,
      3 seven_track bit (1) unal,
      3 not_high_density bit (1) unal,
      3 has_retention_date bit (1) unal,
      3 has_access_indicator bit (1) unal,
    2 mount_message char (120) aligned,
    2 operator_reply char (16) aligned,
    2 tape_label char (80);
```

EXPLANATION OF VARIABLES

1. nents	is the number of entries in the table
2. mount_proc	is the process id of the mount-handling process
3. mount_chan	is the event channel used for mount requests
4. n_mounted	is the current number of reels mounted
5. n_pending	is the current number of mounts pending
6. freep	is the index of the beginning of the free chain
7. pad	is padding
8. array	is the array of per-request entries
9. fill	reserves storage for each table entry
10. tape_status_block	is the description of a single request
11. state	is the state of the request request
12. procid	is the user process id
13. evchn	is the user event channel
14. uname	is the user's person id
15. uproj	is the user's project id
16. tape_reel_id	is the tape reel identification
17. drive_id	is the drive id
18. channel_id	is the tape channel number
19. devx	is the device index, for hardcore calls
20. authentication	is the authentication code for the tape reel

21.	pdtep	is a pointer to the user's PDT entry, for accounting
22.	time_requested	is the time the mount request was made
23.	time_mounted	is the time the operator replied to the mount request
24.	time_dismount	is the dismount time
25.	flags	are flags, copied from the TRDS, describing the tape reel
26.	not_multics_standard	is "1"b if the tape is not in Multics standard format
27.	no_label	is "1"b if the tape is unlabeled
28.	not_yet_labeled	is "1"b if the tape is blank
29.	no_label_check	is "1"b if the tape label should not be checked
30.	no_authenticate	is "1"b if authentication is not required
31.	small_reel	is "1"b if the tape reel is not the standard size
32.	seven_track	is "1"b if the tape is seven-track
33.	not_high_density	is "1"b if the tape is not recorded at the highest density
34.	has_retention_date	is "1"b if tape is date protected
35.	has_access_indicator	is "1"b if special access control will be required for standalone uses
36.	mount_message	is the text of the user's mount request
37.	operator_reply	is the operator's reply
38.	tape_label	is the label as read from the tape

(END)

To: Distribution

From: T. H. Van Vleck

Date: 6/3/72

Subject: Multics Tape Labels and Tape Identification

This memorandum describes a tape labeling scheme and a plan for tape identification which will support the user tape-handling facility and the automatic volume recognition features described in a companion memorandum.

The labeling scheme proposed here is influenced by the American National Standard, "Magnetic Tape Labels for Information Interchange", (X3.27-1969), published by the American National Standards Institute (ANSI).

The tape identification scheme and the labeling scheme have been designed so that they can be added to Multics without changing the appearance of the system to current users of tapes, or requiring users to rewrite tapes now in use on Multics.

Tape identification

As described in the companion memo on User Tapes, each tape at a Multics installation will have a corresponding Tape Reel Description Segment (trds), which will be managed by special programs in ring 1. The trds will be a segment in a special directory, named the same at the tape reel. This "name" for the reel will usually be a number, like

3017

When the user specifies the reel identifier in the "description" argument in a call to ios_$attach, the tape DIM will call into ring 1 and locate the appropriate trds. The ring-1 software will check the user's access to the tape reel, and extract from the trds various attributes needed to accomplish the mounting and verification of the tape reel.

In some cases, however, the user may wish to inform the system of attributes of the tape reel which are not specified in the trds, perhaps because the reel is not yet registered at the installation or because he wishes to over-ride attributes set in the trds. In this case, the "description" argument may consist of a reel identifier, followed by a list of qualifiers separated by commas. For example, a user might specify

3017,9TRACK

or

scratch,NL,comment="Carry tape for batch"

The maximum length of the description argument is 168 characters.

The legal qualifiers are:

NO_LABEL, NL	No tape label
9TRACK, 9T	9-track tape
7TRACK, 7T	7-track tape
ANSI	ANSI labels
DENSITY=, DEN=	Recording density
NO_DATE, ND	Bypass checking of retention date
COMMENT=, COMM=, C=	Type comment at mount request

Although upper case letters are shown, either upper or lower case will be allowed. Other qualifiers may be defined from time to time.

Tape Labels

Multics standard format tapes will continue to be the principal tape format for tapes intended to be read only by Multics. The many features of this format make it particularly useful for recording information which will be read again by the system. Many of these same features, however, make Multics standard format tapes difficult to read with other operating systems. The ANSI labeling conventions have been designed to permit easy interchange of tapes between systems. Multics will therefore support the reading and writing of tapes in ANSI-compatible format, as a supplement to the Multics standard format.

It will still be possible to read and write tapes in "non-standard" formats, which do not have labels recognizable by t the ring-1 tape mounting code; these tapes, however, cannot benefit from the automatic volume recognition code which provides physical security for the data on a reel, and will require operator authentication whenever they are mounted.

Briefly, the ANSI standard provides for labels which identify the tape volume and mark the end of data on the volume, and for file labels which mark the beginning and end of files. These labels are all 80-character records with specified formats, and are separated from the file data records by end-of-file marks. Each tape volume begins with a volume label, whose first four characters are "VOL1". Each file is preceded by a file header label, whose first four characters are "HDR1". Each file is followed by an end-of-file label, whose first four characters are "EOF1". In addition, the standard allows for files which begin on one volume and end on another; if a file is broken in the middle, an "EOV1" label is recorded to mark the end of the volume. The standard also allows for optional labels: user volume labels (UVLa, ...), user file header and trailer labels (UHLa, ..., UTLa, ...); and permits the operating system to record optional labels for its own use (HDR2, ..., EOF2, ..., EOV2, ...).

The simplest form of a tape which conforms to the ANSI standard looks like this:

VOL1 HDR1 * file records * EOF1 * *

where "*" represents a file mark. The standard requires that, for general information interchange, the maximum length of a physical record on the tape should be 2048 characters.

The initial support for ANSI standard tapes within Multics will allow only VOL1, HDR1, and EOF1 labels. Multi-file volumes and multi-volume files will not be supported until later. When a tape is mounted for writing; the VOL1 and HDR1 labels and a tape mark will be written by ring 1. When the tape is dismounted, an EOF1 label will be written. Later, the support code will be

modified to handle multi-volume files, and then multi-file volumes, but the details of that implementation are outside the scope of this memo. Optional labels will be deferred for some time, unless an unforeseeen need for them arises. Users can create such tapes, of course; by specifying NO_LABEL and writing their own code.

The format of the tape labels is given in an appendix.

The Multics standard tape format (see MPM 5.3) is incompatible with the ANSI standard, especially in the use of end-of-file marks. The Multics standard tape header is also recorded in binary mode, and the record is 272 words long, in a special format, instead of the 80 characters required by the standard.

In order to avoid making a change in label formats which would not be downwards compatible, the approach chosen is to add to the Multics label, in unused positions, the standard ANSI VOL1 label. This label will be contained in words 17-36 of the Multics label.

Tapes previously written by Multics will, of course, be acceptable to the system after the new label-handling code is installed, but the system will not be able to perform automatic volume recognition for these tapes; so authentication will be required; and the system will not be able to protect these tapes from unauthorized access by stand-alone programs, since the accessibility indicator will not be available in the tape label.

The following is the pl1 declaration for the VOL1 label:

```
dcl 1 vol1_label based,
    2 label_identifier char (3),        /* "VOL" */
    2 label_number char (1),            /* "1" */
    2 vol_ser_no char (6),              /* reel id */
    2 accessibility char (1),           /* usually blank */
    2 reserved1 char (20),
    2 reserved2 char (6),
    2 owner char (14),                  /* project id */
    2 reserved3 char (28),
    2 label_std_level char (1);         /* "1" */
```

The following is the declaration of the HDR1 label:

```
dcl 1 hdr1_label based,
    2 label_identifier char (3),        /* "HDR" */
    2 label_number char (1),            /* "1" */
    2 file_identifier char (17),        /* file name */
    2 set_identification char (6),      /* reel id */
    2 file_section_number char (4),     /* "1" */
    2 file_sequence_number char (4),    /* "1" */
```

```
    2 generation_number char (4),        /* "0001" */
    2 generation_version_number char (2), /* "00" */
    2 creation_date char (6);            /* BYYDDD */
    2 expiration_date char (6),          /* BYYDDD */
    2 accessibility char (1);            /* as above */
    2 block_count char (6),              /* "000000" */
    2 system_code char (13),             /* installation id */
    2 reserved char (7);
```

The EOF1 label has the same format as the HDR1 label, except that "label_identifier" is "EOF"; and "block_count" is a number, expressed in decimal character form, which is the count of pyhsical records in the file.

The reader is urged to consult the ANSI standard document for more information concerning the label formats and contents.

syscon_msgs_dscr	
ipc_channel	So that process controlling this
current_process_id	virtual console can be wakened.
flags	Possibly not needed
first_message	Index of head of message chain.
last message	Index of tail of message chain.
first_free_buffer	Index of head of free storage list.
last_assigned_buffer	Number of blocks currently in use or on free storage list.
messages	The array of message blocks.
next_message	foreward pointer.
time_sent	
from_user	
from_proj	
flags	
continue	"1"b if message continues in next block.
message_body	

Message Routing Table

This table will be used by the attach entry of the message routing dim (mrd_) to match source (e.g. I/O Daemon, answering service) -- stream (e.g. "user_i/o","error_output") combinations with virtual consoles to which messages are to be directed.

```
dcl 1 mess_route_table aligned based,
        2 lock bit(36),
        2 no_of_sources fixed bin,
        2 source_entry(16) aligned,
            3 source_name char(32),
            3 no_of_streams fixed bin,
            3 stream_entry(16) aligned,
                4 stream_name char(32),
                4 virt_cons(8) char(32);
```

mess_route_table	
lock	
no_of_sources	Number of source rows in this matrix.
source_entry	The source rows
source_name	
no_of_streams	Number of stream columns that have entries in this row.
stream_entry	The virtual consoles for this element of the matrix.
stream_name	
virt_cons	Virtual consoles to which messages should go when originating from this source over this stream.

Virtual Console Table

Associates virtual consoles with physical devices and/or log segments.

```
dcl 1 virt_cons_table aligned based,
        2 lock bit(36),
        2 no_of_entries fixed bin,
        2 entry(32) aligned,
            3 virt_name char(32),
            3 no_of_dest fixed bin,
            3 dest_data(8) aligned,
                4 type fixed bin,
                4 dest_name char(32),
                4 tty_index fixed bin,
                4 seg_ptr ptr;
```

virt_cons_table	
lock	
no_of_entries	
entry	
virt_name	Symbolic name of virtual console
no_of_dest	Number of physical destinations for messages to this virtual console.
dest_data	Where the message is to be sent
type	1 for tty, 2 for segment
dest_name	Symbolic name of physical destination
tty_index	If tty, ref no for calls to hardcore
seg_ptr	if segment, pointer to it.

Appendix B
Operator Commands to Modify Console Table & Routing Table

Commands to Modify the Virtual Console Table

The following operator commands will be provided to give dynamic control over the destination of messages from the system to operations.

Command: define

Effect: Create a virtual console or add physical destinations to a previously existing virtual console.

Usage: define VCONS TYPE1 PHYS_NAME1 ... TYPEn PHYS_NAMEn

The virtual console VCONS will be defined and given the physical destinations PHYS_NAME1 through PHYS_NAMEn. If VCONS is already defined, the destinations PHYS_NAME1 through PHYS_NAMEn will be added to it. The TYPE arguments may have value "file_" or "tw_" to indicate whether the destination is a segment or a console. (This will not be necessary if there is a way for system control to distinguish a console name from a segment name.)

Command: deldest

Effect: Delete a virtual console or remove physical destinations from a virtual console.

Usage: deldest VCONS PHYS_NAME1 ... PHYS_NAMEn

Physical destinations PHYS_NAME1 through PHYS_NAMEn will be removed from the destination list for VCONS. If no PHYS_NAME arguments are given, the entire virtual console will be deleted. (Possibly this should ask if operator wants to do this before proceeding.) TYPE arguments are as described in the description of define above.

Command: redirect

Effect: Change physical destination of messages to a virtual console.

Usage: redirect VCONS PHYS_NAME1 TYPE PHYS_NAME2

This will be equivalent to:
deldest VCONS PHYS_NAME1

```
define VCONS TYPE PHYS_NAME2
```

except that if no PHYS_NAME arguments are given an error message will appear.

Example: Suppose that the initializer is using TTY196 to print daemon messages (i.e. virtual console DaemonErr has one physical destination, TTY196) and that TTY195 is idle. If, for instance, the ribbon should break on TTY196, the operator could go to any console and type:

```
redirect DaemonErr TTY196 tw_ TTY195
```

Messages sent to DaemonErr would now appear on TTY195.

If no console were free when TTY196 became unusable, the operator could, again from any console, type

```
redirect DaemonErr TTY196 file_ DErr_temp_
```

to save subsequent messages until TTY196 is repaired or until another console becomes free.

Commands to Modify the Routing Table

The following commands will be provided to give dynamic control over which virtual consoles recieve messages from which sources.

Command: route

Effect: Associate a set of virtual consoles with an I/O stream of a message source.

Usage: route SOURCE STREAM VCONS1 ... VCONSn

Subsequently when system control function SOURCE writes on I/O stream STREAM the message will be routed to virtual consoles VCONS1 through VCONSn. If a routing entry already exists for SOURCE and STREAM, VCONS1 through VCONSn will be added to the list of virtual consoles.

Command: deroute

Effect: Remove a set of virtual consoles from the routing list of an I/O stream of a source.

Usage: deroute SOURCE STREAM VCONS1 ... VCONSn

Subsequently when system control function SOURCE writes on I/O stream STREAM the message will no longer be routed to virtual consoles VCONS1 through VCONSn. If the VCONS arguments are absent, all routings for this stream-source combination will be removed.

Command: reroute

Effect: Change virtual consoles for a stream-source combination.

Usage: reroute SOURCE STREAM VCONS11 VCONS12 ... VCONSn1 VCONSn2

Equivalent to:

```
deroute SOURCE STREAM VCONS11 ... VCONSn1
route   SOURCE STREAM VCONS12 ... VCONSn2
```

except that if no VCONS arguments are given, an error message will appear.

Examples: The route command would normally be given as part of the procedure for bringing up a daemon or other system control function. The deroute command would normally be given when

bringing such a function down.

For instance, before logging in the Backup Daemon the operator would type

```
route bk user_i/o backup_log_
route bk error_output DaemonErr backup_log_
```

to initialize the routing for messages from Backup.

If it should be discovered that Backup is encountering errors of a nature that require immediate operator attention, the operator could type

```
reroute bk error_output DaemonErr master
```

or

```
route bk error_output master
```

The former would cause the message coordinator to stop sending Backup error messages to the DaemonErr virtual console and route them instead to the master console. The latter would cause such messages to be sent to the master console as well as to DaemonErr. Both, assuming initialization as in the example above, would leave the routing of messages to backup_log_ undisturbed.

After the Backup Daemon is logged out, the operator might type

```
deroute bk user_i/o
deroute bk error_output
```

(Note: It is probably possible and desirable to keep a semi-permanent routing table, using these commands only when new or special conditions call for a change.)

Appendix C
Sundry Other Commands

Command: login

Effect: Cause login of a System Control Function

Usage: login NAME PROJ SOURCE_ID

This command is used to create a process for the system control function NAME (e.g. Dumper,IO) which is given the project id PROJ (e.g. SysDaemon). SOURCE_ID tells the function process what to call itself when sending messages (e.g. bk,io,as,tape).

Example: To start an IO Daemon running type

login IO SysDaemon io

Note that a special bit must be set in the user's pdt entry, in order for him to be logged in by the operator. Normally, this bit will be set only for daemon processes.

Command: reply

Effect: respond to a request for input from a System Control Function (SCF)

Usage: reply SOURCE_ID LINE

The rest of the line after SOURCE_ID will be taken as a message and sent to the SCF whose source id is given as the first argument. This command is used by the operator whenever a SCF requires information from him. The SCF will indicate its need for information by including the sentinel "-->" in the message sent to the operator. To get the attention of the message coordinator long enough to type his reply, the operator should hit the carriage return key before typing "reply". Examples are found on page 5.