TO: MOSN Distribution

FROM: T. H. Van Vleck, Dennis Capps

DATE: 04/17/73

SUBJECT: Changes to Initializer for Message Coordinator

A new facility called the Message Coordinator has been added to the initializer. It allows the initializer to run multiple terminal channels, and lets the daemons run without terminals of their own, sending their messages to the initializer for disposition.

The system also has the ability to login the daemons automatically or on operator request.

Many miscellaneous improvements have been made to the operation of the initializer and to the messages it produces.

This document has two parts: the first part describes the external changes made to the operation of the initializer, from the point of view of the operator. The second part of this document goes into more detail about the internal changes made to the initializer programs, from the point of view of the system programmer.

Changes to Initializer Messages

Several of the messages typed by the initializer have been shortened, to make the system bootload sequence faster.

The first message typed by the system at bootload will be a message of the form

```
   Multics SYSID - MM/DD/YY HHMM.S est DAY
```

giving the system ID and the date and time.

The system will then request one of the ring-1 commands by typing

```
   Command:
```

The legal commands are still "reload," "startup," "multics," "standard," "bos," and "shutdown," but this list will not be typed unless the operator gives an incorrect command.

When the operator types "startup," "multics," or "standard" no comment will be made during the crossing into ring 4. Instead of "Command:" in ring 4, the initializer merely types

```
   R
```

to indicate that another command may be issued.

The time which prefixes initializer messages has been shortened to four digits only. The time will be followed by an abbreviation for the "source" from which the message came. For example,

```
   1322 as  LOGIN 2741  501  tty194 Smith.Multics
```

is the form of a login message. The "as" above identifies the message as coming from the answering service source.

System Startup

A special list of commands can be set up by the system programmers to be executed when the answering service is started. These commands are kept in "system_start_up.ec", and come in two sections: those executed before the answering service is started, and those executed just after the answering service comes up.

Normally, the system_start_up.ec will turn on the message coordinator before running the answering service, and will automatically log in the daemons immediately after the answering service is ready. If the initializer is to operate more than one terminal channel, the additional channels will be attached automatically at this time.

The normal mode of operation for the system will be to use the system master console (BOS typewriter) as the first initializer console, and to automatically add one or more terminal channels to the initializer during startup.

The startup sequence on the system console will look like this:

```
o) BOOT
s) MULTICS 19.1 - 02/14/72 1949.3 EST WED
s) COMMAND:
o) STARTUP
s) R
```

Lines typed by the system are indicated (in this document only) by "s)" and lines typed by the operator are indicated by "o)". After the "R", the system console will not be used for most output except for the usual disk error, tape mount, programmer, and hardcore error messages.

If channel "tty192" is the terminal channel which will be used by the initializer for regular messages, it will be hard-wired to the system or the operator will have dialed it up before typing BOOT, as usual. The output on this console will look like this:

```
s) tty192 attached by system control.
s) 1950 as  Multics 18.6; answering service 6.12
s) 1951 as  LOGIN  Daemon io1     io1     IO.SysDaemon
s) 1951 as  LOGIN  Daemon bk      bk      Backup.SysDaemon
s) 1952 io1 IO DAEMON READY TO START
s) -->  io1
s) 1952 bk  r 1952 4.801 25+99
s) -->  bk
```

The lines beginning with "-->" indicate that the source wants input. They are called "sentinels." To input a line to the daemon, the operator uses the "reply" command.

```
   o) reply io1 init prtdim prta34
   s) R
   o) reply bk start_dump sys_dirs xyz
   s) R
   s) 1953 io1 act_ctl_: IO Daemon accounting initialization.
   s) 1953 io1 Is this the first or second IO daemon?
   s) --> io1
   s) 1953 bk  Enter primary dump-tape label:
   s) --> bk
   o) reply bk IC-75
   s) R
   o) reply io1 first
   s) R
   s) 1954 io1 Type "yes" if prtdim prta34 is correct:
   s) --> io1
```

and so forth. The example above shows how the system intermixes output lines from all of the sources on a single console, and how the operator replies to a request for input from a source.

If more than one terminal channel is connected to the initializer, the output from the various sources (daemon processes, etc.) can be routed to divide the work between several consoles. For example, all the daemons could be handled by one terminal, and the answering service could use another. Or, if all the terminals are broken, the system can be run completely from the system console (but this setup would be bad for the system, since whenever the operator is typing in or the system is typing out on the system console, the entire Multics system is hung; and on a two-cpu configuration, the system may crash if a ring-zero message has to wait too long for the master console.)

All terminals attached to the initializer may input initializer commands. (It is possible to restrict a terminal to only certain commands, but this will not be done at first.)

It is sometimes difficult to input an operator command between output messages on an initializer terminal, because the system keeps interrupting. If the operator types an empty line on an initializer terminal, the system will respond

```
   OPER:
```

and suspend output on that terminal channel. When the operator completes his command, the output will be restarted, with no messages lost. If the operator does not finish his command in one minute, the output will be restarted. (This feature does not work for the bootload console.)

Admin mode and editing of the message of the day can be done from any initializer terminal; but only one terminal can be operating in this mode at a time.

Terminals may also be added to the Initializer dynamically. To do this, the operator dials a terminal into Multics as if he were going to log in, but instead of typing "login", he issues a "dial" command:

```
   s) Multics 18-6: MIT, Cambridge, Mass.
   s) Load = 41.0 out of 50.0 units: users = 41
   o) dial system
```

An optional identifier may be typed after "system," to indicate which terminal has dialed up, or to serve as a password to insure that the command has been issued by an authorized operator. The dialed terminal will then get a message of the form

```
   s) TTY37 405 chn tty196 dialed to Initializer.
```

Also, on the initializer console, a message stating that the terminal has dialed up will be printed.

```
   s) 1137 as  dial_ctl_: TTY37 405 tty196 dialed to Initializer.
```

The operator should then issue a series of commands to accept the terminal channel and to route output to it.

```
   o) accept tty196
   s) R
   o) define vc2 tty tty196
   s) R
   o) route dump user_i/o vc2
   s) R
```

The response on the dialed terminal will be a message saying that the initializer has attached the channel:

```
   s) tty196 attached by system control.
```

followed by whatever messages are routed to the terminal channel.

When the operator is finished with a dialed terminal, or if a curious user tries to dial the initializer without permission, the operator may disconnect the channel from the initializer and make it available for dialups again by typing a "drop" command:

```
   o) drop tty196
   s) R
```

The response on the dialed terminal will be a message like "please reissue dial command," and at this point the terminal may be re-dialed, or used for regular logins, or hung up.

Error Messages

If the operator types a command with any of the illegal characters ";" or "[" in it, the system will respond with the message

```
   syntax error
```

and ignore the command.

If a terminal is restricted to issuing only certain commands and attempts to issue a command it is not allowed, the system will respond

```
   privilege error
```

and ignore the command.

If a terminal attempts to enter admin mode, reconfigure the system, or edit the message of the day, and some other terminal is already doing either of these operations, the system will respond

```
   function busy.
```

and ignore the command.

Other error messages will be found in MOSN-XXX, "Initializer Messages".

Difficulties

Occasionally, the message coordinator may stop operating due to an ITT overflow, a device read error, or other system problem. Try issuing the "reset" command in such a case. This will attempt to restart all channels. Channel restart is also attempted if system control encounters any fault.

Daemon Operation

The daemon processes have not been changed in any way by the installation of the message coordinator. They type the same messages, and expect the same input: their input and output is, however, passed through the initializer on its way to and from the terminal channel.

To cause a daemon to be logged in from the initializer, the operator may type

```
   login PersonId ProjectId sourceId
```

The daemon will then log in, and attach its input and output to the initializer, as a source with name "sourceid." For example, a second IO daemon can be logged in by typing

```
   o) login IO SysDaemon io2
   s) R
   s) 1721 as  LOGIN Daemon  io2 io2  IO.SysDaemon
```

or a complete dump may be started by typing

```
   o) login Dumper SysDaemon dump
   s) R
   s) 1721 as  LOGIN Daemon  dump dump  Dumper.SysDaemon
```

To cause a daemon to log out, the operator issues the "logout" command from the initializer, giving the person and project id of the daemon. Thus, to log out the dumper, the operator types

```
   o) logout Dumper SysDaemon dump
   s) R
   s) 2231 as  LOGOUT Daemon dump dump Dumper.SysDaemon 12:11
```

To cause all daemon processes to be logged out, when the system is being shut down, the operator types

```
   logout * * *
```

Occasionally, the operator needs to send a "quit" to a daemon process. A special command is required because the ATTN or INTERRUPT button on an initializer terminal is connected to the initializer, not to the daemon, and will be ignored by system control. To send a quit to a daemon, the operator must type

```
   quit sourceid
```

and the daemon will accept the quit after it has been passed from the initializer.

Interface Improvements and Bugs Fixed

The fault message typed on an error in the initializer is no longer missing the first character of every line.

Faults during a reload are no longer fatal.

The information typed when a fault occurred in the initializer process used to be slightly garbled: this bug has been fixed, and furthermore once the answering service is running the fault information will be written into a dump file rather than typed on the console.

QUIT is now allowed in the initializer until the answering service or the message coordinator has been started.

On the development machine, if the admin mode password is "*", the password will not be requested and admin mode will be entered immediately.

The "entering admin mode" message has been removed. The ready message from the listener is sufficient confirmation that admin mode has been entered.

The rest of this document is intended for system programmers who need to know how the message coordinator operates, and what the system commands are which affect the use of initializer terminal channels.

Overview

There are three major components to the message coordinator implementation: a major revision of system_control_ to handle multiple input consoles, a new program called the message coordinator which handles multiple ooutput consoles and message routing, and a new device interface module called the "message routing DIM" ("mrd_"), which does input and output to special data segments instead of on a terminal channel.

The message coordinator relies heavily on the interprocess communication event-call facility. The following kinds of event call channels are used in the initializer process:

Event	Procedure Called
tty read completion	system_control_$tty_aught dialup_
message from daemon	message_coordr_$router
output queued for tty	message_coordr_$typer_out
alarm wakeup	dialup_ system_control_$tty_aught act_ctl_ absentee_utility_
daemon attach via mrd_	message_coordr_$protocol
administrator signal	up_sysctl_
user process signal	dialup_ dial_ctl_ absentee
administrator command	system_control_$admin_com_nlr

The first console attached by system_control_ is handled differently from the rest of the initializer terminal channels. Output to it is written on the stream "master_i/o", and input from it is read directly through ios_. The initializer process's single block point is in the DIM which handles this first console. Additional terminal channels are handled by event-call procedures (tty_aught for input, typer_out for output) which perform I/O directly on the channel without the use of ios_.

The main data bases which are used by the message coordinator are:

mc_anstbl	one entry per device channel
MRT	message routing table
vcons_tab	virtual console table
mc_message	incoming messages for initializer
xxx_message	input messages for other sources
ttyxxx_queue	queued output messages for devices

These tables are all completely reconstructed every time the message coordinator is started. All of these segments are kept in >system_control_dir. Their ring brackets should be 4,4,4 and access should be RW for the initializer and for the daemon processes, and null for everybody else.

There are several commands which system programmers may use to find out the status of the message coordinator tables.

dump_mrt	dump Message Routing Table
dump_vct	dump Virtual Console Table
dump_devq	dump device queue
dump_msg	dump source message segment
restart_chn	attempt to restart device queue

Consult the writeups of these commands in the SPS for details of their use.

System startup deck implementation

The segment "system_start_up.ec" located in the directory >system_control_dir is used to cause the whole message coordinator facility to be invoked automatically when the system is started up. This exec_com is invoked twice, once before the answering service is initialized and once after. Its first argument is "part1" on the first call and "part2" on the second call.

The following is an example of a system_start_up.ec:

```
& MIT system startup deck
&
&command_line off
&goto &1
&
&label part1
system_control_$command mc
admin$accept tty192
admin$redefine default_vcons otw_ tty tty192
admin$define scc tty tty192
admin$define asc tty tty192
admin$define tpc tty tty192
admin$reroute sc mc_i/o scc
admin$reroute as severity(1 2 3) asc
admin$reroute tape tape tpc
admin$define ioc tty tty192
admin$define bkc tty tty192
admin$route io1 user_i/o ioc
admin$route io2 user_i/o ioc
admin$route bk user_i/o bkc
admin$route (cd1 cd2) user_i/o bkc admin$define bkc log iolog
&quit
&
&label part2
system_control_$command login IO SysDaemon io1
system_control_$command login Backup SysDaemon bk
system_control_$command reply io1 init prtdim prta34
system_control_$command reply io1 first
system_control_$command reply io1 yes
system_control_$command reply io1 start
&quit
&
& end
```

The example given defines a number of items:

DESTINATIONS

otw_	Operator's console (BOS console)
tty192	Hard-wired terminal channel
iolog	Log file (copy of all IO output)

VIRTUAL CONSOLES

*	Emergency virtual console
default_vcons	Default virtual console
scc	System control console
asc	Answering service console
ioc	IO Daemon console
bkc	Backup console
tpc	Tape message console

SOURCES

as	Answering service (initializer process)
sc	System control (initializer process)
tape	Tape request handler (initializer process)
io1	First IO Daemon
io2	Second IO Daemon
bk	Backup Daemon
cd1	Complete Dump (Dumper.SysDaemon)
cd2	Complete Dump (second Dumper)

Operator Commands for Message Coordinator

Mary new commands have been added to the initializer to support the message coordinator. They divide into six classes:

1. Commands dealing with device channels: accept, substty, and drop.

2. Commands dealing with virtual consoles: define, redefine, and undefine.

3. Commands dealing with routing: route, reroute, and deroute.

4. Commands dealing with sources: reply and quit.

5. Commands dealing with daemons: login and logout.

6. Miscellaneous commands: mc.

All of these commands except "reply" and "mc" are supportec in the answering service program "admin." "admin" obtains and checks arguments, and calls either "message_coordr_" (for 1-4 above) or "daemon_user_manager_" (for 5).

An entry point has been added to system_control_ so that the various exec_com's and administrative commands may execute system control operator commands. Calling

```
     system_control_$command reply x hello
```

will cause the system control command "reply x hello" tc be executed.

Command: accept

Usage: accept a device channel and connect it to initializer

Format: accept TTYXXX -RESTRICT-

This command is used to pick up a terminal channel and add it to the initializer's device complement. If RESTRICT is not specified, or if it is "full", the device will be able to issue all operator commands. RESTRICT may also be

none	no commands allowed
reply	only "reply" is allowed
query	only "who" and "hmu" are allowed

If the channel appears in the answer_table, then it must either have state 0 (not in lines file) or be dialed to the initializer.

Response: TTYXXX attached by system control.

Command: substty

Usage: swap one device for another

Format: substty TTYXXX TTYZZZ

This command causes TTYZZZ to be attached and TTYXXX to be dropped. All output queued for TTYXXX will be placed in the queue for TTYZZZ.

Response: TTYZZZ attached by system control.

same message as for "drop" on TTYXXX

Command: drop

Usage: remove a device channel from system control

Format: drop TTYXXX

This command causes a device channel to be removed from the message coordinator. Any pending output for the channel is lost. If the channel was dialed to the initializer, it is disconnected.

Response: please reissue dial command

(only if channel was dialed)

Command: define

Usage: associate virtual console with channel

Format: define VCONS TYPE DEST

This command creates a new virtual console if VCONS does not already exist. The destination DEST is then added to the destination list for VCONS. A virtual console may have up to 8 destinations. If TYPE is "tty" then DEST must be a channel ID which has been accepted previously. If TYPE is "log" then DEST is the name of a log file to which messages will be added as they are sent to VCONS. (These logs can be printed with "print_log".) If TYPE is "sink" then DEST can be any name: output sent to a sink vanishes.

Command: redefine

Usage: interchange one destination with another

Format: redefine VCONS OLD_DEST NEW_TYPE NEW_DEST

This command removes one destination from a virtual console and adds another. NEW_TYPE and NEW_DEST are as above. If OLD_DEST is a device channel which currently has output queued for it, no more output will be queued but all the queued output will be printed.

Command: undefine

Usage: remove destination from virtual console

Format: undefine VCONS OLD_DEST

This command removes a destination from a virtual console. If VCONS is left with no destinations and output is routed to it, the output will be typed on the bootload console.

Command: route

Usage: direct output from a source to virtual consoles

Format: route SOURCE STREAM VCONS

This command sends the output from the source SCURCE written on the stream STREAM to the virtual console VCONS. If no entry for SOURCE, or for STREAM under SOURCE, exists in the MRT, one will be created. There may be up to 16 sources. Each source may have up to 8 streams, and each stream may have up to 8 virtual consoles. VCONS must have been previously defined. It is acced to the virtual console list for STREAM.

Command: reroute

Usage: change virtual console for a stream

Format: reroute SOURCE STREAM OLD_VCONS NEW_VCONS

This command alters the MRT entry for SOURCE and STREAM to charge a virtual console entry.

Commanc: deroute

Usage: remove virtual console from stream

Format: deroute SOURCE STREAM OLD_VCONS

This command removes a virtual console from the output list for a given SCURCE and STREAM. If the stream is left with no virtual consoles, output will be sent to the cefault virtual corscle, which is usually defined to the system master console.

Command: reply, r

Usage: send input line to a source

Format: reply SOURCE REST OF LINE

This command sends an input line to the given source. The input line is placed in the segment "SOURCE_message" and a wakeup sent to the source. When the source calls to read via mrd_, it will extract the message from the segment.

Command: quit

Usage: send quit to a source process

Format: quit SOURCE

This command sets a flag in the segment "mc_message" indicating that a quit has been sent. If the source process has called

```
     ios_$order (STREAM, "quit_enable", null, status);
```

on one or more of its streams attached through mrd_, the message routing DIM will check every ten seconds for the quit flag, and signal quit if the flag is on.

Command: login

Usage: operator login of daemon

Format: login PERSON PROJECT SOURCE

This command causes the login of a daemon process at operator request. The PERSON.PROJECT must be a registered user with the "daemon" attribute. The outer module for the process being created is forced to be "mrd_".

Command: logout

Usage: operator logout of daemon

Format: logout PERSON PROJECT SOURCE

This command causes the logout of a daemon process at operator request. If PERSON, PROJECT, or SOURCE is "*", all users which match are logged out. SOURCE, or SOURCE and PROJECT, may be omitted, and are then assumed to be "*".

Command: mc

Usage: start message coordinator

Format: mc

This command causes system control to start the message coordinator.

Several additioral improvements to the message coordinator and initializer have been deferred for the present.

1. System control should be modified so that the special characters "[" and ";" can be sent to a daemon.

2. Some way of handling arguments to "startup" so that the operator car skip or modify parts of system_start_up.ec should be invented.

3. For installations which have initializer terminals in multiple locations, some indication that a command has been typed at another terminal should be provided. Currently, is is possible for a terinal to issue a command which causes output on some other terminal, and the output terminal has no indication of what caused the ouput.

4. Along with the above facility, installations with multiple, separated initializer terminals may wish to have an "intercom" feature. Such a facility can be provided now, in an inconvenient fashion.

5. An initializer command which lists the current device complement and routing should be provided. This command should produce output in the form of "route", "define", and so on so that its output can be used to take a temporary change to the standard routing, resulting from initializer message coordinator commands after startup, and make it permanent in system_start_up.ec.

6. A facility for the automatic dprinting of log files, either at a given time interval or whenever the log becomes full, should be worked out.