TO: MSPM Distribution
FROM: H. J. Hebert
SUBJECT: BQ.1.01
DATE: 05/10/68

This reissue of BQ.1.01 represents a minor revision of the System Control Process. Some of the changes are as follows:

(1) The overseer and user control processes are now created only once and by system control.

(2) The communication mechanisms between the system operator and system control are more explicitly stated.

(3) The function of sys_control$response is now handled by sys_control2$reflector.

(4) New items have been added to some of the data structures.

(5) New data segments have appeared.

(6) System Control is now in three segments because of its size.

(7) New directories are used by system control.

Published: 05/10/68
(Supersedes: BQ.1.01, 10/27/67)

Identification

System Control Procedure
C. Marceau and H. Hebert

Purpose

The system control procedure has 3 functions:

1) after initialization of Multics is complete the Initializer Control Program calls the system control procedure to make the system available to users;

2) the system control procedure responds to operator requests concerning the system, e.g. the request to configure the communications lines;

3) at system shutdown time the system control procedure takes appropriate action to shut down all system processes and put the system in a stable state in which it can remain until brought up again. It then returns to the initializer control program which shuts down the hardcore supervisor and hardware control procedures.

Discussion

Multics initialization begins when an operator presses the bootload button on a GIOC in order to "bring up the system". During initialization a Multics environment is created so that the last phase of the Initializer Control Program is executing in a bona fide Multics process which is capable of creating other processes. The Initializer control program then executes the call

```
call Multics;
```

The procedure thus invoked first makes sure that the file system hierarchy is reloaded if necessary, and then creates a new process in a new process-group.

The new process is called the system control process, and its process-group is the system control process-group. The first procedure executed by the new process after initialization is the system control procedure. Now the system control procedure goes on to establish communication with the operator.

First system control creates some input/output machinery to enable the operator to input a login line. To do this system control creates a process in another process-group. This process acts as a universal device manager, doing typewriter I/O for all users. (There may be some exceptions to the "all" as when a systems programmers is testing out a new Device Interface Module within his own process-group. Normally, however, a part of the Device Interface Module serving a user on the system executes outside of the user or system process-group, in a device manager process-group). Later, at the operator's behest, system control may create such universal device managers for each type of device, such as printers or tape drives.

(Eventually the operator should be able to communicate with initialization. The means for doing so without preexistent I/O machinery is not yet available. When it is available, system control will have to disable the operator's console after initialization so that he can log in later using standard I/O).

Now system control creates the other system processes necessary to log in the operator. First it creates a process within its own process-group and causes it to execute the load_control procedure. The Load Control process will be responsible for measuring load on the system and maintaining a ranking of user process-groups.

Load Control is created at this time because it is consulted whenever any user, including the operator, logs in. (The system operator's process-group is always the highest priority group in the ranking, hence is never "bumped" should system load get too heavy).

System control, next, creates another process in its own process-group. This new process is the overseer of all user process-groups and handles all the quits from these process-groups.

Next system control creates a user control process within its own process-group. This process is created now because it handles the logins and logouts of all user process-groups.

Then system control creates another process within its own process-group and causes it to start executing the answering service procedure. The Answering Service Process answers dialups from users and controls all devices over which users may dial up.

The answering service now attaches all the communication lines to itself in an offhook state except the operator's console which is placed onhook. Then the answering service sends an interprocess signal to system control and waits for a shutdown signal (see BQ.2.01). Meanwhile the operator dials up, logs in and types a command.

At this point there are three process-groups: the system control process-group, the universal device manager process-group, and the operator's user-process-group. The system control process-group contains five processes: system control, load control, the answering service, user control and the overseer. The operator's process-group contains one working process to execute the operator's commands.

N.B.: The operator we are discussing is the System Operator, meaning that he controls the operation of the system by his commands. Other operators may be around to handle requests for tapes, to run the printer and card reader, etc. But the term "The operator" in this section refers to the System Operator.

Note that the operator has logged in as a garden-variety user and is not ipso facto known to system control. He must inform system control that he has logged in and be identified as a system operator so that communication can be established between his process-group and system control. This he does as follows: after logging in the operator types an "op_here" command. The command procedure must find the process-id of the system control process and the name of an event channel over which to signal the operator's presence to system control. This information has been placed by system control in the segment "operator_comm" in the system control process-group directory. The command next places the name of the operator and the instance tag of his process-group in the segment "operator_rep", also in the system control process-group directory, and signals the event.

Protection of system control is provided as follows. The system operator logs in as a user, i.e. as a person working on a project. The project under which he logs in is "sys_operator". Only certified system operators may work on this project and system operator commands are available only to users on this project. The segments "operator_rep" and "operator_comm" and the op_here event channel are accessible to any user on this project. After

the operator has placed his personal name and instance tag in the segment "operator_rep", system control changes the access to all operator-accessed segments, except "operator_comm" and "operator_rep", and to the request event channel, so that only this particular operator can access them. It then makes an entry in the system operation log giving the data and time the system came up and the name of the operator. It is thus ensured that there is only one system operator at a time and that he is known to system control. (See below for actions taken to change operators).

The operator may now commence to type commands controlling the operation of the system. The general form of these commands is as follows: the command, executing in the operator's working process, writes the name of the request in the segment "request_name", sends to system control a signal over the request event channel, perhaps after putting more specific information in some segment in the request directory (pathname: > system_process_reserved_storage > request_dir) dedicated to the use of this command. Such segments vary in format according to the command.

Meanwhile, back at the ranch, the signal sent by the operator's process is received by sys_control2$reflector, which wakes up and looks in the segment "request_name" for the name of the request. Sys_control2$reflector determines which process should respond to the request, sends an appropriate event to that process and waits for a request-serviced signal from the process.

The appropriate process wakes up, handles the request, and sends an event signal back to system control, which reflects the signal to the operator's process. (For a description of how these commands appear to the operator's process, see BX.15.00). System control, after signalling the operator, waits for another command.

System operator commands issued soon after system initialization include commands to create and start up other processes in the system control process group, and other system process-groups (see BQ.1.02 for a list of system processes and system process-groups.).

The System Operator may also issue commands to split off operator responsibility to other operators. In particular he can allow a logged-in operator to become media operator or answering service operator. Each such delegation of responsibility is noted in the system log. Such delegation of responsibility has two benefits:

1) the operation of a large system may be divided among several operators;

2) event channel communication may be made more direct between the operator's process-group and the process or process-group to which he communicates his requests. (Changes of operator are still managed by system control).

At a later time the system operator may wish to go home (assuming the system stays up so long) and allow another operator to take his place. First the replacement operator logs in, as a normal user. Then the System Operator types a command indicating that the new operator is to take his place. System control verifies the identity of the replacement, and that he is logged in, then changes the access to its segment and event channels so that only the new operator can access them. It then notes the change of operators in the system log and prints appropriate comments to the former system operator and the new system operator.

System control also provides a channel over which the Load Control process will signal it should the operator log out for any reason (by mistake, or because the connection to his console is broken). Should such a thing happen the operator is guaranteed (by Load Control) to be able to log in again and reestablish his connection with system control. Meanwhile no other operator may log in and assume control of the system. In the unlikely occurence that an operator drops dead, and in the course of doing so logs out (e.g., by falling on the "power off" button of his console) then only the system locksmith can save the situation.

As the sun sinks slowly into the west, the system operator types the shut-down command. In response to this command, system control has the load control process log out all users (last of all the operator), send windup signals to all other system process groups and to the system processes in its process-group, then destroys all other processes except itself, cleans up its own data bases, notes the system shutdown in the system log, and returns to its caller, Multics, which returns to the Initializer control program. Initializer control shuts down the hardware.

Implementation

The above discussion outlines the basic strategy of system control. The remainder of this section is included to clarify some details of implementation.

Calling Sequence

The system control procedure is called by process initialization when initialization of the process is complete. It has no arguments:

```
call sys_control;
```

Tables

System control has the primary duties of creating system processes and reflecting operator requests to these processes. For the first job it uses the system process table.

The table is a PL/I structure into which system control writes pertinent information concerning the processes. When it creates a system process, system control fills in for later use the items in the system process table:

```
dcl  1 system_process_table external static,

     2 n_processes fixed bin (17),  /* number of processes
                                       in table*/

     2 process (100),

       3 name char (32),            /* symbolic name*/

       3 id bit (36),

       3 begin_service bit (70),    /* event channels*/

       3 halt_service bit (70),

       3 shut_down bit (70);
```

To create a system process or process-group, system control calls the create_proc procedure (see BJ.2.01). Among the arguments it passes to create_proc is the process initialization table (PIT) for the process-to-be. All PIT's used by system control are copied from templates found in the PIT directory, (pathname: > system_process_reserved_storage > pit_dir). Each PIT in the directory has as its name the 32-character name of the system process. This table contains the name of the first procedure to be executed after initialization (e.g., ans_init$init) and also contains a free data area. The procedure named in the PIT receives a pointer to this data area as its sole argument. For all processes created by system control, this PIT contains a standard "argument list". By standardizing the argument list for all processes which it creates, system control

can have the advantage of being table driven, and does not need to be rewritten each time someone dreams up a new system process. The "argument list" for all processes created by system control contains the following items ("S" indicates argument supplied by system, "R" indicates return argument).

```
dcl 1 arglist based (argp),

    2 p_g_name char (50),              /* process_group id */

    2 sys_control_id bit (70),         /* id of system control */

    2 init_done bit (70),              /* event channel over
                                          which the process
                                          can signal that it
                                          has finished
                                          initialization (S) */

    2 begin_service bit (70),          /* event channel over
                                          which system control
                                          can signal the process
                                          to begin service (R) */

    2 halt_service bit (70),           /* event channel over
                                          which system control
                                          can signal the
                                          process to halt
                                          service (R) */

    2 shut_down bit (70),              /* event channel over
                                          which system control
                                          can signal the process
                                          to shut itself (or
                                          its process-group)
                                          down so that it can be
                                          destroyed (R) */

    2 shut_down_complete bit (70),     /* event channel over
                                          which the process can
                                          signal that its shut
                                          down is completed
                                          (S) */

    2 request_serviced bit (70),       /* event channel over
                                          which the process
                                          should signal when
                                          it has serviced a
                                          request (S) */
```

```
        2 n_requests fixed bin (17),  /* number of operator
                                         requests which are to
                                         be reflected to the
                                         process (S) */

        2 operator_request (argp->arglist.n_requests),

          3 request_name char (32),   /* name of operator
                                         request (S) */

          3 request_chn bit (70);     /* channel over which
                                         system control can
                                         signal the request (R) */

        2 data char (K);
```

The event channel information in the above list is also recorded in the system process table described above. In fact, system processes could exchange event channel information with system control through that table, but the exchange is done in this way so that all system processes will not have to be revised if there is a change in the format of the system process table.

System control uses the system process table and the PIT's to fulfill its first function, that of creating system processes. To fulfill its second function, acting as mediator for the operator, system control uses the operator request table. This table is input at system initialization time. Together, they inform system control of the limits of its responsibility; that is, they define respectively all the possible system processes and all the possible operator requests. (There may later be some provision made for adding to these tables while the system is running.) The operator request table allows the reflection of operator requests to be table driven.

The operator request table is in two parts. The first part of the table consists of a segment named "request_text", of ascii text, defining possible operator requests. This part is the part that is input at system initialization time and goes into the system control directory (pathname: > system_process_reserved_storage > sys_control). The ascii text is a sequence of lines, one line per request. On each line appear 3 items, separated by semicolons:

1) the name of the request (≤ 32 chars);

2) the name of the process or process_group to which the request should be reflected, e.g., ans_service (≤32 chars);

3) the name of a segment associated with the request, to be created in the request directory (≤32 chars)

An extra semicolon at the end of the last line signals end of table.

The second part of the table is a PL/I structure into which system control writes pertinent information during initialization concerning the requests.

The structure is as follows

```
dcl  1 request_table external static,

       2 n_requests fixed bin (17),   /*number of requests
                                        in table*/

       2 requests (100),

         3 name char (32),            /*name of request*/

         3 segment_name char (32),    /*name of segment
                                        associated with
                                        request*/

         3 process_name char (32),    /*name of process
                                        request is to be
                                        reflected to*/

         3 process_id bit (36),       /*id of process and
                                        event channel to
                                        which event should
                                        be reflected*/

         3 ev_chn bit (70);
```

System control fills in the last 2 items as they become available.

System control also keeps a table of possible operator functions. This table is used to fill in the system operator's function table when he reports in. When other operators report in they get a function table with no entries. When a function is delegated to an operator then it is taken out of the system operator's function table and placed in the other operator's table. This table of possible operator functions is in two parts: an ascii segment which comes in at system initialization time and a PL/I structure kept in external static. The ascii text, "op_functions_text" in the system control directory

(pathname: > system_process_reserved_storage> sys_control), is a sequence of lines, one line per function. On each line appear 4 items separated by semicolons.

1) function name;

2) yes or no, depending on whether the function is an unsolicited request or not. An example of an unsolicited request is a request from a user asking the operator to mount a certain tape reel;

3) name of a function_here procedure. This is a procedure which established interprocess communication between the operator's process and the system process which is associated with the function. This entry may be null;

4) name of an attach_here procedure. This is a procedure which attaches to the operator's process any non-console I/O device which the operator will use in his work, for example a cathode ray tube. This item may be null.

An extra semicolon at the end of the last line signals end of table.

The PL/I structure is as follows:

```
declare 1 op_functions external static,        /* table giving all
                                                  possible opera-
                                                  tor functions */

        2 n_fcns fixed bin (17),                /* number of func-
                                                  tions in table*/

        2 fcns (100),

          3 name char (32),                     /* name of
                                                  function */

          3 op_name char (24),                  /* name of opera-
                                                  tor having this
                                                  function */

          3 unsol_req bit (1),                  /* 1 if an unsoli-
                                                  cited request,
                                                  0 if not */

          3 n_hpc fixed bin (17),               /* no. of charac-
                                                  ters in attach_
                                                  proc */
```

```
3 attach_proc char (32);        /* name of attach_
                                   here procedure */
```

Another segment which system control creates and keeps in its process-group directory is "ucp_comm". This is a segment which provides communication between the answering service and user control and user control and the overseer during login and logout of users. (See BQ.2.01 for details about the segment).

Entries to system control

System control has one entry for initialization, sys_control $sys_control, and one entry for reflecting operator requests, as mentioned above, sys_control2$reflector. The second entry is called as follows. System control creates an event call channel over which it receives all operator requests, as described above. It associates with this channel the procedure sys_control2$reflector. After initializing the system and entering the operator's name on the system log, system control calls the wait entry in the wait coordinator. Thereafter the wait coordinator acts as a dispatcher for system control, returning only when the shutdown event is signalled. Sys_control2$reflector signals shutdown to the system control process when it recognizes a shutdown request.

In addition to the entries sys_control and reflector system control has one entry for each command which is directed particularly to system control (except shutdown, which is signalled over an event wait channel and hence causes a return to sys_control$sys_control.)

Examples of such requests are the request to change system operators or the request to split off control of some part of operations (prime example: media management) to an auxiliary operator. Also, the request to create a system_process is directed to sys_control2$startup.

Another entry is sys_control3$notify. This entry notifies the system process or module associated with a function that it must initialize itself and that it also is getting a new operator as handler of its function.

Directories

In addition to the process and process-group directories, system control uses three other directories, all found in the system_process_reserved_storage directory. They are the pit directory, the request directory and the sys_control directory.

The pit directory (pathname: > system_process_reserved_storage >pit_dir) contains template pits of all the system processes which system control can be called upon to create. The system control process group has read access to all segments in this directory. This directory has all its entries when system control begins executing.

The request directory (pathname: >system_process_reserved_storage >request_dir) does not exist when system control begins executing. This directory comes into being when system control creates the segment needed by the Answering Service commands. In this directory go all the segments needed by the different operator requests and the segment called "request_name". When the operator wishes to signal a request, first he places the request name and a return channel in the segment, "request_name", then signals the request to system control. Only the system operator and system process-groups have read and write access to every segment in this directory. However, all operators have access to the segments "operator_comm" and "operator_rep".

The system control directory (pathname:>system_process_reserved_storage>sys_control) exists and has two ascii segments in it when system control begins execution. These are the segments "request_text" and "op_functions_text". These give a list of all the possible operator requests and functions respectively. Access to this directory is limited to the system control process-group.

System Control In Initial Multics

Initial Multics will include all of system control as described in this section. The difference between the initial version and later versions is that in initial Multics system control will accept only the following operator requests to itself:

1) startup of system processes,

2) shutdown of the system.

Later versions will include commands to change system operators, delegate functions to other operators, etc.

Initial Multics also includes the simplification that load control will not require a separate process but only a system-wide table. Hence instead of creating the load control process, system control creates the system-wide load control table.