TO: Multics Distribution
FROM: Carla Marceau
DATE: December 28, 1967
SUBJ: Overseer, BQ.3.01

The attached minor revision brings the implementation of the overseer up to date. Some declarations are changed, quit and completion events are handled by event calls to entries of the overseer, the name of the overseer data base is changed to working process table and is readable in ring 2.

Published: 12/28/67
(Supersedes: BQ.3.01, 07/11/67)

Identification

The overseer procedure
C. Marceau, P. Belmont

Purpose

The overseer procedure is the soul (a materialistic generation might call it the skeleton) of the Overseer Process. By calls to appropriate subroutines, the overseer procedure carries out the essential responsibilities of the Overseer Process, which are:

a) to initialize the process-group for the user;

b) to set up the appropriate subsystem for the user;

c) to respond to signals initiated by the user (as when the user depresses the break key at his interactive console);

d) to respond to system-initiated signals (e.g., automatic logout).

Discussion

The Overseer Process is created by a User Control Process (see BQ.2.03) after the User Control Process has logged the user in. After initialization the Overseer procedure executes in the Overseer Process. Fig. 1 illustrates the User Control Process and the Overseer.

Alternatively, the Overseer Process may be created by the absentee monitor process when a user has requested an absentee job (see BQ.2.04).

Each user-process-group is either interactive or absentee. An interactive process-group is one which is created by a User Control Process because of a dial-up at a typewriter console. An absentee process-group is one which is created by an Absentee Monitor Process. The procedure which creates a new process-group decides whether the new process-group is to be interactive or absentee, and communicates its decision to the Process-group Ranker and to the overseer of the new process-group. The distinction between interactive and absentee lies entirely in the way these modules view the process-group. Namely, the Process-group Ranker suspends

absentee process-groups, when system load is heavy, without actually logging them out. (A suspended process-group is not allowed to run but is reanimated by the Process-group Ranker without user intervention when system load lightens. See BQ.5.01 for a discussion of the suspension of process groups.) The overseer procedure attaches the user's console and enables the console "quit button" if it is told that the process-group is interactive; for an absentee process group the overseer does not attach a user's console. However, there is nothing to stop an absentee process-group from attaching a console (if it could find a free console) and using it to communicate with a user - just as an absentee process can attach tapes, etc. (An attempt to detach an interactive user's console, a potentially catastrophic action since the user would have no means of reestablishing contact with the now-isolated process-group, will be rejected, viz., cause an error return. This is because the overseer executes in the administrative ring when attaching the console, and no user ring procedure may thereafter detach the console.)

Calling the Overseer

The overseer procedure has no arguments in the ordinary sense, since it is invoked by another process. However, certain information must be communicated to the overseer by its caller. The overseer's caller (User Control Process or Absentee Monitor Process) places the necessary information in a segment in the Overseer's process directory, which is accessible to both the Overseer Process and the calling process. The information is arranged in a structure:

```
dcl 1 args,

      2 caller bit (36),          /*id of calling process*/
      2 auto_logout_chn           /*returned by overseer, the name
        bit(70),                    of an event channel over which
                                    the caller can signal an automatic
                                    logout*/
      2 interactive bit (1),      /* =1 if process-group is inter-
                                    active*/
      2 name char (24),           /*user name*/
      2 projid char (24),         /*project id of user*/
      2 suspend_chn bit (70),     /*event channel over which
                                    absentee monitor can signal
                                    suspension, if interactive=0*/
      2 suspend_response_chn      /*event channel over which
        bit (70),                   overseer should signal after
                                    it has received and processed
                                    a suspend event*/
```

```
      2 ready bit (70),        /*event channel over which
                                 overseer should signal to
                                 caller when it is ready to
                                 receive automatic logout
                                 or suspend signals*/
      2 logout bit (70),       /*event channel over which
                                 overseer should signal
                                 after the user logs out
                                 or it has finished
                                 responding to automatic
                                 logout*/
      2 comm_info_size fixed bin (17),
                               /*number of significant
                                 characters in args.comm_info*/
      2 comm_info char (511), /*communication info, =registry
                                 file name of user's console if
                                 interactive=1, else= path
                                 name of a segment in which
                                 further info can be found*/
      2 wdir_size fixed bin (17),
                               /*number of significant
                                 characters in args.wdir_size*/
      2 wdir char (511);       /*pathname of working directory,
                                 = "" if user's default is to
                                 be used*/
```

If the process-group is interactive, args.comm_info is the registry file name of the interactive device (the registry file name uniquely identifies the device - see BT.0.00). If the process-group is absentee, args.comm_info is the path name of a segment in which further information may be found. For example, the Multics Command Subsystem requires as absentee information the names of segments to be used as input source and destination of output from the commands (i.e., segments to which the I/O streamnames user_input and user_output will be attached).

The elements args.wdir specifies a working directory for the user if the user's default should not be used (see below).

Subsystems

The overseer has the job of initiating a particular subsystem as the user's interface with Multics after he has logged in. The subsystem determines how the user's input to his process group is interpreted. For example, in the Multics command subsystem the listener procedure (see BX.2.02) reads the first line that the user types after logging in, then calls the Multics Shell (see BX.2.00)

to interpret the line as a call to a Multics command or user procedure. In another conceivable subsystem some listening procedure might read all user input lines as requests concerning airline reservations.

The Multics command subsystem provides a general interface for the general-purpose user; other subsystems can serve the special-purpose or the restricted user.

The interface of a subsystem with the overseer procedure consists of three elements:

1) a procedure, termed the login responder, which the overseer causes to be executed after initializing the process group (the Command Subsystem's listener is an example of a login responder);

2) a procedure, termed the quit responder, which the overseer causes to be executed after a quit event (when the user depresses the quit button at his console);

3) a one-bit automatic logout save flag which indicates whether, at automatic logout, the overseer procedure should save the status of processes in the process-group or not (see below for a discussion of saving processes in the process group).

These three elements are contained in the PL/I structure, subsystem, which is located in an Overseer data base:

```
dcl 1 subsystem ctl (p),

    2 login_responder char (32),

          /* path name of the procedure */

    2 quit_responder char (32),

          /* path name of the procedure */

    2 auto_logout_save bit (1);
```

the subsystem structure is located in a segment in the Overseer's process directory. The overseer creates and fills in the subsystem structure segment immediately after login, using information contained in the user profile (see BQ.4.03). The subsystem structure can be modified from the administrative ring.

The overseer obtains a subsystem for the user in the following way:

1) A segment in the user's profile (see BQ.4.03 on user profile directories) specifies what subsystem the user will operate in. This segment can be written in only by the user's project administrator, and specifies both a subsystem and whether the subsystem is mandatory for the user or a default.

2) If the project administrator does not enforce a subsystem on the user, the user may specify his own in another segment in his user profile.

   If the project administrator enforces a subsystem on the user, the overseer fills in the subsystem structure with the specifications of the enforced subsystem. If the project administrator does not enforce a subsystem, the overseer looks to see if the user has specified a subsystem, and if so, uses his specification. If the user has not specified a subsystem, the overseer uses the project-specified default.

After control has passed to the subsystem (viz., to the login responder) the login responder can change the subsystem by modifying the Overseer's subsystem data base. For example, suppose a project administrator's enforced subsystem consists of an enforced identity validation procedure (similar to personal passwords) followed by entry into the Multics Command Subsystem. The project administrator would specify as login responder the identity validation procedure. After the procedure had verified the user's identity, it would modify the subsystem structure (to specify the Multics Command Subsystem) and then return to the overseer. The overseer would call the login responder of the Multics Command Subsystem, i.e., the Listener procedure.

Section BQ.2.06 discusses subsystems in greater detail.

The Login Responder

The subsystem's login responder executes not in the Overseer Process, but in a Working Process created by the overseer. Thus the Overseer Process is at all times ready and willing to respond to user-initiated and system-initiated events (see below). After initiating the login responder, the Overseer Process enters the blocked state, where it remains until either the login responder sends a "completion" event to the Overseer or some other event occurs which causes the Overseer to "wake up". On receiving a "completion"

from the login responder, the overseer destroys the old working processes, creates a new working process, and causes the login responder to begin executing in the new process. The events which can cause the Overseer to interrupt the login responder are:

1) the user issues a "quit" by depressing the break key at his console;

2) the user issues a logout command;

3) the system informs the overseer that the user must be logged out ("automatic" logout);

4) the connection to the user's interactive console is broken (e.g., the user hangs up);

5) the user runs out of funds.

The Overseer process spends most of its life, after it has started up the user's subsystem, waiting for one of the above events to occur. The action taken by the overseer when it notes one of the above events is discussed below. Here our concern is to describe the interfaces between the overseer and the subsystem.

The Quit Responder

After stopping the process-group by quitting each working process, the overseer procedure causes the subsystem's quit responder to be executed in a working process. The quit responder may operate independently of the login responder, but close cooperation between them is possible and usually desirable. (For example, the Multics Command Subsystem's quit responder is an alternate entry to the Listener, its login responder. See BX.2.02.)

Automatic Logout Save Flag

At automatic logout control passes permanently from the user's subsystem. The overseer procedure saves the status of the process-group if the automatic logout save flag is "1"b. (Save is not implemented in initial Multics.)

Flow of Control in the Overseer Procedure

The following discussion is divided into 4 parts:

1) initializing the process-group,

2) starting up the subsystem,

3) responding to events, and

4) logout.

A flow chart for the overseer may be found in figs. 2, 3 and 4.

Initializing the Process-group

The first responsibility of the overseer procedure is to initialize the process-group to be able to do work for the user. To do this initialization the overseer must:

1) Create the machinery for communications between the Overseer and the user control process. First, the overseer creates event channels for automatic logouts and out-of-funds events.

   If the process-group is absentee, the overseer creates an event channel for suspension events. An absentee process-group may be temporarily suspended instead of being logged out. (See below).

2) If the process-group is interactive, the Overseer attaches the user's console (See BF.1) and creates event channels over which quits and hang_ups (See BF.2.25) may be signalled. The console is identified by the registry file name supplied to the Overseer when it was created.

3) Create and initialize the working process table:

```
dcl 1 wpt based (pp),          /*working process table*/
    2 wpp bit (18),            /*relative ptr to most recent
                                 entry in table, =0 if no
                                 entries*/
    2 overseer_process_id      /*overseer process id*/
      bit (36),
    2 logout_chn bit (70),     /*channel over which logout
                                 command can signal logout
                                 to overseer from a working
                                 process*/
```

```
    2 completion bit (70),     /*channel over which working
                                 process can signal completion
                                 event to overseer*/
    2 start bit (70),          /*channel over which working
                                 process can signal start
                                 event to overseer*/
    2 hold bit (70),           /*channel over which working
                                 process can signal hold
                                 event to overseer*/
    2 reset bit (70),          /*channel over which working
                                 process can signal reset
                                 event to overseer*/
    2 wps  area ((4096));      /*area in which entries for each
                                 process are allocated*/


dcl 1 wp based (wpp),          /*entry in wpt, allocated in
                                 wpt.wps*/
    2 forwardp bit (18),       /*relative pointer to next process,
                                 =0 for most recent entry*/
    2 backp bit (18),          /*relative pointer to previous
                                 process, =0 for first entry in
                                 table*/
    2 id bit (36),             /*process id*/
    2 name char (32),          /*symbolic name of process, if any*/
    2 runout_lock fixed bin    /*ring 1 procedures increment this
      (17),                      count by 1 to make themselves
                                 unquittable.  When runout_lock =0,
                                 the overseer can quit this
                                 process*/
    2 i_am_quittable bit (70),/*channel over which working
                                 process can signal to overseer
                                 that it is now quittable.  This
                                 occurs when the working process
                                 declares itself unquittable,
                                 then finds, by checking the
                                 quit_pending flag(wp.quit_pending)
                                 for itself, that the overseer
                                 process is attempting to quit it*/
    2 quit_pending bit(1),     /*=1 iff the overseer process is
                                 attempting to quit the user's
                                 working process*/
    2 quit_flag bit (1),       /*=1 iff this process has been
                                 quit by stop procedure*/
    2 destroy_flag bit (1);    /*=1 iff this process has been
                                 marked for destruction by stop
                                 procedure*/
```

This table, which contains one entry for each working process in the process group, records the current status of each working process (quit_flag, destroy_flag, quit_pending, runout_lock), and also information, including the names of event channels, to be used for certain specific interprocess communications. Except for the quit_inhibit procedure which modifies the runout_lock (see BQ.3.06), the only procedures which modify the working process table execute in the overseer process.

4) Create machinery for communication between the Overseer and the working processes in the process-group. The Overseer creates event channels for completion, start, hold, and reset events. The event channel id's are then written in the top of the working process table and are declared call-type-events, with overseer$completion, ov$start, ov$hold and ov$reset procedures to be called in response to these events, respectively.

The Overseer also creates an event channel for the logout event and stores its id in the working process table. This channel is a wait-type event channel. In general, wait-type event channels are used for all events leading to logouts and call-type channels for all others.

Any ring 32 procedure will be able to signal the Overseer over these channels since the working process table is readable in ring 32 and the event channels are accessible from ring 32.

5) Copy process profile segments (defined in BQ.4.03) from the user's profile into the process group directory. These segments constitute the common process profile for working processes in the process-group. (The Overseer's process profile is distinct from these segments.) The wdir argument to the overseer (if non-null) specifies a change to be made in the just-copied working directory table. If wdir is non-null, the overseer now updates this table by calling session_wdir (see BX.2.12).

6) Create a subsystem data base segment which will be used to record the current subsystem. The overseer obtains an initial subsystem for the user from his user profile (as described above, under "subsystems") and enters it into the data base. Administrative ring procedures may update the subsystem data base as necessary.

7) Finally, inform the user (if interactive) of his successful login.

Setting up the User Subsystem

The overseer now sets up a subsystem (causes it to begin execution) which will act as the interface between the logged-in user and Multics. The three elements defining the subsystem (login responder, quit responder, and automatic logout save flag) have already been placed in the subsystem table in the Overseer process directory. Now the overseer creates a working process and causes the working process to begin executing the subsystem's login responder.

The Overseer "calls" (via a givecall) the login responder with two "arguments":

a) a switch which indicates whether the process-group is interactive or absentee;

b) absentee information (as in call to overseer), if process-group is absentee.

For the time being, the "arguments" are not actual arguments, since givecalls include no argument passing. Instead the overseer places the "arguments" in the "responder_args" segment in the process directory of the working process, before calling the login responder via a givecall. (The overseer passes no arguments to the login responder in initial Multics.)

A login responder should:

a) attach stream names for input/output, either to the ioname "console" or, in an absentee process-group, to segments in the file system hierarchy;

b) do work, for example, by calling other procedures and interacting with the user;

c) return to the overseer for housekeeping (when the login responder returns, the overseer destroys the working process, creates a new working process, and causes the new process to execute the login responder);

d) end the console session by signalling a logout event to the overseer.

This allows the Working Process to handle input/output through stream names and allows the Overseer Process to wait, as soon as possible, for signals requiring immediate attention. For example, the Overseer can process quit events and has an opportunity to save the state of the process-group in case of an automatic logout or suspension event.

User signals to the Overseer

The user (or subsystem) can signal several events to the overseer. A quit is an interrupt issued from the user console to an interactive process-group, or from another process-group to an absentee process-group. A quit event means "Stop the process-group and then do what I say". The overseer does the job of stopping the process-group by calling the stop procedure which quits each process in the group (except the Overseer). The overseer then starts up the subsystem's quit responder, which has the responsibility for deciding what happens next. Some subsystems may ask the user for instructions, others may interpret the "quit" in a fixed manner.

The second user signal to the overseer is logout. When a process in the group sends this event to the Overseer, the Overseer destroys the process-group and returns (causing the group to be logged out). Normally the sending of a logout event will be embedded in a subsystem procedure, for example, the Multics Command Subsystem's logout command (see BX.3.03).

The interactive user can also hang up his console (or his line connection may be cut). The action taken in this case is the same as for an automatic logout (see below).

The Overseer takes the following actions in response to a quit event:

1) calls the stop procedure to quit all working processes in the process-group and to prepare the user's console for further input/output while saving current I/O buffers (see BQ.3.04);

2) creates a new working process;

3) sends an interprocess call to cause the start of execution of the quit responder in the new working process;

4) waits for any event to occur.

Fig. 5 shows the history of the process-group up through its first "quit" event.

When a working process sends a start event the Overseer starts all the quit working process, restores the quit I/O, and destroys the calling process. If there are no quit processes, the Overseer ignores the event. (The start event is set by the quit responder when the user says "restart, I hit the quit button by mistake." All I/O has been preserved and the old process-group carries on as before.)

A hold event means to hold the quit computation so that it will not be destroyed when the next quit occurs.

A completion event means stop and destroy the user's computation and to create a new working process to execute the subsystem's login responder.

A reset event means "destroy all processes which were quit". In other words, the new working process is assuming the role of the login responder and does not need the quit process(es). The reset event is sent to the Overseer by the reset procedure (see BQ.3.04).

Thus at any one time the overseer is waiting for at most nine events:

1. quit
2. logout
3. automatic logout
4. hangup
5. out of funds
6. completion from a working process
7. hold event from a working process
8. start event from a working process
9. restart event from a working process

Note also that when the stop procedure quits the processes of the process-group, it also destroys all previously-quit processes, and the event channels over which they might signal events to the overseer. This cleanup mechanism means that a quit process may be kept around until the next quit event, but no longer, so that the number of processes in the process-group is controlled.

The overseer takes the following actions in response to a logout event:

1) calls the stop procedure to quit all working processes in the group and to halt input/output;

2) deallocates all resources which this process-group has allocated to itself;

3) destroys all working processes in the group;

4) sends a completion event to the User Control Process, which logs the user out).

System Signals to the Overseer

The system can signal three events to the Overseer. An automatic logout signal is sent when the system has decided to log out the user, whether he will or not. The out-of-funds event is signalled to the overseer when the user's account runs dry. Both events are currently handled by the Overseer in the same way (as is a hangup event, caused by the user hanging up his console or by the telephone connection being severed). The suspension event is signalled to an absentee overseer when the process-group is to be temporarily suspended.

The overseer takes the following action to handle automatic logout, out-of-funds, or a hangup event;

1) calls the stop procedure to quit all processes in the group;

2) informs the (interactive) user of the automatic logout;

3) tests the subsystem's automatic logout save flag. If it is "1"b, the overseer calls save (see BQ.3.05) to save the status of the process-group;

4) deallocates all resources which this process-group has allocated to itself;

5) destroys all processes in the group;

6) returns (causing a completion event to be sent to the User Control Process, which logs out the user).

The overseer takes the same action in response to a suspend event as to an automatic logout event, except that the status of the process-group is always saved, regardless of the automatic logout responder bit, and no comments are made to the user. The return causes a completion event to be sent to the Absentee Monitor Process which takes appropriate action to suspend the process-group (see BQ.2.04).

Ring Residence

The Overseer, stop, hold, reset, and start procedures reside in ring 1. These are reliable administrative procedures.

The login responder and quit responder reside in user rings so that any project administrator or user can specify a subsystem without going through the administrative red tape necessary for administrative ring residence.

Overseer data bases, such as the subsystem data base (see above) are in the administrative ring. The user ring subsystem can call an administrative procedure, change_subsystem (see BX.3.03), to modify the subsystem data base. Change_subsystem checks the requested modification to see if it is a permissible one before modifying the subsystem data base.

To facilitate the sending of events from the user ring to the overseer, the working process table is readable in the user ring (so that event channel id's can be read) although writable only in ring 1.

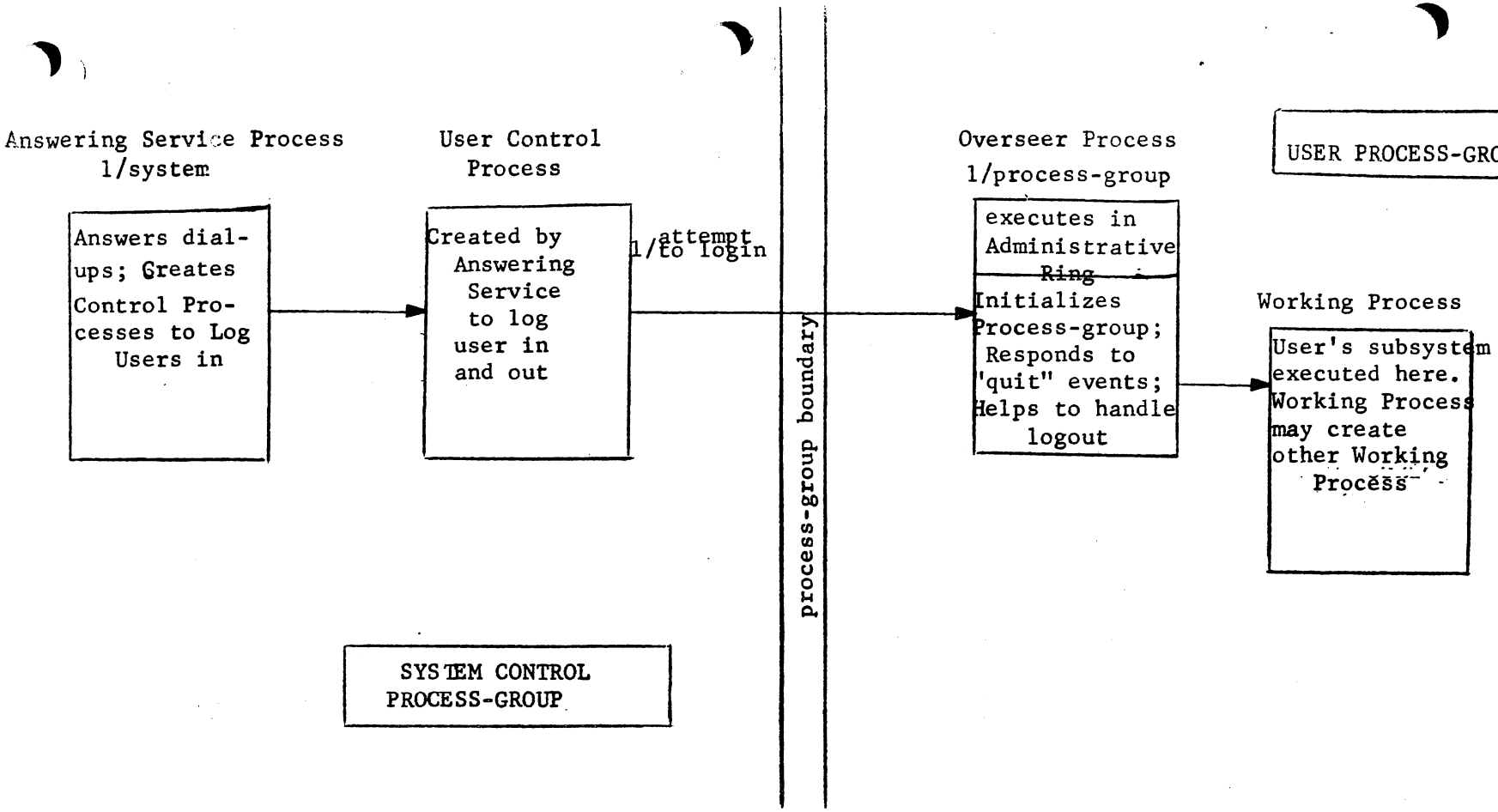


Fig. 1 The Overseer and its Friends

The Overseer is created by a User Control Process after the User Control Process has logged the user in.

The Overseer creates a working Process in which the user's subsystem can run (e.g. which executes the user's commands).

MSPM Fig. 2 Initializing the Process-group and starting up the user's sub-system

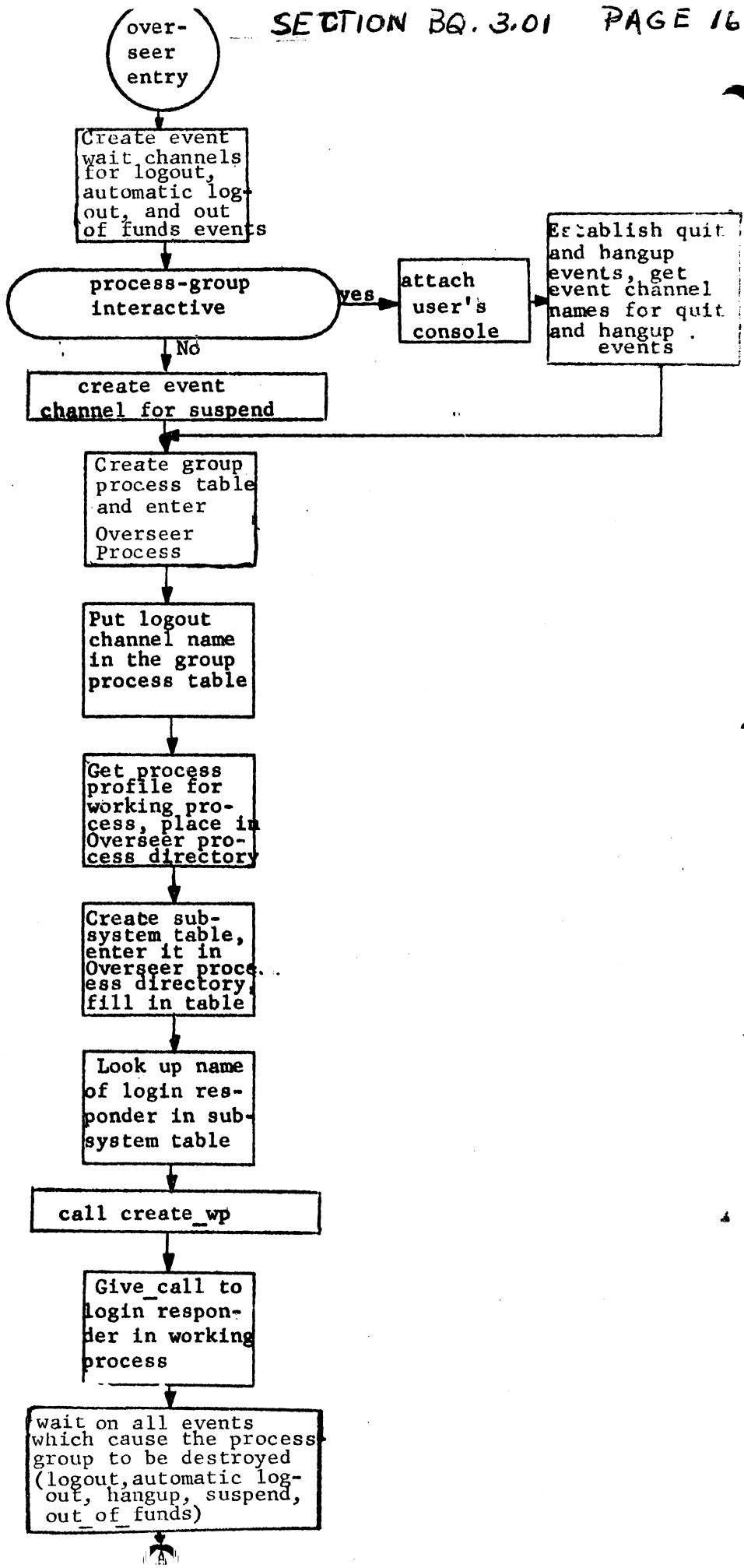
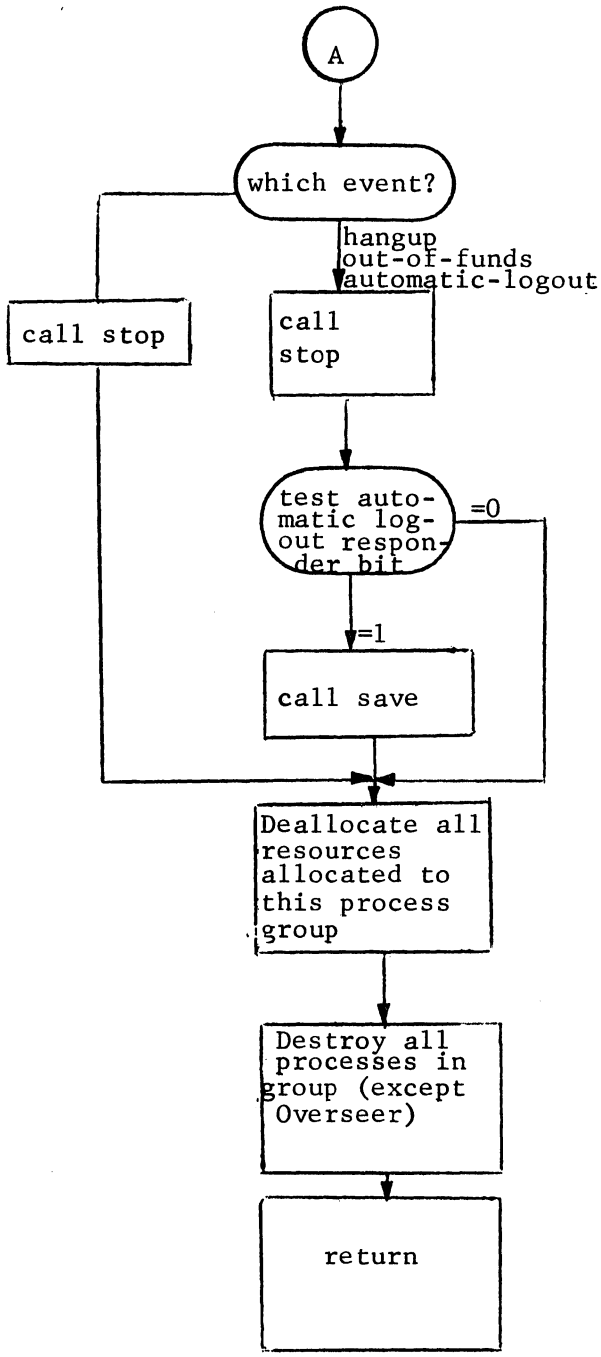
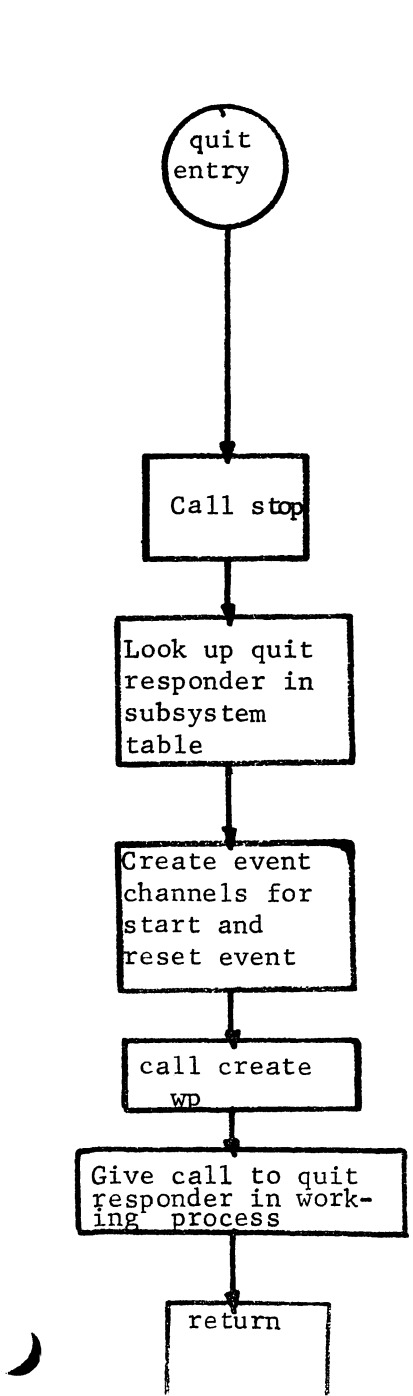


Fig. 3



(A) refers to flow chart in Fig. 2



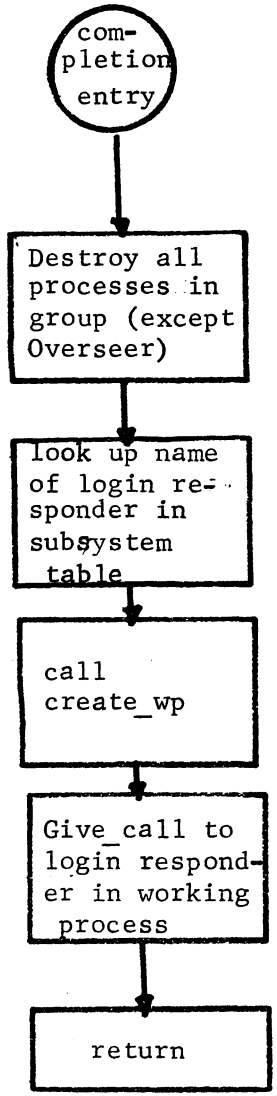


Figure 4. The Quit and Completion entries to the Overseer

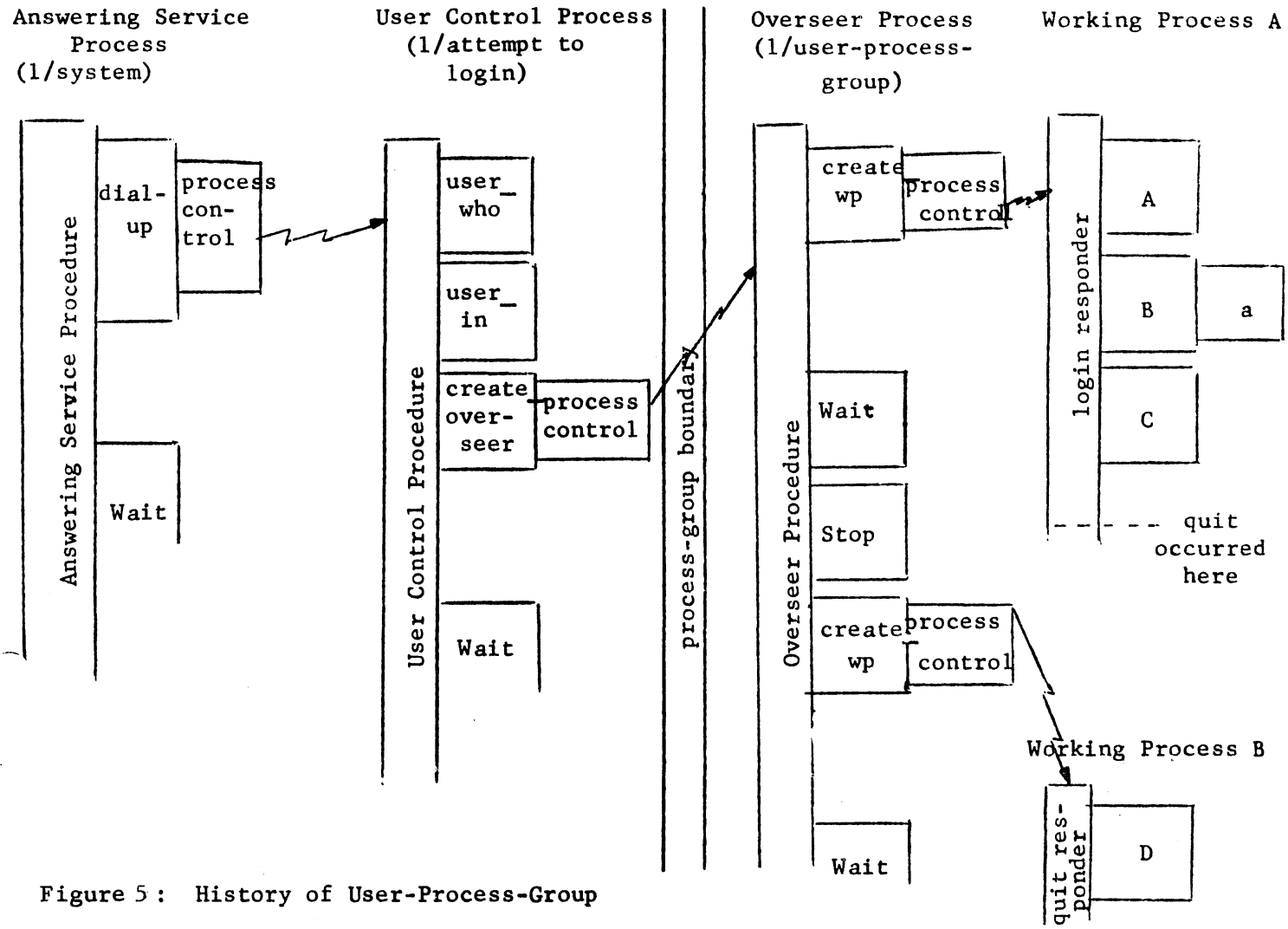


Figure 5: History of User-Process-Group

The figure shows five processes involved in the history of a user process-group. The diagonal double line is a boundary between process-groups.

The flow of control within each process is shown by a process portrain - a "building-block" picture of calls and returns which occur in the process. A horizontal line at the top of a "building block" represents a call from the procedure represented by the "building block" immediately to the left. A horizontal line at the bottom represents a return. A block without a bottom (return) line represents a procedure which has not returned.

A zap (⟿) from Process Control in one process to another process signifies that the sending process has created a process and has caused a procedure to be executed by the receiving process, e.g., Process Control in the Answering Service Process causes the user_control procedure to be executed in the User Control Process. The user process-group illustrated has just received a signal from the user (who pressed the break key at his console). After stop had quit Working Process A, the overseer called create_wp to create Working Process B, then by a give_call caused B to execute the quit responder. Now the overseer is waiting for an event from B.