~~Published: 11/03/67~~

Identification

User-process-groups, an overview

J.H. Saltzer, K.J. Martin, C. Marceau

Purpose

In Multics, a single process is only capable of serial handling of multiple tasks. If programming of multiple tasks is to be organized in a parallel fashion, for either programming convenience, for security, or to take advantage of the multiple processors of the system, more than one process is required. A user-process-group is a collection of processes operating for an instance of a logged-in user. This section is an overview of the organization and operation of a user process group.

The reader of this section should be familiar with the terminology presented in Section BQ.2.00.

Process-Groups

A user-process-group is a process-group that does work for a particular user. Besides user-process-groups, there are system process-groups, which serve the needs of the system as a whole, and act as support for user-process-groups. What, then, is a process-group?

A process-group is a collection of one or more processes working together on a common job. For example, a user-process-group serves one particular user. A process-group as such has certain distinguishing features (even if only one process is in it):

1) Access to segments is by process-group. Since all processes in the group are cooperating to accomplish a common job, they all have common access to procedure and data segments.

2) Resources are allocated to process-groups. If all processes in the group are cooperating on a common job they may also cooperate in writing the results on magnetic tape. On the other hand, other process-groups, since they are working on other jobs, should not have access to the tape drive used by this process-group.

3) Interprocess communication between process-groups requires explicit permission from the receiving process to the sending process. Within one process-group processes are assumed willing to receive messages from each other.

→ In summary, processes within a process-group have common access to segments. Different process-groups are likely to have different access to segments.

Now the question arises: why should there ever by more than one process in a process-group. That is, why can't one process do one job? There are several possible reasons for having more than one process work on a common task with common access privileges. Among these are:

1) the desire to take advantage of the multiple processors of the system;

2) the desire to break a large job down into logical components which may execute concurrently;

3) the need to break a very large job down into smaller components because of restrictions within one process (for example, to avoid an overflow of the descriptor segment);

4) a need for concurrent processing within one job, i.e., the job itself requires concurrency of processing.

We therefore refrain from restricting process-groups to consist of a single process. For the present, we define a user-process-group to be a collection of one or more processes dedicated to serving one logged-in user. We begin by exploring the path by which a user process-group comes into existence.

When the system operator types a command signifying that certain communication lines should be made available for interactive users, the command procedure creates a new process in a new process-group for each communication line. It starts the process off in the User Control module (described in BQ.2.03). The newly created process-group will become the user-process group of the user who successfully logs in over the communication line.

Sometime later the user finishes his work and logs out, or is automatically logged out. In any case, a logout entry of User Control is invoked and terminates the user-process-group. It does this by creating a process in a new process-group and starts this process off in User Control. In this new process User Control destroys the old process-group, thus completing the logout, before settling down to read a login line from the console, and logging in a new user.

The Absentee Monitor Process

The Absentee Monitor Process has 2 tasks:

1) To monitor all requests for the running of absentee computations;
2) to initiate and terminate the running of absentee computations at the direction of load control.

When a user types the login_absentee command, the Absentee Monitor assumes control of the absentee computation and enters it into the user log. The absentee computation begins in the shelved state; that is, although it has a process-group id reserved for it, no process-group is currently existing to execute it. After a time, when system load permits, the Absentee Monitor Process unshelves the computation by creating a user-process-group in which the computation can run. Thereafter, the Absentee Monitor can cause the user-process-group to stop the computation in one of three ways:

1) The Absentee Monitor may send an absentee-stop signal to the overseer, to stop the computation, for example, because system load is heavy. Later, when system load permits, the computation may be ~~resumed.~~ restarted
2) The Absentee Monitor may send a suspend signal to the overseer, indicating that the computation should be saved in its current state, because the system is being shut down. Later, when the system comes up again, the Absentee Monitor will ~~resume~~ cause the computation. to resume execution
3) The Absentee Monitor may send an automatic logout signal to the overseer, e.g., on the request of the user who has logged in interactively. This causes the computation to be ~~logged~~ logged out.

At some time
~~Finally,~~ the process group ~~may~~ sends a completion signal to the Absentee Monitor indicating that the computation is being logged out. The Absentee Monitor then does its part in the logout, i.e., destroys the process-group, notes the logout in the user log, and deletes the computation from its records.

The User-Process-Group

A user-process-group consists of a user's computation and a user control overseer package. The user control logging-in mechanism is necessary in interactive groups for

validating the user's right to use Multics. ~~One~~ The function of the overseer module is to create an environment in which the user may type commands as part of his computation, interact with the command procedures, and control his computation by being able to "quit" it at will. After quitting a computation the user can start it again, begin a fresh computation (reset), or hold the "quitted" computation so that it may be operated on as data. The user control overseer package has responsibility to the system for the user process-group, for example to shut down the process-group in case the system decides to log out the user (automatic logout).

The user control/overseer package executes in the same process(es) as the user's computation, just as the procedures of the file system do. To see how they control the user's computation, let us examine the life of a typical process group.

The ~~Answering Service~~ operator's answer command (or the Absentee Monitor) creates the first process of a ~~the~~ user-process- group, ~~and causes it to execute the overseer procedure.~~ This occurs as soon as the ~~Answering Service~~ command is ~~told~~ alerted to accept dialups over the typewriter line associated with the process-group (or as soon as the Absentee Monitor is told to allow absentee computations to run.)

The new process begins life in the ~~new_user~~ init interactive_group procedure, ~~which~~ ~~If the group is interactive, this procedure~~ calls user_control to log in the user. The user_control procedure now attempts to read a login line from the user console, and waits on the call until some user dials up and types a login line. Now user_control attempts to log in the user (see BQ.2.03) and, if successful, returns to its caller indicating a successful login. Otherwise it returns

with a status indicating that the user is not logged in, and new_user_group calls user_control again to read another login line. In the event of a successful login, new_user_group changes the group id by a call to the hard-core procedure change_group_id (q.v.) and then calls the overseer procedure (see below). The call to change_group_id changes the group's name from that of user control group to the name of the user. It also causes access to all segments outside of ring 0 to be recomputed.

If the process-group is absentee, it begins execution in init_absentee_group, which ~~new_user_group~~ does not call user_control, (whose job is to log in the user) because the user's identity is already known. Instead it calls change_group_id and then calls the overseer procedure.

The Overseer

The Overseer's function is to respond to signals from outside of the user process-group, i.e., to quit and hangup signals from the user's device, to automatic logout signals from the system and possibly to suspension signals from the absentee monitor (in the case of an absentee computation).

The overseer, called at the beginning of the life of the process, makes calls in to the hard core ring requesting that certain conditions be signalled whenever the process receives an interprocess signal. The interprocess signalling mechanism is described elsewhere in greater detail (see BQ.3.01). Briefly, it allows the overseer in a user process to notice the existence of a condition of interest to it whenever the process takes a fault (e.g., a missing page fault or timer-runout fault). The Fault Interceptor Module signals the condition for which the overseer has provided an appropriate handler.

On hangup or automatic logout the overseer saves the current state of the user's computation so that the user can resume it at some later time when he logs in again. Then the overseer deallocates system resources allocated to the process-group and ~~sends a completion signal to the creator of the process group (Answering Service or Absentee Monitor)~~ calls user control at a logout entry. (See above under "process-group" for the termination of the user-process-group.)

On a suspension signal the overseer deallocates resources used by the process group and waits for an event via the interprocess communication facility telling it to restart the user's computation.

The overseer's response to a quit is functionally more complex. On receiving notice of a quit, the overseer stops all work being done in the computation. It then provides for 3 possible actions:

1) restarting the quit computation;

2) ignoring the quit computation while the user starts a new computation in this same process;

~~3) debugging the quit computation;~~

3) starting a new computation in a fresh process (because of possible irreparable damage in the user ring of the user's process).

In addition to handling hangup, automatic logout, and quit conditions, the overseer provides the facility for housekeeping a user's computation. That is, if the user process's address space is cluttered, or perhaps even stuffed, the overseer can provide the computation with a new process in which to execute.

It remains to discuss the interface between the overseer and the user's computation. We first describe the profile of a typical computation, then

discuss how the overseer interacts with it.

User Computations

A user's computation typically consists of the execution of a series of commands, perhaps including parallel execution of some procedures. In the Multics Command System, the order of execution is as follows:

The Listener procedure (see BX.2.02) "listens" at the typewriter for the user to type a command sequence. When he does so the Listener calls the Shell (see BX.2.00) to interpret the command sequence. The Shell calls a command, which calls several procedures, and then eventually returns. If there are more commands in the sequence, the Shell calls the next command. If this is the last command in the sequence, the Shell returns to the Listener, which "listens" for the user to type another command sequence.

A process in which a user's computation executes has certain peculiar features: for example, its options (see BX.12.00) and working directory table (see BX.8.12) are set up to correspond to the user's options and working directory. (In system processes these user-oriented features may be mere appendices. That is, their option-stacks and working directory tables may not exist, and default values for the options and working directory may be assumed.)

The user may, in the course of a command, create other user processes to execute in parallel with his command. These user processes do not necessarily have the format of Listener calls Shell, etc., as above, but are programmed entirely by the user (see BY.5.01 on creating a working process).

The overseer and the user subsystem

The overseer ~~Process~~, after initializing certain process-wide data bases, calls the login responder of the user. The login responder is usually a program which listens for user commands: the usual login responder is the Listener procedure described ~~below~~ above.

After some time.login responder may return to its caller. When that happens the overseer creates a new working process and causes it to execute the login responder again. The Listener procedure (the Multics Command System login responder) returns in order to housekeep, that is to start a fresh computation without old commands and procedures clogging up the address space of its process(es). (Creating a new process is the current method of housekeeping because it is the easiest.)

It may also happen that the user from his console sends a "quit" signal to the Overseer. Then the Overseer halts the current computation and begins a fresh one. It does this by calling the quit responder of the user. The Multics Command System quit responder (that of most users) is cousin to the Listener. This cousin watches out for the commands "start", "new_process", and for debugging commands. This is because start, new_process and the debugging commands actually requests concerning the user's computation, and not "commands" in the usual sense (thus they are usually meaningless in the middle of a command sequence). ~~The Listener also watches for debugging commands.~~

Finally, at some time some working process may call the logout entry of the overseer. When this happens, the overseer destroys the current computation and any quit computation which is still around, and deallocates resources allocated to the process-group. ~~and sends a completion event to its creator.~~