Published: 11/03/67

Identification

User-process-groups, an overview
C. Marceau, J. H. Saltzer, K. J. Martin

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) In general, I/O is per process-group, for the same reason as stated above.

Now the question arises: why should there ever be 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. 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. It will appear in the course of this section that user-process-groups in particular must consist of at least two processes. For the present, however, 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 process-group comes into existence.

The Answering Service

The Answering Service Process (BQ.2.01) is assigned all communication channels not currently in use. When a potential user dials up on one of these channels, the resulting interrupt wakes up the Answering Service Process which interprets the dial-up to mean that some user wishes to attempt to log in to the system. The Answering Service Process therefore creates a new process, named the User Control Process (described in BQ.2.03), to handle this task. It starts the User Control Process off in the user_control procedure, and then returns to its vigil, waiting for more dial-ups.

Sometime later the User Control Process may send a completion signal (by setting an event of the interprocess communication facility). The meaning of this completion signal is that the User Control Process is no longer needed; perhaps the potential user failed to identify himself and was refused permission, or perhaps he logged in, worked for a while and has now logged out. In any case the Answering Service upon receipt of this completion signal reassigns itself the communication channel to watch for future dial-ups, and also destroys the no-longer-needed User Control Process.

The User Control Process

There is one User Control Process for each attempt to log in. The User Control Process has two tasks: establish the user's right to use system resources at this time and create a user-process-group for him to work with. The first task is accomplished by interacting with the user to obtain his opinion of his identification and his password.

If the user satisfactorily identifies himself, the user_in procedure then interrogates the Load Control module to ascertain whether or not the system can allow this particular user to log in now. Assuming he can log in, the User Control process records him as "logged-in" in the User Log, and sets him up in business with his own process-group. This last step is accomplished by creating a process and starting this process off in the "overseer" procedure (see BQ.3.01); this process becomes the "Overseer Process" for the newly logged-in user. The overseer procedure is given as an argument the name of the console on which the user dialed up.

Sometime later the Overseer may send a completion signal to the User Control Process. This completion signal means that the user is to be logged out; so the User Control Process destroys the Overseer, logs out the user, and sends a completion signal to the Answering Service. (Note that the work of User Control processes cannot be done by the Answering Service, because the Answering Service must listen for new dial-ups while users are being logged in (i.e., the job to be done demands concurrent processing.) Similarly, separate User Control processes ensure that users do not have to be logged in or logged out serially.)

The Absentee Monitor Process

The Absentee Monitor Process has 2 tasks:

1) To monitor all requests for the running of absentee computations;

2) to act as user control process for each absentee process-group.

When a user types the login_absentee command, the Absentee Monitor assumes control of the absentee computation. 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 and begins to assume the same role towards it as the User Control Process assumes toward interactive computations. The Absentee Monitor Process creates a Overseer Process for the absentee process-group. At some later time, the Absentee Monitor may send a suspend event to the Overseer, indicating that the computation should be saved in its current state before the Absentee Monitor destroys the Overseer Process. Later, when system load permits, it will again unshelve the computation.

Finally, the Overseer Process may send 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, destroys the Overseer Process, and deletes the computation from its records.

Figure I shows the Answering Service Process, User Control Processes, and the Absentee Monitor Process in their relation to user-process-groups.

(Note that the Absentee Monitor can perform the functions of user control to all absentee process-groups because it does not interact with the user as User Control Processes do. Thus it can take its time and operate serially on all absentee jobs. User Control Processes must be more concerned with providing good response to users who are logging in.)

The User-Process-Group

A user-process-group consists of a user's computation and an overseer module which is considered to be part of the user control package. One 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, destroy it and begin a fresh computation (reset), or hold the "quitted" computation so that it may be operated on as data. Besides its function of creating a friendly environment for the user's computation, the overseer module has responsibility to the system, for example to shut down the process-group in case the system decides to log out the user (automatic logout).

Ideally the overseer module should execute in the same process(es) as the user's computation, just as the procedures of traffic control and the file system operate as part of the user's process(es). Wherever possible, this is indeed the case. However, one requirement on the Overseer is that it be able to respond quickly to signals from the user concerning his computation and signals from the system concerning the process-group. To receive such signals a process must be programmed to enter the wait coordinator (see BQ.6.00), which checks to see if any interprocess signals have been sent to the process. Now a process which is part of a user's computation is programmed by the user and cannot be guaranteed to enter the wait coordinator often enough to give "immediate" response to signals from the user or the system. Hence one process in each user-process-group, called the Overseer Process, contains those modules necessary to respond to user and system signals. The Overseer Process spends most of its life in the wait coordinator, waiting for signals. When, for example, a quit signal arrives, the Overseer immediately executes the quit procedure (see BQ.3.03) and then reenters the wait coordinator. After a user has quit, he may by typewriter request start, reset, or hold. These requests are processed, as far as its possible, in processes which execute the user's computation.

Further, the save and resume procedures (also part of the overseer module) execute for the most part in user processes. (In the initial version of Multics all of the procedures mentioned above will execute in the Overseer Process, for ease of implementation. Later only quits and system-initiated events will be handled in the Overseer Process.)

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 the user's computation executes is called a working process. A working process 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 working processes to execute in parallel with his command. These working processes do not 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 Process and Working Processes

The Overseer Process, created by the User Control Process, is the first process of a user-process-group to come into existence. After initializing certain process-group-wide data bases, it creates a working process, and causes the working process to execute 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 above.

After some time the working process may signal an event to the Overseer Process to indicate that the login responder is returning 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).

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 creating a new working process and causing it to execute the quit responder of the user. The Multics Command System quit responder (that of most users) is a special entry to the Listener. When called at this entry, the Listener watches out for the commands "start", "hold" or "reset", and for commands which imply these commands. This is because start, hold, and reset are 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).

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

Figure II shows the processes of a typical user-process-group, arranged in computations.

Asynchronous Input/Output (I/O)

In general it is desirable to allow a working process to proceed with its work as quickly as possible, for example to solve one equation while the user is inputting the next equation. Thus the next equation is ready for the working process as soon as the working process is ready for it. Similarly, it is desirable that the working process not be required to output the results of the first equation at the slow speed of the typewriter, but that it can proceed to the next computation while the results of the first are being outputted.

Such read-ahead and write-behind imply concurrent processing: that is, the working process computes and the device is monitored concurrently. The process which does the latter job is called (surprise!) a device manager process.

The device manager process which monitors a device used by a user-process-group could reside in the user's process-group, and this may occasionally be the case, as when a system programmer is checking a new device control module (see BF.0). Normally, however, device manager processes for each type of device (e.g., typewriter, or tape) are pooled in a separate process-group. This arrangement has two advantages:

1) it allows for easier protection of device assignments;

2) it permits one device manager to monitor devices for more than one user process group, if this can be done without degrading response time.

Concurrent processing of I/O is handled automatically by the I/O system. Thus a user procedure can simply call the I/O system read procedure, without having to be aware of the fact that a device manager process may already have read in characters from a physical device and stored them in a buffer where the read procedure can find them.

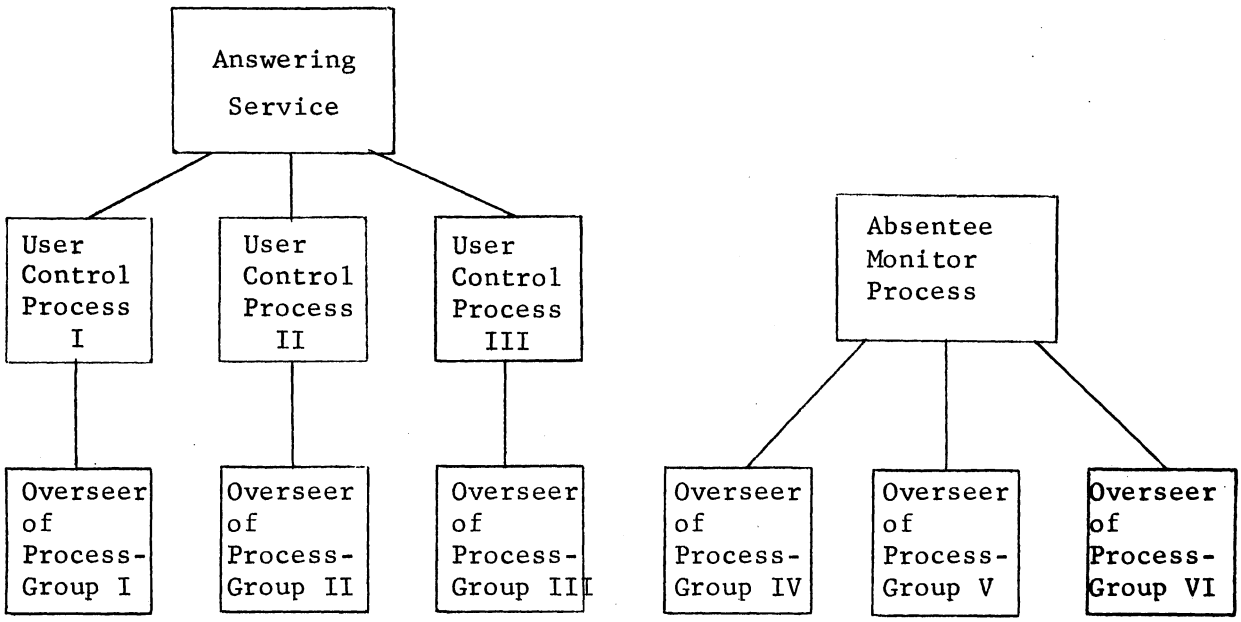
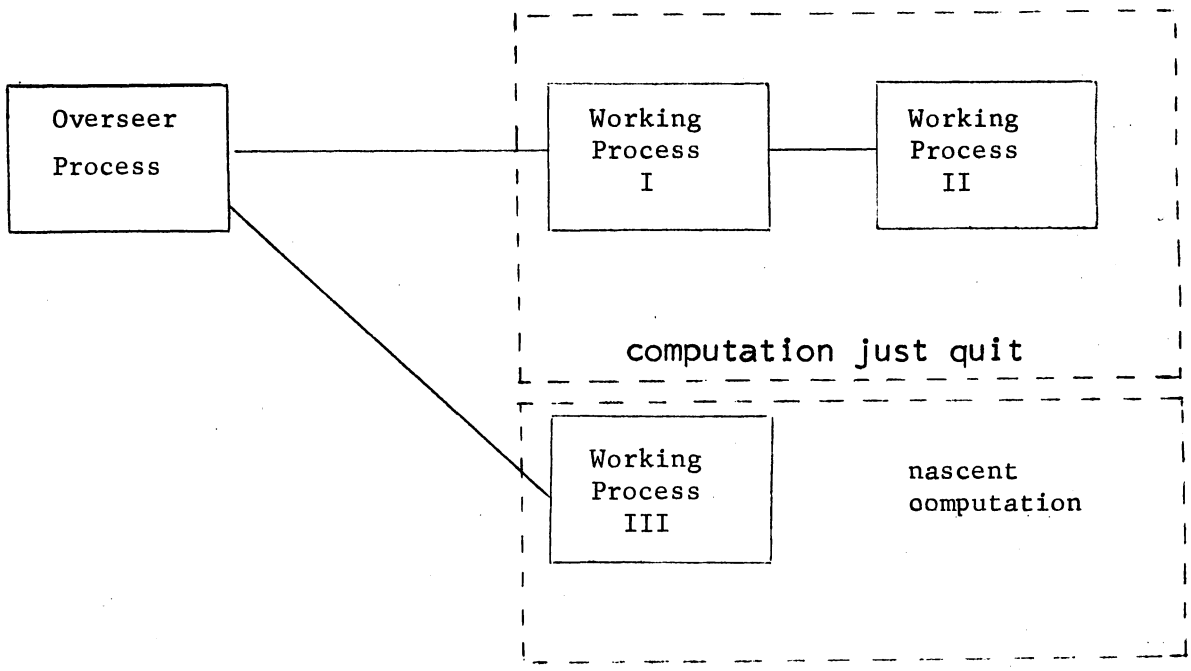


Figure I Overview of User Control

The Answering Service Process, creates one User Control Process for each dial-up. The User Control Process creates an Overseer Process if the user who dialed up logs in successfully.

The Absentee Monitor Process acts as "Answering Service" and "User Control" for all absentee logins.

Figure II - A typical user-process-group



In this example, a user was executing a command which created another process for concurrent processing. Then he pressed the quit button at his console. Now if he types the start command the two processes executing his former computation will be restarted and Working Process III destroyed. Otherwise, Working Processes I and II will be destroyed and he will continue with the fresh computation in Working Process III.