Published: 10/27/67

Identification

Routines to create and destroy working processes
create_wp, destroy_wp.
P. Belmont

Purpose

The routines create_wp and destroy_wp are used to create and destroy working processes. These procedures operate in ring 1 and are called only from ring 1 by the overseer procedure. In later versions of MULTICS, create_wp and destroy_wp will be extended to permit user ring procedures in working processes to call them. This will permit the implementation of parallel processing in user computations.

Summary

As we will see in greater detail below, the creation and use of a working process involves four main steps:

(a) creating a new process (read about process creation in BJ.2.01),

(b) creating an entry for the new process in the working process table, thus making a "working process" of the new process (read about the overseer in the sections of BQ.3),

(c) waiting until the new process is ready to accept an instruction to do some work,

(d) using a "give_call" to cause the new process to execute a call of the form:

```
call subr (arg1,...,argn)
```

(read about interprocess communication in BQ.6, about give_calls in BQ.6.08).

The destruction of a working process involves:

(a') destroying the process,

(b') updating the working process table.

A call to create_wp accomplishes steps (a) and (b). A call to destroy_wp accomplishes steps (a') and (b').

Discussion

A user process group consists of a number of processes working together to accomplish a task for the user. The processes in a user process group are of two kinds: working processes, which execute the users' commands and procedures, and the overseer process, which oversees the working processes (see BQ.3.00). The overseer in each process group maintains a variable length table, the working process table, which contains all the information that the overseer requires about the working processes in the group. When a working process is either created or destroyed, appropriate changes must be made in the working process table.

Create_wp and destroy_wp make it possible for the overseer procedure to create and destroy working processes in a manner consistent with system requirements concerning the working process table.

Let us now give a concise functional description of create_wp. The function of create_wp is to create a working process which will initialize itself, set itself up to receive give_calls, signal this readiness by setting an event in the creating process, and then wait for a sequence of give_calls in its wait_coordinator (see BQ.6.06).

Create_wp creates a new process by calling create_proc.

The procedure create_proc (see BJ.2.01), executing in the creating process, creates and begins the initialization of the new process, schedules the new process and then returns, returning the process_id of the new process. The second part of the initialization occurs in the new process; the last step of the initializing procedure is to call a procedure specified by the caller of create_proc. Create_wp specifies the procedure "start_wp", a standard procedure which makes the new process ready to receive give_calls and then signals this readiness to the creating process. After create_wp returns and the new process has signaled readiness, the creating process may use "give_call" to instruct the new process to do the user's work.

Let us explain the preceding line somewhat by giving a brief discussion of the give_call facility, (see BQ.6.08), an extension of the MULTICS interprocess communication machinery. By means of a call to give_call, a procedure in one process can ask another process to execute a call of the form:

call "ascii_procedure_name" (arg1, arg2,...,argn).

By means of calls to pass_call, any procedure in the second process can execute any give calls which have been sent to its process from other processes. One would not like to initiate parallel processing without some means of knowing when the task running in parallel had finished, and the calling sequence to give_call includes an event channel on which the originating process is to be notified of the return of the give_call'd procedure.

The procedure "start_wp" is programmed so that after initializing itself and signalling to its creating process, it waits for a give_call and executes any give_call'd procedure it receives. If this procedure returns, "start_wp" (see the implementation section) again waits for a give_call. And so on. An arbitrary number of give_calls can thus be directed to the new process, which will execute them in the order received; of course, the use of multiple give_calls makes sense only if each (except perhaps the last) of the give_call'd procedures returns.

Usage

A. Create wp

A call to create_wp takes the form:

```
call create_wp (process_name, process_id,
                event_channel_name, status);
dcl process_name char (32), process_id bit (36),
    event_channel_name bit (70), status fixed bin (17);
```

The user specifies a process name and create_wp returns the process_id, event_channel_name and the status. The new process will signal an event on the specified event channel when it is ready to receive give_calls.

B. Destroy wp

A call to destroy_wp takes the form:

```
call destroy_wp (process_id, status);

dcl process_id bit (36), status fixed bin (17);
```

Destroy_wp searches the working process table for an entry corresponding to `process_id'. If it finds such an entry, it destroys the process and updates the working process table. If it finds no entry corresponding to `process_id' it indicates an error by means of the `status' return argument.

Implementation

Create_wp:

(a) creates an event channel on which the new process will signal readiness for give_calls;

(b) allocates space for a new entry in the working process table and fills in all its items except the process_id; the "i_am_quittable" event channel is created and its name stored;

(c) creates a model process initialization table in its own process directory for the new process (the process initialization table will be copied into the process directory of the new process when that has been created);

(d) calls create_proc to create the new process;

(e) records the process_id of the new process in the new entry, threads this entry into the working process table, deletes the model process initialization table from its own process directory, and returns.

Destroy_wp:

(a) looks in the working process table for an entry corresponding to `process_id';

(b) (if no such entry is found): sets an "error" status, and returns.

(b) (if an entry is found): quits (see NOTE below) and destroys the process, destroys the "i_am_quittable" event channel, unthreads and frees the entry in the working process table, and returns.

NOTE: To quit a working process, set its quit_pending flag and test its quit_inhibit_counter (roughly, a count of locks set on important ring 1 data). If this counter is non zero, wait for an event on its associated i_am_quittable event channel. Then set its wakeup inhibit flag in the event channel table (BQ.6.04) and call quit_process.* The quit_pending flag, quit_inhibit_counter, and i_am_quittable event channel are in the process' entry in the working process table.

---

* Quit_process calls quit (BJ.3.03) but doesn't return until the process is actually quit - implying, for example, that it is out of ring 0.

Start_wp.

The process initialization machinery in create_proc does, among other things, initialize the give_call/pass_call facility, by the establishment of the requisite data structures and event channels. The procedure "start_wp" which is executed following initialization:

(a) looks up (in the process group's give_call table) the name of the event channel which is to be used by external give_calls;

(b) declares this event channel to be a call type event channel, specifying "pass_call" as the procedure to be called;

(c) gives a ready signal to the creating process on an event channel named in the process initialization table (see step c of create_wp);

(d) calls wait.

FIGURE 1

Usage of create_wp and destroy_wp. Empty boxes below indicate activity unrelated to the creation, use, and destruction of the parallel process.