Published: 10/01/68
(Supersedes: BJ.3.00, 12/07/66)

Identification

Overview of Interprocess Communication Entries
Robert L. Rappaport, Michael J. Spier

Purpose

During the `life' of a Multics process, the need arises at least once for this process to have some information furnished by some other process; we say that this process is engaged in "interprocess communication". Interprocess communication implies a synchronization of processes; a process might have to `pause' (idle) for the other process to communicate the information. By convention, for reasons of efficiency, such a process gives its processor away, or `blocks' itself, until the awaited information has been communicated, or until that specific `event' has occurred. It is then taken out of the blocked state and put into the ready state, or `awakened'. The Traffic Controller entries <u>block</u> and <u>wakeup</u> provide those basic functions.

Discussion

An <u>event</u> is anything that is observed by a process and which might be of interest to another process or maybe another procedure of the same process. An event is always associated with some information to be communicated to the interested (receiving) process. Examples of events are: the terminating of a computation, the unlocking of a shared data-base or the arrival of new input from an I/O device. These events happen outside of the hardcore ring and are known as `user-events' to distinguish them from `system-events' which happen in the hardcore ring only and which are discussed in section BJ.2.

Process `A' reaches a point in its execution where it cannot proceed until event `E' has occurred (or in other words, until some information is furnished by some other process.) It therefore calls the Traffic Controller's entry block and abandons the processor. Process `A' is now in the blocked state, which means that it no longer participates in the race for a processor, and will remain in that state until awakened by some other process.

Process `B' now executes, and observes an event. This could be event `E' for which process `A' is waiting, it could also be any other event `Q' in which process `A' might be interested some time in the future; the point is, even though process `B' knows that the observed event is of interest to process `A', it has no way of determining what process `A's current state is, whether it is waiting for some event or whether it is executing. Consequently, the notification mechanism must be such as to allow the preservation of all communicated information even though it might not be of immediate interest to the receiving process.

However, assuming that process `B' did observe event `E', it calls the Traffic Controller subroutine

```
          call wakeup (`A', `E')
```

where `A' is the target process' ID and `E' is the event information.

In order to block itself, process `A' has called

```
          call block (interaction_switch, event)
```

where `interaction_switch' is a flag to indicate whether or not the process is blocking itself awaiting human response (from a console). Process `A' now wakes up, returns from the Traffic Controller and finds in `event' the information communicated by process `B' (namely event `E'). If that information is the one it has waited for, it continues its interrupted execution, otherwise it stores that information somewhere in its memory-space, and calls block again.

Process Synchronization

Both subroutines block and wakeup manipulate the Active Process Table (APT); normally, block puts the APT entry of its own process into the list of blocked processes, wakeup finds the APT entry of the target process in the blocked-list and restores it into the ready-list. However, it is not guaranteed that a call to wakeup in behalf of some process will actually find that process in the blocked state; also, it is not guaranteed that if a process calls block because it is waiting for some event to happen that this event will happen in the futuer, it might already have happened in the past. Evidently, some further interaction is needed between subroutines block and wakeup to insure that event signals do not get lost, and that a process will not mistakenly block itself, never again to be awakened.

This interaction is provided by the process' 'wakeup-waiting' flag. This flag, which can assume one of the two values on/off is located in the process' APT entry. A call to wakeup always sets this flag to 'on'. Then, if the process is blocked, it will be put into the ready-list, else it is left in whatever state it is. A call to block will actually cause the process to abandon its processor only if its wakeup-waiting flag is 'off'; the flag's 'on' state indicates that an event signal (which might be the one awaited) has already occurred, and consequently block returns to its caller. Upon returning, subroutine block always resets its wakeup-waiting flag to 'off'.

In order to insure that no more than one process at a time manipulate the APT, that table is protected by an interlocking convention which is respected by all the processes in the system. The process that currently manipulates the APT sets a lock-word to a non-zero value, all other processes which want to access this table loop-wait until that word is reset to zero. This insures that there will never be any conflict between an awakening and a blocking process which might both try and 'grab' the wakeup-waiting flag at the very same instant.

Transmission of Event Information

Associated with block and wakeup is a paged system-wide data-base known as the Interprocess Transmission Table (ITT). This table contains as many event queues as there are receiving processes in the system. Every receiving process has in its APT entry the head of its ITT event queue. A call to wakeup causes the new event information to be appended to the target process' ITT queue.

Subroutine block, before returning, detaches the ITT queue from the process' APT entry (providing the process with a fresh, zero-length queue) and returns the detached queue to its caller, which then copies the queue's contents into its own memory space and frees the ITT space for future use.

Subroutines block is discussed in detail in section BJ.3.01 and subroutine wakeup is discussed in BJ.3.02.

Subroutines block and wakeup are called by the Interprocess Communication (IPC) facility only. IPC and the ITT are described in sections BJ.10.

Sometimes, a process has to be able to determine the state of some other process. For example, a process which intends to destroy another process can proceed with the destruction of the target process´ directory only after that process has actually entered the stopped state.

```
     call status (process_id, execution_state, load_state)
```

returns the current execution and load state of `process-id´.