Draft 6.13.68

Identification

Overview of the Interprocess Communication Facility

Michael J. Spier

Purpose

In the 'lôfe' of every process in Multics, the need arises at least once for some information to be furnished by some other process. Processes are, by definition, completely independent of one another; an observer in one process can never tell with certainty what is actually happening inside another process at any given time. However, in order to exchange information, processes must be able to communicate and communication means synchronization. The only point in the system where processes are "under control" (mainly because it is there that a process' virtual processor is managed) is in the Traffic Controller; its entries block and wakeup are the basic tools available for process synchronization. The interprocess communication facility (IPC) is the immediate (and only) 'customer' of block and wakeup and offers the additional service of transmitting (in association with each call to wakeup) a limited amount of control information from one process to another.
The Traffic Controller is described in sections BJ, familiarity with it is assumed.

Terminology

An event is anything observed during the execution of one process which may be of interest to another process or perhaps to another procedure of the same process. The IPC handles events which are of interest to non-hardcore procedures and which are known as user-events. Events which

are of interest in the hardcore ring only are named system-events and are handled by a dedicated, wired-down module; the module's name is Process Wait and Notify (PWN) and it is documented in section BJ.2 ; any reference to 'event' in sections BQ.6 implies 'user-event'.

An event is always associated with a call to the Traffic Controller's entry wakeup. A group of one or more events is always known under a collective event channel name which is the symbolic name of an event channel (which we loosely define, for the moment, as a mailbox for events).
For example: All the events which are time observations may be collectively known by event channel name "time" (or "clock" or any other agreed upon symbolic name.) Processes which happen to make time observations may put messages into event channel "time"; A process may interrogate this event channel (mailbox) and find there messages indicating that time readings have been taken at (let's say) 3, 4 and 5 o'clock.
An event channel is the basic IPC variable and is, physically, an entry in an event channel table (ECT).

Introduction

Following is a typical, and oversimplified, example to demonstrate the basics of IPC; the implemented IPC facility is much more complicated largely because of reasons of protection.
Process 'A' (sending process) observes an event 'E' which it knows to be of interest to process 'B' (receiving process'; it knows an event channel name 'C' which belongs to the receiving process and which is the receiving process' collective name for events such as 'E'. Process 'A' calls the IPC and asks it to transmit a message to process 'B' over event channel 'C'; the message contains information about event 'E', process 'A' and event channel 'C'.

The only way messages can be communicated between processes is through the use of shared segments. The IPC maintains a system-wide data base named the Event Transmission Table (ETT) which is known and accessible to all processes in the system. It therefore allocates an entry in that table and puts into it the event message. It now has to associate that message with the receiving process 'B'. To do so, it calls the Traffic Controller entry wakeup, giving it as arguments the event message and process 'B's ID. The Traffic Controller appends the message to process 'B's Active Process Table (APT) entry and wakes the process up. Process 'A' returns from the Traffic Controller to the IPC, and from there to its original procedure.

At some point of its execution, process 'B' needs some information from some other process. It knows that the information will be put in a specific event channel which is the mailbox for that type of information. It calls an IPC module named wait coordinator which interrogates the event channel. If there is a message in it, process 'B' is satisfied and resumes its execution. However, if it is empty, process 'B' must stop executing until such time as the message will be available; it calls the Traffic Controller's entry block and abandons the processor.
When the process returns from block, it knows that it returned because some other process sent it an event message; it finds that message appended to its own APT entry. However, the message may, or may not, be the one for which the process is currently waiting. The wait coordinator therefore first copies the message into the appropriate event channel (remember that the event channel name was part of the message), then loops back to the interrogation of the event channel which is of current interest, and either returns or calls block depending upon whether or not it finds the

awaited message.

Basic Interprocess Communication

We now further define an event channel as being in the receiving process' address space only, and an event channel name as being a unique identifier, This poses a certain problem, because in order to send an interprocess message the sending process needs the "mailbox" event channel name which is unique and can be known only through interprocess communication. The answer is that there is nothing spontaneous or dynamical about interprocess communication. When we say that a process reaches a point in its execution at which it needs some information from another process, we mean that when that particular procedure was coded, the programmer had a very specific type of event in mind and that he knew, at coding time, what the event channel name was to be. For example, when a process accesses a shared data base, by convention it first sets a "lock" word to a non-zero value, and before leaving the data base resets the same lock word to zero. Another process which wishes to access that data base first tests the lock word, and if it finds it non-zero it knows the data base to be "locked", and calls the IPC to wait for an event signal which would anounce the unlocking of that data base. Respecting that convention, the first process, after unlocking, sends IPC messages to all waiting processes. Now this traffic necessitates the knowledge of event channel names. They are made known by the use of the lock word as a mailbox designed for the communication of event channel names. Once that an event channel name is known, the nature of the event channel allows the transmission of some additional information which may be another event channel name thus allowing a growing complexity of event channel networks. However, the fact remains that the very first event channel, namely the lock word, was known at coding time and that without it no interprocess communication would have been possible.

In other words, the fact that an IPC message contains some information which may be an event channel name allows interprocess communication to be recursive and tree-structured; However, the recursion must have a beginning; we name that beginning "basic interprocess communication" and understand it to be any feasible means (going as far as manually-fed inter-console messages) by which a unique event channel name, known to some process, may be made known to some other process.

Normally, this problem is solved by having programmers agree, at coding time, upon some common external symbol (or upon some absolute location) within a segment which is known to both processes. There they communicate the very first event channel name.

Typical examples would be: The Process Initialization Table (PIT) in which the Overseer process communicates to the newly-created process an event channel name, the above-mentioned lock-word in which the locking process puts an event channel name over which all other waiting processes may communicate with it, the Device Signal Table (DST) where a sending process finds the event channel associat with an I/O device, etc.

Protection

IPC is a facility which does for the user what ~~he~~ the user can easily do for himself If the user is expected to devise a way in which to perform the initial basic interprocess communication all on his own, there is no reason why he should not be trusted to do all of his interprocess communication by himself. IPC is provided as a system facility that can be invoked by any user. However, it must be implemented in such a way so as to perform under the very same conditions under which a user-made IPC would have worked. The reason is simple, the user operates under the constraints of a ring

protection mechanism. The IPC facility is a system module and therefore enjoys a broader range of privileges. Carelessly implemented, it would provide the ideal means for the "unfriendly" user to completely circumvent the systems protection mechanism. The facility must therefore be implemented in a way such as to prevent it by hardware from doing in behalf of the user whatever the user cannot do for himself.

The protection of IPC is implemented as follows:

1. IPC is designed to operate in rings 1-63 only.
2. The actual message transmission between processes is done in the ring 0 ETT only. The signalling module is invoked from outside the hardcore ring to transmit a message. It automatically appends (and without the caller's ability to interfere) the caller's process-id and ring number to the ETT message.
3. The receiving process has an event channel table per ring (normally two, for rings 1 & 32, possibly 63). When it returns from block it retrieves all of its ETT messages and (still in ring 0) copies them into the corresponding ECTs by ring number.
4. All IPC modules (but for the signalling module and for the hardcore caller of block) are operating in the process' current ring and are capable of manipulating only that ring's ECT. Thus, if a user decides to either have his own version of IPC or to simply destroy his ECT, only his ring is affected.
5. IPC's ring 0 procedures, mainly the caller of block which upon return transcribes the messages into the various ECTs, must be carefully coded so as to make them indifferent to ECT contents

All control information in the ECT must be either unreferenced by the ring 0 module, or must be kept in ring 0. This is in order to assure that ring 0 will not have to rely on outer ring information.

In order to allow rapid message distribution, the event channel name has been designed in such a way as to make it self-addressing. The event channel name is a 72-bit string and is structured as follows:

```
dcl     1  ev_chn,
        2  ring bit(6),      /* ECT's ring number */
        2  address bit(14), /* channel's relative address within ECT */
        2  password bit(52);/* clock reading at channel creation time */
```

When the process returns from block and retrieves the ETT message, it finds the event channel name. It extracts the ring number, accesses the ECT of that ring at the relative address derived from the name and compares the password with the one stored in the ECT. This insures that only messages addressed to legal event channels will be distributed. Any error detected, for example the absence of a ring or the mismatching of passwords, causes the message to be discarded.

Note: The event channel name's structure will never be of interest to the user, to him it is merely a 72-bit symbolic name. Also, as the 14-bit relative address implies, an ECT is restricted to a maximum size of 16K. It is more than ample, and any overflow would most certainly be due to some faulty system design elswhere.

Implementation

This paragraph briefly describes the implementation of IPC. Details can be found in the appropriate BQ.6 sections.

The interprocess communication facility consists of three major modules:

1. The Event Channel Manager (ECM) which resides in non-hardcore rings.
2. The Wait Coordinator (WC) which resides in non-hardcore rings.
3. The signalling and reception mechanism which resides in the hardcore ring and which is sub-divided into the following
   a. The Process Signal Manager (PSM)
   b. The Device Signal Manager (DSM)
   c. The Message Dispatching Module (MDM)

The event channel manager is a collection of procedures which create, maintain, interrogate and eventually destroy event channels. The ECM operates only within its ring-priviliges and cannot access inner-ring ECTs.

The Wait coordinator is the process' event bookkeeper. It files (in association with the MDM) arriving event messages ~~in~~ into their appropriate event channels, and retrieves these messages (or calls block awaiting their arrival) whenever the need arises.

The PSM is called by a sending process to signal an event which is internal to the system, the DSM is called by a sending process to signal an event which is external to the system (I/O interrupts).