Published: 07/21/67

Identification

How to use the Interprocess Communication Facility
Michael J. Spier

Purpose

Interprocess Communication is basically very simple, in fact so simple that rules and conventions have been made so that interprocess signals be correctly interpreted on both sides of a process boundary. Upon this are constructed a number of superstructures which give the user certain sophisticated services. The set of procedures associated with these services is called the "Interprocess Communication Facility". The facility is comprised of several modules which are described in detail in MSPM sections BQ.6.03-08. This section tries to deal with the use of the Interprocess Communication Facility, it does not explain any terms on the assumption that the reader is already familiar with them, from reading the overview BQ.6.00.

Introduction

As will be seen, the Interprocess Communication Facility is built upon the Basic Interprocess Communication mechanism. In designing the separate modules, a number of constraints -- most of them peculiar to Multics -- had to be considered. Thus, when the reader reads through MSPM sections BQ.6.03-08 he invariably comes across discussions of things such as device signals, rings (!), process-groups, signalling modes, lists, queues and the such, which are discussed in detail in one section and then dropped completely in the following one, so that the reader is justified in asking himself "and where does all this leave me?". The problem is that certain kinds of events and event channels look different when viewed from the sending or receiving ends of Interprocess Communication, and still different when viewed in between sending and reception while they are manipulated within the facility. And yet, in fact, seen from a subjective point of view, be it either sending or reception, Interprocess Communication presents itself as simple and uniform.

Initial Communication

Interprocess Communication always starts with a receiving process' willingness to receive messages from a sending process. The receiving process creates an event channel for that purpose and makes the associated event channel name known to the sending process via the Basic Interprocess Communication mechanism. This means that it places the event channel name within an agreed upon data structure in a segment known and accessible to both processes, assuming that the sending process knows <u>who</u> the receiving process is. If the sending process does not previously know who the receiving process is, then the receiver's process_id must be communicated as well as the event channel id. Example: The prospective receiving process wants to access a systemwide data base and finds it interlocked. It therefore creates an event channel and puts the event channel name plus its own process_id in a location which is associated with and agreed upon by the users of that data base. It then calls <u>wait</u> specifying the newly created event channel as argument. When the process that has locked the data base finally unlocks it, it looks up the associated location in which it finds a process_id and an event channel name. It sends to the receiving process (process id) an event signal over its event channel (event channel name). The receiving process recognizes the received event signal as indicating that the data base is now potentially accessible.

As we can see, this communication depends upon the agreement between both processes (as to the associated location), the sending process' willingness to honor the agreement (looking up the location after unlocking the data base and the sending of an event signal) and the correct interpretation of certain data by both processes (the sending process interprets the associated locations non-zero value as being a process_id and an event channel name, the receiving process interprets the reception of a signal over its event channel as indicating that the data base is now potentially accessible).

Another example would be the arrival of an I/O interrupt. The process which detects the interrupt can be any process. It knows that it has to communicate that specific event to some receiving process yet does not know the receiving process' id nor the associated event channel name. It therefore honors a systemwide agreement by which it puts a message in a mailbox which is associated with the I/O device. In that mailbox it also finds a process_id for which it calls <u>wakeup</u> (remember that a call to <u>wait</u> by a receiving process is associated with a call to <u>block</u> by the wait coordinator).

The receiving process wakes up, looks up its I/O device associated mail boxes and transcribes their contents into the appropriate event channels.

As can be clearly seen from these examples, the kind of event to be signalled (or received) and the degree of ignorance of one another call for different kinds of agreements between communicating processes. As a rule, unless a sending process knows a receiving process' process id and event channel name, no communication via the Interprocess Communication Facility is possible.

Basic Interprocess Communication

The Interprocess Communication Facility cannot be used unless a basic interprocess communication has been transmitted by the (future) receiving process to the (future) sending process. Any means of communication (going as far as interconsole messages) may be used in order to pass an event channel name to a prospective sending process.

By "Basic Interprocess Communication" we refer to the act of a prospective receiving process which places an event channel name (and possibly its own process id) in a location within a segment which it shares with the prospective sending process, in accordance with an agreement between both processes. How and where this information is stored is up to both processes to agree upon. However, by convention, the reception of such a communication is done according to the following rule: The location's zero value is interpreted as "no basic interprocess communication established", and its non-zero value is accepted and interpreted as being an event channel name (and possibly a process id). The receiver of such a communication (the prospective sending process) has to know the kind of event to which this event channel name is dedicated.

This rule is necessary because, contrary to an Interprocess Communication Facility event signal, a basic interprocess communication cannot be accompanied by a control communication (wakeup) and consequently must have one specific reserved value (in this case zero) which is interpreted as an indicator, the remaining possible values consituting the actual message.

How to create an event channel

When a process is interested in being notified about some event by another process, it first has to create an event channel dedicated to that kind of event. A procedure within that process calls create_ev_chn (see MSPM section BQ.6.04), and is handed back, upon return, the newly created channel's event-channel-name which it then communicates (as a basic interprocess communication) to the sending process.

A call to create_ev_chn carries the following implication: Create_ev_chn finds out the caller's validation ring number and stores it in the created event channel. This ring number is the channel's protection level for all calls emanating from within the receiving process. This includes not only calls to modify or delete the event channel, but also calls to the wait coordinator associated with this channel (remember that an event channel cannot be read directly by a user's procedure but must be interrogated through the wait coordinator's entries "wait" or "test_event"). If the user intends to wait upon the event channel from a ring n procedure, he should not create the channel by a procedure which resides in a higher privilege ring or, if he does, he should set the creating procedure's validation ring number to n before calling create_ev_chn.

How to determine an event channel's mode

When a process creates an event channel, it has to specify the channel's signalling mode. It is determined according to the following rules:

1. A device_signal_channel always uses the event_count_mode.

And for all other channels:

2. An event channel (be it an event_wait or an event_call channel) which has only one sending process signalling over it, and where the receiving process does not want to know each individual event's event_id uses the event_count_mode.

3. Only when the receiver is interested in knowing specific event_ids, or when more than one sending process signal over the same event channel and it is of interest to know which event was signalled by what process does one have to use the event_queue_mode.

When interrogating an event channel, it is imperative to know the event channel's signalling mode because upon it depends the amount of precise information returned to the interrogating procedure. The interrogation of an event channel, regardless of its mode, returns a single event indicator at a time. In order to read n event indicators from an event channel, it has to be interrogated n times.

An event_count_mode_channel returns an event indicator which contains the following:

a. Its own event_channel_name

b. An event_id which corresponds to the first event signalled since that channel was last reset to zero.

c. A zero-value process id.

An event_queue_mode_channel returns an event indicator which contains

a. Its own event_channel_name

b. The event's event_id

c. The sender's process_id.

An event channel interrogation always returns the channel's event_channel_name because the calling sequence to the Wait Coordinator allows the caller to specify a list of event channels to be interrogated in which case it is important to be able to identify the event channel whose signal is returned by the Wait Coordinator.

How to grant another user access rights to an event channel

A newly created event channel is by default accessible only to member processes of the creator's process-group. It has a channel-access-list containing a single entry which is the creator's process-group id. By convention, an event channel with no channel-access-list is accessible to any process which knows its event channel name, a channel with a channel-access-list is accessible only to member processes of process-groups whose id is on that list. Consequently, a newly created event channel is out of other process-groups' reach. In order to grant another process-group access to an event channel, a call has to be made to give_access, which has as arguments an access-switch and an access-list.

If the access-switch has the value "0"b, the whole channel-access-list currently associated with the event channel is deleted, including one's own process-group id, and the channel is made accessible to all (in this case, the access_list argument is ignored).

If access_switch has the value "1"b, the specified access-list is appended to the current channel-access list.

How to send an event signal

A procedure within a sending process becomes aware of an event which it knows to be of interest to a receiving process. It wants to signal this event. In order to do so it has to know whether the event is a system interrupt or not.

> System Interrupt The sending process knows the device which originated the System Interrupt. It calls set_dev_signal (device_index), where device_index is a number associated with the I/O device. (See MSPM section BQ.6.07.)
>
> Other than System Interrupt The sending process must know of a location within its address space which contains a receiving process' id (labeled name_of_process) and of another location which contains an event channel name which belongs to the receiving process (labeled name_of_event). Depending upon whether the sending procedure resides in ring 0 or rings 1-63 it calls IPGECM$set_event or ECM$set_event respectively, giving as arguments "name_of_process" and "name_of_event". An additional argument is an event_id generated by the sending process. (See MSPM sections BQ.6.04-05.)

How to be notified of an event

We assume that an event channel has been created by a receiving process and its event channel name made known to a sending process.

The receiving process may simply want to inquire whether or not an event has been signalled over that channel. It invokes test event which immediately returns the required information.

The receiving process may wait for an event to occur by calling wait, which will return to its caller only if the awaited event signal was received over its associated event channel.

The third possibility is for a receiving process to define a procedure which is to be executed whenever an event signal is received without specifically interrogating the event channel (test_event,wait). It associates the procedure with the event channel by calling decl_ev_call_chn (see MSPM sections BQ.6.04-06).

Figure 1 is a diagram to illustrate the different attributes of an event channel as seen from the sending and receiving ends of the Interprocess Communication Facility.

As seen by Sending Process

	Type of event.	
	external to memory (hardware interrupt)	internal to memory
Type of event channel	device signal channel	communication channel
Signalling mode	event count mode	event count mode event queue mode
To signal, call: in ring 0 in rings 1-63	dstm$set_dev_signal	IPGECM$set_event ECM$set_event

As seen by the Receiving Process

	Type of event channel	
	event wait channel	event call channel
receiving mode	event count mode event queue mode	event count mode event queue mode
explicit interrogation by test_event	yes	yes
explicit interrogation by wait	yes	no
implicit execution of an associated procedure	no	yes

Figure 1

Event channels as seen by sending and receiving processes

Note the difference in point of view. The sending process is concerned with the event's origin, the receiving process with the event channel's type.