Published: 07/24/67

Identification

Events and Event Channels
Michael J. Spier

Purpose

Interprocess Communication, as described in the overview (MSPM Section BQ.6.00) is accomplished in the following manner: A process becomes aware of an event that it knows to be of interest to another process. This process shares a common data base with the process that wishes to be informed of the detected event. It therefore places data in the shared data base, then calls "wakeup" for the second process. The second process wakes up (if blocked), reads the data and interprets it to be an Event Indicator.

A process that wishes to receive event indicators from another process must provide a data base to be shared between it and the sending process. Such a data base is called an Event Channel. We use the expression "To signal an event over an event channel" to describe the action of a sending process which writes a certain information in a segment known to him and at least one other process, then calls wakeup for the other process, and the expression "to receive an event signal over an event channel" when we wish to say that the receiving process wakes up, looks into the very same shared data base and finds the information written there by the sending process.

As explained in the overview (MSPM section BQ.6.00), the Interprocess Communication Facility is an extension of the control communication provided by the Traffic Controller (process exchange), associating a limited amount of control information with each call to block and wakeup. This section deals with the definition of events, event indicators and event channels.

Introduction

An event is anything which is observed during the execution of some process (henceforth called the sending process) and which is of interest to some other process (receiving process) or perhaps some other procedure of the first process. An event is a unique occurrence and can happen only once.

Associated with an event is an event id which is a unique bit string. Furthermore, an event is associated with an Event Channel, which has a unique bit string name known as an Event Channel Name. Several events of a similar kind may be associated with a common event channel which has a single event channel name.

An Event Channel is a data base shared between a receiving process and one or more sending processes. An event channel is always associated with exactly one receiving process and exactly one event channel name. It is the receiving process that creates, maintains, reads and eventually deletes an event channel. A sending process can only write into certain parts of an event channel, and that only under the receiving process' control.

An event channel is a first-in first-out queue of Event Indicators. The term "event indicator" refers to the control information associated with an event. Its value may vary, and usually becomes more precise as it is passed from the sending process, via the Interprocess Communication Facility, to the receiving process. An event indicator always implies a unitary value associated with an event and kept track of (internally) by the Interprocess Communication Facility, insuring that all events communicated to a receiving process will be remembered. The event indicator may also contain an event id, and a sending process id. The value of an event indicator depends upon the event channel's mode of signalling.

Whenever a sending process adds a new event indicator to a receiving process' event channel, he also calls wakeup for the receiving process (see Overview, MSPM Section BQ.6.00). Whenever reference is made to the signalling of an event over an event channel, it implies the writing of an event indicator into an event channel and a call to wakeup.

Event Class

An event is signalled by a sending process to an interested receiving process over an event channel which belongs to the receiving process. From the receiving process' point of view, all events are received in the same manner, namely, the process receives a wakeup signal whereupon it looks into the event channel and finds in it the event indicator.

From the sending process' point of view, there are different classes of events, each class necessitating a different method of signalling. Events are divided into two classes by their origin:

a. Process communication
b. Device signal interrupts

Process Communication Events are internal to memory. They are generated by the sending process. Consequently, the sending process is fully aware of the event and of the receiving process to which the event is to be signalled. It therefore knows how to access the associated event channel, and can do so directly. Event channels associated with process communication events are called communication channels.

Device Signal Interrupts are external to memory. They may arrive at any given moment and interrupt any process that may be running at that time on the processor associated with the I/O device that originated the interrupt. This process now becomes a sending process even though it was not the originator of the event and has no means of accessing the event channel associated with that event. The only information accessible to such a sending process is a table called the Device Signal Table. It contains one entry per I/O device. An entry consists of a receiving-process-id and a device-signal-channel. The sending process looks up the device signal table for the entry corresponding to the I/O device from which the interrupt originated. It stores an event indicator in the device signal channel and calls "wakeup" for the receiving process.

The receiving process has a regular event channel associated with each device signal channel in the device signal table. Whenever the receiving process wakes up, it looks up all the entries in the device signal table which are associated with it. If it finds a device signal channel that has been signalled over, it transcribes its contents into the corresponding event channel. This is done automatically by the wait coordinator. The user of the wait coordinator has the impression as if the event indicator associated with the hardware interrupt has been written into the event channel directly by the sending process. (For details see MSPM section BQ.6.07.)

There is a further subdivision of Process Communication Events for reasons of protection

a. Intra-Process-Group Events
b. Inter-Process-Group Events

Intra-process-group events are events which are communicated between processes belonging to the same Process Group.

Inter-process-group events are events which are communicated between processes which do not belong to the same process group.

Inter-process-group communications must be controlled and restricted so as to provide the receiving process with maximum protection from "unfriendly" processes.

Event Channels

An event channel is, by definition, a data base that may be accessed and written by more than one processor at a time. It therefore needs to be made invulnerable to the danger of being destroyed through simultaneous multi-processor access. A way is used in which event channels can be written and read without actually locking the whole data base and making some process wait until it be unlocked. It is described in MSPM Section BQ.6.04.

All event channels belonging to one receiving process are grouped together in a pair of segments in that process' directory called the Event Channel Table and the Event Channel Working Queue. The event channels are manipulated by a set of procedures known as the Event Channel Manager. The working queue is but an extension of the Event Channel Table. Unless otherwise specified, the name "Event Channel Table" will imply both segments.

A receiving process' Event Channel Table and Event Channel Manager reside in the administrative ring. All processes belonging to the same Process Group may access the table in the administrative ring. Consequently, intra-process-group events are signalled by calling a ring 1 procedure which is part of the event channel manager. (The name of this procedure is set_event.)

Processes that do not belong to the same process group are allowed to communicate, but communication must be severely controlled so as to provide maximum protection to the receiving process. Processes that are allowed to communicate inter-process-group events may therefore access the receiving process' event channel table only at a ring 0 level of protection.

Another problem to be considered is ring protection. Every event channel is protected by two ring numbers, ring and signalling ring, indicating the lowest level of protection at which a procedure may access the event channel within the receiving and sending processes respectively.

The protection mechanism is explained in detail in MSPM Section BQ.6.05. The Device Signal Table resides in ring 0 and is in wired down core. It is serviced by the Device Signal Table Manager.

The Event Channel Table

Each event channel entry in the Event Channel Table occupies a contiguously addressable zone of virtual memory. All entries are of the same fixed length and have the same format. An entry consists of one Header and two Sub Channels. The header contains all the control information associated with an event channel and is used by the event channel manager. A sub channel is the part of an event channel into which a sending process writes an event indicator. There are two sub channels per event channel table entry. This is in order to allow safe multiprocessor access to an event channel. Normally, one sub channel is always considered unlocked and a sending process is free to enter information into it. The second sub channel is locked and is used by the receiving process which reads the event indicators out of it. Whenever the locked sub channel is exhausted (reset to zero), the receiving process switches the locking status of both sub channels, thus unlocking the reset sub channel to future sending processes and locking the unlocked one so that it could read it.

A sub channel has capacity enough to accommodate only the most elementary form of event indicator, which is the sum of the event count bits. This mode of signalling is called the Event Count Mode. Event count bits are internal to the Interprocess Communication Facility and invisible to the user. One event count bit corresponds to one signalled event.

If an event is to be signalled giving more precise information, signalling is done by using the Event Queue Mode. In this mode, a list of event cells is appended to each of the sub channels. Each cell of the list can contain one event indicator. This mode allows an event indicator to consist of an event id and a sending process id in addition to the (invisible) event count bit. Section BQ.6.01 explains how to determine an event channel's mode.

An event channel header may be read by any process that has access to it. By convention, it may be written by the receiving process only.

Of the two sub channels, unless they both be interlocked (channel unavailable to a sending process for reasons of creation or deletion), only one at a time is available to sending processes which have the right to write into it.

The table structure

The following describes in detail the structures of the Event Channel Table and the Working Queue. The attached figures 1 & 2 are the corresponding EPL declarations.

(Note: items marked * are implementation dependent and therefore not explained in this section. Items marked ¤ are related to the Wait Coordinator and discussed in MSPM section BQ.6.06.)

The Event Channel Table is a segment consisting of a header -- containing control information -- followed by an array of fixed length event channel entries.

The event channel table header contains:

a. The receiving process' id.

b.* A pointer to the Working Queue segment.

c.* A pointer to the Event Channel Hash-code Segment. This segment is used in order to retrieve an event channel entry by name, using a hash-code algorithm. This segment is invisible to the user and therefore not described in this section. (See MSPM section BQ.6.09.)

d. The receiving process' wakeup switch. (See MSPM section BQ.6.04.)

e.¤ The Wait Coordinator's call-wait priority switch.

f.¤ A pointer to the head of the event-call-channel list.

g. A pointer to the head of the device-signal-channel list.

h.¤ A pointer to the associated-procedure list (see working queue).

i.* The current dimension of the event channel entry array.

j.* The current dimension of the working cell array.

k.* The current dimension of the process-group id array.

l.* The length-2 of en event channel table entry.

m.* The length-1 of a working queue cell.

n.* The number of process-group-id pointers in a channel access cell.

o. The event channel entry array.

Each event channel table entry consists of a header followed by the two sub-channels.

The event channel header contains:

a. The event channel name identifying the event channel.

b. Information associated with the event channel protection

   b1. A pointer to a channel access list in the working queue segment. If the pointer has the value zero then, by convention, the channel is accessible to any sending process that knows the channels' name. Otherwise, access is granted only to member processes of process-groups whose ids are entered in this channel's access list.

   b2. Ring number. It is the validation ring number of the procedure which created this event channel and serves to protect the channel from being accessed (other than for signalling purposes) by procedures residing in lower privilege rings.

   b3. Signalling ring number. This ring number is specified by the procedure that created this event channel and indicates the lowest privilege ring from which a sending-process procedure may access this channel.

c. The event channel's type as seen from a sending or receiving process' point of view.

   c1. Sending type. The type indicates either communication ("0"b) or device signal ("1"b) channel.

c2. Receiving type. The type indicates either event-wait ("0"b) or event-call ("1"b) channel.

d. Mode. This is the event channel's signalling mode and indicates either event-count ("0"b) or event-queue ("1"b) signalling.

e. Information associated with event call channels.

e1. A pointer to the associated procedure cell, in the working queue.
e2. A pointer to the associated data base.

e3. The event call channel level number, which controls recursive calls to the associated procedure.

e4. The event-call-channel list thread. All event call channels are linked together in a list structure for easy searching.

e5. The event call channel's merge priority number, specified by the channel's creator, which indicates the channel's position within the event-call-channel list.

f. Information associated with device signal channels.

f1. Device index. It is an index which identifies the associated entry in the Device Signal Table. It is internal to, and set up by, the Device Signal Table Manager.

f2. The device-signal-channel list thread. All device signal channels are linked together in a list structure for easy searching.

(In reference to items e4 & f2 note that a single event channel may be part of both lists concurrently.)

<u>The two sub channels contain</u>:

g. A common header

g1. The sub-channel interlock word. It can assume the values 0-2 only and is an index pointing to neither or the currently <u>unlocked</u> sub-channel.

g2. Two usage counters, indicating how many sending processes are currently engaged in manipulating the corresponding sub-channels.

h. The two sub-channels, each consisting of:

h1. Variables used for the event-count mode signalling. They are always used by the sending process, the event-queue mode being an extension of the event-count mode.

h11. Event-id corresponding to the first event signalled over this sub-channel since it was last reset.

h12. Event-count. A count of the events which have been signalled over this sub-channel.

h2. A pointer to the event queue list in the working queue. It is used in case of event-queue mode signalling.

The Working Queue segment contains two arrays:

1. An array of process-group ids
2. An array of working cells.

The process-group ids are in a separate array in order to keep one entry per process-group (these things are 50 characters long!).

A process-id entry contains:

a. An interlock word indicating whether or not this entry is currently free.

b. The process-group-list thread. All entries are linked together in a list structure for easy searching.

c. A usage counter indicating how many event channels have pointers to this entry. When the usage counter gets reset to zero, the entry is automatically deleted.

d. A 50-character string process-group id.

The working cell array

There are three types of working cells:

1. Event queue cell
2.* Associated procedure cell
3. Channel access cell

All working queue cells have a fixed format header:

a. An interlock word, indicating whether or not the cell is free. The value stored in this location is the index of the event channel for which the cell is reserved. A cell with a zero-value interlock word is considered to be unassigned.

b. A type. It may assume the values 1-3 to indicate whether this is an event-queue (1), associated-procedure (2) or a channel-access (3) cell.

c. A list thread. Cells are usually linked into list structures, depending upon the type of cell (as explained below).

In addition to the above header, the cells contain:

Event queue cell:

d. An event id.

e. A sending process id.

There is one event-queue-list per event sub-channel, provided that it is an event-queue-mode channel.

Associated procedure cell:

d. An inhibit word. This word contains the current recursive call level number of the event call channel that has caused the associated procedure to be called. A recursive call will be inhibited unless the calling channel has a higher level number.

e. A pointer to the associated procedure's entry point.

f. A usage counter indicating the number of event-call-channels currently associated with this procedure. When the usage counter resets to zero, this cell gets automatically deleted.

All associated-procedure cells are linked to form one associated-procedure-list.

Channel access cell:

d. An array of pointers to process-group ids in the process-group id array.

Normally, there is one access cell associated with each event channel entry (unless the channel is accessible to all processes, in which case there is no channel access cell). If the channel has an access list which can not be accommodated in one access cell, additional cells are appended to form a channel-access-list.

Declaration of the Event Channel Table header

```
declare  1 ev_chn_tbl ctl(my_ect),
         2 rec_prcs bit(36),
         2 my_wrq pointer,
         2 my_echt pointer,
         2 wakeup bit(1),
         2 cl_wt_prior bit(1),
         2 cl_list fixed bin(17),
         2 dev_sigl_list fixed bin(17),
         2 ass_prcd_list fixed bin(17),
         2 ect_dim fixed bin(17),
         2 wrkq_dim fixed bin(17),
         2 ids_dim fixed bin(17),
         2 entr_rest fixed bin(17),
         2 cell_rest fixed bin(17),
         2 grp_id fixed bin(17),

/* Declaration of the ECT entries array */
         2 entr(my_ect->ev_chn_tbl.ect_dim),
           3 name bit(70),
           3 rest (my_ect->ev_chn_tbl.entr_rest)  fixed bin(17);
```

Declaration of an Event Channel Table entry

```
declare  1 chn ctl(my_ect),

/* The event channel header */
         2 hdr
           3 name bit(70),
           3 acc,
             4 ptr fixed bin(17),
             4 ring fixed bin(17),
             4 signl_ring fixed bin(17),
           3 type,
             4 send bit(1),
             4 rec bit(1),
           3 mode bit(1),
           3 ev_cl,
             4 p_ptr fixed bin(17),
             4 d_ptr pointer,
             4 level fixed bin(17),
             4 list fixed bin(17),
             4 prior fixed bin(17),
           3  dev_signl,
             4 inx fixed bin(17),
             4 list fixed bin(17),
```

Figure 1

The Event Channel Table

```
/* The two sub-channels */
          2 intlk,
            3 word fixed bin(17),
            3 use(2) fixed bin(17),

          2 sub(2),
            3 ev_id bit(70),
            3 count fixed bin(17),
            3 queue bit(36);
```

Figure 1 (continued)

The Event Channel Table

Declaration of the Event Channel Table Working Queue

```
declare  1 wrk_q ctl(my_ect),

/* The process-group id array */
         2 ids(my_ect->ev_chn_tbl.ids_dim),
           3 intlk bit(36),
           3 list fixed bin(17),
           3 usage fixed bin(17),
           3 grp_id character(50),

/* The working cell array */
         2 cell(my_ect->ev_chn_tbl.wrkq_dim),
           3 intlk bit(36),
           3 rest (my_ect->ev_chn_tbl.cell_rest) fixed
                  bin(17);
```

Declaration of the three cell structures

```
/* The event-queue cell */
declare  1 ev_q ctl(my_ect),
         2 intlk bit(36),
         2 type fixed bin(17),
         2 list bit(36),
         2 ev_id bit(70),
         2 send_id bit(36);

/*The associated-procedure cell */
declare  1 ass_prcd ctl(my_ect),
         2 intlk bit(36),
         2 type fixed bin(17),
         2 list bit(36),
         2 inhib fixed bin(17),
         2 ptr pointer,
         2 usage fixed bin(17);

/* The channel access cell */
declare  1 acc ctl(my_ect),
         2 intlk bit(36),
         2 type fixed bin(17),
         2 list bit(36),
         2 group_id(my_ect->ev_chn_tbl.grp_id) fixed bin(17);
```

Figure 2

The Working Queue