Published: 07/27/67

Identification

The Wait Coordinator
Michael J. Spier

Purpose

Within the life of a Multics process, the need arises at least once for some information to be provided by another process. Unless this need be satisfied, the process cannot continue its execution. The traffic controller provides an entry point block to be called in such a case. The process that is awaiting the information calls block, knowing that some other process will wake him up again as soon as the information becomes available.

The Wait Coordinator is a supervisor module which coordinates the process' calls to block with the event signals that it receives. It allows a procedure within that process to wait for a specific event to happen, momentarily ignoring all other signals which might have been received during the waiting period. Except for a small number of hardcore supervisor routines, no procedure within a process is allowed to call block directly, all such calls must go through the Wait Coordinator.

The reader should be acquainted with the Interprocess Communication Facilities as described in MSPM sections BQ.6.00 - .05.

Introduction

The Wait Coordinator is the receiving end of Interprocess Communication. Normally, whenever a receiving process is blocked, it is blocked in the Wait Coordinator, and it is there that it wakes up as a result of some sending process' event_signalling. The Wait Coordinator knows for what specific kind of event the process is waiting, so whenever the process wakes up in the Wait Coordinator, a check is made to determine whether or not the wakeup was associated with the event that is currently waited for. If not, the Wait Coordinator calls block again. If it is, a return is made to the Wait Coordinator's caller.

The Wait Coordinator may be called to wait for one of several different kinds of events to happen. It returns to its caller upon detection of the first of these events to have been signalled.

Consider the following example. A process is engaged in reading punched cards, transcribing them onto magnetic tape and listing them on a printer. Each I/O device has a <u>Device Manager Process</u>. Before our process may initiate a new I/O operation, it must <u>wait</u> for an event - signalled by the device manager - indicating that the previous I/O operation has successfully been terminated. In order to understand how the Wait Coordinator functions, let us analyze the following 3 ways of writing such a card-transcription program.

<u>Solution 1.</u>

```
1. read a card
2. wait for card to be read
3. write magnetic tape
4. wait for tape to finish
   writing
5. print a line
6. wait for printing to be
   finished
7. go to step 1.
```

Solution 1 is linear in its nature. By using a buffering system, I/O waiting times may be reduced:

<u>Solution 2.</u>

```
1. read first card
2. wait for first card to be
   read
3. read next card
4. write previous card on
   magnetic tape
5. print previous card on printer
6. wait for card to be read
7. wait for tape to finish writing
8. wait for printing to be
   finished
9. go to step 3.
```

Even though the events relevant to steps 7 and 8 might have happened, no return will be made from the Wait Coordinator in step 6 before the card reader has finished reading. If, while the process is still blocked in step 6, event signals associated with the magnetic tape drive or printer are received, these signals will be momentarily ignored by the Wait Coordinator.

The next solution calls for very large I/O buffers.

Solution 3.

1. Wait for either card-reader or tape drive or printer to be available.
2. If card reader is available read card into buffer then go to step 1.
3. If tape drive is available and buffer not empty write magnetic tape then go to step 1.
4. If printer is available and buffer not empty print a line then go to step 1.

In step 1 we call the wait coordinator specifying 3 kinds of events to wait upon. A return will be made from the wait coordinator as soon as any one of these 3 events has happened. It is up to the user to determine which event it was.

Event channels used to be waited upon in the way described above are called Event Wait Channels. The wait coordinator, when asked to wait upon such an event channel does not return to its caller nor take notice of other event wait channels until the specified event has been signalled.

There is a second kind of event channel called an Event Call Channel. Such a channel is not waited on. The receiving process is always, implicitly, waiting for events to be signalled over event call channels and whenever such a signal is received, it causes the Wait Coordinator to automatically call a procedure which is associated with that event call channel.

Let us reconsider solution 3 of the example. Let us change steps 2 3 and 4 into subroutines, as follows

SUB 2. Read card into buffer. Return.

SUB 3. If buffer not empty, write magnetic tape. Return.

SUB 4. If buffer not empty, print a line. Return.

Step 1, and the conditions "is device available" in steps 2, 3 and 4 may be dispensed with, simply by transforming the card reader, tape drive and printer event-wait-channels into event-call-channels and by associating them with SUB 2, SUB 3 and SUB 4 respectively. Whenever an event is received over an event call channel (and the receiving process is executing in the wait coordinator) a call will automatically be made by the wait coordinator to the event call channel's associated procedure.

From the receiving process' point of view, the wait or call attributes of an event channel are mutually exclusive, and correspond to the receiving process' explicit or implicit interest in an event. The reaction of the Wait Coordinator to a signal received over an event channel differs accordingly. An event signal received over an event-wait-channel will result in a return from the wait coordinator, whereas an event signal received over an event-call-channel will result in a call being issued by the wait coordinator.

A user may make a call to the wait coordinator in order to be informed whether or not an event has been signalled over an event channel regardless of the event channel's wait/call attributes (test_event).

Calls to the wait coordinator

A process may inquire whether or not one or more events have happened by invoking:

```
f=wc$test_event(chn_list,ev_ind,sts)
```

```
declare (chn_list(n),ev_ind(3)) bit(70), sts bit(36),f bit(1);
```

chn_list is an array of event channel names corresponding to the event channels to be interrogated.

ev_ind is an event indicator as described in MSPM sections BQ.6.01,03,04.

sts is a return status.

Test_event returns to its caller as soon as

a. An event indicator has been found (f="1"b), or
b. All the specified event channels (chn_list) have been tested (f="0"b).

A process may wait for an event to happen by calling

```
call wc$wait(chn_list,ev_ind,sts)
```

where the arguments are the same as those for test_event. For both procedures, "sts" returns error status information such as channel access violation.

The "wait" procedure will not return to its caller unless an event signal was received over one of the event channels specified in "chn_list". Chn_list is interrogated sequentially.

Event call channels

An event-wait-channel may be declared to be an event-call-channel by calling

```
call ecm$decl_ev_call_chn(ev_chn,proc_ptr,data_ptr,
                          prior,level-sts)
```

An event-call-channel may be redeclared to be an event-wait-channel by calling

```
call ecm$decl_ev_wait_chn(ev_chn,sts)
```

(The arguments to these calls are defined in MSPM section BQ.6.04).

An event-call-channel is associated with a procedure to be called by the wait coordinator whenever an event signal is received over the event-call-channel. The associated procedure may be any user's procedure and is invoked by the wait coordinator as follows

```
call [associated procedure] (data_ptr,ev_ind)
```

```
declare data_ptr pointer, ev_ind(3) bit(70);
```

where data_ptr points to a data base associated with the event-call-channel (more than one event-call-channel may be associated with the same procedure) ev_ind   is the event indicator read out of the event channel.

Note: The associated procedure may freely call the wait coordinators entries "wait" and "test_event".

The receiving process' event-call-channels are linked in a list structure called the event-call-channel-list, which is searched sequentially by the wait coordinator. A user may wish to assign a certain search priority value to decl_ev_call_chn's "prior" argument, which determines the event-call-channel's place in the list. The value of "prior" may be any arbitrary value and is determined by the user.

Consider (figures 1-3 may be a helpful reference) the case in which event-call-channel X has several event indicators queued up. A certain procedure of the receiving process calls wait, a call that cannot be satisfied because the specified event has not yet happened. The Wait Coordinator therefore looks up the event-call-channel-list and reads one event indicator out of channel X whereupon it calls the associated procedure which in turn calls the Wait Coordinator's entry "wait" for event-wait-channel Y. Now, if channel Y has not been signaled over,the Wait Coordinator will once more look up the event-call-channel-list, read a second event indicator out of channel X and attempt to issue a recursive (and erroneous) call to the associated procedure.

In order to avoid such recursive calls, the Wait Coordinator possesses a special interlocking mechanism. When declaring an event-call-channel (see decl_ev_call_chn, MSPM section BQ.6.04), the caller specifies as argument a level number to be associated with the event-call-channel (and which is not associated with either of the channel's ring numbers). When calling the associated procedure, the Wait Coordinator memorizes this level number in the associated-procedure-cell (see MSPM section BQ.6.03) and considers the procedure to be inhibited to any event-call-channel which has an equal or lower level number. The original contents of the associated-procedure-cell's inhibit word is saved in the Wait Coordinator's current stack frame and restored upon return from the associated procedure.

A second possible error condition may arise if a procedure, (especially the event call channel's associated procedure) calls wait for the event-call-channel. A convention was made by which such calls to wait will result in an error return from the Wait Coordinator, without having accessed the specified event-call-channel.

The user who wishes to wait upon a channel declared to be an event-call-channel will first have to change the channel's type into an event-wait-channel (by calling decl_ev_wait_chn) and upon return from the Wait Coordinator redeclare the channel to be an event-call-channel.

Decl_ev_call_chn checks with the file system to make sure that its caller is not trying to circumvent the system's protection mechanism by declaring an event-call-channel so that he could indirectly call a procedure that is directly inaccessible to him. The call to decl_ev_call_chn is rejected if it turns out that the associated procedure is not to be accessed from the caller's ring.

When considering all the arguments, implied restrictions and possible error conditions associated with (ecch) event-call-channels, the reader must remember that, --once properly declared-- the event-call-channels and their associated procedures are capable of performing "off line" tasks, thus exploiting more efficiently the processor time alloted to the receiving process.

Event channel protection

Event channels are protected by their ring number which determines from out of which rings they may be accessed. All the Event Channel Manager modules (but for set_event and read_event) check this ring number against their caller's validation ring number before accepting the call. Set_event, as described in MSPM section BQ.6.05, performs a similar validation check in conjunction with the event channel's signaling ring number. Read_event is the only Event Channel Manager module to accept calls without any checks, because it is inaccessible to the user and can be called by the Wait Coordinator only. It is the Wait Coordinator who, when called, first checks to make sure that the caller resides in a ring from which he has access to the event channels he wishes to wait on.

The reason for taking this responsibility away from read_event and giving it to the Wait Coordinator is as follows: The caller may specify a list of event channels to wait on. The Wait Coordinator issues separate calls to read_event for each one of these event channels, and whether or not he calls read_event for all these channels depends upon whether or not an event indicator was read (remember that the Wait Coordinator is satisfied with the first event indicator which read_event returns). Consequently, the illegal access condition may or may not be detected depending upon the way event_chn_list was specified and upon the arrival of specific event signals.

The Wait Coordinator, when called (be it wait or test_event), gets its caller's validation ring number (sb|3) and compares it to the ring numbers of all the event channels to be interrogated. Any access violation causes an error status return from the Wait Coordinator.

When the Wait Coordinator interrogates the event-call-channel-list, no validation check is made because the channel is interrogated in behalf of a known user (the procedure that declared the channel to be an event-call-channel) and all checking was performed by decl_ev_call_chn.

Description of the Wait Coordinator

The wait coordinator has two functions

a. Test the channels specified in the "chn_list".

b. Test all of this process' event call channels (the event-call-channel-list).

As already stated, a positive result to the first check causes the wait coordinator to return to its caller, a positive result to the second check causes the wait coordinator to issue a call to the event call channel's associated procedure. Both functions a and b are independent (and exclusive) of one another. The user may specify the order in which both functions are to be carried out by setting the wait_call_priority switch in the Event Channel Table Header. This is done by calling one of the following entries to the Event Channel Manager

```
call set_wait_prior

call set_call_prior
```

The switch then remains set in the specified position until another such call.

The attached figures 1-3 are general flow charts of the Wait Coordinator. Figure 1 shows the Wait Coordinator's main-program, its entries test_event and wait and the way the call_wait_priority switch is interrogated. Figures 2 and 3 show the two primitives event_wait and event_call which are called by the Wait Coordinator's main program.

The Wait Coordinator invokes the Event Channel Manager function read_event which returns either the "event indicator found" or the "event indicator not found" conditions.

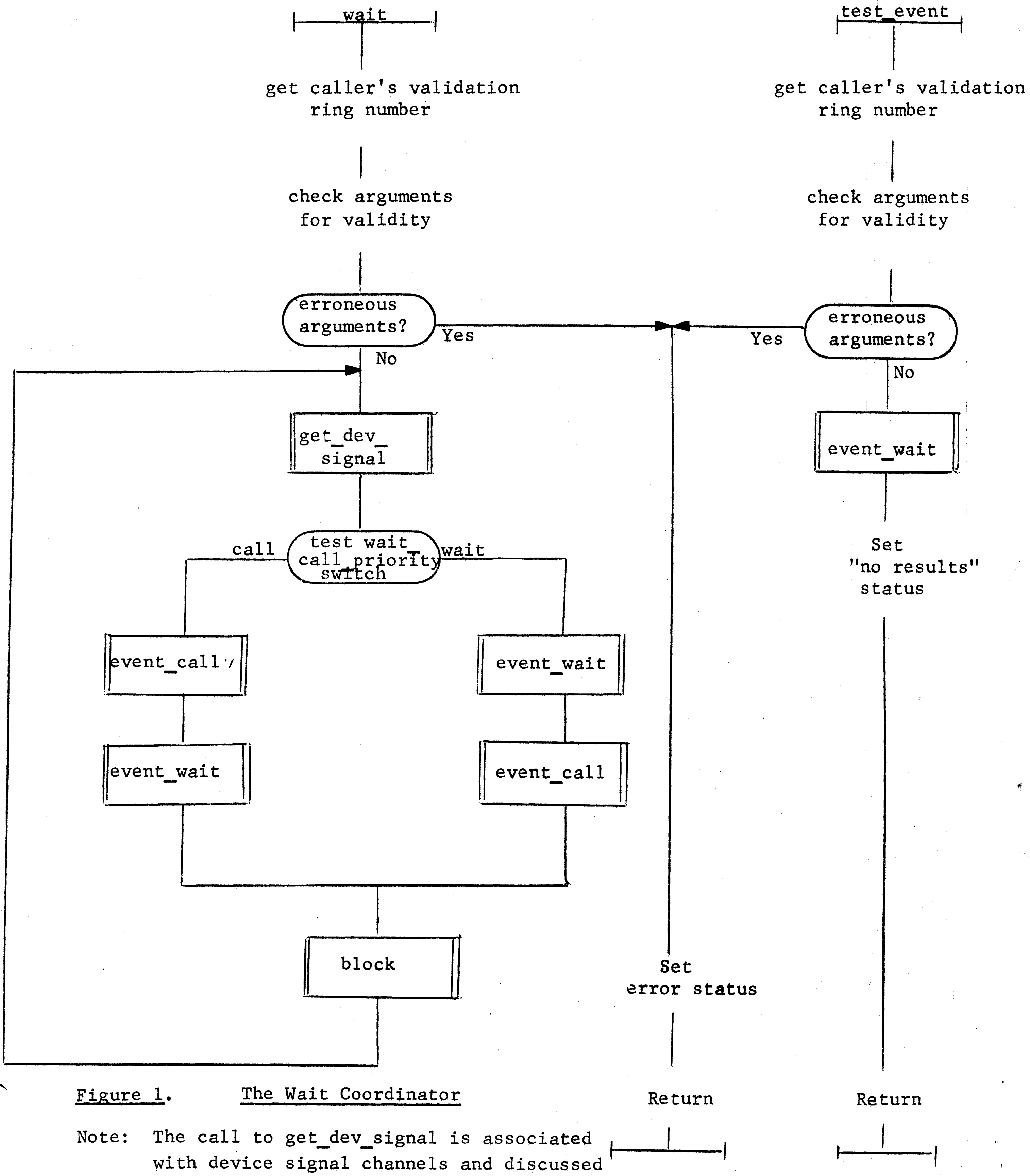


Figure 1. The Wait Coordinator

Note: The call to get_dev_signal is associated with device signal channels and discussed in MSPM section BQ.6.07,

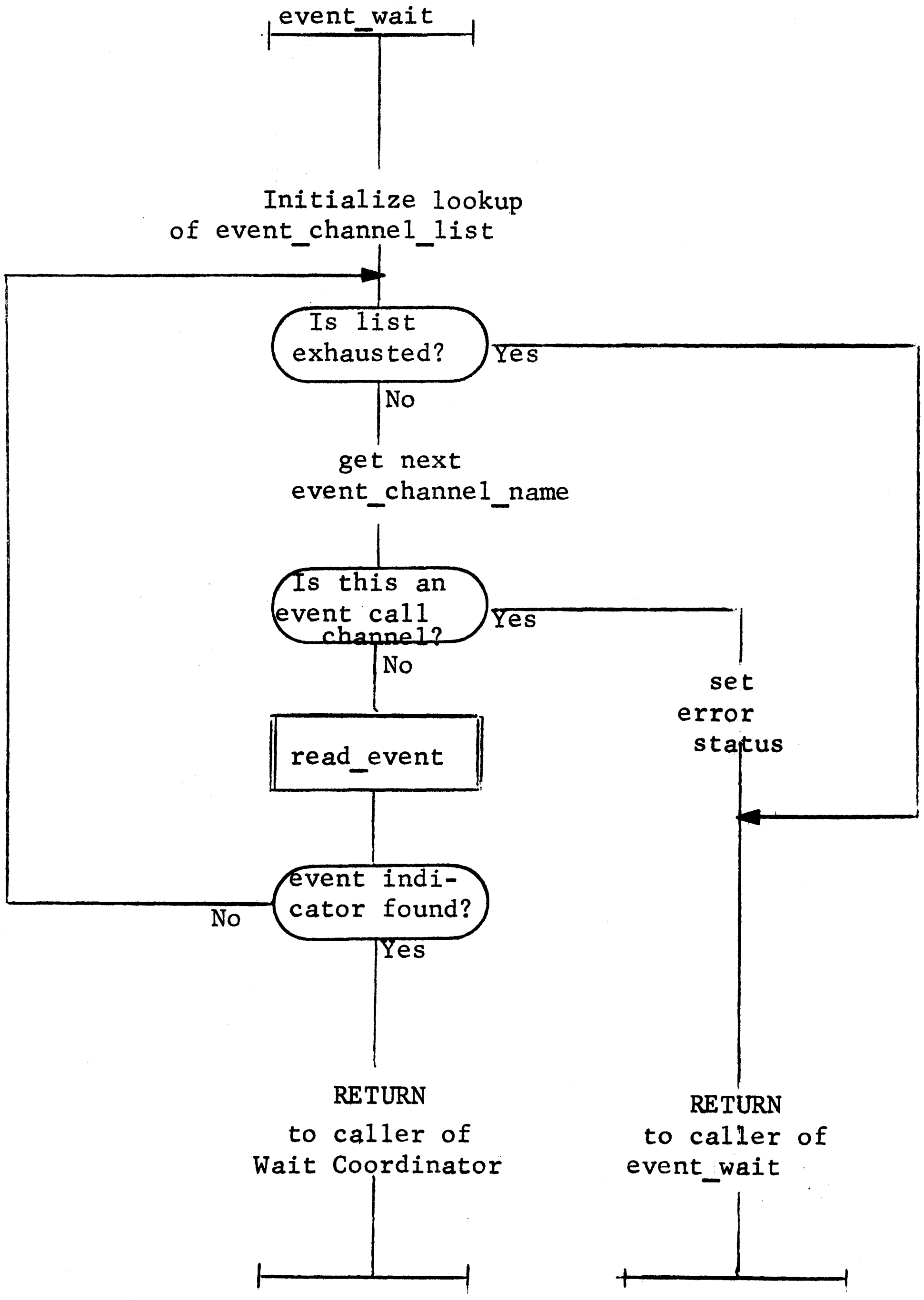
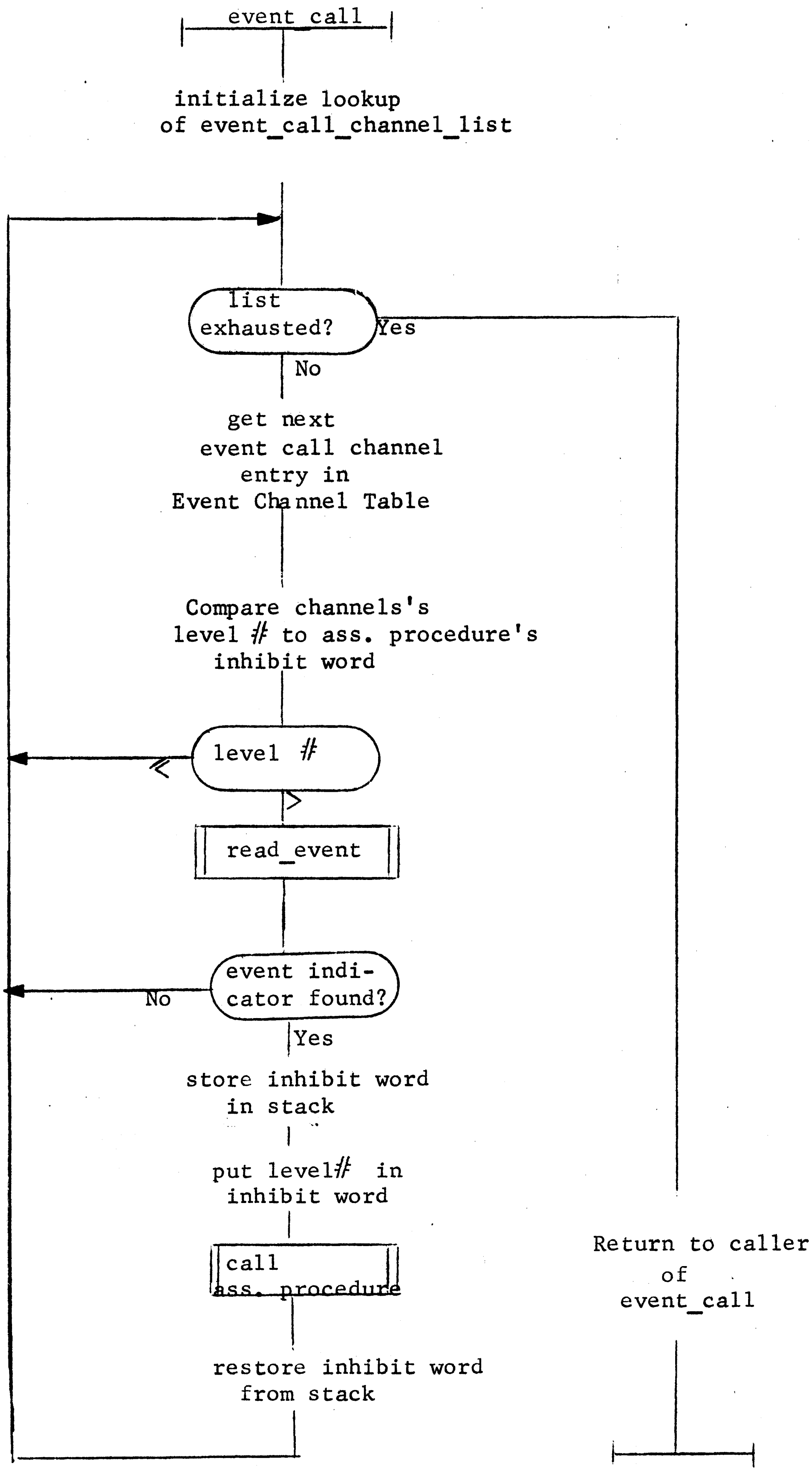

event_wait
Initialize lookup of event_channel_list
Is list exhausted?
Yes
No
get next event_channel_name
Is this an event call channel?
Yes
No
read_event
set error status
event indicator found?
No
Yes
RETURN to caller of Wait Coordinator
RETURN to caller of event_wait

Figure 2.

event_wait

Figure 3.


event call
initialize lookup
of event_call_channel_list
list
exhausted?
Yes
No
get next
event call channel
entry in
Event Channel Table
Compare channels's
level # to ass. procedure's
inhibit word
level #
<
>
read_event
event indi-
cator found?
No
Yes
store inhibit word
in stack
put level# in
inhibit word
call
ass. procedure
restore inhibit word
from stack
Return to caller
of
event_call