Published: 8/29/66

Identification

The Call-Passer
K. J. Martin, B. A. Tague (See note)

Purpose

Within Multics it is necessary that a procedure in one process be able to call a procedure in another process, passing arguments. For instance, every time a user logs in, three processes are created which must be able to communicate with one another. (The processes are known as the Overseer, the Device Manager and the Working Process, and are, collectively, a process-group of the user). Both processes involved in an interprocess call must belong to the same process-group created for one user. Other means (described in Section BD.8.03 on inter-process-group communications) are available for communication between process-groups.

Discussion

Refer to Figure 1 during the following discussion. Solid lines are calls; small numbers indicate the order of calls. Dashed lines indicate references to data. When a procedure, the caller, wishes to make an interprocess call it issues the following call:

call give_call (process-name, entry-name, event-name, arg1, arg2, ..., argn).

process name — is the symbolic name of the process in which the called procedure should be executed. Each process-group contains a process-group-wide data base associating an assigned symbolic name with a process id for each process of the process-group. The symbolic name of a process is unique within the process-group and is standard for each type of process. Section BO.1.04 on logging in defines the method of symbolically naming processes. Give_call converts the symbolic name to the process id.

entry-name — contains the ASCII name of the entry point which is being called.

event-name — is a bit string in which give_call can store an event identification number by calling set_event in the Wait-Coordinator.

argi are the arguments to the procedure "entry-name.

The procedure give_call is one of the two procedures which make up the call-passer. Give_call exists as a procedure in the same process as the calling procedure, or "caller". The other procedure of the interface module, accept_call, exists as a procedure in the same process as the called procedure, or "callee".

The job of give_call is to accept a call of any form and make the arguments available to another process. Since the number and mode of the arguments of the call are not known before the call occurs, give_call cannot readily be written in PL/I. Give_call takes the arguments of the call and makes up an entry in the work queue, a data base which is common to both processes. The information in each entry is:

1. Symbolic name of entry-name, the procedure to be called.
2. Process identification of the process from which the call is originating.
3. The event, event-name.
4. A count of the number of arguments, argi.
5. For each argument:
   a) six words containing the four- or six-word specifier of the argument if there is a specifier associated with the argument. If the argument is a non-string scalar and therefore has no specifier, the six words contain four meaningless words and an ITS pointer to the data.
   b) The slot name locations in the file system of the execution copies of the segments containing the arguments, dope and specifications. Slot names are obtained by a special call to directory control, supplying a segment number. Two slot names are returned for each segment number. The slot names identify the directory where the segment may be found, and the entry (or slot) within the directory which points to the segment.

The unique identification number of event_name was established by give_call by calling set_event in the Wait Coordinator.

After having made the arguments available to the called process, give_call calls the wakeup entry in the Traffic Controller and wakes up the called process. Give_call then returns to the caller procedure. The caller procedure may do as it wishes, knowing that accept_call will note the event, event_name when the callee returns.

The called process wakes up in the procedure, accept_call. The calling process must be sure that accept_call will be invoked or returned to as a result of the wakeup. Accept_call operates on each of the arguments, argi, in an attempt to make the segments which contain the arguments and their dope and specifications known to the called process. The call, estblseg, in directory control is given the slot name location of a segment, and makes the segment known to the process. The new segment number of the segment is then placed in the argument list pointer or in the pointer(s) of the specifier (item 5a of the work queue entry) for the appropriate argument.

When all arguments are known to the process, accept_call builds a call to the callee procedure using items 1, 4 and 5 of the work queue entry. Note that the actual arguments have not been touched. Only the argument list pointers and pointers in specifiers have been altered to make the arguments known in the called process. Note also that pointer variables and labels (which are data items and not specifiers) are not suitable arguments for inter-process calls because the segment number in the pointer variable or label is likely to be wrong in the called process. Since error returns (label data) are not allowed, status return arguments can be used to notify the caller of errors detected by the callee. On-condition and signal PL/I statement pairs may not occur across an inter-process call because of the existence of two stacks instead of one.

Accept_call calls the callee procedure with the artifically built call. Because accept_call has built a normal-looking call, the callee operates normally; it never knows the call came from another process. When it has finished it returns to accept_call. Accept_call then notes the event which is item 3 of the work queue entry, with item 2 as the process to wake up. Accept_call calls block in the Traffic Controller and remains blocked until another call occurs.

The caller procedure can detect the completed event as it sees fit.

Note:

The author of this section, K. J. Martin, is being transferred to another subproject. B. A. Tague joined the Central System Control subproject shortly before publication of this section. Although he was not involved in the authorship of this section, he is aware of the issues and will be working on the task. Therefore, comments and criticism of the ideas presented may be directed to him.

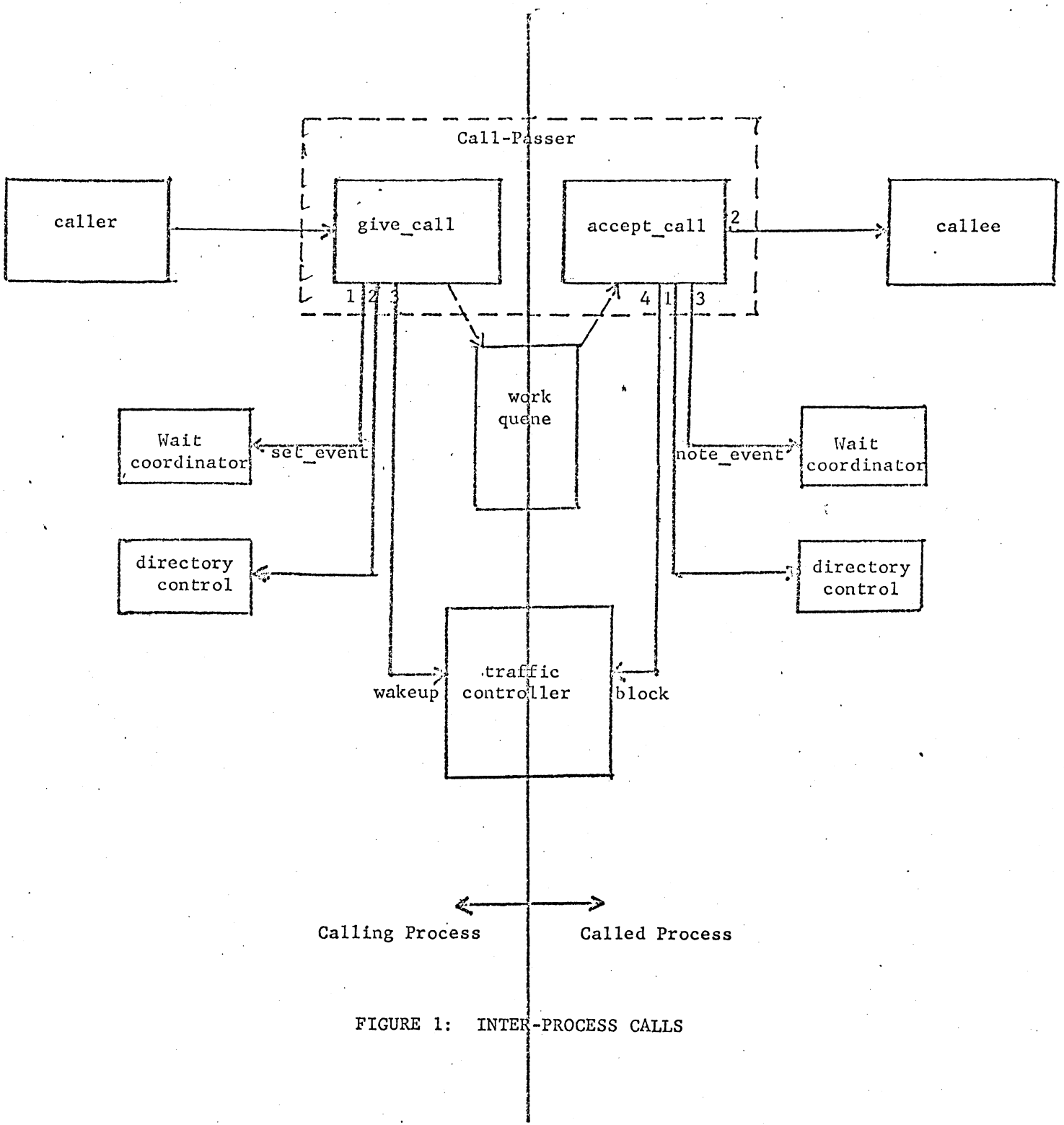

FIGURE 1: INTER-PROCESS CALLS