To: Distribution
From: Eleanor Donner
Date: 07/11/79
Subject: Improvements to IPC

SUMMARY

This MTB proposes that the IPC (interprocess communication) facility be reimplemented and enhanced. The primary reasons are:

1) Maintainability

Originally coded in 1968, the collection of programs comprising IPC can be recoded and restructured to be more straightforward and clearer. New features of PL/I, new system facilities and structured programming techniques can be utilized.

2) Better management of ECT (event channel table)

The contorted management of the ECT can be made simpler using areas. The restriction that all ECT entries be placed in a single segment will be removed. The circumstances of copying only some of the ITT messages from hardcore per call will be eliminated.

3) Performance

As part of the implementation of better maintainability and ECT management, performance improvements will accrue. Further, consideration of very large users of IPC, such as the answering service, will be given.

The MTB consists of a discussion of the current problems and proposed solutions of ECT storage, a summary of format changes in ECT entries, a list of possible future extensions, and is followed by module descriptions and PL/I structures of the data concerned.

Multics project internal working documentation. Not to be reproduced or distributed outside the Multics project.
PROBLEM

Background

The ECT (Event Channel Table) is a per ring database containing

1) ECT header

2) Entries for event wait channels
   These generally remain in the ECT from the time they are
   created until the process is destroyed.

3) Entries for event call channels
   These generally remain in the ECT from the time they are
   created until the process is destroyed.

4) Entries for additional information needed for event call
   channels
   These are known as trailer entries.
   These generally remain in the ECT from the time they are
   created until the process is destroyed.

5) Entries for event messages associated with either event
   wait or event call channels
   These remain in the ECT for a short time, until the
   messages for a channel are read.

6) Entries for itt messages
   These are copied from the ITT (Interprocess Transmission
   Table) in ring 0.
   These remain in the ECT for a short time until they are
   converted to event message entries.
The first 5 objects are managed entirely in the outer ring receiving IPC messages. The ECT entries for itt messages are constructed from slots in the ITT - a per system ring 0 data base containing all IPC messages for all processes. (In the ITT, there is a single linked list of messages for each process managed by the traffic controller. The head of the list for each process is pointed to by a variable in the pds. The list contains IPC messages for all rings. IPC messages are added to the end of the list. The freeing operation currently is defined to deallocate the entire list of ITT messages for a process.) These messages in the ITT are converted into itt ECT entries when the hardcore block primitive is called. The entries for ECT event messages are constructed from ECT itt messages entries by user ring IPC when block is called in the outer ring.

Allocation of ECT entries

Currently the header of the ECT and the first 15 entries are stored in internal static of a process. If more than 15 entries are needed, hcs_assign linkage is called to allocate a block of 15 more entries. In the first block, a fixed number of entries (currently 10) is reserved for itt message entries. This reserved block is the last 10 entries of the first block of the ECT.

There are some serious problems with the manner in which the ECT is managed.

1) If there are more than 10 messages in the ITT to be copied into a given ring's ECT, ring 0 is unable to copy them all. If this situation occurs, a bit is set in the ECT header indicating that the user's ECT is full. The outer ring must call into ring 0 again to copy out the ITT messages. If there are many messages waiting, repeated calls must be made. (1)

2) The scheme of creating itt message entries in the user ring's ECT by the hardcore block primitive works improperly in a multiple ring environment. If more than one ring has messages in the ITT, the present behavior can result in lost wakeups. Messages from the ITT for all rings are copied into the appropriate ring's ECT. If the ECT for a ring (other than the calling ring) becomes full before any messages for the calling ring are retrieved, wakeups are lost. This has happened more frequently as more use of ring 1 is made. It

(1) This situation was worse in the past. Under some circumstances, ECT itt message entries could be overwritten as event message entries were created.
is more prone to occur in the initialiser process.

3) The scheme of allocating a block of 15 ECT entries has performance ramifications when a process either creates a large number of channels or receives a large number of messages. The blocks can be allocated in different pages without any control by the system.

4) Allocating the entries in internal static may have some problems in a multitasking environment.

SOLUTION

One solution to the itt message entry problem is to copy only the messages for the calling ring. This involves changing the traffic controller's discipline of managing the ITT - specifically allowing the freeing of individual messages. Since the problem of controlling the overflow of the ITT itself has not been solved, I am reluctant to do so at this time. This change to IPC would make overflow more likely. Once the ITT overflow is under control, IPC should be changed in this manner.

While the ITT message entries do not strictly speaking have to be put in the ECT, it is very convenient to do so for debugging purposes and to conserve segment and page usage.

A solution to the above difficulties is to use an area of "reasonable" size, which is itself allocated in the system free area. All components of the ECT will be allocated and freed using the standard system area mechanism.

Using an area rather than allocating individual entries has the effect of minimizing scattering of ECT entries. The ECT won't be located using internal static but via a pointer in the stack header; thus it will work properly for run units and for tasking.

When the area is exhausted, another area of a larger size will be created. This can be done in one of two ways:

1) Make the first area extensible. The area package will create a new segment in the process directory. If the size of the initial area is chosen properly, a second segment will be created only for heavy users of IPC.

2) Adopt a slightly more complicated scheme of creating a list of areas. When an overflow occurs, the current area will be marked as full. A new area will be allocated and will become the current area. The IPC allocator will create a list of areas of progressively larger sizes. The maximum size of these areas will approach a page. Initially there
will be two sizes 1) an area large enough to hold 20 assorted entries and 2) one large enough to hold 40 entries.
(These numbers may be changed upon experimentation.)
I prefer the first approach but would welcome comments.

SUMMARY OF ECT ENTRY FORMAT CHANGES

1) Event wait channels and event call channels will be doubly threaded into two lists. Currently only event call channels are threaded. This change will allow a display tool to be written that can find ECT entries easily. It will allow more comprehensive consistency checks of ECT entries to be made.

2) Packed pointers will be used for threading purposes and in the event channel name instead of relative pointers. This will allow placement of ECT entries in multiple segments. Currently the process is terminated if more ECT entries are needed than can be put in a single segment. Occasionally this has occurred in the initializer process and might reasonably occur in a transaction processing setting.

3) The procedure associated with an event call channel will be changed to be a PL/I entry variable rather than a pointer variable. New entry points will be added to create an event call channel and to convert an event wait channel to a call channel. Both of these will supply an entry variable. Need has arisen to make active internal procedures the targets of an event call. In order to save space, the entry variable kept in the event call entry will consist of two packed pointers - a pseudo packed entry variable.

4) The total number of wakeups received over a channel will be kept rather than a bit indicating that the channel has been used.

5) Static channel information in the header will be removed, as they are obsolete.

6) Trailer entries will be combined with event call entries.

7) The pointer to the ECT header will be placed in the stack header.

8) The size of each ECT entry will increased from 8 to 12 words.
OTHER CHANGES

1) An entry point will be provided for subsystem writers and system use to set the ECT area size, overwriting the system default. If the ECT is not yet created, the ECT initial area will be created using the size defined by the caller. If the ECT is created already, a status code will be returned to indicate that the ECT has been initialized. If the second allocation scheme is adopted, all subsequent areas will be of the given size. This is envisioned as a useful tool for large users of IPC, like the initializer process or transaction processing.

2) A tool will be provided to display the ECT. It will be able to dump it in octal and display it symbolically. It optionally will accept as input the pathname of an ECT so that dead processes may be debugged.

STATUS CODES

The practice of returning nonstandard status codes will be continued (unfortunately) for current entry points of IPC. New entry points will return standard status codes.

FUTURE IMPROVEMENTS

Optimize the act of a process sending itself wakeups. Eliminate calls into ring 0. This would speed up the answering service.

Provide the ability for IPC to be replaced in the user ring. Remove the knowledge of the ECT format from ring 0. Provide new entry points to ring 0 block and change management of the ITT by pxss.

Provide an IPC command similar to the io_call command.

Implement a more efficient search of the event call list, using a hashing scheme.
Name: ipc_

Entry: ipc_$create_event_call_channel

This entry point creates an event-call channel in the current ring.

Usage

declare ipc_$create_event_call_channel entry (entry, ptr,
fixed bin, fixed bin(71), fixed bin(35));

call ipc_$create_event_call_channel (procedure_value,
data_ptr, priority, channel_id, code);

where:

1. procedure_value (Input)
   is the value of the entry point of the procedure to be invoked when an event occurs on the new channel. The procedure entry point may be an internal procedure.

2. data_ptr (Input)
   is a pointer to a region where data to be passed to and interpreted by that procedure entry point is placed.

3. priority (Input)
   is a number indicating the priority of this event-call channel as compared to other event-call channels declared by this process for this ring. If, upon interrogating all the appropriate event-call channels, more than one is found to have received an event, the lowest-numbered priority is honored first, and so on.

4. channel id (Output)
   is the identifier of the event channel.

5. code (Output)
   is a standard system status code.
Entry: ipc$_dcl_event_call_channel

This entry point changes an event-wait channel into an event-call channel.

Usage

declare ipc$_dcl_event_call_channel entry (fixed bin(71),
  entry, ptr, fixed bin, fixed bin(35));

call ipc$_dcl_event_call_channel (channel id,
  procedure_value, data_ptr, priority, code);

where:

1. channel id (Input)
   Is the identifier of the event channel.

2. procedure_value (Input)
   Is the value of the entry point of the procedure to be invoked when an event occurs on the specified channel. The procedure entry point may be an internal procedure.

3. data_ptr (Input)
   Is a pointer to a region where data to be passed to and interpreted by that procedure entry point is placed.

4. priority (Input)
   Is a number indicating the priority of this event-call channel as compared to other event-call channels declared by this process for this ring. If, upon interrogating all the appropriate event-call channels, more than one is found to have received an event, the lowest-numbered priority is honored first, and so on.

5. code (Output)
   Is a standard system status code.
Name: set_ect_size_

This is an internal interface to be used by system programs that create a large number of event channels and for whom the performance of interprocess communication is an issue. This entry point sets the initial block of event channel table entries for the current ring to a specified size. Normally it is called before the event channel table for the current ring is initialized.

Usage

```plaintext
declare set_ect_size_ entry (fixed bin, fixed bin(35));
call set_ect_size_ (ect_size, code);
```

where:

1. **ect_size** (Input)
   is the size in entries of the initial block of the event channel table for the current ring.

2. **code** (Output)
   is a standard system status code. It may be `error_table_$ect_already_initialized`. 
display_ect
display_ect

Name: display_ect

The display_ect command prints the state of an ECT (Event Channel Table). The ECT to be displayed can be indicated by a pathname or by a virtual pointer or may be omitted. In this last case, the ECT for the user's process for the current ring is selected. Several options are provided to select types of ECT entries displayed and to select various formats.

Usage

display_ect {pathname} {-control_args}

where:

1. pathname
   is either the pathname of a segment or a virtual pointer to a segment, containing the ECT to be displayed. If a virtual pointer with an offset is supplied, it is assumed to point to the ECT header. If no offset is specified, the command uses a heuristic to find the ECT header. If no pathname is specified, the ECT for the user's process for the current ring is selected.

2. control_args
   may be chosen from the following:

   -channel channel-id, -chn channel-id
     prints information about event channels whose name is a substring of channel-id.

   -wait, -wt
     prints information about event-wait channels. This is the default.

   -no_wait, -nwt
     does not print information about event-wait channels.

   -call, -cl
     prints information about event-call channels. This is the default.

   -no_call, -ncl
     does not print information about event-call channels.

   -itt
     prints event messages copied out of ring 0 from the
display_ect

ITT.

-no_itt, -nitt
does not print event messages copied from the ITT.
This is the default.

-all, -a
prints information about all event channels,
including unused ones.

-queued, -queue, -q
prints information only about used channels. This is
the default.

-ring n, -rg n
displays the ECT from ring n in the user's process,
rather than the current ring. This control argument
can be used only if a pathname was not given.

-brief, -bf
suppresses or shortens some of the output.

-long, -lg
prints the full text. This is the default.

-header, -he
prints the information in the ECT header.

-no_header, -nhe
does not print the information in the ECT header.
This is the default.

-interpret, -it
interprets process identifiers in event messages.
This is the default. See the Access Required section
below.

-no_interpret, -nit
does not attempt to interpret process identifiers in
event messages.

-octal, -oc
prints the contents of event channels in octal
format.

-no_octal, -noc
does not print event channel information in octal.
This is the default.
-short, -sh
prints octal information formatted for an 80 character width carriage.

-no_short, -nsh
prints octal information formatted for a larger carriage. This is the default.

-debug, -db
prints forward and backward threads in each item.

-no_debug, -ndb
does not print forward and backward threads. This is the default.

Access Required

Read access is required to the answer table in order to interpret process identifiers. If this access requirement is not satisfied, process identifiers will not be interpreted.
BEGIN INCLUDE FILE ... ect_structures.incl.pl1 ... June 1979 */

dcl ectp    ptr;  /* pointer to ECT header */
dcl ectep    ptr;  /* pointer to individual ECT entry */

dcl 1 ect_header  aligned based, /* structure of the Event Channel Table header */
   2 counts,  /* size in entries of ECT */
   3 entry_count  fixed bin,  /* number of event wait channels */
   3 wait_count  fixed bin,  /* number of event call channels */
   3 call_count  fixed bin,  /* number of event message entries */
   3 ev_message_count  fixed bin,  /* number of itt message entries */
   3 itt_message_count  fixed bin,

2 entry_list_ptrs,  /* head of event wait channel list */
   3 first_waitp  ptr unaligned,  /* tail of event wait channel list */
   3 last_waitp  ptr unaligned,  /* head of event call channel list */
   3 first_callp  ptr unaligned,  /* tail of event call channel list */
   3 last_callp  ptr unaligned,  /* head of itt message list */
   3 first_ittp  ptr unaligned,

2 area_info,  /* head of ect area list */
   3 first_areap  ptr unaligned,  /* tail of ect area list */
   3 last_areap  ptr unaligned,  /* pointer to current area */
   3 current_areap  ptr unaligned,  /* size in words of next ect area */
   3 next_area_size  fixed bin (18),

2 meters,  /* total wakesps sent on all channels */
   3 total_wakesps  fixed bin (33),  /* wakesps sent on wait channels */
   3 total_wait_wakesps  fixed bin (33),
   3 total_call_wakesps  fixed bin (33),

2 flags,  /* space for various flags */
   3 wait_or_call_priority  bit (1) unaligned,  /* = "0"b if wait chns have priority */
   3 unused  bit (17) unaligned,  /* = "1"b if call chns have priority */
   3 mask_call_count  fixed bin (18) unsigned unaligned,  /* number of event call chns masked */

2 fill  (18) fixed bin;  /* pad to 36 words */
fixed bin static options (constant) init (1);
fixed bin static options (constant) init (2);
fixed bin static options (constant) init (3);
fixed bin static options (constant) init (4);

1 wait_channel -
2 word_0
3 unused1
3 type
2 next_wait_champ
2 prev_wait_champ
2 word_3
3 unused2
3 inhibit_count
3 wakeup_count
2 name
2 first_ev_messp
2 last_ev_messp
2 unused3

aligned based,
aligned,
fixed bin (17) unaligned,
fixed bin (17) unaligned,
/* Event wait channel - type = WAIT */

ptr unaligned,
ptr unaligned,
aligned,
itbit (1) unaligned,
fixed bin (17) unaligned unsigned,
/* number of times message reception has been inhibi:
fixed bin (18) unaligned unsigned,
/* number of wakeups received over this channel */
bit (72),
ptr unaligned,
ptr unaligned,
(4) fixed bin;
/* event channel name associated with this channel */
/* pointer to last message in queue */

aligned based,
aligned,
fixed bin (17) unaligned,
fixed bin (17) unaligned,
/* Event call channel - type = CALL */

ptr unaligned,
ptr unaligned,
aligned,
itbit (1) unaligned,
fixed bin (17) unaligned unsigned,
/* "1'b if call to associated proc in progress */
fixed bin (18) unaligned unsigned,
/* number of times message reception has been inhibi:
bit (72),
ptr unaligned,
ptr unaligned,
(4) fixed bin;
/* event channel name associated with this channel */
/* pointer to last message in queue */
/* pointer to associated data base */
/* procedure to call when message arrives */
/* pointer to entry point */
/* pointer to stack frame */
Event message - type = EV_MESSAGE */

pointer to next message for this channel */
event message as returned from ipc_specific */
72 bit message associated with wakeup */
process id of sender */

"1"b if device signal */
"0"b if user event */
ring of sending process */
pointer to associated event channel */

ITT message - type = ITT_MESSAGE */

pointer to next itt message entry in ECT currently */
event message as returned from ipc_specific */
event channel name */
72 bit message associated with wakeup */
process id of sender */

"1"b if device signal */
"0"b if user event */
ring of sending process */
<table>
<thead>
<tr>
<th>dcl</th>
<th>1 event_channel_name</th>
<th>aligned based,</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 ecte_ptr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ptr unaligned,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fixed bin (3) unaligned</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/* pointer to channel entry in ECT */</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/* ring number of ECT */</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/* identifier unique to the process */</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 unique_id</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bit (33) unaligned;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dcl</th>
<th>1 special_channel_name</th>
<th>aligned based,</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 zero_if_special</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fixed bin,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fixed bin (3) unaligned</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/* =0 special channel */</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/* ^=0 full event channel */</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 ring</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fixed bin (3) unaligned</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/* target ring number */</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 mbz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bit (15) unaligned,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fixed bin (17) unaligned;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/* number of special channel */</td>
<td></td>
</tr>
</tbody>
</table>

END INCLUDE file ... ect_structures.incl.pl1 */