Published: 02/07/68

Identification

Media Request Management
R. C. Daley, C. M. Mercer, J. H. Saltzer

Purpose

This section specifies a ring 0 queue management procedure, the Media Request Manager (MRM), designed to buffer requests that must be serviced by a media operator. The kind of request conveyed by MRM prescribes media operator action in the handling of detachable I/O media. Procedure calls are defined for use in performing initialization, queueing requests, and accessing requests for service.

General

The reader must be thoroughly familiar with the Interprocess Communication Facility, including the Wait Coordinator, described in Section BQ.6.

The following terms are used extensively in describing MRM:

Requester - any process group that indicates the need for media operator service by calling mrm$put_request.

Responder - a single process group that is the recipient of requests queued by any requester using mrm$put_request. The responder uses the calls mrm$get_request and mrm$put_status in performing its functions.

The procedures that constitute MRM are available to more than one process group, although the function performed by a given process group in relation to MRM determines what procedures that process group may successfully invoke. For example, requesters (each a process group) are not permitted to use procedures dedicated to the authorized responder (a single process group). The call to mrm$put_request is available to all process groups. A restricted MRM procedure performs its functions only if the current process group id (found in the process_info segment) agrees with the one authorized.

The Media Request Table

The system-wide data base used by MRM is known as the Media Request Table (MRT) and resides in ring 0. The structure of the MRT is illustrated as an implementation aid. Requests for operator service reside in slots in the MRT in one of the following states:

1. New - the request has not been examined by the media operator process.

2. Active - the request is being serviced.

An additional state, void, is used to indicate slots that are no longer occupied by requests. Void slots form a chain linked by a next slot index and exist as a pool to be used for storing new requests.

Slots containing new requests form a chain with links sequenced such that requests enter the active state on a first-in first-out basis.

The address space occupied by the MRT is comprised of a table control section followed by a serial table containing sequentially-indexed slots, each of which is in one of the three states mentioned above. A request remains in the same slot throughout its residence in the MRT, and may be referenced by an index while in the active state. The MRT address space expands when new requests are received and no void slots are available, and contracts when the slot referenced by the highest index allocated passes from the active state. Except where conditions cause MRT expansion or contraction, a slot passes successively through the void, new, and active states; then returns to the void state. Slots released by contraction are considered to be in the null state.

Because the MRT could otherwise be accessed simultaneously from multiple process groups, interlock procedures are used by MRM to prevent multiple access errors.

Initialization Calls

Before requests can be routed from the requester to the responder, the MRT must be initialized and a communications path must be established. Procedures mrm$init_mrt and mrm$op_group_id, both described below, perform their functions only if the process group id in effect when they are called is that of System Control.

1. Call issued by System Control to initialize the MRT:

```
call mrm$init_mrt (cstat);

   dcl cstat fixed bin (18);  /* set by MRM to
                                 indicate call status
                                 0 = valid call
                                 1 = caller not
                                     system control */
```

This call produces an empty MRT with no connection to a responder.

2. Call issued by System Control to identify the authorized responder:

```
call mrm$op_group_id (op_grp, cstat);

   dcl op_grp char (50), /* responder process group
                            id */

   cstat fixed bin (18); /* set by MRM to indicate
                            call status
                            0 = valid call
                            1 = caller not system
                                control */
```

MRM interprets a blank op_grp as an indication to cease signalling a previously authorized responder when subsequent mrm$put_request calls are received.

If a call to mrm$op_group_id is issued when there are active requests in the MRT, these requests are linked to the beginning of the new request chain and retrieved for the responder before any new requests. A responder call to mrm$get_request retrieves a request and an indication that it is either new or active.

Calls to MRM that are available only to the authorized responder are validated by a comparison between the process group id in effect at the time of the call and that supplied in the call to mrm$op_group_id.

3. Calls issued by the responder to supply MRM with the identity of an event channel:

```
call mrm$op_event (event, cstat);
```

```
dcl event bit (70),          /* responder event channel
                                name */

cstat fixed bin (18);        /* set by MRM to indicate
                                call status
                                0 = valid call
                                1 = responder group id
                                    error */
```

Each time a new request becomes available, MRM uses the Interprocess Group Event Channel Manager (IPGECM) to signal over the event channel supplied by the responder. IPGECM requires a process id as an argument and the one supplied by MRM is the one in effect (available in the process_info segment) when mrm$op_event is called.

Prior to calling mrm$op_event, the responder must issue at least two Event Channel Manager calls to initialize the event channel:

```
/* create event channel for ring 0 sending
   procedure */

   call ecm$create_ev_chn (ev_chn, mode, 0);

/* open event channel for all process groups,
   argument acc_list is ignored */

   call ecm$give_access (ev_chn, "0"b, acc_list);
```

Calls by Requester

A single call is defined for use by a process that needs media operator service: mrm$put_request. If a medium or device is specified in the request (as indicated by the appropriate character string being neither null nor blank), assignment is validated before the request is accepted for transmission to the responder. In order to use MRM, the requester must first create an event channel and its identity must be supplied as an argument in the call to mrm$put_request. Changes in request status that occur during service are made known to the requester in the form of an event signal and an event id.

The event id returned to the requester contains two components: (1) an end-of-service flag recognized by MRM, and (2) a status component having values known only to the requester and responder. The requester may monitor the progress of its request using Wait Coordinator options.

The requester must issue at least two Event Channel Manager calls to initialize an event channel for notification of request status changes:

```
/* create an event channel for use by ring 0
   sending procedure */

   call ecm$create_ev_chn (ev_chn, mode, 0);

/* open event channel access to any authorized responder,
   argument acc_list is ignored */

   call ecm$give_access (ev_chn, "0"b, acc_list);
```

After an event channel is initialized, the requester initiates media operator service with the following call:

```
call mrm$put_request (op, type, medium, device_type,
                      device, event, cstat);

/* arguments op, type, medium, device_type, and device
   may each be replaced with a literal character string
   no longer than 32 characters */

     dcl op char (32),           /* media operator action */

         type char (32),         /* medium description */

         medium char (32),       /* medium name */

         device_type char (32),  /* device description */

         device char (32),       /* name of device */

         event bit (70),         /* event channel name */

         cstat fixed bin (18);   /* set by mrm to indicate
                                    call status,
                                    0 = request accepted,
                                        operator present
                                    1 = request accepted,
                                        operator absent
                                    2 = rejected, medium
                                        unassigned
                                    3 = rejected, device
                                        unassigned
                                    4 = rejected, both
                                        unassigned */
```

Calls by Responder

Each time the responder is signalled over the event channel made known to MRM through an mrm$op_event call, a new request is waiting for service. A request is retrieved each time the responder calls mrm$get_request. The function performed by the responder, be it a service operation or a scheduling operation, is of no significance to MRM. The responder may notify the requester of status changes during request service by calling mrm$put_status. When service is terminated, the responder must issue a final mrm$put_status call with an end-of-service flag set. MRM interprets this flag as a signal to output final status and delete the request from the MRT.

When a request is retrieved using mrm$get_request, additional information is also retrieved including:

Index - a value that must be furnished in each mrm$put_status call for the request retrieved

Key - a value that must be furnished along with index

State - a character string indicating that a request is either new (state = "new") or has been retrieved by a previous responder but not completed (state = "actv").

A request is retrieved using the following call:

```
call mrm$get_request (op, type, medium, device_type,
                      device, state, index, key, cstat);

     dcl op char (32),          /* operator action */

         type char (32),        /* medium description */

         medium char (32),      /* medium name */

         device_type char (32),/* device description */

         device char (32),      /* device name */

         state char (4),        /* set by mrm to "new"
                                   or "actv" */

         index fixed bin (18), /* set by mrm for put_status
                                   call */
```

```
        key bit (70),          /* set by mrm for put_status
                                  call */

        cstat fixed bin (18);  /* set by mrm to indicate
                                  call status
                                  0 = no special conditions
                                  1 = responder group id
                                      error
                                  2 = no requests queued */
```

The following call is used to notify the requester of a change in the status of a media request:

```
call mrm$put_status (index, key, status, eos, cstat);

    dcl index fixed bin (18),   /* saved from get_request */

        key bit (70),           /* saved from get_request */

        status bit (18),        /* status of media
                                   request */

        eos bit (1),            /* "1"b = end of service */

        cstat fixed bin (18);   /* set by mrm to indicate
                                   call status
                                   0 = valid call
                                   1 = responder group
                                       id error
                                   2 = illegal index and/or
                                       key */
```

Summary

In addition to summarizing interface between a requester and responder, the following paragraphs detail the exchange of status information and describe the final disposition of a request.

The requester indicates media operator action is necessary by calling mrm$put_request which stores the request in the MRT and informs the responder that a request is waiting. The requester may then use Wait Coordinator options to detect changes in request status. The Wait Coordinator test_event and wait options return an array (ev_ind) of three elements. The second element, ev_ind (2), contains an event id set when the responder calls mrm$put_status. The requester may use the following substr function references to access status information:

```
end_flag = substr (ev_ind (2), 1, 1); /* get end-of-service
                                         flag
                                         "0"b = request still
                                                active
                                         "1"b = service
                                                terminated */

status = substr (ev_ind(2), 2, 18);   /* get status
                                         information set by
                                         responder */
```

If the Wait Coordinator test_event option is used and indicates that no event indicator has been found, no status information has been set and the request is either waiting or being serviced. If the Wait Coordinator is used to retrieve an event indicator in which the end-of-service flag is set, the final status set by the responder is available.

A request that has not been previously seen by the current responder is retrieved each time mrm$get_request is called. MRM uses ipgecm$set_event to signal the requester each time the responder calls mrm$put_status designating a particular request and its new status. If mrm$put_status finds the end-of-service flag (eos) set, final status is output and the request is deleted from the MRT.

Media Request Table Format

```
/* media request table */

     dcl 1 mrt based (p),

/* miscellaneous */

     2 lock bit (36),          /* interlock word */

     2 rsp_gid char (50),      /* responder process group id */

     2 rsp_pid bit (36),       /* responder process id */

     2 rsp_evc bit (70),       /* responder event channel name */

     2 alloc fixed bin (18),   /* index of highest slot
                                  allocated */

     2 first_empty fixed bin (18),  /* index of first empty
                                       slot */

     2 first_new fixed bin (18),    /* index of oldest
                                       unserviced request */

     2 last_new fixed bin (18),     /* index of newest unserviced
                                       request */

/* slot structure */

     2 slot (5000),

       3 rqstr_pid bit (36),        /* requester process id */

       3 rqstr_evc bit (70),        /* requester event channel
                                       name */

       3 rqst_op char (32),         /* requested operation */

       3 medium_type char (32),     /* e.g. mag tape, paper,
                                       etc. */

       3 medium_name char (32),     /* tape name, paper size,
                                       etc. */

       3 device_type char (32),     /* e.g. mag tape drive,
                                       printer, etc. */

       3 device_name char (32),     /* tape drive name, printer
                                       name, etc. */

       3 slot_state char (4),       /* "void", "new", "actv", or
                                       "null" */

       3 next_slot fixed bin (18); /* next slot index */
```