Published: 01/25/68

Identification

DIM service procedures
R. K. Rathbun

Purpose

This section describes all of the service procedures internal to the File System DIM. (See BG.10.00 for an overview of the File System DIM.) The procedures represent tasks whose code would otherwise appear in more than one of the DIM's procedures. Herein, the term request-initiator refers to the procedure dim_command and the term hardware-interface refers to any of the procedures drum_ctl, disc_ctl, race_ctl, or the like.

Manipulation of the I/O queue and the done-list

From the time the DIM receives a new request to the time the request is completed, the request has associated with it a primary I/O queue entry. The queue entry serves as a working space for the request-initiator, as an error repository for its associated hardware-interface, and in general, as an identification of the request. When all of the hardware commands generated by the request have been completed, the queue entry is placed in the done-list where it remains until the DIM signals that the request is completed. Once completion is signaled (by a call to iodone), the entry is taken from the done-list and is placed in the free-list from which it was allocated when the request was initiated.

For certain pathological write-requests (see BG.10.02), an additional (or secondary) entry is required by the request-initiator. An entry of this type is merely a temporary memory for the request-initiator. When the secondary entry is no longer needed, it is placed in the free-list directly -- it is not placed in the done-list as an intermediate step.

Regardless of how pathological the request, it needs at most one secondary entry at a given moment. However, since every request potentially requires two queue entrys, there must be at least:

number of processors + 1

entries in the free-list at initialization-time. A smaller number of entries permits a total lock-out situation in the DIM.

The queue has the following overlay for the management procedures:

```
dcl 1  q_linkage based (xx),
         2 free_done,
           3 lock bit (36),
           3 first_free bit (18),
           3 free_count bit (18),
           3 first_done bit (18),
           3 done_count bit (18),
         2 normal_link (1024),
           3 fill1 bit (54),
           3 linkage bit (18),
           3 fill2 bit (36);
```

where the items of the structure are as follows:

free-done is the header for both the free-list and the done-list. The next five sub-items constitute the current linkage-status of the queue.

lock is set "on" before any attempt is made to link or unlink an entry from either the free-list or the done-list.

first-free is an index to the first free entry, provided that the free-list is not empty.

free-count is the number of entries in the free-list.

first-done is an index to the first entry in the done-list, provided that the done-list is not empty.

done-count is the number of entries in the done-list.

normal-link is an array each element of which overlays one normal queue entry.

fill1 is a filler to align the next item.

linkage is an index to the next entry in the list, provided that the entry itself is in either the done-list or the free-list and that the entry is not the last entry in the list.

fill2 is a filler to align successive queue entries.

To obtain a free queue entry (primary or secondary), the DIM makes the following call:

```
call dims$gf (qx, err);
```

where:

```
dcl   qx fixed binary (35),   /* queue index */
      err fixed binary (35);  /* error-code */
```

If free_count is non-zero, the entry whose index is first_free is removed from the free-list. Items linkage and fill2 of this entry are set to zero as initial values. The parameter qx is set to the index of the newly removed entry and the error-code err is set to zero (i.e., no error); these values are then returned.

If, on the other hand, free_count is zero, dims$service_done_list is called to have entries in the done-list moved to the free-list. If free_count is now non-zero, the removal is performed as above. If the count is still zero, then dims$wait is called to cause processing of other requests and dims$service_done_list is called to have entries in the done-list moved to the free-list. The sequence of a call to dims$wait followed by a call to dims$service_done_list is repeated until a free entry appears.

If ever dims$wait returns an error-code, the error-code is returned to the caller, thus signaling that every file system device is inoperative; i.e., no queue index is returned.

To link an entry to the free-list, the following call is made:

```
call dims$lf (qx);
```

The queue entry whose index is qx is placed at the top of the free-list.

To obtain an entry from the done-list, the DIM makes the following call:

```
call dims$gd (qx);
```

If the done-list is empty, then qx is returned as zero. Otherwise, an entry is removed from the top of the done-list and its index is returned as qx.

To link an entry to the done-list, the following call is made:

```
call dims$ld (qx);
```

The queue entry whose index is qx is placed at the top of the done-list:

Posting hyper-commands

Whenever a request to a device control module is found to be completed, the requesting module must be notified of this fact. This action is called posting. For every hyper-command handed to a hardware-interface, some type of posting is done when the hyper-command has been run. (Briefly, a hyper-command is that group of DIM commands necessary to move a hyper-record; cf. BG.10.00). Since the actual posting is device-independent, the hardware-interfaces share the same posting procedure. The procedure is called as follows:

```
call dims$post (id, flag, type, indx, arg);
```

where:

```
dcl   id fixed binary (35),     /* id of device */
      flag fixed binary (35),   /* error flag */
      type fixed binary (35),   /* posting type */
      indx fixed binary (35),   /* user id */
      arg fixed binary (35);    /* posting argument */
```

The arguments have the following definitions:

id is the device id of the device belonging to the hardware-interface which called the posting procedure.

flag is the error flag pertaining to the hyper-command being posted. A value of zero indicates successful completion. A non-zero value indicates unsuccessful processing. By the poster, no significance is attached to the particular value of a non-zero flag, the value is simply passed on to the initiator of the request, whenever this is possible.

type identifies the initiator of the request, or equivalently, the type of posting to be done. The following are the valid codes:

```
0 = initialization posting
1 = primary queue posting
2 = secondary queue posting
3 = free-storage posting
other = call panic
```

Additional types may be added, if the need arises.

indx is the identification of the user within the class type. Initialization has no indx. For queue types, indx is the index of the primary queue entry. For free-storage, it is the index of the free storage buffer associated with device id.

arg is an argument passed from the initiator of the hyper-command to the poster via the hardware-interface. For secondary queue type posting, it is an index into the secondary queue entry where is contained a hyper-record number and its hyper-sector address. This parameter is not used for other types of posting.

The particular action taken for the various posting types is described here.

(Type 0). Since the system is so "primitive" at initialization time, a non-zero flag is a command to call panic; on the other hand, a zero flag causes only a return.

(Type 1). The flag is or'ed into the status-word of the queue entry whose index is indx. The count of the number of hyper-commands outstanding for this queue entry is decremented. If the count is still non-zero, nothing else is done. If the count is now zero, then the request is completed and the queue entry is linked to the done-list.

(Type 2). The request-initiator may interlock a hyper-record of a file when it is necessary to initialize that hyper-record. To do this, the request-initiator requires the temporary use of a secondary queue entry in order to identify the hyper-record and to save the hyper-sector address.

A secondary queue entry has the following overlay:

```
dcl 1 ioq2 (0:1024) based (xx),
      2 fmo bit (18),      /* relp to file-map */
      2 count bit (18),    /* inits outstanding */
      2 fill1 bit (1),     /* fill to align next */
      2 hrn1 bit (17),     /* h-record number */
      2 hra1 bit (18),     /* h-record address */
      2 fill2 bit (1),     /* fill to align next */
      2 hrn2 bit (17),     /* h-record number */
      2 hra2 bit (18),     /* h-record address */;
```

where "h-record" means hyper-record.

The secondary queue entry is located via the linkage of the primary queue entry (designated by indx) associated with the request. Posting amounts to locating the file-map associated with the request, and then replacing the current hyper-record address with the one stashed in the secondary

queue entry. Arg specifies which of the two possible replacements carried in the secondary queue entry is to be performed. If this replacement is the last (of the two), then the secondary queue entry is unlinked from the primary entry and is linked to the free-list. The primary entry is then posted as above.

(Type 3). The flag is stashed in the status word associated with device id and free storage buffer index thereof. If the flag is zero, then the buffer is designated as having no errors in transmission. Lastly, a switch is set to indicate that the I/O has been completed (see BG.10.04).

(Type not listed above). The poster calls panic.

Servicing the done-list

The done-list contains all queue entries for which I/O has been completed. The DIM may cause the done-list to be emptied and the entries therein to be "posted" (i.e., Page Control is informed) by the following call:

```
call dims$service_done_list;
```

For each entry in the done-list, parameters of the entry are removed and passed to the page control entry iodone as follows:

```
call iodone (op, fmo, state, mem, status);
```

where:

```
dcl   op bit (3)          /* file operation */
      fmo bit (18),       /* relative pointer to file-map */
      state bit (10),     /* see below */
      mem bit (18),       /* memory address */
      status bit (18);    /* error code */
```

All of the parameters, except status, were handed to dim$file_io when the request was initiated. The or'ed error codes from all hyper-sector I/O operations is stored in status. The parameter state is an argument from the caller of the DIM to iodone and is not altered by the DIM.

After its parameters are removed, each entry from the done-list is linked to the free-list, thus replenishing the supply.

Passing the time waiting for I/O

Occasionally, a portion of the DIM will be unable to continue operation until one or several I/O operations are complete. For example, if there are no free queue entries, then at least one request must be hammered to completion before an entry will be available. A similar situation exists for free storage; i.e., when its load is heavy and its service from the hardware-interface is lagging.

Although a portion of the DIM may be unable to proceed, the DIM itself can still do useful work. At such times, the following call is made:

```
call dims$wait (id, err);
```

where:

```
dcl  id fixed binary (35),     /* device id */
     err fixed binary (35);    /* error-code */
```

Generally, this call causes the hardware-interface(s) to be run; running a hardware-interface speeds up posting and thereby increases effective throughput.

If the inability to proceed is due to a particular device (as with free storage), then the caller sets id to the device identification of that device. In this case, the hardware-interface for only that device is run. If the hardware-interface returns an error-code, that code is returned to the caller. This indicates, of course, that the device is inoperative and further "waiting" is futile.

If the inability to proceed is not attributable to a particular device (as with dims$gf), then the caller sets id to zero, the identification of an illegal device. This causes the hardware-interface of every operative device to be run. If every hardware-interface returns an error-code, then an error-code is returned by dims$wait, indicating that every device is inoperative. On the other hand, one zero error-code indicates that useful work can still be done, and so dims$wait returns a zero error-code.

A return from dims$wait with a zero error-code does not imply that the difficulty has been removed. Indeed, it is the responsibility of the caller to check that and to call again, if necessary.

Regardless of the setting of id, all errors returned from hardware-interfaces are recorded in dims_dct (described in BG.10.01).