TO: MSPM Distribution
FROM: M. A. Padlipsky
SUBJECT: BG.10.01, BG.10.03
DATE: 01/25/68

Sections BG.10.01 and BG.10.03 are being released at this time, in advance of an updated Overview, in order to document segments which are currently on the Segment Library, especially as to calling sequences. It should be noted that they deal with that section of code in the File System Device Interface Module which is common to all devices used by the File System (sometimes known as "the generic DIM"). Note also that, for conciseness, the File System DIM is usually referred to in these documents simply as "the DIM". Specific device-oriented information will be found in sections BG.11 - BG.13.

Published: 01/25/68

Identification

DIM Driver
R. K. Rathbun

Purpose

The Dim Driver is the main logic program of the File System DIM. (See BG.10.00 for an overview of the File System DIM). As a request processor, the DIM interfaces with the rest of the File System via the two entries dim$file_io and dim$run. The first entry is used to initiate a request and the second is used to cause further processing of already-initiated requests. This section describes in greater detail the action taken by each entry.

Initiating a request

A new request is handed to the DIM with the call:

```
call dim$file_io (id, op, astp, fmp, state, mem, rec, cnt, err);
```

where:

```
dcl  id bit (4),                    /*device id*/
     op bit (3),                    /*request type*/
     astp pointer,                  /*ast pointer*/
     fmp pointer,                   /*file-map pointer*/
     state bit (10),                /*argument of iodone*/
     mem bit (18),                  /*starting memory*/
     rec fixed binary (18),         /*starting record*/
     cnt fixed binary (18),         /*number of records*/
     err bit (18);                  /*error-code*/
```

Upon entry to file_io, a queue entry is obtained from the free pool of queue entries. If it is impossible to obtain a free entry, then bit 18 of "err" is set to "1"b and file_io returns to the caller. This type of error can occur only when every device attached to the file system is out of operation.

Assuming a queue entry was successfully obtained, "id", "ptr$rel(astp)", "state", and "mem" are stashed in the entry. In accord with "op", one of the following calls is then made:

```
/*op = "000"b, request is read*/
     call dim_command$read (id2, fmp, rec, cnt, qx, errcode);
/*op = "001"b, request is write*/
     call dim_command$write (id2, fmp, rec, cnt, qx, errcode);
/*op = "010"b, request is delete*/
     call dim_command$delete (id2, fmp, rec, cnt, qx, errcode);
```

where "id2" is fixed (id, 35), "qx" is the index of the queue entry described above, and "errcode" is an error-flag set by dim_command.

Upon return from dim_command, "errcode" may be either "on" (≠0) or "off" (=0). In the event that the flag is "on", the device was found inoperative and processing of the request was terminated. It cannot be determined to what degree the request had been processed before the device became inoperative. In any case, the queue entry is returned to the free pool, bit 18 of "err" is set to "1"b, a note is made that device "id" is inoperative, and file_io returns.

If "errcode" is "off", then the request has been fully initiated without error. To allow the hardware interface to process the initiated requests, a call to dev_ctl$run is made. If dev_ctl finds that the device is out of operation, it frees all queue entries associated with requests that been fully initiated and returns with an error-flag, say "errcode2" set "on". Otherwise, a return is made with "errcode2" set "off".

If "errcode2" is "off", a call is made to service_done_list to signal that certain (previous) requests have been completed (successfully or unsuccessfully). File_io then returns.

If "errcode2" is "on", bit 18 of "err" is set to "1"b, a note is made that device "id" is inoperative, and file_io returns.

It should be noted that a return from file_io with "err" non-zero means that the request was not completed, that it will not be completed, and that the caller is not obliged to make any form of a "cleanup" call to the DIM. If "err" is zero, then there is no guarantee that the request has been completed (successfully or unsuccessfully); only a call to iodone from service_done_list can indicate completion. That is, this type of return merely indicates that the request has been fully initiated.

Processing a fully initiated request

In response to a hardware interrupt a special process (the File System Device Monitor Process; see also BL.11.01) is awakened. Upon receiving a wakeup, this process makes the following call to the DIM to update the status of previously initiated requests. The file system may also facilitate the processing of fully initiated, but uncompleted requests by making this call.

```
call dim$run;
```

This call to the DIM is interpreted as a command to run the hardware interface for every device attached to the file system. The DIM makes the following call for every device, provided that the device is not known to be inoperative:

```
call dev_ctl$run (id, 0, errcode);
```

If upon return, "errcode" has been set "on", that fact is noted, and dev_ctl$run is not called again for device "id".

When this sequence of calls is completed, service_done_list is called to signal that certain requests previously initiated have been completed.

A user of the DIM cannot force a particular request to completion by a direct call to the DIM. Instead, he must continue to call dim$run until service_done_list calls iodone to signal completion.

Data bases

There is only one data base used by the dim$file_io and dim$run which is not used more exclusively by other DIM procedures. This data base is the DIM's device configuration table. The table is merely a compact version of the File System Device Configuration Table with some additional items pertinent to the running of the DIM:

```
dcl  dims_dct static external,
     2 ndevices bit (5),          /*number of devices
                                    assigned*/
     2 cross (0:15) bit (5),      /*cross reference
                                    ...array into descr*/
     2 descr (16),                /*device description*/
       3 did bit (4),             /*device id*/
       3 blocksize bit (5),       /*device-oriented
                                    record-size*/
       3 down_sw bit (1);         /*inoperative flag*/
```

ndevices. Interpreted as a count of the number of devices assigned to the file system. An ordinal index between 1 and ndevices is used to index the structure descr.

cross. An array that maps the id of a device into its index into descr. That is, descr(cross(id)) refers to the entry in the descr array pertaining to device "id".

descr. An array-structure which describes one file system device per entry as follows:

did. The file system's identification of the device.

blocksize. The number of 64-word blocks of device space per hyper-record. That is, the number of 64-word blocks described by an address found in a file-map which belongs to a file residing in this device.

down-sw. A flag which indicates whether the device is believed operative. The flag is initialized to "0"b for every device assigned and is set to "1"b when dim$file_io or dim$run is handed a non-zero-flag.