Published: 03/13/68

Identification

Dim Command Module
S. W. Jones

Purpose

The module dim_command is responsible for servicing file system requests which involve transferring pages of segments to and from on-line secondary storage devices.

Introduction

The function of dim_command is to process read, write, and delete requests for files on secondary storage. To accomplish this, dim_command constructs pseudo-commands for device control and maintains a file map describing the file as it resides on the device.

dim_command is called by dim$file_io at one of the three entry points, $read, $write, or $delete, depending upon the request to be processed. Accordingly, pseudo-commands are constructed and passed to dev_ctl$new_io which is responsible for building the actual device commands (DCW's). Control is then returned to dim$file_io whether or not the device commands have been executed.

A non-zero error code on return from dim_command indicates that the device is inoperative and that the request cannot be completely serviced. A zero error code indicates only that the responsibility of completing the request has been turned over to the (asynchronous) device control module.

Records on the device

Pages of segments when stored on a secondary device are known as records of files. Each record contains 64 words and is stored on the device as part of a hyper-record (a group of 2, 4, 8, or 16 contiguous 64-word blocks). The number of records within a hyper-record depends upon the device; the records within a hyper-record are called sub-records whenever confusion may arise from the term record. Hyper-records of a file are stored randomly on the device; the device address of each hyper-record is stored in the file's associated file map.

Before the first write request is issued for a given file, its file map contains all null addresses (octal 777777). The null address denotes to dim_command that any hyper-record having this as its address does not have space allocated for it on the device. It is therefore a signal to dim_command that space must be allocated for the hyper-record before any of its sub-records may be written. The address of the allocated hyper-record replaces the null address in the file map for future use.

Every record will appear to contain all zeroes on the device until it has been explicitly referenced in a write request. Different techniques are required to achieve this effect depending upon the request. When attempting to read records of a hyper-record having a null address, dim_command interprets them as records containing all zeroes and calls a routine to physically store zeroes (instead of reading zeroes from the device). Similarly, for a write request involving a hyper-record with a null address, those sub-records not specified are zeroed on the device.

The null address is one of two reserved addresses; the function of the other reserved address (the marked address, octal 777776), is described later in this section.

Records in core memory

Records to be transmitted to or from core memory must be read into or written from contiguous 64-word blocks. In addition, dim_command deals only with the left-most 18 bits of the 24-bit memory address, so that I/O is always started on a 64-word page boundary (i.e., the right-most 6 bits are conventionally zeroes).

On the calling of dim command

The entries of dim_command are called as follows:

```
call dim_command$read(did,fmptr,recno,count,ioqx,err);
call dim_command$write(did,fmptr,recno,count,ioqx,err);
call dim_command$delete(did,fmptr,recno,count,ioqx,err);

dcl  did fixed binary (35),
     fmptr pointer,
     recno fixed binary (35),
     count fixed binary (35),
     ioqx fixed binary (35),
     err fixed binary (35);
```

did is the identification of the device on which the file resides.

fmptr is a pointer to the file's file map. The file map contains a device address for each hyper-record of the file and an interlock to avoid certain types of simultaneous access to the file map.

recno is the record number of the first record of the request. Record numbering begins with 1 (one).

count is the number of records to be affected by the request.

ioqx is the index of this request as it exists in a table of outstanding requests. The table contains additional information related to the request.

err is the error code.

IOQ

The IO queue (IOQ) is a table which is totally internal to the DIM. A primary entry is allocated with each new call request, to hold the arguments to dim$file_io; the entry is de-allocated by dims$service_done_list just before iodone is called. While the entry is allocated, portions of it serve as a working area for both dim_command and dev_ctl.

The IOQ appears as follows:

```
dcl 1 ioq (0:1024) based (xx),
      2 astep bit (18),          /* aste relp */
      2 did bit (4),             /* device id */
      2 op bit (3),              /* operation */
      2 state bit (10),          /* arg to iodone */
      2 filler bit (1),
      2 memadd bit (18),         /* memory address */
      2 linkage bit (18),        /* inter-q link */
      2 status bit (18),         /* accumulated status */
      2 hrc_count bit (18);      /* number of pseudo-
                                 ... commands generated */
```

astep is the pointer to the file's ast entry. This parameter is not used by the DIM. (It is merely passed to iodone when the request is completed.)

did is the identification of the device.

op is the operation to be performed on the file. (op is equal to 0 for read, 1 for write, and 2 for delete.)

state is the state of the file as seen by the file system. This parameter is not used by the DIM. (It is passed to iodone when the request is completed.)

memadd is the left-most 18 bits of the absolute starting memory address. The remaining 6 bits are conventionally zeroes.

linkage is a working area for dim_command. If the value is non-zero, then it is the index of a secondary IOQ entry in the table; this secondary is used only by $write and is described later.

status is the "worst unrecoverable" hardware error to occur with DCW's for this request. The precise meaning of this parameter is explained in BG.10.07.

hrc count is the number of pseudo-commands generated by this request. Initially, the value is the number of hyper-records within the scope of the request which will generate DCW's. As the request is being processed, the count is decremented when a set of DCW's for a hyper-record has been executed. This is done by dims$post when called by dev_ctl to signal that a set of DCW's has been completed. When the count reaches zero, processing of the request is essentially completed.

Strategy of dim_command$read

$read services the requests which involve transferring records of files from a secondary storage device to core memory. If a record is requested and the hyper-record containing that record has space allocated for it on the device, then $read extracts the device address from the file map and constructs pseudo-commands for device control to read that record into core. If the address of a record is a reserved (null or marked) address, then it is assumed that the record contains all zeroes. To avoid the overhead of reading zeroes from the device for records of this type, $read calls a routine which zeroes contiguous pages of core.

Strategy of dim_command$write

$write services the file system's requests which involve transferring records from core memory to secondary storage. If a record is requested and that record has space allocated for it on the device (that is, the address from the file map is not a reserved address), $write simply extracts the associated device address from the file map and constructs the pseudo-commands to over-write the previously written records.

If, however, a record requested does not have space allocated for it (that is, its address in the file map is null), then $write calls the device free storage routine to obtain an unused address. This address is placed in the file map when every record of the hyper-record is to be written; processing then proceeds as above. If not all the sub-records of the hyper-record are to be written, that is, for example, given a hyper-record size of four, write records three and four only, then the real address is used for writing the records as required by the request (records three and four) and for zeroing the remainder (records one and two). While this "initializing" is taking place, the address in the file map is replaced with a marked address (octal 777776) and the real address is saved in a second ioq entry (see below). After the DCW's associated with this hyper-record have been executed, the real address replaces the marked address in the file map.

To dim_command, the marked address, which is the second reserved address, has the following meaning:

a) to $read, the sub-records contain all zeros (same treatment as with the null address).

b) to $write, the sub-records cannot be written until the marked address is replaced with the free address. A marked address for a record therefore implies that space is being allocated for the record.

c) to $delete, the sub-records have been deleted.

The marked address is necessary since the DIM requires only that two simultaneous requests cannot reference the same record of a file. If the restriction were stronger, namely that no two simultaneous requests fall within the same hyper-record of a file, then the marked address would not be needed.

Secondary IOQ entries

The IOQ has an additional over-lay which corresponds to the secondary entries; that is, a secondary IOQ entry is merely an IOQ entry with a different format and use. A secondary entry is of interest only to $write and to dims$post.

$write requires a secondary entry when a null address is being replaced by a marked address (which is ultimately to be replaced by a "normal" address). In this case, a secondary entry is obtained (if one does not already exist for this request), and its index is stashed in "linkage" of the primary IOQ entry. The secondary entry itself serves as the temporary store for the "normal" address while some of its sub-records are being zeroed. More precisely, the secondary entry is filled with the following information: the relative pointer of the file map, the index of the hyper-record's address slot within the file map, and the hyper-record address. There is room for two indexes and two addresses, corresponding to the first hyper-record and the last hyper-record within the scope of the request. If the first hyper-record is also the last hyper-record, then the second index and address will not be used. To reiterate, the secondary entry is needed only when the request is "write" and the first (or last or both) hyper-record address within the scope of the request is null.

The secondary entries appear as follows:

```
dcl  ioq2 (0:1024) based (xx),
     2 fm_relp bit (18),          /*file map relp*/
     2 count bit (18),            /*items used*/
     2 tfskb1 bit (1),            /*disc only*/
     2 fm_index1 bit (17),        /*first fm index*/
     2 dev_add1 bit (18),         /*first address*/
     2 tfskb2 bit (1),            /*disc only*/
     2 fm_index2 bit (17),        /*second fm index*/
     2 dev_add2 bit (18);         /*second address*/
```

The parameter count is set to one (1), when the secondary entry is obtained and is set to two (2) if the second index/address pair are needed. Possibly, dev_ctl will issue and post the first set of DCW's before the last hyper-record is processed by $write; in this case, $write must allocate another secondary entry (with a count of one) as a replacement. dim_command does not know that

the secondary ioq entry is a replacement. (Note that when the count is decremented to zero by dims$post, all hyper-records to be initialized have been, and so the secondary entry is de-allocated and "linkage" in the primary entry is zeroed. In any case, dims$post is responsible for replacing the marked address in the file map with the address saved in the secondary entry; that is, $write initiates the mechanism and dims$post terminates it -- just as for any other type of I/O operation).

It should be noted that when hyper-page size is an even multiple of hyper-record size, the DIM is most efficient and the need for a secondary IOQ entry is eliminated.

Strategy of dim command$delete

$delete services the file system's requests to delete records of files residing on a secondary storage device. For each hyper-record within the scope of the request, $delete does one of the following:

1. If every sub-record of the hyper-record is to be deleted and the hyper-record is not null, the device address is returned to the device free storage routine, and its address in the file map is replaced by the null address.

2. If the hyper-record address is null or marked, the appropriate records are assumed to be deleted already. More precisely, either space has never been allocated (null), or the space being allocated does not overlap the records being deleted (marked); in either case, the deleted records will contain zeroes if later read.

3. Otherwise the sub-records to be deleted are zeroed on the device.

The condition of deleting an entire hyper-record, when the hyper-record is marked, is in violation of the rule against simultaneous record accessing.

It should be noted that an address is returned to free storage only when the entire hyper-record is within the scope of a delete request. Thus, for example, deleting an entire file one record at a time (that is, one record per request) is not the same as deleting the entire file with one request. In the first case, no addresses are released to free storage, and in the second, all the addresses are released. However, this is not a problem at the current time because page control deletes records in multiples of 1024 words.

Summary of calls made by dim command

Following is a list of all DIM procedures used by dim_command accompanied by a definition of the input parameters and brief explanation of the service each procedure performs.

1. call zero (address, null, pgct);

```
dcl  address fixed bin (35),
     null ptr,
     pgct fixed bin (35);
```

address is the memory address divided by 64.

null is a null pointer.

pgct is the page count (in small pages).

The routine is used by $read when a reserved address is encountered. Pgct number of consecutive pages starting at the specified memory address are zeroed.

2. call dev_ctl$new_io (did, iocmd_array, memadd, devadd, posting_cmd, posting_index, notify_parameter, block_size, err_code);

```
dcl  did fixed bin (35),
     iocmd_array (16) fixed bin (35),
     memadd fixed bin (35),
     devadd fixed bin (35),
     posting_cmd fixed bin (35),
     posting_index fixed bin (35),
     notify_parameter fixed bin (35),
     block_size fixed bin (35),
     err_code fixed bin (35);
```

did is the device identification.

iocmd_array is the array of pseudo commands (read, write, idle, write zeros). One element of the array corresponding to one record in the hyper-record.

memadd is the address relative to the first record in the hyper-record at which I/O will occur.

devadd is the device address relative to the first record in the hyper-record at which I/O will occur.

posting_cmd is the indication of what should be done after the issued device commands have been executed. (Refer to BG.10.05 for detailed explanation).

notify parameter is a parameter associated with posting_cmd (see BG.10.05), and here is the indication of which file map address must be updated after the DCW's for this given address have been executed.

block size is the number of records in a hyper-record.

err code is the error indication returned by dev_ctl$new_io if the device is not operative.

$read, $write, and $delete construct pseudo-commands to process block_size number of records (a hyper-record) at a time. dim_command completes processing each hyper-record when it passes the above information from which dev_ctl constructs DCW's.

3. call dims$wait (did,err_code);

```
dcl  did fixed bin (35),
     err_code fixed bin (35);
```

did is the device identification number.

err code is the error status returned if the device is inoperative.

This routine is used when a marked address is encountered by $write. dims$wait is called to speed up the execution of the outstanding DCW's for the hyper-record involved.

4. call free_store$withdraw (did,address, err_code);

```
dcl  did fixed bin (35),
     address fixed bin (35),
     err_code fixed bin (35);
```

did is the device identification number.

address is an unused device-address.

err code is the error indication returned if the device is not operative.

This routine is used by $write when a hyper-record does not have device space allocated. An unused device-address is obtained from the device free storage routine.

5. call free_store$release (did, address, err_code);

```
dcl  did fixed bin (35),
     address fixed bin (35),
     err_code fixed bin (35);
```

did is the device identification number.

address is the device address.

err code is the error indication returned.

$delete returns a device address to the device free storage routine.

6. call dims$gf (ioqx,err_code);

```
dcl  ioqx fixed bin (35),
     err_code fixed bin (35);
```

ioqx is the ioq entry index.

err code is the error status returned.

A secondary ioq entry with index, ioqx, is allocated for $write.

7. call dims$lf (ioqx);

```
dcl  ioqx fixed bin (35);
```

ioqx is the ioq entry index.

This routine is used by $write to return an ioq entry to the ioq free list.

8. call dims$ld (ioqx);

```
dcl  ioqx fixed fin (35);
```

ioqx is the ioq entry index.

This routine is used by $read and $delete to link an ioq entry to the done list.

External system lock routines used by dim command

1. call ilock$looplock (p,err);

2. call ilock$loopunlock (p);

Explanation of function and parameters is given in BG.15.02.