Identification

External DIM Functions
R. K. Rathbun

Purpose

For each file residing on a file system device, the DIM maintains a file map which defines the location (on the device) of the file's records. A file map is of use only to the DIM, but must exist somewhere in the Multics hierarchy as long as the file exists. When the file is active, its file map must reside in wired-down core; when the file is inactive, its file map resides in the directory which contains the branch defining the file.

To permit file system procedures external to the DIM to manipulate file maps without being aware of the file map structure, several primitive procedures are provided. The procedure dimf$fm_max_size returns the number of bits required for a file map needed to describe a file of known maximum length. This procedure is used when a file is created, to determine how large an area in core is needed to hold its file map. The procedure dimf$fm_init initializes a file map so that it describes (to the DIM) a file which has no device space yet allocated. A call of this type immediately follows the allocation of an in-core area for a new file map. The procedure dimf$fm_move is called by directory control to move a file map within a directory whenever the maximum length of the related file is changed. It should be noted that a file map cannot be shortened until the trailing records have been deleted via calls to the DIM.

Also provided is dimf$optimum_page_size which returns the "best" page size for files residing on a given device. Files so paged receive service with minimum overhead from the DIM.

A dimf constant

The file maps generated and manipulated via calls to dimf are device independent in the sense that the length of a file's map is the same no matter on which device the file actually resides. This requires that dimf consider

every file system device to have a hyper-record size which is (at most) equal to the smallest hyper-record size of all file system devices. This smallest hyper-record size is a constant between complete file system reloads. Thus a file with "x" 1024-word blocks requires (at most) a certain number of hyper-record addresses, denoted by da(x), which value is independent of the device containing the file. The value of da(x) is given by the integral part of:

```
(16 * x + t - 1) / t
```

where t is the smallest hyper-record size.

File map size

The file map is declared as follows:

```
dcl  fm based (xx),
     2 lock bit (36),
     2 add (1024) bit (18);
```

where:

lock is an interlock to prevent simultaneous modifications to the map by the DIM.

add(i) is the hyper-record address of the i-th hyper-record of the file.

The calling sequence to determine the number of bits needed for a file's file map is given by:

```
call dimf$fm_max_size (size, bits);
```

where:

```
dcl  size bit (9),
     bits fixed binary (35);
```

and:

size is the length of the file in 1024-word blocks (specified by the caller).

bits is the number of bits needed for the file's map (returned by DIMF).

The value of bits is given by:

```
36 + 18 * da(size)
```

as can be seen from the declare.

Initializing a file map

A file map is initialized by the call:

```
call dimf$fm_init (size, mp);
```

where:

```
dcl  size bit (9),
     mp pointer;
```

and:

size is the length of the file in 1024-word blocks.

mp points to the space allocated for the map.

This causes the lock to be zeroed and the first da(size) hyper-record addresses to be set to "null" addresses in the file map pointed to by mp. (See BG.10.02 for the definition of a null address).

Moving a map

Whenever the maximum length of a file is changed, the file map is moved by the call:

```
call dimf$fm_move(osize, omp, nsize, nmp);
```

where:

```
dcl  (osize, nsize) bit (9),
     (omp, nmp) pointer;
```

and:

osize and nsize are the old and new maximum lengths of the file in 1024-word blocks.

omp and nmp are the old and new pointers to the file map space.

If the length of the file is unchanged, then the lock is zeroed and da(osize) hyper-record addresses are copied from the old file map into the new map.

If the length of the file is increased, then the above is performed and then the addresses in the map from da(osize)+1 to da(nsize) are set to null.

If the length of the file is decreased, then the lock is zeroed and the first da(nsize) hyper-record addresses are copied into the new map. The addresses from da(nsize)+1 to da(osize) in the old map are untouched; that is, they are expected to be nulls resulting from delete requests to the DIM.

Paging

If a file resides on a device whose hyper-record size is hrs, then optimum service can be expected from the DIM if the file has a page size of hrs. In this case, the DIM does not have to go through the overhead of breaking hyper-records into their component records in order to satisfy requests. The next best page size is a multiple of hrs, since this still requires no breaking down. Although other page sizes are not optimal, they are still functional.

The optimum page size is obtained via:

```
call dimf$optimum_page_size (did, page_size);
```

where:

```
dcl  did bit (4),
     page_size fixed binary (35);
```

and:

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

page-size is the returned hyper-record size for the indicated device as obtained from a list of initialization-time constants.