Published: 04/24/67

Identification

File System Initialization (Part 1)
R. C. Daley

Purpose

This section provides the specification of the procedures which perform the first part of file system initialization. These procedures run under the control of the Multics initialization control program during the first part of Multics initialization. The main purpose of this part of file system initialization is to initialize all secondary storage devices used by the file system and to initialize some of the data bases used by the file system. After these initialization procedures have been run, they may be deleted allowing the storage to be reused by the second part of Multics initialization.

Introduction

When the Multics initialization control program passes control to the first part of the file system initializer, the system is in the following state.

1. Some of the segments of the hardcore supervisor have been loaded and all of their external segment references have been prelinked.

2. The GIOC interface module (GIM) and its data bases has been initialized and is available for use by the file system.

3. The system interrupt interceptor and interrupt handlers for drum and GIOC interrupts have been initialized.

4. Dummy versions of the traffic controller procedures "block" and "wakeup" have been provided.

File System Initialization

At the appropriate point during the first part of system initialization, the Multics initialization control program makes the following call to initialize the file system.

```
call fs_init_1;
```

Upon receiving this call, the following steps are taken to initialize the file system.

Step 1

To initialize all secondary storage devices available to the file system, control is passed to an initialization procedure by means of the following call.

```
call initialize_devices;
```

If the file system hierarchy has been destroyed or must be reloaded, this procedure calls the define_device entry of the device utility package (see BG.17) to re-initialize each secondary storage device (e.g. prepare free storage maps, etc.).

Step 2

To define the areas of secondary storage to be used by the version of Multics currently being initialized a call is made to the following initialization procedure.

```
call define_partitions;
```

This procedure makes successive calls to the get_status entry of the device utility package to obtain the file pointer and current length of the root directory and the master hyperrecord addresses (see BG.17) defining the areas of secondary storage to be used by this version of Multics. This information is placed in the file system device configuration table (see section BL.10.04) for use during parts 2 and 3 of file system initialization.

Step 3

The file system device disposition table (DDT) is initialized by issuing the following call.

```
call initialize_ddt;
```

This procedure initializes the DDT from information in the file system device configuration table and presets the multilevel criteria parameters (see section BH.1).

Step 4

The file system device interface modules (DIMs) are initialized by means of the following call.

```
call initialize_fs_dims;
```

This procedure first initializes the I/O queues which are common to all DIMs and then proceeds to initialize a DIM for each secondary storage device available to the file system. The initialization of a file system DIM includes the following steps.

1. The master hyperrecord is read into core to obtain the device hyperrecord size and the record addresses of the free storage maps. (The address of the master hyperrecord has previously been stored in the file system device configuration table during step 2)

2. The deposition and withdrawal buffers are initialized by reading in the appropriate records of the free storage maps.

3. The DIM history table is initialized along with other DIM dependent data bases.

Step 5

The system segment tables (SST) consisting of the active segment table (AST) and descriptor segment table (DST) and the process segment table (PST) are initialized by means of the following call.

```
call initialize_sst;
```

This procedure initializes the SST free storage area and allocates an AST hash table. Each entry in this hash table is set with the vacant switch ON to indicate an empty AST. Upon return from this call the SST is initialized but contains no AST, DST or PST entries.

Step 6

The file system core map is initialized by means of the following call.

```
call initialize_core_map;
```

This procedure initializes a core map of the size necessary to accomodate the amount of core available to Multics as defined in the system configuration table. Each block of core is marked as available and currently unassigned.

Step 6

The wired-down process waiting table (PWT) is initialized by means of the following call.

```
call initialize_pwt$wired_pwt;
```

This procedure initializes the wired-down PWT to appear as empty (i.e. no processes waiting).

Step 7

Zero length segments are loaded during initialization for the sole purpose of reserving segment descriptors for specific uses by the hardcore supervisor. To set up some of these segment descriptors to point to specific segments, a call is made to the following initialization procedure.

```
call update_descriptors;
```

This procedure sets the descriptor words of the zero length segments "hardcore_ds" and "current_ds" to point to the current descriptor segment. The descriptor word for the current process data segment "pds" is set to point to the segment loaded by the name "process_data". (During normal Multics operation, the segment descriptor for "pds" points to an interim process data segment if the process is being loaded and points to the real process data segment (i.e. "process_data") while the process is in the loaded state.)

Step 8

Control is returned to the Multics initializer. The above initialization procedures are no longer needed and may be deleted.