Published: 04/24/67

Identification

File System Initialization (Part 2)
R. C. Daley

Purpose

This section provides the specification of the procedures which perform the second part of file system initialization. These procedures run under the control of the Multics initialization control program during the second part of Multics initialization. The main purpose of this part of the file system initialization is to initialize that portion of the file system necessary for dynamic memory allocation. Once this part of the file system is initialized, the remaining segments of the hardcore supervisor can be loaded into a "virtual memory" provided by the file system.

Introduction

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

1. All of the wired-down segments of the hardcore supervisor have been loaded and all external segment references have been prelinked. These segments include all of the segments necessary to process a missing-page fault (e.g. page control, core control, file system DIMs).

2. An interim fault interceptor has been provided. This procedure will call the Multics procedure ~~pagefault~~ page_fault to handle missing-page faults and will call an interim procedure ~~interim_segfault~~ interim_2_segfault to handle missing-segment faults.

File System Initialization

At the appropriate point during the second part of initialization, the Multics initializer makes the following call to initialize the file system.

```
call fs_init_2;
```

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

Step 1

The necessary AST, DST and PST entries are added to the system segment tables (SST) by means of the following call.

```
call update_sst;
```

This procedure creates a DST entry for the hardcore descriptor segment using the current descriptor segment as described in the segment loading table (SLT). Each unused entry in the descriptor segment's page table is set to point to this new DST entry. The necessary software switches are then set in all of the page table words for the descriptor segment.

Next, a PST entry is created for the Multics initializer (which will eventually become the first Multics process) and this new PST entry is linked to the newly created DST entry. The unique identifiers and AST pointers to the initializer's per-process segments (i.e. the KST, hardcore stack, process definitions segment and process data segment) are initialized to zero since this information is not yet available. A relative pointer to the newly created PST entry is then placed in the process data segment (referenced as segment "pds").

Finally, AST entries are created for each segment listed in the segment loading table (SLT) except for the wired down segments of the hardcore supervisor for which the per-process switches are OFF. Pointers to these AST entries are then placed back in the SLT with the related SLT entries. Certain items for these newly created AST entries cannot yet be provided and must be set to zero. The following is a list of these items.

1. The unique segment identifier (id)

2. Pointer to AST entry for parent directory segment (astparent)

3. Index of branch in parent directory (xbranch)

4. Active meter table index (amtindex)

The active file trailer (AFT) for the new AST entry must have the device identification set for the first available device listed in the file system device configuration table and the file length and file pointer items set to zero. This action will ultimately cause the segment to be written on the device specified by the AFT.

The software switches in the page table words must be set appropriately. Page table words for pages currently in use are set to show that the page has been modified. This action will ultimately cause a copy of the page to be written in secondary storage. Unused page table words (marked with directed fault 0) are set to point to the related AST entry. The segment status in the segment loading table entry for the segment is then tested and one of the following actions is taken.

1. If the segment status is wired (i.e. the segment must remain wired down), the wired-down segment count is set to "1".

2. If the segment status is loaded (i.e. the page table must remain in core), the page-table-hold count is set to "1".

3. If the segment status is active or normal, the entry-hold count is set to "1" and a call is made to the segment control utility routine maketrailer to create a process trailer for the AST entry.

Step 2

To update the core map to indicate that all core used by segments already loaded is assigned, a call is made to the following initialization procedure.

```
call update_core_map;
```

This procedure creates core map entries for all core used by segments (other than zero-length segments) listed in the segment loading table (SLT). All of the information needed for creating these entries is provided in the SLT entry and page table of each segment. Entries for hyperpages for segments in wired-down status must be set to indicate that the hyperpage is wired down.

Step 2

Control is returned to the Multics initializer. The remainder of the hardcore supervisor can now be loaded using the virtual memory provided by the file system missing-page fault handler and the interim missing-segment fault handler described below.

The Interim Fault Interceptor

To allow missing page and segment faults to be handled while the remainder of the hardcore supervisor is being loaded and initialized, an interim fault interceptor is initialized by the Multics initialization control program. This fault interceptor responds only to missing page and segment faults (i.e. directed fault 0).

Upon receiving a directed fault 0, the interim fault interceptor processes the fault in the same manner as the normal fault interceptor (see BK.3) with the following exceptions.

1. Since much of the Multics protection ring mechanism has not yet been loaded or initialized, all faults are assumed to have occured in ring 0. (At this stage of system initialization ring 0 is the only ring.)

2. The interim fault interceptor calls the interim segment fault handler (see below) to process missing segment faults.

Note: missing-segment faults are always processed by calling the segment fault handler after switching to the ring 0 pageable stack (i.e. "hardcore_stack") unless the fault occured while referencing this stack. If a segment fault occurs as a result of a reference to the normal ring 0 stack, the call to the segment fault handler must be made using the process concealed stack which is always wired-down.

The Interim Segment Fault Handler

The interim segment fault handler is provided to handle missing-segment faults while the remainder of the hardcore supervisor (including the regular segment fault handler) is being loaded and initialized. During part 3 of file system initialization there is a short period of time (from step 6 to step 10) in which both the interim and normal segment fault handlers must be operational. Therefore,

the interim segment fault handler must determine which segment faults it should handle and which should be passed on to the normal segment fault handler. Segment faults for segments having corresponding entries in the segment loading table (SLT) are processed by the interim segment fault handler while segment faults for segments not listed in the SLT must be passed on to the normal segment fault handler.

When a missing-segment fault is encountered by the interim fault interceptor, the following call is made to the interim segment fault handler.

```
call interim_segfault (scuptr, dbr, ringno, errcode);
```

The parameters used in this call are the same as described for the normal segment fault handler segfault (see BG.3.1). However, the ringno parameter is guaranteed to be zero during system initialization and may be ignored by the interim segment fault handler.

Upon receiving this call, a check is made to determine if the missing segment has a corresponding SLT entry. If no SLT entry exists for this segment, the segment fault is passed to the normal segment handler by calling the segfault primitive of segment control.

If an entry exists in the SLT for the missing segment, the SLT entry may contain a pointer to a previously created AST entry for the missing segment. If no AST entry yet exists for the segment, a new AST entry is created as described in step 1 and a pointer to the new AST entry is placed in the corresponding SLT entry for the missing segment.

Once the AST entry for the segment is found (or created if necessary) a call is made to a page control primitive (getloaded) to provide a page table for the missing segment. Upon return from page control, the page table address in the AST entry is combined with other information in the AST and SLT entries for the segment to make up a segment descriptor word. This segment descriptor word is placed in the appropriate location of the descriptor segment and control is returned to the fault interceptor.