Published: 10 April 1967

Identification

Initializing Core Manager
D. H. Johnson

Purpose

The initializing core manager does core storage allocation for the Multics Initializer until the file system is capable of performing this function. It consists of procedures to handle missing page and segment faults and to maintain the interim core map.

Introduction

From the beginning of the Initializer Control Program (see MSPM BL.5.01) until the end of part 2 when procedure interim_fi$use_mode_2 is called to switch the core allocation responsibility to the file system, the initializing core manager does all core storage management for the Multics Initializer. The initializing core manager uses the interim core map and the segment loading table (SLT) as its primary data bases. The initializing core manager module resides in three segments. Their names and responsibilities are outlined below.

1. core_manager

This segment consists of three procedures;

assign_core - removes a hyperpage of the requested size from the interim core map and revises the core map accordingly,

update_core_map - expands the interim core map using the exact core memory configuration for Multics described in the system configuration table, and

free_core - releases the core assigned to 'temporary' segments and updates the interim core map to include the freed core.

This segment also contains the interim core map;

core_map - an array of entries describing blocks of unused core storage.

2. interiml_pagefault

This segment contains one procedure;

interiml_pagefault - assigns a hyperpage to a segment when a missing page fault occurs.

3. interiml_segfault

This segment consists of a single procedure;

interiml_segfault - assigns a page table to a segment when a missing segment fault occurs and also assigns a hyperpage corresponding to the segment address referenced.

The interim core map and the core manager procedures are described in this document. The SLT is described in MSPM BL.2.01.

Interim core map

The interim core map is a fixed length table of entries describing available but currently unassigned core storage. For convenience, the table is kept in the segment, core_manager. Each entry in the table contains two items which describe an available contiguous area of core. The items are:

a. the absolute address (0 mod 64) of the first word of an area of core
b. the number of contiguous 64 word blocks in the area

The interim core map is initialized by the Bootstrap Initializer (see MSPM BL.4.02) with one entry which defines the unused core based on an assumed configuration. Other entries are added to the core map when the Multics hardware configuration is known and when 'temporary' segments are released. These two events are described below in procedures update_core_map and free_core respectively. The core map entries are ordered in ascending sequence of the absolute address items.

The core map is declared by the following PL/I statement.

```
dcl 1 core_manager$core_map ext,
    2 current_size fixed,  /* current number of entries
                              in the core map */
    2 bootstrap_ceiling fixed,
                           /* highest address in memory assumed
                              by Bootstrap Initializer */
    2 entry(N),            /* N is decided upon at compilation */
    3 address bit(18),     /* absolute address 0 mod 64 */
    3 blocks bit(18);      /* number of 64 word blocks */
                           /* if 0, the entry is vacant */
```

Procedures

The primary core manager procedure is assign_core, which satisfies requests for blocks of core storage. It is called by the interiml_pagefault and interiml_segfault procedures. This procedure is entered by the statement:

```
call core_manager$assign_core(search_sw,page_64_sw,
                              hyperpage_size,abs_loc) ;
```

The arguments are declared in the PL/I statement:

```
dcl search_sw bit(1),        /* ON if bottom up,
                                OFF if top down */
    page_64_sw bit(1),       /* ON if 64 word page,
                                OFF if 1024 word page */
    hyperpage_size bit(8),   /* in units of page size */
    abs_loc bit(18);         /* absolute location
                                of hyperpage assigned */
```

This procedure examines the core map for an entry which describes an area of core that satisfies the request. The address of the core area is returned in the argument, abs_loc. If the request can not be satisfied, system initialization stops.

The search algorithm used on the core map represents a simplified attempt to keep the wired down hard-core supervisor separate in core from the rest of the Multics Initializer. This is done to minimize the holes in the wired down area of memory. The core map entries are searched in the order specified by the argument, search_sw. The procedures that call assign_core will set search_sw OFF only when requesting blocks for the wired down hard-core supervisor (wired down segments, page tables for wired down segments, and page tables for loaded segments). This has the effect of putting the wired down supervisor in lower addressed core and the initializer and active supervisor in higher addressed core.

The steps taken by procedure assign_core are:

1. Determine the mode of searching the core map. If search_sw is ON, begin searching at the last core map entry and, if necessary, move up the list. Also, examine the core map entry from the bottom of the area it describes. If search_sw is OFF, begin searching at the first entry and, if necessary, move down the list. Examine the core map entry from the top of the area it describes.

2. Determine whether the entry examined is capable of providing a hyperpage of the desired page size and length. Arguments page_64_sw and hyperpage_size describe the requested area.

If the core area described by the entry contains a hyperpage of the requested size, place the address of the beginning of the hyperpage in abs_loc. Update the core map entry. If the hyperpage was taken from the middle of the area, the blocks skipped over are lost from the interim core map. Return to the caller.

If the entry can not satisfy the request, go to step 3.

3. Determine whether there is another entry in the core map. If there is, go to step 2. Otherwise, stop system initialization.

When the interim fault interceptor, operating in mode 1 (see MSPM BL.5.02), gets control after a directed fault zero, it calls either the interim page fault handler or the interim segment fault handler for that mode. The page fault handler is called as follows:

```
call interiml_pagefault(scuptr)  ;
```

where scuptr is a pointer to three double words containing information stored by an scu instruction when the fault occurred. This procedure does the following steps:

1. The scu information is interpreted to determine if the missing page is a descriptor segment page. If it is not a descriptor segment page, go to step 6.

2. Call procedure assign_core to obtain a 64 word page for the descriptor segment. The procedure arguments are set as follows:

```
search_sw - OFF
page_64_sw - ON
hyperpage_size - 1
```

3. Prepare a page table word pointing to the assigned page and store it into the descriptor segment page table. The access control field is set to write permit, slave access, master procedure.

4. Fill the assigned page with directed faults (segment faults).

5. Return to the calling procedure.

6. The scu information is examined to determine the number of the segment getting the missing page fault. This number is then used to reference the SLT entry for the segment.

7. Call procedure assign_core to obtain a hyperpage from the core map. The SLT entry items are used to set the arguments passed to assign_core. If the status item in the SLT entry indicates that the segment is wired down, set search_sw to OFF; otherwise, set it to ON. The page_64_sw and hyperpage_size

arguments are taken directly from the SLT entry and passed to assign_core.

8. The page table word(s) are filled in with the address(es) of the hyperpage. The access control field in each page table word is set to write permit, slave access, master procedure. The hyperpage is filled with zeros.

9. Control returns to the caller.

The segment fault handler is called as follows:

call interiml_segfault(scuptr) ;

where scuptr has the same meaning as in the call to interiml_pagefault. This procedure does the following steps:

1. The scu information is used to determine the number of the segment getting the missing segment fault. The SLT is checked to see if the specified segment number corresponds to a valid entry in the SLT. If the segment number is not valid, system initialization stops.

2. Call procedure assign_core to obtain a hyperpage to be used for a page table. The SLT entry items are used to set the arguments passed to assign_core. If the status item indicates that the segment is wired down or loaded, set search_sw to OFF; otherwise, set it to ON. The argument, page_64_sw, is set ON. The SLT entry items maximum length and 64 word paged switch are used to determine the size of the page table and also the descriptor boundary field value used in step 4. The page table size is set in the argument hyperpage_size.

3. The directed fault zero in the descriptor entry word is replaced by a temporary descriptor. The temporary descriptor allows the interiml_segfault procedure to reference the page table as an unpaged data segment and fill it with directed faults (page faults).

4. The real descriptor for the segment is placed in the descriptor entry word. The address continues to point to the page table. The boundary field value , calculated in step 2, is placed in the boundary field. The SLT item 64 word paged switch is placed in the page size bit in the descriptor word. The paging bit is set to indicate the segment is paged. The SLT item access is placed in the right most 6 bit field of the descriptor word.

5. Call procedure interiml_pagefault to assign the hyperpage which corresponds to the address being referenced when the fault occurred.

6. Control returns to the caller.

There are two procedures that expand the interim core map. After the Initializer Control Program has loaded the system configuration table it issues the following call:

```
call core_manager$update_core_map ;
```

This procedure does the following steps:

1. Examine the system configuration table to find out the exact core memory available to Multics. The system configuration table (see MSPM BL.3.01) information examined includes:

a. memory address range for each system controller
b. processor base address
c. mailbox addresses for each GIOC
d. mailbox addresses for each EMM

2. Interim core map entries are constructed for all of the Multics memory that is not represented in:

a. Bootstrap Initializer assumed memory, or
b. GIOC and EMM mailboxes that are not within the Bootstrap Initializer assumed memory.

The Bootstrap Initializer assumed memory begins at the processor base address and ends at the address given in the interim core map item bootstrap_ceiling.

3. The interim core map is sorted on the absolute address field to keep the entries in ascending order.

4. Control returns to the caller.

Before the Initializer Control Program enters part 2, it calls the following core manager procedure:

```
call core_manager$free_core ;
```

This procedure releases all initialization segments that are no longer needed and returns the core occupied by them to the interim core map. The procedure does the following steps.

1. A pass is made through the initializer entries in the SLT to locate the segments to be released. Each initializer entry is examined as follows:

a. If the temporary segment switch is OFF, this segment may not be released. The entry is skipped and the next entry is tested.

b. Otherwise, the segment may be released. The descriptor entry word for the segment is modified so that its page table may be referenced by free_core as an unpaged data segment. The page table entry words are examined and used to construct core map entries. After all of the pages have been returned to the core map, the page table itself is returned to the core map. A directed fault seven is placed in the descriptor segment entry. Control then goes to step a. to examine the next SLT entry.

2. When all of the 'temporary' segments have been released, the core map is sorted on the absolute address field to keep the entries in ascending order.

3. Control returns to the caller.