Published: 10 April 1967

Identification

The Fault Initializer
Chester Jones

Purpose

This section provides the specification of the fault initializer which is run under control of the Multics initialization control program (MSPM Section BL.5.01). The basic goal of the fault initializer is to initialize the fault interceptor module (MSPM Section BK.3.02) and to establish the data bases associated with fault handling. This section assumes a thorough knowledge of Multics fault handling and a general knowledge of Multics system initialization.

Introduction

When the initialization control program passes control to the fault initializer, the hardcore supervisor is in the following state.

1. All of the segments of the hardcore supervisor, which are required at initialization time, have been loaded into core and all external segment references have been linked by the prelinker. (All of the data bases used for fault handling have been provided by the segment loader although their contents are zero.)

2. The system configuration table which describes the hardware available to the Multics system has been built by the configuration table generator (MSPM BL.5.03). Of particular interest to the fault initializer are the system controller-processor assignments, the memory port number assignments, and the address ranges assigned to each system controller.

3. The initialization linker has been provided by the initialization control program; symbolic intersegment references made by the fault initializer cause linkage faults which are handled by the initialization linker. (Although the hardcore supervisor segments have been pre-linked, the fault initializer has not.)

4. The process concealed stack (MSPM Section BJ.1.05) and the processor stack (MSPM Section BK.1.03) have been provided and initialized by the Bootstrap program.

Fault Initialization

In order to initialize the fault interceptor module, the initialization control program makes the following call.

```
call fault_init1
```

Upon return from this call, the fault interceptor module is fully initialized, but not yet in operation. The steps required for this initialization are as follows.

1. The segment descriptor word for the fault interceptor segment is changed (temporarily) from "master procedure" to "data, writing permitted."

2. The linkage pointer and linkage base (lp-lb) values for the fault interceptor are obtained from the Segment Loading Table and stored "inside" the fault interceptor procedure. (Since the fault interceptor is not called, it must be able to establish its own linkage values using an "internal" address.) To obtain the pointer to the linkage section for the fault interceptor, the fault initializer performs the following call.

   ```
   call slt_manager$get_link_seg_ptr(text_ptr,link_ptr,errtn)
   ```

3. Pointers to the various locations in the process concealed stack and the processor stack are built and stored "inside" the fault interceptor procedure. (The fault interceptor must save and restore the processor state using only "internal" addresses.)

4. The segment descriptor word for the fault interceptor segment is changed from "data, writing permitted" to "master procedure."

5. The processor communication tables (MSPM Section BK.4.01) are initialized. In particular, the following arrays are built based on information obtained from the system configuration table.

   a. time-out pattern array

   b. pre-emption pattern array

   c. quit pattern array

   d. time-out pointer array

   e. pre-emption pointer array

   f. quit pointer array

   g. connect operand word array

   (At first, the processor communication tables will be read from tape. This step is included here as a reminder for later.)

6. Entries in the process definition segment (MSPM Section BJ.1.06) that are used by the fault interceptor are initialized. Of particular interest to the fault interceptor is the 'stacks' array which contains pointers to the base locations of the various paged stacks for the user process. (When a process is created, it must be christened with a non-empty 'stacks' array.) The first entry of the 'stacks' array is set to point to the base location of the initial process' ring 0 paged stack.

7. Control is returned to the Multics initialization control program.

After all modules that are called by the fault interceptor have been initialized, the Multics initialization control program issues the following call to initialize the fault vector and to switch to the Multics fault interceptor.

```
call     fault_init2
```

Upon return from this call, the Multics fault handling mechanism is fully operable. (An important implication of this is that the Multics protection ring mechanism is in effect; the Gatekeeper must have been initialized.)

To initialize the remainder of the fault vector, the fault initializer "compiles" instruction pairs of the form

```
scu      a,*
tra      b,*
```

where a is an ITS-pointer to the concealed stack and b is an ITS-pointer to the fault interceptor. The fault initializer builds the necessary pointers in segment 'vector_redirector'.