Published: 11/03/67

Identification

init_proc
R. L. Rappaport

Purpose

This document describes one of the subroutines involved in process initialization. An overview of process initialization is presented in BJ.2.02.

Introduction

As explained in the overview of process initialization, initializing a process means initializing the address space of the process. This includes creating several segments (e. g. stack segments) and prelinking a path through the linker, in the new address space, so that linkage faults can be handled dynamically.

A new process begins execution in subroutine swap_dbr (see BJ.5.01). Among the items placed in the new process address space, by the creator of the new process, is a flag which is used to notify swap_dbr the first time the new process begins execution. In other words, if swap_dbr finds the flag on in a particular address space, then swap_dbr turns it off and sets the new process on the path to self initialization. Swap_dbr in fact calls init_switch (see BJ.9.06) which in turn calls init_proc. Init_proc never returns from this cell.

Discussion

Subroutine init_proc is called with no arguments. That is the calling sequence is simply:

```
call init_proc;
```

The stack in use at the time of the call to init_proc is the fault_stack (see BJ.1.06).

Init_proc is basically a simple driving program which performs several tasks by calling upon other modules to do them. The tasks are:

1. The creation of a call stack in the hardcore ring and a call stack in the administrative ring.

2. The pre-linking of the segments involved in dynamic linking.

3. The passing of control from the hardcore ring of the new process into the administrative ring.

The actual actions taken by init_proc are discussed below.

Creating stacks is simply done by calling subroutine create_stack (see BD.9.08)specifying the ring in which the stack will reside. Therefore init_proc merely calls this routine twice to create the needed segments.

The prelinking of the linker segments is accomplished by calling subroutine pre_linker_driver (see BJ.9.02). Pre_linker_driver uses as data the table produced by create_linker_segs (see BJ.8.03) at process creation time (i.e., the pre_linker driving table).

Finally, init_proc gives up control by calling subroutine gate_init (see BJ.9.03). Gate_init never returns to init_proc since it passes control outward to the administrative ring.