TO: MSPM Distribution
FROM: M. A. Padlipsky
SUBJ: BL.11
DATE: 02/27/68

The attached revision of BL.11 reflects the actual implementation
of the Traffic Controller Initializer.

Identification

Traffic Controller Initializer
R. L. Rappaport, G. Benjafield

Purpose

This section provides the specification of the Traffic
Controller Initializer which is called by the Initializer
Control Program (see Section BL.5.01). The basic goal
of this initializer is to initialize the data bases associated
with the Process Exchange and the Process Creation modules
and to initialize the hardware processors of the system.
This section assumes a thorough knowledge of the Process
Exchange, the process creation machinery, and the Multics
Initializer.

Introduction

At the time the Traffic Controller Initializer is invoked, the
system is in the following state:

1.   All hardware processors, except the one on which the
      Initializer is executing, are in a disconnected state.
      That is, they are executing the DIS (Delay until Interrupt
      Signal) instruction.

2.   The Multics Initializer, as far as the Process Exchange is
      concerned, is not yet a process.

3.   The block and wakeup entries to the Process Exchange are
      disabled. That is, the two data switches block_enable
      and wakeup_enable, which reside in segment tc_data are off.

4.   The drain switch (see Section BJ.6) in the Processor data
      segment being used by the Initializer is on. This switch
      effectively disables timer-runout interrupts.

5.   The Active Process Table is not initialized.

6.   The Process Creation data bases have not been initialized.
      These data bases are the template segments used in creating
      and loading processes.

7.   The File System Device Monitor process has not yet been
      created.

8.   The Basic File system has been initialized.

The goal of the Traffic Controller initializer is to negate
the first seven statements above. Before discussing the
overall strategy of the Traffic Controller initializer,
some of the problems involved in the tasks outlined above
are presented.

First, there are two reasons why block and wakeup are disabled
at this time.

1.   During file system initialization, both entries are called.
      Since, from the vantage point of the Process Exchange, the
      initializer is not yet a process, these calls must be
      diverted. This is accomplished by disabling the entries.

2.   The second reason for disabling the entries is that the
      File System Device Monitor process has not yet been created.
      In order for paging to work without this process, block
      must be disabled and an alternate procedure must be executed
      every time block is called (see Section BL.5.04).

Description

The calling sequence for the Traffic Controller initializer
is simply:

      call tc_init;

The steps taken by tc_init are as follows:

1.   The Active Process Table (APT) is initialized. That is,
      the entries are cleared and a thread is built linking all
      the entries into the "empty list".

2.   Several subroutines are called to initialize various data
      bases. They are:

          a)   apt_hash$init (see BJ.7.02)
          b)   build_template_dseg   (see BL.11.04)
          c)   build_template_pds   (see BL.11.05)
          d)   build_template_pdf   (see BL.11.06)
          e)   get_proc_id$init   (see BJ.7.03)

      The names of these subroutines imply their functions.

3.   The initializer process is made to resemble a process from
      the vantage point of the Traffic Controller. That is, an
      APT entry is allocated for this process and initialized
      (by calling the internal procedure alloc_apt_entry). The
      process is defined to be running on the BootIoad processor.

Other items initialized in this entry are the process_id
(obtained from pds$processid), the process segment table
entry pointer (obtained from pds$pstep), and the segment
descriptor word (SDW) for this process descriptor segment
(obtained from the current descriptor segment).

4.   An idle process is created and initialized for each distinct
      processor in the system. The following paragraph
      describes how these idle processes are created and
      initialized. Note that the idle process for the Bootload
      processor is handled in a slightly different manner than
      the others.

      The descriptor segments and process data segments for idle
      processes are created by calling the system
      initialization procedure create_hardcore_proc. While
      a processor data segment need not be created for the
      Bootload processor, one must be created for each of the
      other processors and its SDW stored into the descriptor
      segment of the appropriate idle process. File system
      subroutine make_seg is called to create the processor
      data segment and create_hardcore_proc stores the SDW.
      APT entries are allocated for each idle process and
      initialized; these APT entries indicate that they will
      only run on the processors with which they are associated.
      While the Bootload idle process is initialized to be in
      the blocked state, the other idle processes are
      initialized to the running state.

      After this step the APT indicates that each processor is
      executing. On the bootload processor, the APT indicates
      that the initializer process is executing while each
      other processor appears to be executing its own idle process.
      In reality only the bootload processor is running at
      this point.

5.   The drain switch (prds$drain_switch) is set to zero. This
      enables timer-runout interrupts to be effective.

6.   The File System Device Monitor (FSDM) process is created
      as are the idle processes by calling create_hardcore_
      proc. Once this process has been created and initialized,
      Traffic Controller subroutine block can be enabled. (The
      FSDM process itself sets the data item tc_data$block_enable
      to "1"b at the appropriate time.) After initializing the
      APT entry of the FSDM process, the initializer process
      sets tc_data$wakeup_enable to "1"b and calls wakeup for the
      FSDM. As a result, the FSDM process is positioned at the
      head of the ready list.

In order to give control of the Bootload processor to the
FSDM process, the initializer sends itself a timer-runout
interrupt (by calling master_mode_ut$set_cell, see BK.5.04).
When the initializer regains control of the processor, it
tests the status of tc_data$block_enable. If this data item
is still zero (an indication that the FSDM process is not
yet finished), the initializer interrupts itself again. It
remains in this interrupting and testing loop until the data
item becomes equal to "1"b.

7.   Finally, the other processors in the system are initialized.
      This is done by a call to init_processor (see BL.11.03) for
      each processor. This procedure interrupts the appropriate
      processor with the initialize interrupt which has the effect
      of loading the descriptor segment base register of the
      processor with the sdw of the descriptor segment of the
      appropriate idle process and transferring to an initialize
      entry in init_processor.

      At this point, tc_init returns to its caller.