TO: MSPM Distribution

FROM: R.M. Graham

SUBJ: BL.4.00

DATE: April 5, 1967

Please add to BL.4.00 dated 3/24/67: (Supersedes: BL.4.00, 10/21/66)

Published: 03/25/67
Supersedes BL.4.00 10/21/66

Identification

Bootstrap Initializer -- Overview
A. Bensoussan

Purpose

The Bootstrap Initializer constitutes the interface between the bootload program (BC.4.01) and the Multics Initializer. Its purpose is to change the environment left by the bootload program to the appropriate environment required by the Multics Initializer for being able to stand alone.

This section describes what is required by the Multics initializer, why it is required and how it is provided.

General Discussion

The Multics Initializer is a GE 645 program which presents the following characteristics: (see figure 2 at the end of this section)

1. It runs in appending mode; therefore a descriptor segment must be created and the appending mode set.

2. It consists of a main program, the "Initializer control" program, which issues various calls to appropriate initialization procedures; therefore:

   a. the initializer control program has to be loaded

   b. the linkage mechanism must be made available; this requires:
      - to load a linker, the "initialization linker",
      - to initialize the linkage sections used by the linker
      - to create and initialize the "SLT" (segment loading table) which contains information about all the segments described in the decriptor segment, and that is used by the linker to find the segment number of the referenced segment.
      - to initialize the linker so that it can reference the SLT without getting a linkage fault,

- to provide a fault catching mechanism such that recursive linkage faults can be handled properly This means that a fault interceptor, the "interim_fi", is needed and has to be loaded and initialized for calling the initialization linker when it recognizes a linkage fault. The fault vector must be initialized to direct a linkage fault to the interim_fi. In addition, the process data segment of the future Multics initializer process is loaded and initialized, providing the concealed stack needed for storing control unit information without getting a page fault.

c. the standard call, save, return macro must work properly; this implies that:
   - a stack is provided
   - the base address registers are paired and initialized according to Multics conventions.

Rather than providing an interim stack, the hardcore stack of the future Multics initializer process is created.

3. It starts calling a procedure, the "segment loader" capable of loading segments when they are needed. Therefore:

   a. the segment loader must be loaded

   b. all the procedures called by the segment loader must be loaded; they are the "tape reader" and the "SLT manager".

   c. all the data bases used when the segment loader is called must be created and initialized; they are:
      - the SLT, needed by the SLT manager (it was already mentioned as a segment needed by the initialization linker)
      - the "physical record buffer", needed by the tape reader; it contains the last physical record read from the MST and the current pointer to the first logical word available in this buffer.

4. When the segment loader is executing, it allocates core each time a missing page fault occurs and it creates a descriptor word and a page table each time a missing segment fault occurs. Therefore:

a. a missing page fault handler, the "interim1_pagefault" segment has to be loaded,

b. a missing segment fault handler, the "interim1_segfault" has to be loaded,

c. the interim_fi mentioned above has to be initialized to direct missing page and segment faults to the appropriate handler,

d. the fault vector must be initialized to direct these faults to the interim_fi.

5. When the tape reader issues a CONNECT GIOC, it is expecting the interrupt to be directed to its own interrupt handler. Therefore the interrupt vector has to be initialized in such a way that this condition is satisfied.

6. When any unexpected faults or interrupts occur, the Multics Initializer expects them to be directed to the segment "stop". Therefore, the fault-interrupt vector must be initialized in such a way that this condition is satisfied.

It is easy to see, after this general discussion, that 2 different kinds of actions have to be taken in the Bootstrap Initializer: the first one consists of loading or creating segments; the second one consists of initializing certain critical segments that have been loaded.

That is why the bootstrap initializer is implemented in 2 distinct segments:

Bootstrap1 which loads and creates segments, and

Bootstrap2 which initializes the loaded segments.

Environment before the Bootstrap Initializer

It is summarized in Figure 1, located at the end of the present section. A complete description of this environment can be found in BC.4.01. Numerical values of standard tape format parameters h, l and t are given in BB.3.01.

Environment after the Bootstrap Initializer

It is summarized in Figure 2, located at the end of the present section.

1. The following segments and their associated linkage sections (if any) are in core in an operable form:

   - Descriptor segment
   - Fault vector
   - Mailbox
   - Bootstrap1
   - Bootstrap2
   - Initializer control program
   - Segment loader
   - Tape reader
   - Physical record buffer
   - SLT
   - SLT manager
   - Interim fault interceptor
   - Interim1 page fault
   - Interim1 seg fault
   - Initialization Linker
   - Process data segment
   - Hardcore stack
   - Stop segment

2. Each of these segments has an entry in the SLT

3. Three kinds of faults can be handled:

   a. Linkage fault

   b. missing page fault

   c. missing segment fault

   Any other faults are directed to the segment STOP.

4. Only the interrupts coming from the bootload GIOC status channel 0 or 1 can be handled.

   Any other interrupts are directed to the segment STOP.

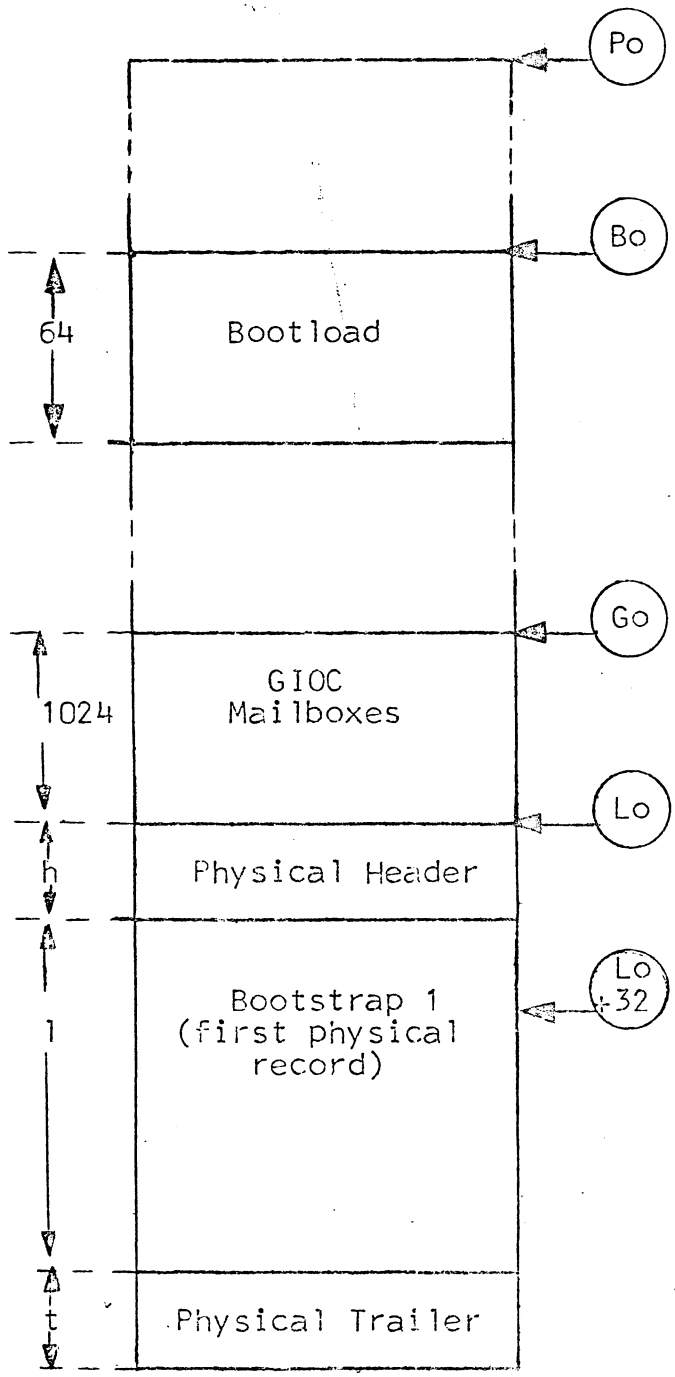

Figure 1: Environment before BOOTSTRAP INITIALIZER

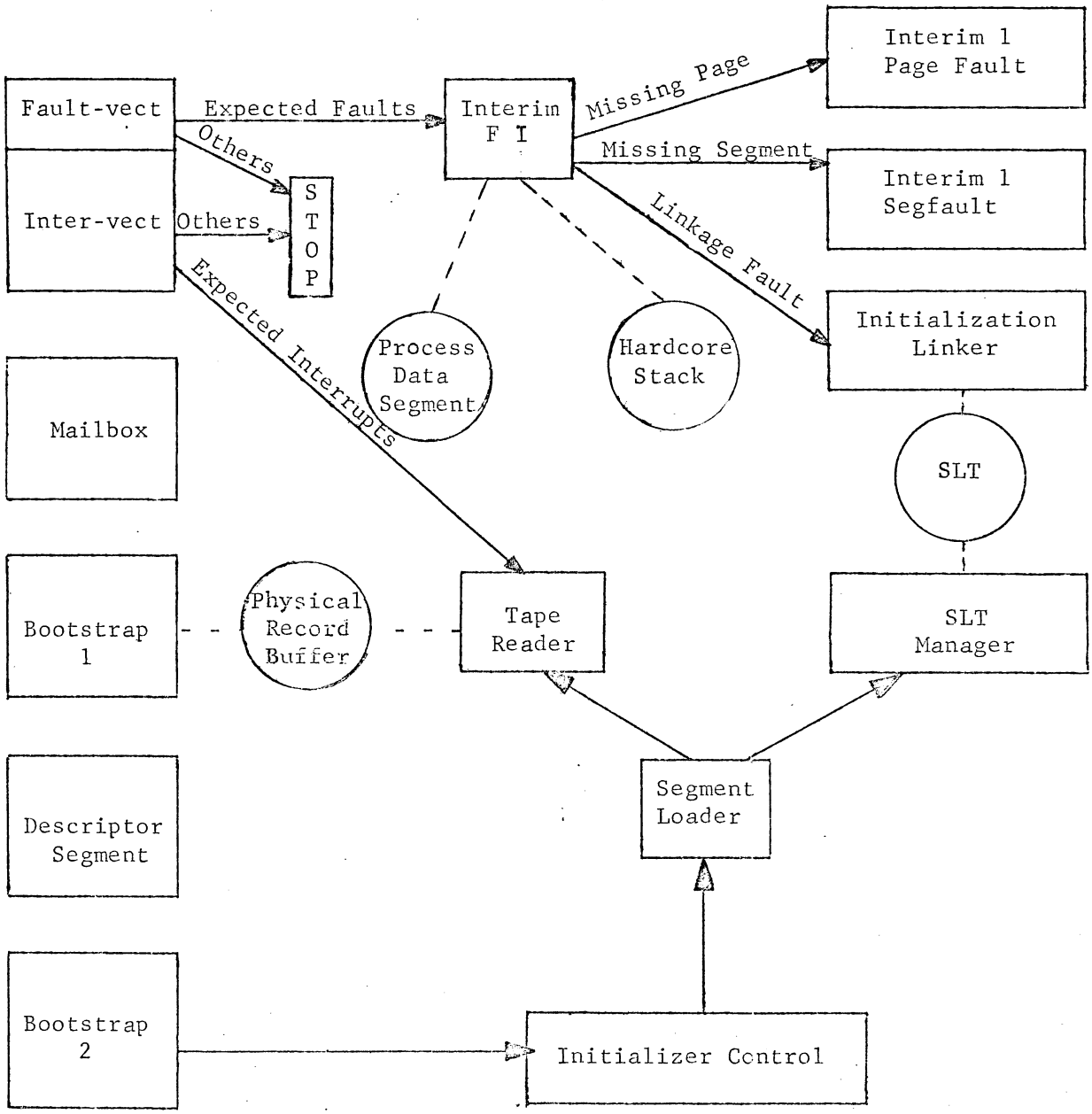

Figure 2: Environment after BOOTSTRAP INITIALIZER