MCR10093
Fix move_non_perm_wired_segs.pl1
Revision 1.2

Eric Swenson

October 5, 2021

1 Problem Description

When booting Multics with certain config decks, the system crashes with the error:

```
 0921.5 initializer: 2033 establish_config_deck state 1 phase 2.
 1721.5 initializer: 3009 scas_init state 1 phase 3.
 1721.5 initializer: 3010 tc_init state 1 phase 3.
 1721.5 initializer: 3011 init_sst$normal state 1 phase 3.
 1721.7 initializer: 3012 disabling slt allocation state 1 phase 3.
 1721.7 initializer: 3013 initialize_faults$interrupt_init state 1 phase 3.
 1721.7 initializer: 3017 init_pvt state 1 phase 3.
 1721.7 initializer: 3018 read_disk$init state 1 phase 3.
 1721.7 initializer: 3019 init_root_vols state 1 phase 3.
 1721.7 initializer: 3027 load_mst$make_permanent state 1 phase 3.
 1721.7 initializer: 3028 scs_and_clock_init$date_time state 1 phase 3.
 0921.7 initializer: 3029 io_config_init state 1 phase 3.
 0921.7 initializer: 3030 ioi_init state 1 phase 3.
 0921.7 initializer: 3031 init_toehold$save_safe_config_deck state 1 phase 3.
 0921.7 initializer: 3033 establish_config_deck state 1 phase 3.
 0921.7 initializer: 3034 init_partitions state 1 phase 3.
 0921.7 initializer: 3035 make_segs_paged state 1 phase 3.
 0921.7 initializer: 3036 collect_free_core state 1 phase 3.
 0921.7 initializer: 3037 delete_segs$temp 1 state 1 phase 3.
CONSOLE: ALERT
 0921.7 page_fault: fatal error at loc 2330
 0921.7 Multics not in operation; control process: Initializer.SysDaemon.z.
```

Investigations performed by Charles, Gary, and Eric revealed that the crash is in the collection 1 program `delete_segs$temp`. This program attempts to delete a segment, one of whose pages is also claimed as a page of a wired segment. The wired segment has taken ownership of the CME that describes the page. Attempting to delete the paged segment, `delete_segs$temp` calls `pc$truncate` to cleanup ownership of that segment's active pages by zeroing their PTWs, marking their CMEs as free, and decrementing the count of in-memory pages in their ASTE.records.count. But the CME is no longer associated with the paged segment; it was usurped by the wired segment. `CME.astep`

is invalid, having been previously zeroed, and creating a pointer to the ASTE results in a bogus pointer to the header of the SST. The resulting value of `ASTE.records` is 0 (invalid because not pointing to a real ASTE). When `pc$truncate`, as one of its functions, decrements that count, it becomes negative, leading to the crash.

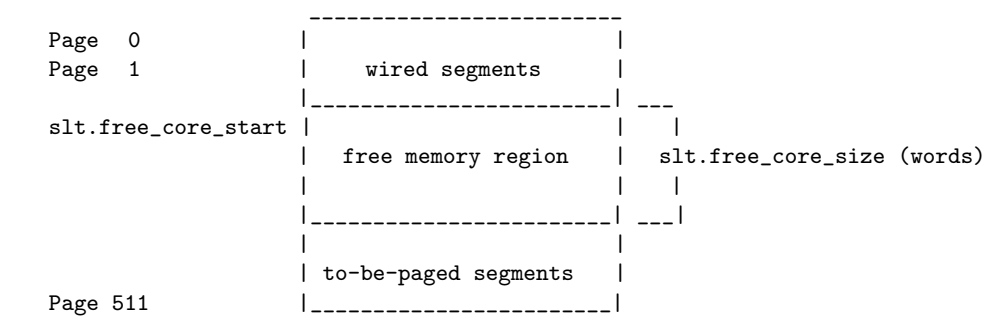
The bug is not, however, in `delete_segs$temp`, but in another collection 1 program, `move_non_perm_wired_segs`, which runs earlier during initialization.

A ticket describing the problem can be found here: https://multics-trac.swenson.org/ticket/242. A detailed description of the analysis that led to the discovery of the bug, can be found in the section "Analysis", found later in this document.

2 Description of Bug

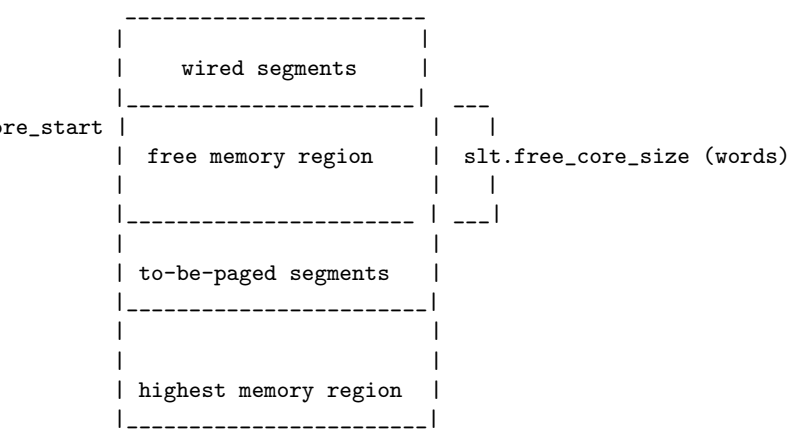
The Multics system bootload is performed in four major stages, called collections. Each collection adds mechanisms of the Multics segmented/paged PL/I execution environment, then loads program/data segments needed to run the next collection.

Collections 0 and 1 store segments in three regions of bootload memory: the first 512 (decimal) pages of real memory. The lowest region contains segments which will eventually be permanently wired into real memory after the full Multics service environment is setup. The highest region contains segments which will eventually be paged segments. These two regions are separated by an adjustable region containing unused pages in real memory. This free memory region shrinks as segments are loaded into the other two regions, or grows as segments are removed from those other memory regions.



Each segment from the Multics System Tape (MST) is loaded into one or more contiguous memory pages in either the region of wired segments or to-be-paged segments, with an `sdw.address` identifying the starting page address of the segment, and `sdw.length` set to size of that segment in words. Page tables are not generally used for segments loaded in collections 0 and 1.

When collection 1 runs, an early phase locates the config deck. The `move_non_perm_wired_segs` program reads MEM config cards to learn how much memory is available to Multics, then moves the to-be-paged segments to a highest memory region (beyond page 511 if such locations are configured as online).



When selecting segments to be moved, the `move_non_perm_wired_segs` algorithm compares each segment's `sdw.address` field to ensure that it resides in the to-be-paged segments region. However, a faulty comparison is used:

```
if  sdw.address <= slt.free_core_start + slt.free_core_size, then SKIP_THE_SEG_MOVE.
```

The above code is pseudo-code. See the `compare_ascii` output later in this MCR for the actual code, which actually appears twice in `move_non_perm_wired_segs`.

The comparison should have used the < operator rather than <= for this check. The result is that the last initialization segment processed is not moved. This segment has a base address matching exactly the boundary between free memory and to-be-paged memory (meaning it starts at the word just after the end of free memory). However, the pages of memory occupied by this segment (and all correctly moved segments) are afterwards considered free. The free memory region is expanded to include these pages, even though one segment still has a memory range in the free region.

Later, during initialization, another segments gets allocated at least one of these pages, resulting in two segments sharing at least one page of memory. Still later, page control notices that a page it is freeing has already been marked free, and crashes the system.

This description is deliberately simplified so as not to bore the reader. Those interested can read all the details in the "Analysis" section.

3 Proposed Change

The fix needs to be made to both places where the comparison operator is incorrect – the one that processes initialization segments and the other, that processes supervisor segments. The latter isn't strictly necessary, because that condition will never be met, but nonetheless the <= is wrong.

The fixed code is:

```
    if init_si.address < slt.free_core_start + slt.free_core_size then init_si.faulted =
        "1"b;
```

compare_ascii output of the comparison between the MR12.7 (also MR12.4, MR12.5, MR12.6x) version of this program and the fixed version can be found below:

```
cpa [lpn move_non_perm_wired_segs.pl1] ==

A172                                if init_si.address <= slt.free_core_start +
    slt.free_core_size then init_si.faulted = "1"b;
Changed by B to:
B172                                if init_si.address < slt.free_core_start +
    slt.free_core_size then init_si.faulted = "1"b;


A187                                if sup_si.address <= slt.free_core_start +
    slt.free_core_size then sup_si.faulted = "1"b;
Changed by B to:
B187                                if sup_si.address < slt.free_core_start +
    slt.free_core_size then sup_si.faulted = "1"b;

Comparison finished: 2 differences, 4 lines.
r 13:05 0.986 15
```

4 Analysis

This section details the investigation and analysis that led to the discovery of the root of the problem and pointed to the code in which the bug lay. A reader not interested in this detail may skip to the next section.

As mentioned above, the immediate reason for the crash is that `delete_segs$temp` tries to delete the initialization segment, `free_area_1`, segment number 436, by calling `pc$truncate` to remove each page from memory and mark its CME as free. Its page 0 PTW points to page 337, which has `cme.astep` equal to `"0"b`, so its ASTE appears to overlay the header of the `sst$` segment. The value of `astep` is 0, as is the value of `aste.records`, and an attempt to evict one the segment's active pages decrements this value. The result is that the value goes negative, a condition that page control checks against, and upon finding a negative value, crashes the system with the message described in the first section of this MCR.

The reason why the CME for page 337 page 0 of `free_area_1` is zero is because in a prior initialization step, `collect_free_core`, while processing the segment `idle_pdses`, segment number 60, zeroed out the CME for its page 0 (except for the `cmd.abs_w` bit). Its page 0 references page 337 of memory. At this point, there were two segments: `free_area_1` and `idle_dsegs` whose page 0 referenced page 337 of memory. `collect_free_core`, which runs after all pageable segments have been allocated disk records in the hardcore partition (and thus no longer needing memory allocated to them), collects the newly freed memory and causes the pages to be added to the free list.

Note: During the course of analyzing this problem, Charles, Gary, and I repeatedly confused ourselves with segment numbers. Specifically, there are two segments of interest with similar segment numbers: `free_area_1` (segment 436), and `fw.msp800.msp8` (segment 446). The significance of thissegment will be revealed shortly, as it plays an important role in leading to the crash.

In order to understand why we got to a situation where both `free_area_1` and `idle_pdses` shared page 337, we have to go back a ways in initialization. As mentioned already, the crash is detected in `delete_segs$temp`, which is preceded by `collect_free_core`. Prior to this, `make_segs_paged` is run. This program "makes segments paged" by creating a new SDW and ASTE for those segments marked pageable. It assigns a disk record in the hardcore partition for each non-null page in the segment as well as a free CME representing a page of memory. These are stored in the PTWs for the segment. It copies each page of the old segment to the new segment and marks the CMEs for the old pages as free, threading those pages onto the `sst$usedp` list of free CMEs.

It seemed likely that "the bug" was in `make_segs_paged` (it wasn't) and this is where I started my investigation and debugging. I added `syserr` calls to `make_segs_paged` (printf debugging) to see what segments were made paged, what their old memory base address was, and specifically looked for references to page 337. What I found, surprised me because a third segment became relevant. It was the disk firmware segment, `fw.msp800.msp8` (segment 446) and its starting address in contiguous memory was 676000, or page 337.

Now segment 446 is among three disk firmware segments (not marked as firmware in hardcore header, because only the tape firmware, needed in early collection 0, are so-marked). The other two are segment 444 and segment 445. All three of these segments are treated (or intended to be treated) in exactly the same way. But the `syserr` calls revealed that segments 444 and 445 had base addresses near the top end of physical memory, but segment 446 had a low memory address. The obvious distinction between segments 444 and 445, and segment 446 is that segment 446 is the last of the initialization segments. I immediately suspected a fencepost or off-by-one problem somewhere earlier in initialization.

I looked back to how collection 0 loaded collection 1 (where these firmware segments are loaded). The previously-described SLT allocator is used to allocate memory, using the scheme discussed earlier in this MCR. As `bootload_loader` loads the disk firmware segments from the Multics System Tape (MST), it allocates space for them in bootload memory in the region of memory referred to in this MCR as "to-be-paged memory". The segments are allocated in the order in which they are defined in the MST, first segment 444, then 445, and finally 446 – the last segment allocated. This allocation strategy would place all three of these segments near the end the 512 pages of bootload memory. But the `syserr` calls indicated that segments 444 and 445 resided at the end of physical memory, while segment 446 was still near the top of bootload memory.

This suggested that some mechanism had moved 2 of the 3 segments to the top of memory, but somehow the third segment was missed. I immediately suspected that whatever mechanism did this moving had intended to move all three segments (along with others not relevant to this issue), and that most likely, the memory consumed by all three segments (and others) was considered free, leading, later to the reallocation of pages still allocated to the missed segment, `fw.msp800.msp8`, or segment 446.

I found that the program whose responsibility it was to move these segments "high" was `move_non_perm_wired_segs`. This is a collection 1 program, loaded by collection 0. It has rather complicated logic and I added syserr calls to see whether it was even processing segment 446. I found that it was indeed processing segments 444, 445, and 446, but only moving 444 and 445 high.

This led to the discovery of the bug in `move_non_perm_wired_segs`, described earlier in this MCR.

The following timeline highlights the important steps and consequences that have bearing on this bug.

- Collection 0 loads segments which compose collection 1 programs and data. Those tagged as firmware or initialization segments are allocated to contiguous low memory (pages 0 to 255 decimal, or 0 to 377 octal). `fw.msp800.msp8` is allocated pages 337 (octal) through 350 (octal).
- In Collection 1, phase 1, the config deck is established, and the true extents of available memory are determined.
- `move_non_perm_wired_segs` is invoked to move non-retained, but wired segments to the end of physical memory. Due to the bug described herein, `fw.msp800.msp8` (segment 446) is not moved, but retains its continuous allocation starting in memory frame 337 (octal). Its old pages remain in its page table. The memory, however, is marked as free (through the adjustment of `slt.free_core_size`.
- In Collection 1, phase 2, `tc_init$early` is invoked to initialize the traffic control segments, including `tc_data`, `idle_dsegs`, and `idle_pdses`. These are segments are initialized for the BCE environment and are later rezied in phase 3.
- In Collection 1, phase 3, `tc_init` is invoked to re-initialize and reallocate `tc_data`, `idle_dsegs`, and `idle_pdses` for Multics operation. The old memory for these segments is discarded and made available to page control. However, `idle_pdses` page 0 is allocated memory frame 337 (octal). Now, this frame address appears in the page tables of two segments: `fw.msp800.msp8` (segment 446), and `idle_pdses` (segment 60).
- `init_sst$normal` is called to initialize the System Segment Table (SST), which contains the Active Segment Table Entries (ASTEs), each followed by the page table for that segment. All ASTEs are marked as free, and all PTWs are marked as invalid. The Core Map Entries (CMEs), describing the status of the real memory frames, are marked as free (unused), though they are not threaded into the list of free CMEs yet.
- `make_segs_paged` makes pageable segments paged, as described earlier. The old pages of `fw.msp800.msp8`, segment 446, are put on the free CME list. This includes page 337 (octal). Then, this page is assigned to another initialization segment, `free_area_1`, segment 436. This segment is made paged to permit it to be deleted as a temporary segment by `delete_segs$temp`. Note that now, page 337 is owned to `idle_pdses`, segment 60, and `free_area_1`, segment 436.
- `collect_free_core` is called to discard no-longer-needed segments, as described earlier, and zeros out the CME of page 0 of `idle_pdses`, or pa 337, except for the `cme.abs_w` bit.
- `delete_segs$temp` is called to delete temporary segments. As described earlier, page control (when invoked from `pc$truncate`) crashes the system because while processing `free_area_1`'s page 0, it find `aste.records` 0 for this segment 0, decrements it, and believes that this value has gone negative. Again, `aste.records` is 0 because `cme.astep` is 0 (and points to a bogus ASTE in the SST header) since the CME was zeroed previously.

5 An Evil Config Deck

An example of a config deck that caused Multics to crash as described in this MCR follows. Note that there are many such config decks, and we haven't really been able to isolate exactly what config decks will cause the issue. Since the bug is related to memory allocation, and various config decks result in differently-sized data structures, it is difficult to pin down any specific config deck that will cause the issue to manifest itself. Note: it is quite possible that even "good" config decks provoke a similar problem to that described in this MCR, and possible corruption might occur in different places – even if the system doesn't crash in early initialization.

```
clok -delta 8. -zone pst
cpu -tag a -port 7 -state on -type dps8 -model 70. -cache 8.
mem -port a -size 4096. -state on
mem -port b -size 4096. -state on
mem -port c -size 4096. -state on
mem -port d -size 4096. -state on
iom -tag a -port 0 -model imu -state on
prph -device opca -iom a -chn 36 -model 6001. -ll 256. -state on
prph -subsys tapa -iom a -chn 12 -nchan 1 -model 8200. -number 16.
prph -subsys dska -iom a -chn 13 -nchan 1 -model 3381. -number 4.
prph -subsys dskb -iom a -chn 14 -nchan 1 -model 3381. -number 4.
prph -subsys dskc -iom a -chn 30 -nchan 1 -model 3381. -number 4.
prph -subsys dskd -iom a -chn 31 -nchan 1 -model 3381. -number 4.
prph -subsys dske -iom a -chn 32 -nchan 1 -model 3381. -number 4.
prph -subsys dskf -iom a -chn 33 -nchan 1 -model 3381. -number 4.
prph -subsys dskg -iom a -chn 34 -nchan 1 -model 3381. -number 4.
prph -subsys dskh -iom a -chn 35 -nchan 1 -model 3381. -number 4.
chnl -subsys dska -iom a -chn 37 -nchan 1
root -subsys dska -drive 00a -subsys dska -drive 00b -subsys dska -drive 00c
tcd -apt 1000. -itt 2000.
intk warm 0.
sst -4k 400. -16k 150. -64k 50. -256k 20.
part hc dska 00a
part dump dska 00a
parm loud
```

Note that the config deck is not intended to actually boot a running Multics service. It is just enough to provoke the bug. With the new hardcore and this config deck, the system doesn't fully come up, but it gets past the crash. Adding more config deck cards will allow the system to boot.

6 Testing

Once the fix was identified and made, a new version of hardcore was built. Multics was booted with the "Evil Config Deck" and no crash occurred. The hardcore tape was also tested on the GHM_Test Multics system, and the `ci-kit` was run with this hardcore as well. No issues were found. Further testing will be done, and the GHM system will be updated with this hardcore soon.

The following output shows a test system being successfully booted with the Evil Config Deck.

```
eswenson@eswenson-ThinkPad-W520:~/Multics/systems/ECD$ ./dps8 boot.ini
```

```
DPS8/M simulator X2.0.1-rc2-eswenson/test-20210911b+410
Options: LOCKLESS
  Commit: 799446d96387220ce29724b084d0cc6b7e1e3ad0
System state restored

Please register your system at https://ringzero.wikidot.com/wiki:register
or create the file 'serial.txt' containing the line 'sn: 0'.

FNP telnet server port set to 6180

TAPE: unit is read only
CPU a thread created
[FNP emulation: listening to 0.0.0.0 6180]
CONSOLE: ALERT
bootload_0: Booting system test19 generated 09/21/21 1026.1 pdt Tue.
1726.3 announce_chwm: slt.free_core_start = 420000
1726.3 announce_chwm: slt.free_core_size = 252000
1726.3 announce_chwm: 428. pages used of 512. in wired environment.
1726.3 announce_chwm: 706. words used of 1024. in int_unpaged_page_tables.
find_rpv_subsystem: Enter RPV data: M-> [auto-input] rpv a11 ipc 3381 0a

1726.3 load_mst: 947. out of 1048. pages used in disk mst area.

bce (early) 1726.3: M-> bce

System was last shutdown at:
Tuesday, September 21, 2021 6:29:47 pst
Current system time is: Tuesday, September 21, 2021 9:26:30 pst.
Is this correct? M-> y

1726.5 initializer: 2009 scas_init state 1 phase 2.
1726.5 initializer: 2010 tc_init$early state 1 phase 2.
1726.5 initializer: 2011 init_sst$early state 1 phase 2.
1726.5 initializer: 2012 disabling slt allocation state 1 phase 2.
1726.5 initializer: 2013 initialize_faults$interrupt_init state 1 phase 2.
1726.5 initializer: 2016 load_disk_mpcs state 1 phase 2.
1726.5 initializer: 2017 init_pvt state 1 phase 2.
1726.5 initializer: 2018 read_disk$init state 1 phase 2.
1726.5 initializer: 2019 init_root_vols state 1 phase 2.
1726.5 initializer: 2020 establish_temp_segs state 1 phase 2.
1726.5 initializer: 2021 find_file_partition state 1 phase 2.
1726.5 initializer: 2025 load_mst$init_commands state 1 phase 2.
1726.5 initializer: 2028 scs_and_clock_init$date_time state 1 phase 2.
0926.5 initializer: 2029 io_config_init state 1 phase 2.
0926.5 initializer: 2030 ioi_init state 1 phase 2.
0926.5 initializer: 2031 init_toehold$save_safe_config_deck state 1 phase 2.
0926.5 initializer: 2032 bce_get_to_command_level state 1 phase 2.
bce (boot) 0926.5: M-> boot

0926.5 initializer: 2033 establish_config_deck state 1 phase 2.
1726.5 initializer: 3009 scas_init state 1 phase 3.
1726.5 initializer: 3010 tc_init state 1 phase 3.
```

```
1726.5 initializer: 3011 init_sst$normal state 1 phase 3.
1726.6 initializer: 3012 disabling slt allocation state 1 phase 3.
1726.6 initializer: 3013 initialize_faults$interrupt_init state 1 phase 3.
1726.6 initializer: 3017 init_pvt state 1 phase 3.
1726.6 initializer: 3018 read_disk$init state 1 phase 3.
1726.6 initializer: 3019 init_root_vols state 1 phase 3.
1726.6 initializer: 3027 load_mst$make_permanent state 1 phase 3.
1726.6 initializer: 3028 scs_and_clock_init$date_time state 1 phase 3.
0926.6 initializer: 3029 io_config_init state 1 phase 3.
0926.6 initializer: 3030 ioi_init state 1 phase 3.
0926.6 initializer: 3031 init_toehold$save_safe_config_deck state 1 phase 3.
0926.6 initializer: 3033 establish_config_deck state 1 phase 3.
0926.6 initializer: 3034 init_partitions state 1 phase 3.
0926.6 initializer: 3035 make_segs_paged state 1 phase 3.
0926.7 initializer: 3036 collect_free_core state 1 phase 3.
0926.7 initializer: 3037 delete_segs$temp 1 state 1 phase 3.
0926.7 initializer: 3038 disk_reader$init state 1 phase 3.
0926.7 initializer: 3039 segment_loader 2.0 state 1 phase 3.
0926.7 initializer: 3040 pre_link_hc 2.0 state 1 phase 3.
0926.7 initializer: 3041 initialize_faults$fault_init_two state 2 phase 3.
0926.7 initializer: 3042 getuid$init state 2 phase 3.
0926.7 initializer: 3043 init_vtoc_man state 2 phase 3.
0926.7 initializer: 3044 dbm_man$init state 2 phase 3.
0926.7 initializer: 3045 init_scavenger_data state 2 phase 3.
0926.7 initializer: 3046 init_dm_journal_seg state 2 phase 3.
0926.7 initializer: 3047 init_sys_var state 2 phase 3.
0926.7 initializer: 3048 dir_lock_init state 2 phase 3.
0926.7 initializer: 3049 ioi_page_table$init state 2 phase 3.
0926.7 initializer: 3050 fnp_init state 2 phase 3.
0926.7 fnp_init: Warning: no FNP's configured.
0926.7 initializer: 3051 tc_init$part_2 state 2 phase 3.
0926.7 initializer: 3052 init_syserr_log state 2 phase 3.
0926.7 initializer: 3053 init_str_seg state 2 phase 3.
0926.7 initializer: 3054 init_sst_name_seg state 2 phase 3.
0926.7 initializer: 3055 init_hardcore_gates state 2 phase 3.
0926.7 initializer: 3056 accept_rpv state 2 phase 3.
0926.7 initializer: 3057 init_lvt state 2 phase 3.
0926.7 initializer: 3058 init_root_dir state 2 phase 3.
0926.7 initializer: 3059 kst_util$garbage_collect state 2 phase 3.
0926.7 initializer: 3060 init_branches state 2 phase 3.
0926.7 initializer: 3061 syserr_seg_manager$initialize_log_names state 2 phase 3.
0926.7 initializer: 3062 init_stack_0 state 2 phase 3.
0926.8 initializer: 3063 delete_segs$temp 2.0 state 2 phase 3.
0926.8 initializer: 3064 load_system state 3 phase 3.
0926.8 initializer: 3065 disk_reader$final state 3 phase 3.
0926.8 initializer: 3066 tc_init$start_other_cpus state 3 phase 3.
0926.8 initializer: 3067 init_proc state 4 phase 3.
0926.8 rcp_init: tapa_00 deleted.
CONSOLE: LOCK
disk_table_: New disk_table created
Multics test19 - 09/21/21 0926.8 pst Tue
Command:
```

Note: this config deck isn't really complete and wont actually result in a running system (e.g. no FNP), but we got well past the fault in `delete_segs$temp` and made it all the way to ring-1 Initializer command level.

7 Debugging MST with ECD Fix

For those who want to prove to themselves that the new hardcore fixes the issue, an MST can be found here: https://s3.amazonaws.com/eswenson-multics/public/ecd_fix.tap

This is otherwise a normal MR12.7 hardcore and can be run with any prior release of Multics for testing.

8 Documentation

There are no documentation changes for this change.

9 Version History

2021-09-29	1.0	Initial version of the MCR.
2021-10-04	1.1	Updated with Gary's comments and suggestions.
2021-10-05	1.2	Updated with Gary's final review comments.