Subject: MCR10129, analyze_multics (v. 2.4) Improvements

Author: Gary Dixon

Date: January 03, 2023

Multics Tickets:

#297: New/enhanced requests for analyze_multics: cme, slte, page_control_check, aste

#298: copy_deadproc and copy_liveproc incorrectly capture UID for segments in the saved pdir

#293: unpaged_page_tables.incl.pl1 has an invalid declaration of the upt header.

#296: numeric_to_ascii_base_ generates 8-digit octal integer strings that cv_oct_ cannot convert

Introduction

The recent team effort investigating a Multics bootload problem described by:

> **MCR10093: *Fix move_non_perm_wired_segs.pl1***
> (https://s3.amazonaws.com/eswenson-multics/public/mcrs/MCR10093.pdf)

identified several features that would be useful in the analyze_multics (azm) subsystem. These features can be summarized as follows.

analyze_multics (azm) should include new requests to:

- display contents of a core_map entry (CME)
- display contents of a segment loader table entry (SLTE)
- check contents of SST and core_map against the set of rules used by page control and its supporting routines. These rules stem from the relationship between:
  - each active segment table entry (ASTE) and its associated Page Table; and
  - the core map entries (CMEs) describing content of each memory frame (page of real memory in hardware).

ASTEs or CMEs that do not adhere to these relationship rules are often symptoms of an underlying hardcore supervisor problem. But much checking is required to identify such broken-relationship symptoms.

Fortunately, a subroutine already exists within Multics page control which performs these checks: pc_check_tables_.pl1. This subroutine is designed to operate in two modes:

- If Multics crashes, the Emergency Shutdown mechanism attempts to shut down the Multics Storage System to avoid data loss from in-progress page control operations that were interrupted by the crash. If segments or directories are found in an inconsistent state, their ASTE, page table or related CMEs may be marked as “damaged” and their inconsistent pages may be emptied (set to all-zeros) to remove the inconsistency.
- If invoked from the user ring, the subroutine tries to find and report any inconsistencies without changing the page control data.

In practice, this passive user ring mode of pc_check_tables_ is impractical to use on a running system because page control is constantly changing ASTEs and their Page Tables and referenced CMEs.

However, during a Multics crash, system memory is saved as a static crash image. This crash image could be examined by the pc_check_tables_ in passive mode to quickly locate and report meaningful inconsistencies between items in the SST (System Segment Table) and core_map; these items include each ASTE with its associated Page Table, and each CME.

How Page Control and the Storage System Work

To understand the meaning of pc_check_tables_ reports, the person analyzing the dump image must know how Multics hardware and software manage the relationship between segments stored on disk and data presented to a process workspace by the Multics virtual memory mechanisms.

Readers who already understand these mechanisms should skip forward to the ***Existing subroutine: pc_check_tables_*** section.

What is a Virtual Segment, an SDW, and a Page Table with PTWs?

Multics hardware presents a *virtual memory workspace* to each user process. The workspace is divided into 512 *virtual segments*: individual data regions. Each segment holds a different kind of data (executing program instructions, stack history of called subroutines and their automatic variables, input or output data, library programs, gates into inner-ring supervisor subsystems, those inner-ring programs, etc.). Each segment has its own access control settings which grant the process specific access rights to data in the region.

A virtual segment is defined by a *segment descriptor word (SDW)* describing: process access rights to the data region (read, execute, write); a location (sdw.add) within physical hardware memory where the virtual segment resides in memory; and a data length (sdw.bound) of the virtual segment specified in 16-word blocks of 36-bits/word. Most segments make use of a hardware-supported paging mechanism that divides the segment into 36864-bit pages (`36 bits/word * 1024 words/page`). If sdw.unpaged = "0"b, the sdw.add element gives the real memory location of a *page table*. This page table is an array of *page table words (PTWs)*; the sdw.bound element tells the hardware how many PTWs are included in the virtual segment's page table. Each ptw.add element gives the real memory location that holds a page of the virtual segment's data. The order of PTWs in the page table array defines the sequence of pages included in the virtual segment. The ptw.df and ptw.df_no elements tell the paging hardware whether contents of the page are valid: if ptw.df = "0"b, then page contents has not been loaded into memory and a *page fault* occurs if any program running in the process attempts to access that page.

The process' array of 512 SDWs is held in the *Descriptor Segment* of the process (its dseg segment).

What are Disk Segments, SSFs, MSFs and Records?

The Multics Storage System defines a *disk segment* as a collection of related data stored as a stream of data bits on disk. This data stream is recorded as: a sequence of fixed-length *records*, each holding 36864 bits (`36 bits/word * 1024 words/record`) of data; and a total segment length in bits (its *bit count*). Each such disk segment is stored in one of two ways:

- As a *single-segment file (SSF)* consisting of from 1 to 256 records plus a bit_count.
- As a *multi-segment file (MSF)* consisting of a sequence of SSFs named by numbers from 0 to N (decimal integers) collected in an MSF directory whose bit_count is set to N+1. The overall bit_count for an MSF is the sum of bit counts on the SSFs comprising the MSF.

For the purposes of describing analyze_multics changes, the remaining sections of this MCR will refer only to "disk segments" as the primary method used to store segment contents on an external disk.

A disk segment is identified to the system in by information stored in its *directory entry*: an identifying bit string that is unique on the Multics system (entry.uid); by one or more entry names attached to its directory entry (entry.nnames, entry.name_frp and entry.name_brp); and by a pathname of superior directories up to the root directory on that Multics system (the directory whose pathname is: ">") which locate the directory containing its directory entry.

Additional attributes of the disk segment (stored in its directory entry) include: an identifier for the disk volume containing segment records (entry.pvid) and index of a Volume Table of Contents Entry (VTOCE) on that physical volume (entry.vtocx); a physical segment bit count (entry.bc); access control attributes (read/execute/write modes for particular groups or projects; ring brackets and access class), etc.

The *VTOCE* record includes: current segment length (vtoce.csl) in records; count of disk records in-use by the segment (vtoce.records); array of disk addresses holding records of the segment (vtoce.fm – for file map) stored in sequential order. Note that vtoce.csl may be larger than vtoce.records because some records in mid-file may never have been written to disk (may be empty). Such missing disk records have a *null disk address* in their corresponding vtoce.fm array element.

How is a Disk Segment Connected to a Virtual Segment?

A program running in a user process must ask the Multics hardcore supervisor to connect a particular disk segment to a virtual segment (memory region) directly accessible by the program. This supervisor request is called "initiating a segment" or "making a segment known to the process". The request specifies minimum access rights needed by the calling program, and pathname of the directory entry describing the disk segment.

The supervisor must perform the following make-known services to enable the user process to access a specific disk segment (target segment) as one of the 512 virtual segments in its process workspace.

A. Make-Known Check: Does the requesting user process already have within its dseg an SDW that references the target segment, perhaps by a different pathname? The segment's unique identifier (entry.uid) can be searched for in the hash table of the process' known segment table (kst segment). If found, the kst_entry identifies an existing SDW in use by the requesting process to access the target segment. The supervisor returns a pointer referencing the segment number of that existing SDW (existing virtual segment known to the process).

   If the target segment is not already known to the process, an unused SDW in the process' SDW array must be chosen to access the disk segment. If no free SDW is available in its kst segment, the user process must first terminate some existing virtual segment to free an SDW; then try again to make the segment known.

B. Access Rights Check: Do the access authorizations of the Person_ID.Project_ID combination which created the user process grant any rights to read, execute, and/or write the target segment? The supervisor compares ownership of the process, current ring of execution, and AIM access authorization of the process with access control attributes in the segment's directory entry (ACL, ring brackets, AIM access class) to determine what rights may be granted. If the requested rights are granted, initiation of the segment can proceed.

C. Activation Check: Does the target segment already appear in the system's list of active segments? The supervisor searches its active segments list for the segment's unique identifier (entry.uid). If a match is found, the supervisor can use the page table assigned to that active segment.

   If not already active, the supervisor must locate an unused page table large enough to hold the existing records of the target segment (vtoce.csl).

   The system limits the number of segments in its *active segments working set* by pre-allocating storage for a fixed count of page tables in four popular lengths: 4, 16, 64 and 256 pages long. If a page table of the appropriate size is not found on the unused list, then the supervisor must locate the least-recently-used segment of that page table size and deactivate that segment to provide a page table for the target segment. The processes which were using that little-referenced active segment will each take a segment fault if they reference it in the future; but the segment will remain known to each of those user processes.

D. Activating the Segment: Supply data for the ASTE and page table describing the disk segment to the system. (See the "What are Active Segments and ASTEs?" section below.)

E. Making the Segment Known to the Process: Fill-in the chosen SDW for the process asking for the target segment to be "made known". (See "Adding the Virtual Segment to the Process' Workspace" section below.)

What are Active Segments and ASTEs?

With checks A, B and C satisfied, the supervisor can add the target segment to its list of *active segments*. This list is kept in the *System Segment Table (SST)* that provides storage for the pre-allocated page tables. Preceding each page table is an *Active Segment Table Entry (ASTE)* that holds information about the disk segment being accessed by that page table. This information includes segment characteristics: unique id (aste.uid); segment record count (aste.records) and current length (aste.csl); segment type (aste.dirsw); segment location on disk (aste.pvtx, aste.vtocx); etc. The group of ASTEs form the system's *list of active segments*: its active segment working set.

When activating the target segment, the supervisor copies attributes describing characteristics from the segment's directory entry and *Volume Table of Contents Entry (VTOCE)* to the ASTE elements. The supervisor then loads the sequence of disk addresses from the VTOCE address list (vtoce.fm) into corresponding ptw.add fields of the selected page table. Each loaded ptw.df is set to "0"b indicating the record has not been loaded into a memory frame.

As an active segment is read or written, the supervisor updates information in the ASTE to describe any changes in segment length, segment date_used and date_modified, disk addresses assigned to hold contents of new records, etc. The ASTE elements and associated PTWs now hold the most accurate description of these disk segment attributes; ASTE information therefore supersedes equivalent data elements stored in the segment's directory entry and VTOCE. The directory entry and VTOCE for the disk segment will be updated only when the active segment is deactivated (removed from the system's list of active segments).

Adding the Virtual Segment to the Process' Workspace

With steps A through D completed, the system has activated the target segment (assigned an ASTE and page table to record its current characteristics, and its list of disk addresses). But this active segment is still not associated with a virtual segment known to the requesting process.

The rights granted to the requesting process for access to the target segment (determined by step B) must now be recorded in the process' SDW chosen to describe its new virtual segment (determined by step A). Access rights include: sdw.read, .write, .execute and sdw.r1, .r2, and .r3 ring bracket elements. This tells the hardware how much access the user process has to contents of the virtual segment. Location of the active segment's page table is stored in sdw.add, with sdw.unpaged = "0"b. The virtual segment's current length (in 16-word blocks) is calculated from aste.csl and stored as sdw.bound. Finally, sdw.df is set to "1"b, enabling the SDW for use by the process.

This mechanism allows several processes to access the same active segment in differing ways. For one process, the segment may contain instructions that can be read and executed. For another process, the segment may contain data that can be rewritten by a compiler generating instructions for the executable program. A third process may read instructions in the segment as data items using a debugger. All three processes use the same page table to access pages of the active segment. But each user process gains only specific access rights to the active segment through its virtual segment's SDW. The SDW's *segment number* (index of the SDW within the process dseg's array of SDWs) differs in each process.

As one or more processes operate on an active segment, the segment may grow or shrink in length. Those length changes must be updated in all SDWs sharing that active segment. To enable such updates, the supervisor attaches an *ASTE trailer record (str)* to the ASTE describing the active segment. Each trailer record gives: the ASTE offset of a sharing process' dseg segment (str.dstep); and segment number of that process' SDW which references the active segment (str.segno). This allows the supervisor to update the SDW in the workspace of each process referencing the active segment to reflect any changes in length as well as changes in other segment attributes: changes in access rights granted by ACLs; activation status of the segment; deletion of the segment; etc.

What are CMEs?

When a disk segment is made known to a process, some of its disk records may have been loaded into real memory frames by other processes which know about this disk segment.

But when the disk segment is activated by the first such make-known request, none of its pages are loaded. The ptw.add values in the active segment's page table contain disk addresses; and all ptw.df bits are set to "0"b (page fault occurs when referenced). None of these PTWs have a memory frame (location in real memory) assigned into which the disk record could be loaded. The paging system defers loading of each page of the active segment until that page is first referenced by one of the processes sharing its page table. Such reference causes a page fault to interrupt program execution until the referenced disk record is loaded into a real memory frame by the page_fault handler.

The page_fault handler walks its list of real memory frames (using the famous "clock" algorithm) to decide what actions are needed to resolve the page fault, and to track those actions over a period of time. This list of memory frames is the array of *core map entries (CMEs)* stored in the core_map segment. The array contains a CME describing contents of each configured memory frame (no matter whether the containing SCU is online or offline). Two main groups of CMEs exist:

- CMEs describing memory frames that are managed by page_control are threaded into a single circular list starting at sst.usedp, with thread offsets stored in the cme.fp and cme.bp elements.
    - These CMEs are vacant if cme.astep, cme.ptwp = "000000"b, and in-use if these elements are non-zero.
- CMEs describing wired supervisor pages whose content is loaded during system bootload operation. Each has cme.fp, cme.bp = "000000"b and cme.abs_wired = "1"b.

The page_fault handler performs the following steps to resolve the page fault that occurs when a process references a PTW with ptw.df = "0"b.

A. Walk the list of CMEs looking for one that is vacant. As the list is traversed, check whether the PTW associated with each CME has been used (ptw.phu) during the prior pass around the circular list of CMEs. If not referenced, schedule I/O to evict that page from memory to create a vacant memory frame; if referenced, turn off the ptw.phu bit so the next pass around the list can check for another reference. [Note: the paging hardware turns on the ptw.phu bit each time the page is referenced by a process for any reason.]

B. When a vacant CME is found, assign the memory frame described by that CME to hold the active segment's disk record (set cme.ptwp to point to the faulting PTW; set cme.astep to point to the ASTE which precedes the page table containing the faulting PTW; copy the disk address from ptw.add into cme.add; and store the absolute address of the memory frame in ptw.add). ptw.df remains set to "0"b (faulted).

C. Schedule a page-read operation:

   a. Look in the ASTE (associated with the page table for the faulting PTW) to get ID of the physical disk which holds records of the disk segment (aste.pvtx).

   b. Use that pvtx plus the cme.add disk address to schedule a read operation to bring that disk record into the assigned memory frame (ptw.add).

   c. Set cme.io = "0"b indicating that reading (input of the page) is in progress.

   d. Set cme.notify_requested = "1"b to indicate that a process needs to be notified when the I/O operation completes.

D. Call pxss$page_wait to change state of the process taking the page fault. This call tells the scheduler that the process must wait until the disk record has been read into the memory frame. It specifies a wait identifier which is the offset of the PTW within the SST segment; this identifier is stored in process' *Active Process Table Entry (APTE)* as apte.wait_event. The scheduler removes the process taking the page fault from its CPU and reassigns another process to run on the CPU.

When the page-read operation completes, the interrupt handler recognizes input completion into a memory frame, references the CME describing that memory frame, updates the PTW pointed to by cme.ptwp to set ptw.df = "1"b (not faulted), and calls pxss$page_notify with offset of the PTW as the wait_event. pxss finds all processes waiting on that wait_event and changes their state to ready-to-run.

When the pxss scheduler restarts the page-faulting process, the instruction causing the page fault is restarted (possibly in mid-instruction execution) and re-references the page. The page's PTW now finds the page in-memory so no page fault recurs.

Existing subroutine: pc_check_tables_

The threaded list of CMEs are the central data objects used by page_control in managing operations that tie each disk record to a page in an active segment shared by one or more user processes. CME data elements link contents of the memory frame described by that CME to:

- a record in an disk segment (cme.add);
- overall attributes of that active segment as described by an ASTE (cme.astep); and
- one specific PTW in the ASTE's associated page table (cme.ptwp) whose page table index equals the disk record's index within the segment's VTOCE (the vtoce.fm index).

This method of accessing a disk segment in the Multics Storage System as a virtual segment (memory region) in a process workspace imposes several rules on page control.

1. A particular disk segment can be activated only once to tie its records to the pages of an active segment.
    a. All user processes that want to reference records in this disk segment must do so by sharing that active segment.
2. Therefore:
    a. Each disk segment's attributes will be recorded in only one ASTE at a time.
    b. Each disk record will be loaded into only one memory frame at a time, and can therefore appear in only one active segment at a time, and only in the specific PTW of that active segment corresponding to the record's offset within the segment (its vtoce.fm array index).

What Relationships Does pc_check_tables_ Verify?

The pc_check_tables_ subroutine checks contents of the ASTEs, page tables, and CMEs and reports any variance from the above rules.

- Each non-vacant CME should have a cme.ptwp value which gives the offset of the one-and-only PTW having a ptw.add giving the absolute address of the memory frame described by that CME.
- Each non-vacant CME should have a cme.astep value which gives the offset of the ASTE preceding the page table containing the PTW referenced by cme.ptwp.
- Each in-use ASTE should have the correct count of disk records loaded into active segment's pages (aste.np), as indicated by count of PTWs in its associated page table having ptw.df = "1"b.

pc_check_tables_ checks all non-vacant, page_control CMEs and all PTWs in page tables held in the SST and reports any items that violate these rules.

- If a CME has a non-zero cme.astep with a zero cme.ptwp value, it is reported.
- If a CME has a non-zero cme.ptwp with a zero cme.astep value, it is reported.
- If a PTW has a ptw.add value referencing a memory frame whose CME does not have cme.ptwp pointing to that PTW, it is reported.
- If an ASTE has aste.np ^= count(PTWs in its associated page table with ptw.df = "1"b), it is reported.
- If an ASTE has aste.csl ^= count(PTWs with valid disk address or memory frame abs-addr), it is reported.

What Data Is Reported by pc_check_tables_?

Each inconsistency reported by pc_check_tables_ contains only minimal information describing the inconsistency. For example, a report for one type of inconsistency begins with a line naming the inconsistency, followed by two lines giving pointers to the CME and PTW entries which are inconsistent.

```
_____________________________________
CME ptwp ^= ptw address.
 --> PTW at  102|1275
 --> CME at   47|144
_____________________________________
```

The pc_check_tables_ subroutine identifies inconsistent items by providing a pointer to each PTW or CME or ASTE. It cannot use system mechanisms to lookup a segment name associated with each entry. This meager level of information obscures the meaning of these messages.

Why are these messages so skimpy?

During Emergency Shutdown, the runtime execution environment is highly restricted. The pc_check_tables_ subroutine cannot do any look-up of segment-related information, or interpretive displays of item contents. The OS machinery to perform such operations may be broken by events causing the crash. More damage to storage system contents could occur if ESD attempted such interpretation.

How Might an azm Request Enhance these Reports?

However, by recreating these same messages while examining a crash image, azm provides a rich runtime environment that can enhance each report with meaningful details about the items involved and details of the relationship inconsistency. The enhancements could identify segments involved, and could justify some inconsistencies as being expected and therefore not symptomatic of an underlying bug in the operating system or a hardware failure.

For example, the message above could be enhanced as:

```
_____________________________________
CME ptwp ^= ptw address.
 --> PTW at SST|1275
     ASTE for wi_linkage (Seg 406  in Proc 0), at SST|1260.
 --> CME at core_map|144    breakpoint_page
     CME for breakpoint_page (Seg 14|0  in Proc 0), at core_map|144
           EXPECTED INCONSISTENCY:  the breakpoint_page page ends many hardcore segs.
            cme.ptwp intentionally set to "000000"b3, matching none of these PTWs.
_____________________________________
```

Identifying the CME as describing the breakpoint_page segment provides a valuable clue about the inconsistency. breakpoint_page is a special segment used by the bce_probe command to track breakpoints created in hardcore supervisor segments.

The regular probe command tracks breakpoints created by a given user in any program by storing data about the breakpoint in the user's PERSON_ID.probe segment (kept in the user's home directory). When a breakpoint is reached, the probe command looks in this shared .probe segment to find details about the breakpoint, actions to be taken when it has been reached, etc.

However, the bce runtime environment does not support demand paging. So a PERSON_ID.probe cannot be located in a separate segment that is brought into memory when a supervisor breakpoint is reached. Instead, this shared breakpoint data is stored in a single page which gets appended to the end of each wired supervisor program segment in which breakpoints might be created. The bootloader extends the page table of each such supervisor segment by adding a PTW pointing to the shared breakpoint_page. When a breakpoint is reached, bce_probe can then look in the final page of the breaking segment to access details of that breakpoint.

When the bootloader has appended the shared breakpoint_page to many supervisor segments, each has a page table pointing to the same memory frame. Since the CME describing a memory frame does not contain an array of cme.ptwp values pointing to each of these PTWs, the bootloader cannot adhere to pc_check_tables_ rule that each CME should have a cme.ptwp giving the offset (within the SST) of the one-and-only PTW which references that memory frame. The bootloader intentionally violates the rule by setting cme.ptwp = "000000"b3: since the CME cannot point to all the PTWs referencing the breakpoint_page memory frame, it instead will point to none of those PTWs. This is a safe practice for wired supervisor segments, and thereby is a planned exception to the pc_check_tables_ rule. That explains why the report above can be labeled as an *expected inconsistency*.

Adding a request to display the Segment Loader Table entry (SLTE) for segment 406 would further justify the expected nature of this inconsistency. An **slte** request could show that segment 406 has the "init_seg" attribute: it is a segment loaded to assist with initializing (or bootloading) the system. And segment 406 has the SLTE "breakpointable" attribute which instructs the bootloader to append a PTW for the breakpoint_page to the end of the page table for that segment.

```
azm:  slte 406 -header

                        Supervisor Segments      Initializing Segments
                          First:    0              First:  400
                           Last:  115               Last:  465

wi_linkage             406  (0, 0, 0) read execute write wired
  wired_init_linkage       init_seg breakpointable
                           wired length: 1024 words; paged length: 2 pages;
                           max length: 1 pages;
```

Similar reports not referring to an expected inconsistency can still be enhanced with added information and even with suggested diagnostic requests. For example, the following report shows the level of information that could be included to investigate the problem corrected by MCR10093.

```
_____________________________________
CME ptwp ^= ptw address.
 --> PTW at SST|16370
     ASTE for free_area_1 (Seg 436  in Proc 0), at SST|16354.
 --> CME at core_map|1604

2 PTWs reference memory frame described by core_map|1604:

  CME_OFS  FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT -----  --- FLAGS ---
     1604    0      0       0      0    W   Seg  60|0  idle_pdses    abs_wired  scu(A)
     1604    0      0       0      0    W   Seg 436|0  free_area_1   abs_wired  scu(A)

     DIAGNOSTIC REQUEST(s): aste -ptw 16370 -pt; cme -ptw 16370; cme -search 1604;
_____________________________________
```

Such enhanced output can be provided by having a new azm **page_control_check** (**pcc**) request call the pc_check_tables_ subroutine; the pcc request can then add extra details to each output line as inconsistencies are reported. It can use outputs from a new **cme** request to describe content of inconsistent CMEs; and an enhanced **aste** request to get the name of a segment owning the page table containing a given PTW. Adding an **slte** request to display Segment Loader Table entries known to the system at the time of crash would be beneficial as well.

This MCR proposes to make such changes and additions to the analyze_multics subsystem.

Proposed Changes

The following list summarizes the nature and scope of the proposed changes. Most of the changes occur in bound_azm_ sources (the azm command and its requests), and in bound_amu_ (the analyze Multics utility subroutines that provide segment-style access to data in a crash dump image of memory – a dump fileset). Items 1 and 2 encompass the major changes and additions. Most of this MCR will focus on those changes. Other changes will be summarized in brief without a detailed explanation.

Modifying azm and its supporting utilities in any significant fashion poses many challenges in program design and implementation. azm was developed over many years by knowledge experts in particular areas of the operating system. Much of their knowledge is embodied in azm code without full explanations or even documentation of internal azm interfaces. With many programmers involved, azm code follows a variety of coding styles which can further confuse code enhancement efforts.

In implementing the proposed changes, I encountered many unreported bugs in azm code which had to be repaired to permit the new requests to function correctly (or even function at all). Some unreported bug fixing and coding style cleanup was done to help improve the overall subsystem. Only the most significant of these are touched upon in this change proposal. But all will be identified for review by the auditor.

The main improvements and bug fixes are summarized by the 4 Multics Tickets listed in the header at the beginning of this MCR. Please refer to those tickets for details of the bugs, or justification for the proposed enhancements. Multics Ticket #297 (https://multics-trac.swenson.org/ticket/297) suggests the enhancements in items 1.b and 2 below.

1. Extend capabilities of the following azm requests:
    a. req: absolute_address, absadr (in azm_requests_2_.pl1)
    b. req: aste (in azm_requests_1_.pl1)
2. Add new azm requests:
    a. req: core_map_entry, cme (in azm_requests_3_.pl1)
    b. req: page_control_check, pcc (in azm_requests_3_.pl1)
    c. req: slt_entry, slte (in azm_requests_3_.pl1)
3. Fix bugs in the following bound_azm_ requests and subroutines:
    a. req: display, d (in azm_requests_1_.pl1)
    b. req: display_absolute, da (in azm_requests_1_.pl1)
    c. azm_display_am_.pl1
    d. azm_display_fdump_events.pl1
    e. azm_dump_mem_.pl1
    f. azm_request_table_.alm
    g. azm_str_util_.pl1
    h. azm_verify_dump_ams_.pl1
    i. azm_why_.pl1
    j. copy_pdir_.pl1 Ticket #298 (https://multics-trac.swenson.org/ticket/298)

4. Add new azm-related subroutines and include file:
    a. azm_sst_util_.pl1
    b. azm_sst_util_ptws.incl.pl1
5. Fix bugs or add new features to the following bound_amu_ subroutines:
    a. amu_.alm
    b. amu_do_translation_.pl1
    c. amu_et_.alm
    d. amu_fdump_mpt_.pl1
    e. amu_get_name_.pl1
    f. amu_hardcore_info_.pl1
    g. amu_info_.pl1
    h. amu_parse_ptr_args_.pl1
    i. amu_print_.pl1
6. Fix bugs and/or add features in the following commands, subroutines and include files:
    a. analyze_multics, azm
    b. numeric_to_ascii_base_.pl1
        Ticket #296 (https://multics-trac.swenson.org/ticket/296)
    c. pc_check_tables_.pl1
7. Correct problems and/or add elements in the following azm-related include files:
    a. unpaged_page_tables.incl.pl1 and structure_library_5_.cds
        Ticket #293 (https://multics-trac.swenson.org/ticket/293)
    b. amu_hardcore_info.incl.pl1
    c. azm_info.incl.pl1
    d. slte.incl.pl1
8. Move copy_erf_seg_.pl1 (which references many bound_amu_ routines) from bound_meter_util_ into bound_azm_. The source segment itself is not changing.
9. Improve documentation for the following commands:
    a. copy_deadproc.info
    b. copy_liveproc.info
10. Add documentation for new azm requests, and improve documentation for existing requests:
    a. aste.info
    b. core_map_entry.info
    c. display.info
    d. display_absolute.info
    e. list_processes.info
    f. page_control_check.info
    g. sdw.info
    h. slt_entry.info

1.a) Improved request: absolute_address, absadr

Change the absolute_address (absadr) request to properly handle segments having a page table in either unpaged_page_tables and int_unpaged_page_tables segments.

Existing code works fine for the more prevalent segments which use one of the standard page tables provided in the SST segment. But many segments used during bootload or comprising critical parts of the hardcore supervisor are defined by page tables not residing in the SST. absadr needs to properly handle a virtual address referencing these segments.

Such segments are loaded from the Multics System Tape directly into wired memory pages by the Multics bootloader. They are not managed by page_control, and are not visible outside of the ring-0 supervisor. But they are important to system operation and are often factors in a dump analysis effort.

1.b) Improved request: aste

The existing aste request displays the Active Segment Table Entry (ASTE) describing an active segment, the page table associated with that ASTE, and any ASTE trailer records which identify processes that know about that segment (including the segment number they use to access that active segment). The request uses a robust set of methods for determining the name of the disk segment tied to the active segment. It can even identify segments loaded at bootload time which are defined by a page table stored in the unpaged_page_tables segment.

It would be useful if these methods for segment identification: were expanded to include segments defined by a page table in the int_unpaged_page_tables segment; and made callable from the new page_control_check (pcc) request to identify the disk segment associated with an inconsistent ASTE report from pcc.

For example, the following inconsistency reports need segment identification lines:

```
_____________________________________
Bad counter for ASTE.
 --> ASTE at 102|3040
np = 0, should be 4
_____________________________________
```

Segment ID might also be used to identify the active segment which is using an inconsistent PTW, such as in the following pcc report:

```
_____________________________________
CME ptwp ^= ptw address.
 --> PTW at  102|1275
 --> CME at   47|144
_____________________________________
```

This would require adding three new control arguments for selecting an ASTE, or choosing the output data to be displayed.

```
Control arguments (selecting an aste):
-offset ASTE_OFFSET,
-ofs ASTE_OFFSET
   octal offset within sst_seg of the ASTE selected for display.
   Other data structures usually refer to this offset as the XXX.astep
   element.
-ptw PTW_OFFSET
   octal offset within SST, unpaged_page_tables or
   int_unpaged_page_tables of a PTW (Page Table Word).  The segment
   associated with the Page Table containing that PTW is selected
   for display.

Control arguments (displayed data):
-brief_header, -bfhe
   displays only the segment reference name or path name, segment
   number, and ASTE location.
```

Code to identify an active segment from a pointer/offset to its ASTE or PTW will also be needed in the proposed new core_map_entry (cme) request. So this code will be consolidated in several new azm_sst_util_ subroutine entry points (see section "4.a-b" described below).

2.a) New request: core_map_entry, cme

A core map entry (CME) is the main structure used by page_control to tie together a disk record to an active segment page. Because it plays a pivotal role in many system operations, a person reviewing a system crash might need to select the CME to display in different ways:

- Display the CME that describes a given absolute address in real memory.
- Display the CMEs describing in-memory pages of an active segment, given:
    - segment name
    - segment number in the selected process
    - offset of an ASTE describing the active segment
    - offset of a PTW locating a page of the active segment
- Display the CME at a given index within the core_map array of CMEs; or at a particular offset within the core_map segment.

The information displayed for each CME should have a 1-line format to support display of CMEs in a fashion that demonstrates their relationship to one another. For example, CMEs describing pages of a single active segment are displayed in page_table order:

```
azm:  cme -seg 436

CMEs  for      ASTE for free_area_1 (Seg 436  in Proc 0), at SST|16354.

  CME_OFS     FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT -------------  --- FLAGS ---
     1604       0      0       0      0    W   Seg 436|0       free_area_1      abs_wired  scu(A)
   176144  176140   1604   16354  16371        Seg 436|2000    free_area_1      io=output  scu(D)
   176140  176134 176144   16354  16372        Seg 436|4000    free_area_1      io=output  scu(D)
   176134  176130 176140   16354  16373        Seg 436|6000    free_area_1      io=output  scu(D)
   176130  176124 176134   16354  16374        Seg 436|10000   free_area_1      io=output  scu(D)
   176124  176120 176130   16354  16375        Seg 436|12000   free_area_1      io=output  scu(D)
   176120  176114 176124   16354  16376        Seg 436|14000   free_area_1      io=output  scu(D)
   176114  176110 176120   16354  16377        Seg 436|16000   free_area_1      io=output  scu(D)
   176110  176104 176114   16354  16400        Seg 436|20000   free_area_1      io=output  scu(D)
                                               CA-MAX 21777
```

Or layout of unpaged segments in the first four memory frames. CMEs are display in absolute address order:

```
azm:  cme [index_set 0 [calc 1024*4-1] 200o -octal]

 ABS_ADDR     FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT -------------  --- FLAGS ---
        0       0      0       0      0    WU  Seg   4|0       fault_vector     abs_wired  scu(A)
      200       0      0       0      0    WU  Seg   4|200     fault_vector     abs_wired  scu(A)
      400       0      0       0      0    WU  Seg   4|400     fault_vector     abs_wired  scu(A)
      600       0      0       0      0    W   Vacant memory.                  abs_wired  scu(A)
     1000       0      0       0      0    W   Vacant memory.                  abs_wired  scu(A)
     1200       0      0       0      0    WU  Seg  11|0       iom_mailbox      abs_wired  scu(A)
     1400       0      0       0      0    WU  Seg  11|200     iom_mailbox      abs_wired  scu(A)
     1600       0      0       0      0    WU  Seg  11|400     iom_mailbox      abs_wired  scu(A)
     2000       0      0       0      0    WU  Seg  11|600     iom_mailbox      abs_wired  scu(A)
     2200       0      0       0      0    WU  Seg  11|1000    iom_mailbox      abs_wired  scu(A)
     2400       0      0       0      0    WU  Seg  11|1200    iom_mailbox      abs_wired  scu(A)
     2600       0      0       0      0    WU  Seg  11|1400    iom_mailbox      abs_wired  scu(A)
     3000       0      0       0      0    WU  Seg  11|1600    iom_mailbox      abs_wired  scu(A)
     3200       0      0       0      0    WU  Seg  11|2000    iom_mailbox      abs_wired  scu(A)
     3400       0      0       0      0    WU  Seg   3|0       dn355_mailbox    abs_wired  scu(A)
     3600       0      0       0      0    WU  Seg   3|200     dn355_mailbox    abs_wired  scu(A)
     4000       0      0       0      0    WU  Seg   3|400     dn355_mailbox    abs_wired  scu(A)
     4200       0      0       0      0    WU  Seg   3|600     dn355_mailbox    abs_wired  scu(A)
     4400       0      0       0      0    WU  Seg   3|1000    dn355_mailbox    abs_wired  scu(A)
     4600       0      0       0      0    WU  Seg   3|1200    dn355_mailbox    abs_wired  scu(A)
     5000       0      0       0      0    WU  Seg   3|1400    dn355_mailbox    abs_wired  scu(A)
     5200       0      0       0      0    WU  Seg   3|1600    dn355_mailbox    abs_wired  scu(A)
     5400       0      0       0      0    WU  Seg   3|2000    dn355_mailbox    abs_wired  scu(A)
     5600       0      0       0      0    WU  Seg   3|2200    dn355_mailbox    abs_wired  scu(A)
     6000       0      0       0      0    WU  Seg   3|2400    dn355_mailbox    abs_wired  scu(A)
     6200       0      0       0      0    WU  Seg   3|2600    dn355_mailbox    abs_wired  scu(A)
     6400       0      0       0      0    W   Vacant memory.                  abs_wired  scu(A)
     6600       0      0       0      0    W   Vacant memory.                  abs_wired  scu(A)
     7000       0      0       0      0    W   Vacant memory.                  abs_wired  scu(A)
     7200       0      0       0      0    W   Vacant memory.                  abs_wired  scu(A)
     7400       0      0       0      0    W   Vacant memory.                  abs_wired  scu(A)
     7600       0      0       0      0    W   Vacant memory.                  abs_wired  scu(A)
```

The most significant elements of the CME structure can be included in this 1-line format. The main omission is the disk address of the record that is (or will be) loaded into the memory frame. Disk address is not usually significant unless there is a problem with the disk hardware. In such cases, the disk address could be displayed using the aste request.

The following output shows proposed display for segment 327 by the cme request. This is followed by display of all CME elements for the first page of that segment using the display request's structure display mechanism. Most elements are included in the 1-line format. The final part of the output by the aste request shows the disk addresses for records (pages) of segment 327.

```
azm:  cme -seg 327

CMEs  for      ASTE for >sl1>ioi_ (Seg 327  in Proc 0), at SST|3660.

  CME_OFS     FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT -------------  --- FLAGS ---
   172604  172610 172600    3660   3674        Seg 327|0       >sl1>ioi_        io=output  scu(D)
   172610  173070 172604    3660   3675        Seg 327|2000    >sl1>ioi_        io=output  scu(D)
                                               CA-MAX 3777


azm:  display core_map|172604 -as cme
cme                           @ 47|172604
  fp = "172610"b3, bp = "172600"b3, devadd = "000010000100101011010 0"b, contr = "3"b3,
  ptwp = "003674"b3, astep = "003660"b3,  pin_counter = 0, synch_page_entryp = "000000"b3
  ON:  io
  OFF: synch_held, er, removing, abs_w, abs_usable, notify_requested, phm_hedge


azm:  aste 327 -pt

ASTE for >sl1>ioi_ (Seg 327  in Proc 0), at SST|3660.

PAGE      PTW       DEVADD    PD COPY   at SST|3674

   0  752760401145   20453                phu phm phu1
   1  753000400045   20454                phu1
   2  377012000001    null
   3  377012000001    null
```

The proposed user interface for the cme command is described in the following excerpt from the new cme.info segment.

```
azm:  help cme

   (58 lines follow; 294 lines in info)

2022-07-11  core_map_entry, cme

Syntax:  cme ABS_ADDRs
         cme -control_arg

Function:  Displays information from a core map entry (CME).  Each
describes how a given page of real memory is currently used.

Arguments:
ABS_ADDRs
   absolute octal address of a location in real memory.  One or more
   addresses may be given.  The request displays the CME for the page
   containing each of those real memory addresses, and tries to
   identify which segment contains that page.
```

```
Control arguments:
   One of the following control arguments may be given instead of the
   ABS_ADDRs argument described above.
-index CME_INDEX ...,
-ix CME_INDEX ...
   zero-based decimal index in the core map array of one or more CMEs
   to be displayed.  This is also the zero-based memory-frame number
   (page number) in real memory.  Index values range from 0 to 16383
   in a system with 4 SCUs configured with largest memory blocks.
-offset CME_OFFSET ...,
-ofs CME_OFFSET ...
   octal offset in the core_map segment of one or more CMEs to be
   displayed.  Offsets range from 10o through 200010o in a system with
   4 SCUs configured with largest memory blocks.  Given the size of a
   CME (4 words), offsets must end with a 0 or 4 octal digit.

-ptw PTW_OFFSET
   octal offset of a Page Table Word (PTW) within the System Segment
   Table (SST), unpaged_page_tables or int_unpaged_page_tables.  If
   the PTW is active (not faulted), the request displays the CME
   describing the memory frame at the ptw.add absolute address
   location.
-aste ASTE_OFFSET
   octal offset of an Active Segment Table entry (ASTE) within the
   System Segment Table (SST), or of a PTW within a page table in
   unpaged_page_tables or int_unpaged_page_tables.  The request
   displays the CME for each in-memory PTW containing a page of the
   segment.

-segment {SEGNO | SEGNAME},
-seg {SEGNO | SEGNAME}
   octal segment number or name of a segment known to the selected
   dump process and listed in an Active Segment Table entry (ASTE)
   within the System Segment Table (SST), or with a page table in
   the unpaged_page_tables or int_unpaged_page_tables segment.
   The request displays every CME for each PTW that is in-memory
   (not faulted).
-search CME_OFFSET,
-srh CME_OFFSET
   octal offset in the core_map segment of one or more CMEs to be
   displayed.  Searches all in-memory PTWs in page tables in the SST,
   unpaged_page_tables and int_unpaged_page_tables.  Any PTWs which
   reference the memory frame described by that CME are displayed as a
   CME reference.

List of core map entry fields:
   Each CME is displayed on a single line containing several fields.
   One of the first four entries below is given to identify the CME.
CME_IDX
   The CME is the Nth entry in the core_map array of CMEs, where N
   ranges from 0 to 16383.  That CME describes the Nth memory frame
   in real hardware memory.
```

```
CME_OFS
   The CME is located at octal offset CME_OFS within the core_map
   segment.  The core_map begins with an 8-word heading followed by an
   array of 4-word CMEs.  Thus, CME_OFS can range from 10o through
   200010o.
ABS_ADDR
   Given an octal absolute address in real hardware memory, the CME
   describing the memory frame covering that address is displayed.
PTW_OFS
   The CME describes the memory frame referenced by the PTW at octal
   offset PTW_OFS in the SST, unpaged_page_tables, or
   int_unpage_page_tables (segments holding page tables).

FWD, BACK
   Octal CME offset within the core_map of the next CME in a threaded
   list of CMEs, or the prior CME in such threaded list.  The main
   threaded list locates CMEs describing memory frames whose contents
   are managed by page_control.  Tracing such threads is simplified
   when displayed CMEs are identified by their CME_OFS value.
   Frames not managed by page_control usually contain segment pages
   permanently wired in real memory, or contiguous wired pages
   referenced by an unpaged SDW).  Such CMEs usually have 0 FWD and
   BACK offsets.

ASTEP
   Octal offset within the SST segment of an ASTE (Active Segment
   Table Entry) whose associated page table references the memory
   frame described by this CME.
PTWP
   Octal offset within the SST segment of the specific PTW (Page Table
   Word) that references the memory frame described by this CME.

DFWU
   Three switches describing the page currently stored in the memory
   frame.
     DF is displayed if the SDW used to locate the CME is inactive
        (has its sdw.df field set to cause a fault).
      W is displayed if the cme.abs_w bit is "1"b: the page content
        is abs-wired and its absolute address must not be changed.
      U is displayed if sdw.unpaged = "1"b: the segment is defined
        as a set of contiguous pages in real memory without using a
        page table.  An example is the fault_vector segment.

FRAME CONTENT
   A field summarizing content of the frame.  It usually provides an
   octal segment number|offset w/in that segment, followed by the
   reference name or pathname of that segment.  Some memory frames
   contain disk pages that are not defined by the storage system as a
   segment: for example, a physical volume bit map.  Other frames may
   be vacant (contain no data).  A frame may contain an active page
   whose content cannot be identified by data included in the dump
   image.
```

```
FLAGS
   Describes other elements of the cme data structure, if they have
   non-zero value.  These include:
     io=output
        Content of the memory frame is being written to disk.  If not
        present, content of disk might be queued to be read from disk,
        or no I/O operation may be queued.
     io_error
        The most recent input/output operation encountered an error.
     removing
        Memory frame is being removed by a reconfiguration operation.
     abs_wired
        Contents of memory frame must not be relocated to a different
        memory frame (different absolute address).

     abs_usable
        Memory frame eligible to hold an active page which must be
        abs_wired in memory (is not relocatable).
     notify_requested
        An I/O operation is pending.  When it completes, a waiting
        process needs to be notified of that completion event.
     phm_hedge
        pc$flush_core needs to write content of this memory frame to
        disk.

     scu(X)
        ID letter (A, B, C, or D) of the System Control Unit
        containing the real memory page described by this CME.
     pin_counter=N
        number of times to skip eviction of content from this memory
        frame.
     synch_entryp=PC_OFFSET
        Memory frame holds a page of a Data Management synchronized
        segment.  PC_OFFSET is a relative pointer within page_fault
        of entry point transferred to when synchronizing content of
        memory frame with its disk record.
```

2.b) New request: page_control_check, pcc

The proposed new page_control_check (pcc) request has a very simple user interface. When invoked without arguments, it displays enhanced output from the pc_check_tables_ subroutine which includes DIAGNOSTIC REQUEST(s) that can be entered by the user to get further information about the inconsistent items.

```
azm:  sld >old_dumps>21
Could not find apte for process_idx 0
     dbr = 54012
This is an early dump.

  ERF 21  in directory >old_dumps dumped at 09/19/21  0638.4 pst Sun.
Proc   0 DBR     54012 empty     last on cpu    Initializer.SysDaemon.z
```

```
azm:  pcc
_____________________________________________
Bad counter for ASTE.
 --> ASTE at SST|1240
     ASTE for config_deck (Seg 2  in Proc 0), at SST|1240.
 np = 0, should be 4
             DIAGNOSTIC REQUEST(s): aste -offset 1240;
_____________________________________________
Bad counter for ASTE.
 --> ASTE at SST|35300
     ASTE for disk_mst_seg (Seg 434  in Proc 0), at SST|35300.
 csl = 15, should be 31
 records = 0, should be 31
             DIAGNOSTIC REQUEST(s): aste -offset 35300;
_____________________________________________
Statistics:
         2 Bad aste counter
```

If invoked with the -long control argument, it automatically runs the suggested DIAGNOSTIC REQUEST(s) as part of each output report. For example, the first report above is expanded as shown below.

```
azm:  pcc -long
_____________________________________________
Bad counter for ASTE.
 --> ASTE at SST|1240
     ASTE for config_deck (Seg 2  in Proc 0), at SST|1240.
 np = 0, should be 4

             DIAGNOSTIC REQUEST: aste -offset 1240

ASTE for config_deck (Seg 2  in Proc 0), at SST|1240.

  1240      0 001260015560 000000000000 000002000000 000000000000
  1244      4 004001777777 410000300000 000000000000 000000000000
  1250     10 000000000000 000000000000 004024004000 000000000002

uid = 000000000000, vtocx -1 on pvtx 1 (dska device 0)
max len 4, 4 recs used, 0 in core, cur len 4
Not updated as used.
Not updated as modified.
Hardcore segno = 2

Flags: usedf hc_sdw ehs nqsw dnzp ddnp

PAGE      PTW       DEVADD    PD COPY   at SST|1254

   0  000120401025   23501                phu wired
   1  000140400025   23502                wired
   2  000160400025   23503                wired
   3  000200400025   23504                wired
_____________________________________________
```

2.c) New request: slt_entry, slte

There are several reasons for proposing a new request to display Segment Loader Table entries (SLTEs). As a convenience, it is easier to get properties for a segment loaded from the Multics System Tape (MST) using an azm request than to manually search for such information in the >ldd>hard>info>hardcore.header or hardcore.checker segment.

However, a more significant reason stems from the bootloader practice of building the SLT entries incrementally as segments are loaded from the MST; and editing or eliminating those entries as segments loaded in low-memory during early bootload operations get moved to more permanent locations; or as segments needed only during initialization of the system get deleted. The online SLT grows and shrinks as bootloading progresses, and segment attributes get modified from the starting values specified in the hardcore.header segment.

The new slte request displays information about an SLT entry as it was known to the bootloader when the system crashed. Segments not loaded in memory (because they haven't yet been read from the MST, or because they have been used and then deleted when no longer needed) won't show up in slte output. And attributes of segments displayed in SLTEs may have changed if the bootloader moves them to higher areas of real memory, or fabricates >sl1 directory branches for the segments, or otherwise repurposes utility segments (like firmware segments or free_area segments).

The slte request will list header information from the SLT, followed by information for selected hardcore segments. SLTEs can be selected by segment name, segment number, or by segment attributes. For example:

```
azm:  slte 406 –header

                        Supervisor Segments      Initializing Segments
                          First:    0              First:  400
                           Last:  115               Last:  465

wi_linkage             406  (0, 0, 0) read execute write wired
  wired_init_linkage        init_seg breakpointable
                            wired length: 1024 words; paged length: 2 pages;
                            max length: 1 pages;
```

Interface for the slte request is summarized by the help -brief output below. The request accepts 0, 1 or 2 segment_number/segment_name combinations. If no values are given, information is displayed about all segments known to the bootloader at time of the crash. If two values are given, they select a range of segment entries to display.

```
azm:  help slte -bf

   (257 lines in info)

2022-11-22  slt_entry, slte

Syntax:
   slte  {SEGNO | SEGNAME}  {SEGNO | SEGNAME}  {-control_args}

Syntax as an active request:
  [slte  {SEGNO | SEGNAME}  {SEGNO | SEGNAME}  {-control_args}]

Arguments:
 SEGNO   SEGNAME

Control arguments:
 -header, -he      -all_names       -attribute ATTRIBUTEs,
 -no_header, -nhe  -primary, -pri     -attr ATTRIBUTEs

List of attributes:
 privileged, priv        wired      gate              temp_seg, temp
 encacheable, cache      ^paged     abs_seg, abs      per_process, proc
 breakpointable, break   firmware   init_seg, init    branch
```

3.a-i) Fix bugs in bound_azm_ requests and subroutines

The most significant of the unreported bugs occurs in azm/amu modules that temporarily switch the selected process to look for something in another process' workspace: usually an SDW referencing a page table. The intent of these modules is to switch back to the process which was selected when that module was called; but sometimes the module exits without reverting to the original selected process; or the user quits out of azm to abort some output; then uses program_interrupt to reenter azm.

Modules with this problem are calling either amu_$fdump_mpt_change_idx, or the trio of subroutines: amu_$fdump_mpt_temp_change_idx, then one or more amu_$fdump_mpt_change_idx calls, and finally amu_$fdump_mpt_revert_idx. The interface for the trio of routines also fails to handle the case when module A calls amu_$fdump_mpt_temp_change_idx, and makes 0 or more amu_$fdump_mpt_change_idx calls; but then calls module B which also uses this trio of routines, thus losing track of the original selected process seen by module A.

The user interface for the trio of subroutines was corrected to have each caller save the earlier caller's starting selected process value, and then restore that value before it returns. And all modules using any of these routines were examined and corrected if necessary in their use of the returns, proper setup of cleanup on-units to restore the starting selected process, and guaranteed return through an exit label that restores the starting process.

3.j) copy_pdir incorrectly captures UID for segments in the saved pdir

The copy_deadproc and copy_liveproc commands both call the copy_pdir_ subroutine to copy segments in a process directory, and to capture UID and segment numbers of those pdir segments so they can later be examined by azm's saved_pdirs dumping mechanism.

However, Multics Ticket #298 reports a bug in the mechanism for capturing a UID in a text string representation. Multics UID bit strings were guaranteed unique within a given Multics system by generating them from a 36-bit seed taken from the middle of a Multics calendar clock reading made when the system was booted. For many years, the left-most bit of this 36-bit string was "0"b, so converting the 36-bit string to a fixed bin(35) integer did not lose any meaningful data.

However, since January 13, 2008 at 09:55:20 pst, the calendar clock value has had a "1" in the leftmost bit of the seed UID. This caused the copy_pdir_ method for converting the 36-bit string to a 12-character octal representation to fail, because it ignored that left-most bit.

copy_pdir_ was changed to avoid this problem by using an ioa_ control string to convert the 36 bits to octal characters: call ioa_$rsnnl ("^12.3b", …, uid);

4.a-b) New subroutines: azm_sst_util_$XXX

Both the aste request and the new cme request will call two new subroutine entry points to locate an ASTE structure and/or a page table.

```
ENTRY POINT:  azm_sst_util_$find_aste_given_offset

FUNCTION:     Given an offset into the SST, unpaged_page_tables, or
              int_unpaged_page_tables segments, this entry point finds
              the:
               - ASTE (for an SST offset) and its page table (and PT
                 size), and a pointer to a PTW in that page table
                 IFF the offset points into the page table region,
                     instead of to its leading ASTE structure; OR
               - wired page_table (for an offset into one of the other
                 two segments (and PT size), and a pointer to a PTW in
                 that page table.
```

The calling sequence is as follows:

```
  call azm_sst_util_$find_aste_given_offset (amu_info_ptr,
       offset,
       process_idx, segnum, astep, ptp, pt_size, pt_name, ptwp, code);
```

```
ENTRY POINT:  azm_sst_util_$find_aste_given_segno

FUNCTION:     Given a segment number and process_idx in which that
              segment is known, this entry point finds:
               - (for a segment defined in the SST) the ASTE and its
                 page table (and PT size); OR
               - wired page_table (for an offset into one of the other
                 two segments (and PT size).
```

The calling sequence is designed to be as close to the prior subroutine's sequence as possible.

```
     call azm_sst_util_$find_aste_given_segno (amu_info_ptr,
          process_idx, segnum,
          sdwp, astep, sptp, spt_size, spt_name, ptp, code);
```

The cme –search CME_OFFSET request will call the following new entry point.

```
ENTRY POINT:  azm_sst_util_$find_ptws_with_address

FUNCTION:     Given an input structure containing a mem_addr element
              (absolute memory address), this entry point locates
              PTWs in either int_unpaged_page_tables, unpaged_page_tables,
              or SST segments whose ptw.add references that memory frame
              address.  These three segments contain all of the page
              tables used by Multics.
```

The calling sequence is shown below. It includes a pointer to the ptws structure declared in azm_sst_util.incl.pl1.

```
      call azm_sst_util_$find_ptws_with_address (amu_info_ptr,
           addr(ptws), code);
```

```
/*  START OF:       azm_sst_util_ptws.incl.pl1                * * * * * * * * * * * * * */


  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  /*                                                                                  */
  /* NOTE:  This is a private structure for use by components of bound_azm_ and should not be used */
  /*        in other bound segments.  Its contents are subject to change at any time to suit the   */
  /*        needs of the azm subsystem.                                               */
  /*                                                                                  */
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  dcl 1 ptws aligned based (ptws_ptr),
      2 mem_addr     fixed bin(24) unsigned,    /* memory frame address to look for in ptw.add  (In) */
      2 countN       fixed bin,                 /* count of possible results to look for.       (In) */
      2 resultsN     fixed bin,                 /* number of ptws found in search.             (Out) */
      2 results      (ptws_countN refer(ptws.countN)),
                                                /* array of possible results.                  (Out) */
        3 ptrs,
          4 astep    ptr,                       /*  ptr to ASTE preceding page_table w/              */
                                                /*  matching PTW                                     */
                                                /*   - non-null only if PTW found in SST             */
                                                /*   - null if PTW found in unpaged_page_tables      */
          4 ptp      ptr,                       /*  ptr to page_table holding matching PTW           */
          4 ptwp     ptr,                       /*  ptr to matching PTW                              */

        3 pt_name    char(32) var,              /*  name of segment holding the matching page table. */
        3 numbers,
          4 pt_size  fixed bin,                 /*  size of page_table holding matching PTW          */
          4 segnum   fixed bin,                 /*  segment number associated with page_table        */
                                                /*   - set to -1 for page tables found in SST        */
                                                /*   - set to segment number for unpaged_page_tables */
       ptws_ptr      ptr,                       /* ptr to this structure.                            */
       ptws_countN   fixed bin;                 /* dimension of ptws.results sub-structure to alloc. */

/*  END OF:         azm_sst_util_ptws.incl.pl1                * * * * * * * * * * * * * */
```

5.a-i) Fix bugs or add new features to the following bound_amu_ subroutines

amu_.alm: add entry points amu_$get_slt and amu_$get_slte entry points. These transfer to procedures in amu_get_name_.pl1 which contains other code that accesses System Loader Table entries.

amu_do_translation_.pl1:

A) Setup cleanup on-unit to restore selected process (bug described in 3.a-i above).

B) Consolidate return statements to reduce replication of complex code.

amu_et_.alm:

A. Add errors: $address_negative, $address_not_octal, $address_too_big, $aste_big_offset, $aste_freed, $aste_seg_not_in_use, $aste_small_offset, aste_volume_map, $cme_bad_offset, $cme_big_index, $cme_big_offset, $cme_no_aste, $pt_not_found, $ptw_array_overflow, $ptw_bad_offset, $ptw_no_upt, $ptw_not_found, $ptw_paged_out_of_memory, $ptw_small_offset, $ptw_small_offset_upt, $sdw_address_invalid, $sdw_address_uneven, $seg_not_active, $seg_not_paged, $segno_from_slte

B. Rename:

$free_core	to	$memory_frame_vacant,
$negative_offset	to	$offset_negative,
$non_existant_mem	to	$memory_frame_offline,
$not_octal_offset	to	$offset_not_octal.

C. Re-alphabetize the error codes by entry point name.

amu_fdump_mpt_.pl1:

A. Change calling sequence of $temp_change_idx and $revert_idx entry points to add a P_save_idx parameter to support recursive use of these two entry points.

B. Change $revert_idx entry point to avoid any change if amu_info.proc_idx_hold = -1.

C. Change get_data_ internal proc (used by amu_fdump_mpt_) for new calling sequences in item A.

amu_get_name_.pl1:

A. Fix one subroutine exit to properly check ret_ptr_sw instead of always trying to return a pathname string.

B. Add $get_slt and $get_slte entry points (needed by the new azm slte request).

C. Change get_init_seg_slt and get_sup_seg_slt internal procedures to properly reference pathname attached to SLTE using the path structure in slt.incl.pl1.

D. Fix error message for SLT name table not found in dump.

E. Fix errors in get_ast_name code.

amu_hardcore_info_.pl1:

A. Change get_and_set to avoid changing its input argument.

B. Change get_ptr_from_slt to correctly set ring 0 stack segno in an "early dump" in no slte found for one of expected segments in the dump.

C. Change fixed(baseno(PTR),17) to segno(PTR) throughout program.
D. Change fixed(rel(PTR),17) to wordno(PTR) throughout program.
E. Change amu_hardcore_info_$fdump to fill-in new elements of hardcore_info.breakpoint_page substructure. These elements fill a former hardcore_info.pad1 field at end of the structure.
F. Correct code indentation issues.

amu_info_.pl1:
In $create entry point, initialize amu_info.proc_idx_hold value.

amu_parse_ptr_args_.pl1:
A. Change to use revised calling sequence for amu_$fdump_mpt_temp_change_idx and amu_$fdump_mpt_revert_idx.
B. Create cleanup on-units to revert changed process_idx by calls to amu_$fdump_mpt_temp_change_idx.

amu_print_.pl1:
A. Change to use revised calling sequence for amu_$fdump_mpt_temp_change_idx and amu_$fdump_mpt_revert_idx.
B. Create cleanup on-units to revert changed process_idx by calls to amu_$fdump_mpt_temp_change_idx.

6.a) Change command: analyze_multics (azm)

Enhance the azm command program to:

A. Increase the subsystem version from 2.3 to 2.4.

B. When debugging the analyze_multics subsystem, add the directory containing the analyze_multics object segment as first entry in the subsystem's info-directory search paths so any changed/added .info segments will be displayed by azm's help request instead of any installed version of the command.

6.b) Avoid bug in numeric_to_ascii_base_.pl1

Ticket #296 (https://multics-trac.swenson.org/ticket/296) reports a problem in using the improved index_set active function as an active request in analyze_multics. When using index_set to generate 8-digit octal integers for use as azm absolute memory address values, an address ending in 6- or 7-zeros (e.g., 1000000o) gets converted to E-format: 1e+6o. But E-format octal numbers are not accepted by the cv_oct_ subroutine used by analyze_multics requests to process absolute address character strings.

As a work-around for this problem, change the trigger point for consecutive trailing zeros used by numeric_to_ascii_base_ to change the result to E-format only if the integer result ends in >7 trailing zeros. The prior value was >5 trailing zeros.

A full remedy for this problem would involve either changing cv_oct_ (and cv_binary_, cv_dec_, and cv_hex_ to accept input character strings in E-format); or adding a new numeric_to_ascii_base_ entry point that enables the caller to specify whether E-format output strings are ever acceptable, and the trigger point(s) at which they are used. Both of these envisioned remedies are significant projects deemed outside the scope of this MCR.

6.c) Fix bugs in pc_check_tables_.pl1

Change the pc_check_tables_.pl1 subroutine to permit it to be called by the new azm page_control_check (pcc) request. These are minor changes which do not impact the rules checked by the subroutine, or information content of the inconsistency reports generated by checking. The changes only enable those reports to all employ the same format and statistics counting.

A. Change order of display_aste location message to follow starting
   message for ASTE content errors.
B. Add statistics for bad ASTE counter errors. Most dumps have several such reports.
   They are usually non-significant warnings.
C. Track lengthy do-group start/end points using PL/I labels so the compiler
   matches do-statements with corresponding end-statements.
D. Remove variables declared but never referenced, per Multics programming standards.

7.a) Correct error in unpaged_page_tables.incl.pl1

A. Change declaration of upt.first_entry to denote only the location of
   the first Page Table + header (rather than declaring its full
   contents). This prevents infinite loop in the azm request:
     display unpaged_page_tables -as upt
   which thinks upt.first_entry includes a very large number of PTWs.
B. Add declaration of upt_entry_header so size of this header can be
   determined in PL/I code.
C. Expand comments to explain use and layout of the segments.

Multics Ticket #293 (https://multics-trac.swenson.org/ticket/293) describes this problem in more detail.

Recompile structure_library_5_.cds which adds the unpaged_page_tables.incl.pl1 declarations to the azm (and dump_segment) known structure library.

7.b) Add elements to structures in amu_hardcore_info.incl.pl1

A. Add iupt and int_unpaged_page_tables elements.

B. Add 3 elements describing the breakpoint_page segment and its single PTW (locating its data page in a wired, real-memory frame.) This breakpoint_page gets added to end of many wired breakpointable hardcore supervisor segments.

C. Add/clarify comments describing various elements.

D. Add pad elements to facilitate future addition w/o need to recompile many components not otherwise affected by those additions.

Recompile the existing source programs that reference changed areas in amu_hardcore_info.incl.pl1. These sources are not changing; only their object components will be installed.

- amu_deadproc_.pl1
- amu_definition_.pl1
- amu_return_val_.pl1
- amu_tc_data_.pl1
- azm_syserr_.pl1

7.c) Add declaration to azm_info.incl.pl1

A. Declare azm_area to make the existing azm-specific “standard multics area” easier to use.

7.d) Change slte.incl.pl1 comments

A. Move comment denoting “End of word 2” to the correct element of slte.

8) Move copy_erf_seg_.pl1

Move copy_erf_seg_.pl1 (which references many bound_amu_ routines) from bound_meter_util_ into bound_azm_. The source segment itself is not changing.

9-10) Changed documentation (see the Documentation section of this MCR).

mbuild Description of Proposed Changes

The build script segment for this MCR involves changes to 6 bound segments, 5 include files, and 10 info segments. It is shown below in its entirety to present a summary description of the proposed changes that includes information useful to the developer, auditor, and other reviewers.

```
                    azm01.mb  12/29/22  1048.6 pst Thu


Description --
bound_amu_:
 M1) Change amu_get_name_.pl1 to: add get_slt and get_slte entry points;
     repair bugs.
 M2) Change amu_.alm to map amu_$get_slt and amu_$get_slte onto the new
     entry points in amu_get_name_.  Change bound_amu_.bind to retain
     those new entry points.
 M3) Change amu_hardcore_info_.pl1 to: avoid changing input arg; use
     newer PL/I builtin functions to simplify code.
 M4) Change amu_fdump_mpt_.pl1 to fix bug permitting recursive use of
     $temp_change_idx and $revert_change_idx entry points.  Both existing
     and new requests need to use these subrs recursively (and are probably
     incorrectly doing so now even though full support is NOT present.
 M5) Change amu_do_translation_ to establish cleanup on-unit when changing
     selected process.
 M6) Change amu_parse_ptr_args_.pl1 and amu_print_.pl1 for new calling
     sequence of amu_$fdump_mpt_temp_change_idx and its enhanced cleanup
     protocol.
 M7) Recompile for changes to amu_hardcore_info.incl.pl1:
        amu_definition_.pl1
        amu_return_val_.pl1
        amu_deadproc_.pl1
        amu_tc_data_.pl1


bound_meter_util_:
 M1) Move copy_erf_seg_.pl1 to bound_azm_ (where other callers of
     bound_amu_ reside).


bound_azm_:
 Z1) Change analyze_multics.pl1 to: use -referencing_dir-like search
     to find updated .info segs when running dev version of azm; to
     increment azm version from 2.3 to 2.4 to reflect new requests added.
 Z2) Change azm_requests_1_.pl1 to extend request: aste
     Change azm_requests_2_.pl1 to extend request: absadr
     Change azm_requests_3_.pl1 to add new requests:
      cme, page_control_check (pcc), slt_entry (slte);
     (Ticket #297)
 Z3) Change azm_str_util_.pl1 to assist in getting segname for an ASTE.
 Z4) Add azm_sst_util_.pl1 to assist in locating an ASTE by offset w/in SST.
 Z5) Change several sources to correct cleanup, quit, and conversion
     condition handling: azm_requests_1_.pl1, azm_requests_3_.pl1,
      azm_display_am_.pl1, azm_display_fdump_events.pl1,
      azm_verify_dump_ams_.pl1, azm_why_.pl1
 Z6) Change azm_dump_mem_.pl1 to: fix indentation; use size builtin function
     instead of literal constants; use WORDS_PER_PAGE system constant.
 Z7) Change azm_why_.pl1 to: remove unneeded code; use mc_area; correct use
     of unspec builtin; use segno and wordno builtins.
```

```
 Z8) Change bound_azm_.bind: to add azm_sst_util_ object; to add missing
     Order keyword required by Multics programming standards.
 Z9) Change copy_pdir_ to correctly transfer segment UIDs into the pdir_info
     annotation segment.  (Ticket #298)
 Z10) Recompile for changes to amu_hardcore_info.incl.pl1:
        azm_syserr_.pl1
 Z11) Move copy_erf_seg_.pl1 from bound_meter_util_ to bound_azm_.


bound_library_wired_:
 W1) Change pc_check_tables_.pl1 to: fix order of ASTE location message in
     output report; add stats for "Bad ASTE counts" errors; track very long
     do-groups in code using PL/I labels.


bound_structure_lib_:
 S1) Recompile structure_library_5_.cds to use revised upt structure in
     changed unpaged_page_table.incl.pl1.  Multics ticket #293


bound_multics_bce_:
 B1) Change numeric_to_ascii_base_.pl1 to avoid switching to E-format output
     to reduce output length when 8 or more consecutive zeros appear at the
     end of an integer output string.  Before this change, 6 or more
     consecutive zeros caused conversion to e-format.  This conversion prevents
     index_set from being able to generate all possible Multics 8-digit octal
     absolute address values in a format accepted by cv_oct_ subroutine (which
     does not support E-format input strings).  Multics ticket #296


Includes:
 C1) Fix errors in existing slte.incl.pl1 comments.
 C2) Add azm_area declaration to azm_info.incl.pl1.
 C3) Add breakpoint_page elements to end of amu_hardcore_info.incl.pl1.
     This is only change to data elements.  Recompile: amu_hardcore_info_.pl1,
     amu_info_.pl1, azm_requests_1_.pl1.  Add/clarify comments for many elements.
 C4) Change unpaged_page_tables.incl.pl1 declaration of upt.first_entry to
     avoid loop in azm request:  display unpaged_page_tables -as upt
 C5) Add new azm_sst_util_ptws.incl.pl1 include file to support the
     azm_sst_util_$find_ptws_with_address interface.


Infos:
 I1) Change aste.info to revised input arguments.
 I2) Add core_map_entry.info describing new cme request.
 I3) Add page_control_check.info describing new pcc request.
 I4) Add slt_entry.info describing the new slte request.
 I5) Correct typos & vi-reported errors in:
     display.info, display_absolute.info, list_processes.info,
     sdw.info.
 I6) Merge and update copy_deadproc.info and copy_liveproc.info
     to provide info needed to understand/use these commands.
     (Ticket #298)
```

Changes --

```
 Bound_obj:    bound_azm_                        IN: tools  UPDATE;
   bindfile:   bound_azm_.bind                    REPLACE;
   source:     analyze_multics.pl1                REPLACE  compiler: pl1 -ot;
   source:     azm_display_am_.pl1                REPLACE  compiler: pl1 -ot;
   source:     azm_display_fdump_events.pl1       REPLACE  compiler: pl1 -ot;
   source:     azm_dump_mem_.pl1                  REPLACE  compiler: pl1 -ot;
   source:     azm_request_table_.alm             REPLACE  compiler: alm;
   source:     azm_requests_1_.pl1                REPLACE  compiler: pl1 -ot;
   source:     azm_requests_2_.pl1                REPLACE  compiler: pl1 -ot;
   source:     azm_requests_3_.pl1                REPLACE  compiler: pl1 -ot;
   source:     azm_sst_util_.pl1                  ADD      compiler: pl1 -ot;
   source:     azm_str_util_.pl1                  REPLACE  compiler: pl1 -ot;
   source:     azm_syserr_.pl1                    REPLACE  compiler: pl1 -ot;
   source:     azm_verify_dump_ams_.pl1           REPLACE  compiler: pl1 -ot;
   source:     azm_why_.pl1                       REPLACE  compiler: pl1 -ot;
   source:     copy_erf_seg_.pl1                  ADD      compiler: pl1 -ot;
   source:     copy_pdir_.pl1                     REPLACE  compiler: pl1 -ot;

 Bound_obj:    bound_meter_util_                 IN: tools  UPDATE;
   bindfile:   bound_meter_util_.bind             REPLACE;
   source:     copy_erf_seg_.pl1                  DELETE   compiler: pl1 -ot;

 Bound_obj:    bound_amu_                        IN: tools  UPDATE;
   source:     amu_.alm                           REPLACE  compiler: alm;
   source:     amu_deadproc_.pl1                  REPLACE  compiler: pl1 -ot;
   source:     amu_definition_.pl1                REPLACE  compiler: pl1 -ot;
   source:     amu_do_translation_.pl1            REPLACE  compiler: pl1 -ot;
   source:     amu_et_.alm                        REPLACE  compiler: alm;
   source:     amu_fdump_mpt_.pl1                 REPLACE  compiler: pl1 -ot;
   source:     amu_get_name_.pl1                  REPLACE  compiler: pl1 -ot;
   source:     amu_hardcore_info_.pl1             REPLACE  compiler: pl1 -ot;
   source:     amu_info_.pl1                      REPLACE  compiler: pl1 -ot;
   source:     amu_parse_ptr_args_.pl1            REPLACE  compiler: pl1 -ot;
   source:     amu_print_.pl1                     REPLACE  compiler: pl1 -ot;
   source:     amu_return_val_.pl1                REPLACE  compiler: pl1 -ot;
   source:     amu_tc_data_.pl1                   REPLACE  compiler: pl1 -ot;

 Bound_obj:    bound_library_wired_              IN: hard  UPDATE;
   source:     pc_check_tables_.pl1               REPLACE  compiler: pl1 -ot;

 Bound_obj:    bound_multics_bce_                IN: hard  UPDATE;
   source:     numeric_to_ascii_base_.pl1         REPLACE  compiler: pl1 -ot;

 Bound_obj:    bound_structure_lib_              IN: tools  UPDATE;
   source:     structure_library_5_.cds           REPLACE  compiler: cds;

 Include:      amu_hardcore_info.incl.pl1        IN: sss.include  REPLACE;
 Include:      azm_info.incl.pl1                 IN: sss.include  REPLACE;
 Include:      azm_sst_util_ptws.incl.pl1        IN: sss.include  ADD;
 Include:      slte.incl.pl1                     IN: sss.include  REPLACE;
 Include:      unpaged_page_tables.incl.pl1      IN: sss.include  REPLACE;

 Info:         copy_liveproc.info                IN: priv.info  DELETE;
 Info:         aste.info                         IN: azm.info  REPLACE;
 Info:         copy_deadproc.info                IN: priv.info  REPLACE;
                add_name:  copy_liveproc.info;
 Info:         core_map_entry.info               IN: azm.info  ADD;
                add_name:  cme.info;
```

```
 Info:         display.info                      IN: azm.info  REPLACE;
                add_name:  d.info;
 Info:         display_absolute.info             IN: azm.info  REPLACE;
                add_name:  da.info;
 Info:         list_processes.info               IN: azm.info  REPLACE;
                add_name:  lsp.info;
 Info:         page_control_check.info           IN: azm.info  ADD;
                add_name:  pcc.info;
 Info:         sdw.info                          IN: azm.info  REPLACE;
 Info:         slt_entry.info                    IN: azm.info  ADD;
                add_name:  slte.info;



Information --

 Phase:             developing;
 Descriptor:        >t>multics_libraries_;
 LogDir:            -library *.log;

 Approve_IDs:       MCR10129;
 Install_ID:        ;

 Developers:        GDixon.Multics;
 Auditors:          ;
 Installers:        ;


Status --
TICKETS:
 #293:  unpaged_page_tables.incl.pl1 has an invalid declaration of the upt header.
 #296:  numeric_to_ascii_base_ generates 8-digit octal integer strings that cv_oct_
cannot convert
 #297:  New/enhanced requests for analyze_multics: cme, slte, page_control_check, aste
 #298:  copy_deadproc and copy_liveproc incorrectly capture UIDs for segments in the
saved pdir

    AREAS                                PROGRESS
 ------------------------------------   --------------

 -- Design
  amu_fdump_mpt_.pl1                    complete
   fix significant non-recursive
   design in: $temp_change_idx and
              $revert_change_idx
  azm_str_util_.pl1                     complete
   extension to get segname for
   aste located by .astep offset
   (rather than segno/segname)
  azm_sst_util_.pl1                     complete
   new routine with three new
   entry points to facilitate various
   searches for ASTE and Page Tables.


 -- Coding
  amu_hardcore_info.incl.pl1            complete
   - Add new elements to the
     hardcore_info structure:
      .pointers.iupt
      .segno.sst_seg
```

```
     .segno.int_unpaged_page_tables
     .breakpoint_page.bkpt_ptw_add
                     .bkpt_cme_offset
                     .bkpt_segno
  - Add new element to the
    hardcore_cur structure:
     .iuptp
  - Reorganize declarations to
    hardcore_info and hardcore_cur
    structures.  Existing names are
    NOT changing.
  - Recompile ONLY the following
    sources that using this %include
    which are not otherwise changing.
         amu_definition_.pl1
         amu_return_val_.pl1
         amu_deadproc_.pl1
         amu_tc_data_.pl1
         azm_syserr_.pl1
         copy_erf_seg_.pl1


analyze_multics.pl1                    complete
 - make new request info segs visible
   while testing changes to azm.

azm_request_table_.alm add new         complete
   request definitions:
 - cme                                  done
 - page_control_check, pcc              done
 - slt_entry, slte                      done

azm_requests_1_.pl1                    complete
- requests for hardcore data structs
  $aste                                 done
    - new ctl: -offset ASTE_OFFSET
    - new ctl: -ptw PTW_OFFSET

azm_requests_2_.pl1                    underway
- make absadr request find init_segs
  with pts in int_unpaged_page_tables

azm_requests_3_.pl1                    complete
- events request                        done
   - repair cleanup, conversion,
     and quit condition handling
- requests for hardcore data structs
  $cme  (new request)                   done
  $slte (new request)                   done

  $page_control_check  (new request)    done
    - use new get_core_map              done
    - page_control_tables_ co-rtns      done
       display_aste_                    done
         report segname of ASTE using
         aste.strp
       display_cme_                     done
         report segname of frame owner
         using cme.astep and aste.strp
       display_ptw_                     done
         report segname of page owner
```

```
          from absadr(ptw) by finding
          preceding ASTE for page_table
        display_ptr_                    done
        display_pvname_                 done

azm_sst_util_.pl1                     complete
 - $find_aste_given_offset             done
   new routine to find ASTE/pt
   in SST, unpaged_page_tables &
   int_unpaged_page_tables for
    req: aste -offset ASTE_OFFSET  or
    req: aste -ptw PTW_OFFSET      or
    req: cme -aste ASTE_OFFSET     or
    req: cme -ptw  PTW_OFFSET
 - $find_aste_given_segno              done
   new routine to find ASTE/pt
   in SST, unpaged_page_tables &
   int_unpaged_page_tables for
    req: aste SEG_NAME      or
    req: aste SEG_NUMBER    or
    req: cme -seg SEG_NAME  or
    req: cme -seg SEG_NUMBER
 - $find_ptws_with_address             done
   new routine to find ptws which
   reference same memory frame for
   req: cme -srh PTW_OFFSET
azm_str_util_.pl1                     complete
 - $one_segment_number
   new ep to get segname for
   ASTE located by .astep offset
   to be called by:
   req:  aste -ofs ASTE_OFFSET
   req:  pcc
     co-rtn:   display_aste_

azm_dump_mem_.pl1                     complete
 - Clarify code by:                    done
    - fixing code indentation for
      $azm_dump_mem entry point;
    - using size(STRUCTURE_NAME)
      built-in (instead of literal
      values) to explain code
      accessing cme and sdw entries;
    - using WORDS_PER_PAGE constant.
 - New azm_dump_mem_$get_cme_sdw       done
   entry point.

copy_pdir_.pl1:                       complete
 - fix problem transferring seg UIDS
   info pdir_info annotation created
   when deadproc/liveproc is copied
   to >dumps>saved_pdirs.
   Ticket #298

pc_check_tables_.pl1                  complete
  - minor changes to order of calls
    to co-routines.


amu_.alm                              complete
  - add 2 new entry points:
```

```
     amu_$get_slt and amu_$get_slte

amu_do_translation_.pl1               complete
  - add cleanup on-unit
  - consolidate return handling

amu_fdump_mpt_.pl1                    complete
  - fix $temp_change_idx and
    $revert_change_idx to handle
    recursive calls.

amu_$fdump_mpt_temp_change_idx
    and amu_$fdump_mpt_revert_idx
  - change callers (reported by pcref)
    to use revised calling sequence:
      amu_fdump_mpt_.pl1               done
       - add cleanup on-units
      amu_parse_ptr_args_.pl1          done
       - add cleanup on-units
      amu_print_.pl1                   done
       - add cleanup on-units
      azm_display_am_.pl1              done
       - add cleanup on-units
      azm_display_fdump_events.pl1     done
       - add cleanup on-units
       - correct do-group array bounds
      azm_requests_1_.pl1
      azm_verify_dump_ams_.pl1         done
       - add cleanup on-units
      azm_why_.pl1                     done
       - add cleanup on-units
       - simplify/correct code

amu_get_name_.pl1                     complete
  - exit bug for one entry point.
  - add $get_slt and $get_slte eps.
  - change get_init_seg_slt and
    get_sup_seg_slt procs to properly
    ref pathname attached to slte.

amu_hardcore_info_.pl1                complete
  - $get_and_set to:                   done
     - avoid setting input arg
     - set ring 0 stack segno
       in an "early dump"
  - use new PL/I built-ins to          done
    clarify code
  - add breakpoint_page info to        done
    hardcore_info.

amu_info_.pl1                         complete
  - init amu_info.proc_idx_hold when   done
    amu_$create is called.
  - move allocation of hardcore_info   done
    into amu_hardcore_info_.pl1


pc_check_tables_.pl1                  complete
 - minor corrections in order of
   calling co-routines to report
   anomalies.
```

```
 numeric_to_ascii_base_.pl1           complete
  - change consecutive zero count
    at which output format changes
    to E-format when outputting
    integer values from >5 to >7
    consec. 0s before radix point.
    NB: Only fixes part of ticket
        #296.

 structure_library_5_.cds             only_include_file_changes
  - No change to actual source
    for this cds segment; only the
    following include file changes.
    Only this cds source is affected
    by that include file change.
    Ticket #293.


 azm_info.incl.pl1                    complete
  - add azm_area declaration.
 azm_sst_util_ptws.incl.pl1           complete
  - create new include as part of
    azm_sst_util_$find_ptws_with_address
    interface definition.
 slte.incl.pl1                        complete
  - move comment to correct element
    of the slte structure.
 unpaged_page_tables.incl.pl1         complete
  - change declaration of
    upt.first_entry to provide
    only an offset of first structure
    rather than repeating dcl of all
    structure elements.  Otherwise,
    probe and dump_segment -as
    misread the sub-structure dcl and
    go into lengthy loop trying to
    display an infinite first_entry.ptws
    array.


-- Testing
 - aste request (changes)             complete
 - cme  request (new)                 complete
 - pcc  request (new)                 complete
    >dumps>1 and >old_dumps>(22 23 25
    021817.1749.0.7 030121.1749.0.7)
    provide useful input data for
    testing.
 - slte request (new)                 complete


-- Documentation                     complete
 - aste.info                          revised
    - doc new -offset APTE_OFFSET
      control arg
 - cme.info                           done
 - copy_deadproc.info                 done
    - merged with: copy_liveproc.info
 - display.info                       revised
    - clarify incorrect/incomplete doc
```

```
 - display_absolute.info                revised
    - correct doc errors
 - list_processes.info                  revised
 - page_control_check                   done
 - slte.info                            done
    - new -attr ATTRIBUTE ctl           done

-- History Comments
 reasonably complete for above segs     complete

-- MCR Approval
 no MCR number obtained yet

-- Auditing

-- Installing
```

Testing

analyze_multics is a large subsystem with many requests that display data from a crash image. While it is composed of two large bound segments (bound_azm_ and bound_amu_), these are executed as a single subsystem: bound_azm_ does not function without the dump image per-process addressing capabilities of bound_amu_.

The GHM system has a variety of saved crash images representing system interruptions taken at several points during bootload of the system, as well as during full system operation. These crash images contain initialization-related segments that only exist which the bootload is in progress; as well as full-operation environments with OS data segments populated with a variety processes and running applications. This assortment of crash images was very helpful in testing additions to the aste command extending its display capabilities to: wired hardcore supervisor segments (which actually have no ASTE); and to ASTEs used to access physical volume bit maps – ASTEs that are never accessed as a segment; and finally to initialization segments that do exist when Multics is in full operation.

All new functionality was tested to ensure that each request located the correct underlying data in the dump image, and then properly displayed that data to the user. Several new algorithms were devised for locating ASTEs and page tables by offset within their containing data segment. These algorithms were tested extensively to ensure the correct data was located by ASTE or PTW offset.

The new page_control_check request provided to excellent test bed. Its reports identify items that were inconsistent with normal page control links between related data. Ensuring that various azm requests can access and display each of the reported items provided some focused testing across a variety of crash situations. (The pcc request can be used to examine a crash taken at any point in Multics operation.)

Several quick tests were run using saved process directories from a “dead process” and “from several live processes”. With correction of the UID capture algorithm, stack tracing and display of data from non-stack segments worked again for these saved process image types.

Finally, functionality of the revised number conversion tools (MCR10101 and MCR10118) used within analyze_multics was thoroughly tested. A minor problem was found in numeric_to_ascii_base_ generation of octal strings, and a suitable avoidance was implemented and successfully tested. No further issues with number conversion in azm surfaced during all testing.

In summary, focused testing of the new features, and each enhancement and bug fix were completed successfully. No new problems or issues have been introduced by the changes proposed in this MCR.

Documentation

The following command documentation has been enhanced to better describe how “dead processes” and “live processes” may be examined using analyze_multics.

copy_deadproc.info

```
2022-12-05  copy_deadproc

Syntax as a command:
  copy_deadproc DEADPROC_NAME {-ctl_args}
  copy_deadproc -name DEADPROC_NAME {-ctl_args}

Function:  Gather information about a "dead process" and move its
process directory to >dumps>saved_pdirs directory.  For more
information, see the "Notes on a dead process" section below.

Arguments:
   Either a DEADPROC_NAME or the -name control argument is required.
DEADPROC_NAME
   is the name of the dead process directory to be copied.  If
   DEADPROC_NAME is not an absolute pathname, the default path is
   >process_dir_dir>DEADPROC_NAME.  All components of the
   DEADPROC_NAME must be given.  Such names have the format:
      PERSON_ID.PROJECT_ID.f.TTY_NAME
   For example:  Swenson.Multics.f.d.h003

Control arguments:
-name DEADPROC_NAME, -nm DEADPROC_NAME
   specifies the name of the dead process to be copied.
-delete, -dl
   specifies that after the dead process is copied, the original
   directory in >process_dir_dir is deleted.  Status and modify access
   to the containing directory is needed.  If access is lacking, the
   user is queried about whether to continue copying.
-no_delete, -ndl
   specifies do not delete the dead process directory after copying is
   complete.  This is the default.

-owner, -ow
   specifies that the process owner's PERSON_ID.PROJECT_ID be given
   "s" access to the preserved directory.  This permits the owner to
   examine the directory using analyze_multics.  Otherwise, system
   administrator access is required to examine the preserved directory
   contents.

Notes on a dead process:  Any process whose operation is terminated
abnormally (e.g., termination by a "fatal process error") will be
treated as a "dead process" if the process was owned by
Person_ID.Project_ID and the following conditions are true at the time
of process termination:

 - The system administrator has previously set the "save_pdir"
   attribute in the System Administration Table (SAT) entry for
   Project_ID.

 - The Project_ID project administrator has previously set the
   "save_pdir" attribute in Person_ID's entry within Project_ID's
   Project Definition Table (PDT).
```

The Answering Service is notified whenever any process terminates; the notification includes a reason for termination. The Answering Service normally deletes the process directory owned by the terminating process. However, if termination was abnormal and the "save_pdir" attribute has been set as described above, then the Answering Service preserves the process directory by renaming it using a "dead process" name format:

```
rn >pdd>!BPlDbdwbBBBBBB PERSON_ID.PROJECT_ID.f.TTY_NAME
```

Information contained in the "dead process" name format plus the date-time-entry-modified on the renamed process directory serve to identify ownership of directory contents and date_time of its termination.

Notes on using copy_deadproc: After the Answering Service has renamed the process directory for the "dead process", a copy_deadproc command can save this dead process directory to a protected location where it may be examined using the analyze_multics (azm) tool. This command annotates contents of the process directory, and moves it to the preserving location with the new name:

```
mvd >pdd>PERSON_ID.PROJECT_ID.f.TTY_NAME >dumps>saved_pdirs>=.=.pdir
```

If a PERSON_ID.PROJECT_ID.pdir directory already exists at that location, it is first renamed to:

```
PERSON_ID.PROJECT_ID.N.pdir
```

where N=1 before moving the latest dead process directory. Similar renaming occurs if directories with higher values of N already exist in the saved_pdirs directory.

Access to the copied pdir is determined by the directory initial ACL on the >dumps>saved_pdirs directory. Access to segments within that directory does not change.

Several hardcore segments needed by analyze_multics are also added to the directory. Two such segments are: pdir_info and uid_hash_table which contain the annotations made by copy_deadproc. These annotations are used by analyze_multics when translating segment numbers used by the dead process. Any segments pointed to by links in the dead proc directory are also copied in place of those links.

Access required: Use of the copy_deadproc command requires access to the phcs_ gate, as well as sma access on the >dumps>saved_pdirs directory. A user can copy his own dead process if he has "sma" on the saved_pdirs directory and access to the >sl1>phcs_ gate.

Access to the hphcs_ gate is needed if the dead process was not owned by the person using the copy_deadproc command.

Access to the system_privilege_ gate is needed when copying a dead process which ran at a different authorization level than the process doing the copying. Such copying can only be done only by the system administrator or security administrator.

copy_liveproc.info

```
2022-12-05  copy_liveproc

Syntax as a command:
   copy_liveproc PROCESS_DIR_NAME PERSON_ID.PROJECT_ID {-ctl_args}

Function:  A "live process" is one which is still connected to the
user's terminal but has stopped responding to the user's inputs.  This
prevents the user from interrupting operation of the process to invoke
a debugger.

A copy_liveproc command issued from another process can copy the
process directory for the "live process" to a protected location where
it may be examined using the analyze_multics (azm) tool.  This command
moves the process directory to:
   >dumps>save_pdirs>PERSON_ID.PROJECT_ID.pdir

Arguments:
   Either PROCESS_DIR_NAME and PERSON_ID.PROJECT_ID arguments may be
   given, or -directory and -name control arguments may be used.
PROCESS_DIR_NAME
   is the name of the live process directory to be copied.  If just
   an entryname is given, it is assumed to be an entryname in >pdd
   (i.e., >process_dir_dir>!BPbDbdxBBBBBBB).
PERSON_ID.PROJECT_ID
   identifies the user owning the process.  This name is used
   when creating the saved process directory, ending with a .pdir
   standard suffix.

Control arguments:
-directory PROCESS_DIR_NAME, -dr PROCESS_DIR_NAME
   specifies the name of the process directory to be saved.
-name PERSON_ID.PROJECT_ID,
-nm  PERSON_ID.PROJECT_ID
   specifies the name of the user owning the process.
-owner, -ow
   specifies that access be set appropriately so the owner of saved
   directory can used analyze_multics himself to examine the hung
   process.

Notes on using the copy_liveproc command:  The print_apt_entry (pae)
command/AF can be used to obtain the TTY_NAME and PROCESS_DIR_NAME for
a non-responding "live process" to be copied.  See "Examples" below.

The copy_liveproc command moves segments from the original process
directory of the "live process" to the protected location.  Do not use
copy_liveproc to copy your current process, or to copy any process
essential to system operation (such as the Initializer process).
Moving its process directory will abnormally terminate that process as
its stack or other essential segments are deleted.
```

```
If the copy_liveproc command discovers that an existing
PERSON_ID.PROJECT_ID.pdir directory in >dumps>saved_pdirs, that
existing directory is renamed to:
  PERSON_ID.PROJECT_ID.N.pdir
where N=1 before copying the dead process directory.  Similar renaming
occurs if directories with high values of N already exist.

Access to the copied pdir is determined by the initial_acl for
directories set for the >dumps>saved_pdirs directory.  Access on
segments within the directory does not change.

Several hardcore segments needed by analyze_multics are also copied
into the directory.  Two segments are:, pdir_info and uid_hash_table.
These are used by analyze_multics when examining the live process.
Any segments pointed to by links in the dead proc directory are also
copied in place of those links.

Access required:  Use of this command requires access to the phcs_
gate, as well as sma access on the >dumps>saved_pdirs directory.  A
user can copy his own process if he has "sma" on the saved_pdirs
directory and access to the >sl1>phcs_ gate.

Access to the hphcs_ gate is needed if the dead process doesn't belong
to the process doing the copying.

Access to the system_privilege_ gate is needed when copying a dead
process which ran at a different authorization level than the process
doing the copying.  Such copying can only be done only by the system
administrator or security administrator.

Examples:
A system administrator might preserve a non-responsive Backup
project directory using the command:
  copy_liveproc [pae Backup -process_dir] Backup.SysDaemon
```

The following info segments describe azm requests that are new, or have been enhanced; or info segments that have been clarified without changes to the azm code.

aste.info

```
2022-11-06  aste

Syntax:  aste  { SEGNO | SEGNAME }  {-control_args}

Function: displays an Active Segment Table entry (ASTE), page table,
and trailer information.  These components may be displayed
individually or selected as a group.

Arguments:
SEGNO
   octal segment number of a segment in the selected process whose
   ASTE is to be displayed.
SEGNAME
   reference_name of a segment whose ASTE is to be displayed.  It must
   be the name of a stack_R segment (where R is a ring number), or a
   name defined in the Segment Loading Table (SLT).  For a list of such
   ref_names, use the azm request:  slte

Control arguments (selecting an aste):
-offset ASTE_OFFSET,
-ofs ASTE_OFFSET
   octal offset within sst_seg of the ASTE selected for display.
   Other data structures usually refer to this offset as the XXX.astep
   element.
-ptw PTW_OFFSET
   octal offset within SST, unpaged_page_tables or
   int_unpaged_page_tables of a PTW (Page Table Word).  The segment
   associated with the Page Table containing that PTW is selected
   for display.

Control arguments (displayed data):
   If no control arguments are given, the aste and page table
   information is displayed.
-aste
   displays active segment table information for the selected entry.
-page_table, -pt
   displays page table information for the selected segment.
-trailer, -tr
   displays trailer information describing the selected segment.
-brief, -bf
   displays everything but the page table.  This is
   equivalent to specifying -aste and -tr.
-long, -lg
   displays all components.  This is equivalent to specifying -aste,
   -pt and -tr.

-brief_header, -bfhe
   displays only the segment reference name or path name, segment
   number, and ASTE location.
```

```
Examples:
The following shows display of an ASTE, with its associated page table
and the trailer listing processes which know about this segment.

azm:  aste hcs_ -long

ASTE for hcs_ (Seg 161  in Proc 0), at SST|103144.

103144      0 000000000000 102460000000 037636103630 544045573066
103150      4 011001000040 414100300000 000000000000 000000000000
103154     10 000000000011 000000000000 011424011011 000000000102

uid = 544045573066, vtocx 40 on pvtx 1 (dska device 0)
max len 9, 9 recs used, 9 in core, cur len 9
Not updated as used.
Not updated as modified.
Par astep = 103630, Son = 0, brother = 102460
Trailer thread = 37636
     Quota (S D) = (0 9)

Flags: usedf hc_sdw aaon ehs nqsw fmch dnzp ddnp

PAGE      PTW       DEVADD    PD COPY   at SST|103160

   0  746760401145   20276                phu phm phu1
   1  747000401045   20277                phu phu1
   2  747460400045   20274                phu1
   3  747500400045   20275                phu1
   4  747520400045   20272                phu1
   5  747540400045   20273                phu1
   6  747560400045   20270                phu1
   7  747720400045   20271                phu1
  10  747740400045   20266                phu1
  11  377012000001    null
====
  17  377012000001    null

Known as:
317 in Proc 0 DBR 17171450 running   on cpu a  Initializer.SysDaemon.z
```

core_map_entry.info, cme.info

```
2022-07-11  core_map_entry, cme

Syntax:  cme ABS_ADDRs
         cme -control_arg

Function:  Displays information from a core map entry (CME).  Each
describes how a given page of real memory is currently used.

Arguments:
ABS_ADDRs
   absolute octal address of a location in real memory.  One or more
   addresses may be given.  The request displays the CME for the page
   containing each of those real memory addresses, and tries to
   identify which segment contains that page.

Control arguments:
   One of the following control arguments may be given instead of the
   ABS_ADDRs argument described above.
-index CME_INDEX ...,
-ix CME_INDEX ...
   zero-based decimal index in the core map array of one or more CMEs
   to be displayed.  This is also the zero-based memory-frame number
   (page number) in real memory.  Index values range from 0 to 16383
   in a system with 4 SCUs configured with largest memory blocks.
-offset CME_OFFSET ...,
-ofs CME_OFFSET ...
   octal offset in the core_map segment of one or more CMEs to be
   displayed.  Offsets range from 10o through 200010o in a system with
   4 SCUs configured with largest memory blocks.  Given the size of a
   CME (4 words), offsets must end with a 0 or 4 octal digit.

-ptw PTW_OFFSET
   octal offset of a Page Table Word (PTW) within the System Segment
   Table (SST), unpaged_page_tables or int_unpaged_page_tables.  If
   the PTW is active (not faulted), the request displays the CME
   describing the memory frame at the ptw.add absolute address
   location.
-aste ASTE_OFFSET
   octal offset of an Active Segment Table entry (ASTE) within the
   System Segment Table (SST), or of a PTW within a page table in
   unpaged_page_tables or int_unpaged_page_tables.  The request
   displays the CME for each in-memory PTW containing a page of the
   segment.

-segment {SEGNO | SEGNAME},
-seg {SEGNO | SEGNAME}
   octal segment number or name of a segment known to the selected
   dump process and listed in an Active Segment Table entry (ASTE)
   within the System Segment Table (SST), or with a page table in
   the unpaged_page_tables segment.  The request displays every
   CME for each PTW that is in-memory (not faulted).
```

```
-search CME_OFFSET,
-srh CME_OFFSET
   octal offset in the core_map segment of one or more CMEs to be
   displayed.  Searches all in-memory PTWs in page tables in the SST,
   unpaged_page_tables and int_unpaged_page_tables.  Any PTWs which
   reference the memory frame described by that CME are displayed as a
   CME reference.

List of core map entry fields:
   Each CME is displayed on a single line containing several fields.
   One of the first four entries below is given to identify the CME.
CME_IDX
   The CME is the Nth entry in the core_map array of CMEs, where N
   ranges from 0 to 16383.  That CME describes the Nth memory frame
   in real hardware memory.
CME_OFS
   The CME is located at octal offset CME_OFS within the core_map
   segment.  The core_map begins with an 8-word heading followed by an
   array of 4-word CMEs.  Thus, CME_OFS can range from 10o through
   200010o.

ABS_ADDR
   Given an octal absolute address in real hardware memory, the CME
   describing the memory frame covering that address is displayed.
PTW_OFS
   The CME describes the memory frame referenced by the PTW at octal
   offset PTW_OFS in the SST, unpaged_page_tables, or
   int_unpage_page_tables (segments holding page tables).

FWD, BACK
   Octal CME offset within the core_map of the next CME in a threaded
   list of CMEs, or the prior CME in such threaded list.  The main
   threaded list locates CMEs describing memory frames whose contents
   are managed by page_control.  Tracing such threads is simplified
   when displayed CMEs are identified by their CME_OFS value.
   Frames not managed by page_control usually contain segment pages
   permanently wired in real memory, or contiguous wired pages
   referenced by an unpaged SDW).  Such CMEs usually have 0 FWD and
   BACK offsets.

ASTEP
   Octal offset within the SST segment of an ASTE (Active Segment
   Table Entry) whose associated page table references the memory
   frame described by this CME.
PTWP
   Octal offset within the SST segment of the specific PTW (Page Table
   Word) that references the memory frame described by this CME.

DFWU
   Three switches describing the page currently stored in the memory
   frame.
     DF is displayed if the SDW used to locate the CME is inactive
        (has its sdw.df field set to cause a fault).
      W is displayed if the cme.abs_w bit is "1"b: the page content
        is abs-wired and its absolute address must not be changed.
```

```
      U is displayed if sdw.unpaged = "1"b: the segment is defined
        as a set of contiguous pages in real memory without using a
        page table.  An example is the fault_vector segment.

FRAME CONTENT
   A field summarizing content of the frame.  It usually provides an
   octal segment number|offset w/in that segment, followed by the
   reference name or pathname of that segment.  Some memory frames
   contain disk pages that are not defined by the storage system as a
   segment: for example, a physical volume bit map.  Other frames may
   be vacant (contain no data).  A frame may contain an active page
   whose content cannot be identified by data included in the dump
   image.

FLAGS
   Describes other elements of the cme data structure, if they have
   non-zero value.  These include:
     io=output
        Content of the memory frame is being written to disk.  If not
        present, content of disk might be queued to be read from disk,
        or no I/O operation may be queued.
     io_error
        The most recent input/output operation encountered an error.
     removing
        Memory frame is being removed by a reconfiguration operation.
     abs_wired
        Contents of memory frame must not be relocated to a different
        memory frame (different absolute address).

     abs_usable
        Memory frame eligible to hold an active page which must be
        abs_wired in memory (is not relocatable).
     notify_requested
        An I/O operation is pending.  When it completes, a waiting
        process needs to be notified of that completion event.
     phm_hedge
        pc$flush_core needs to write content of this memory frame to
        disk.

     scu(X)
        ID letter (A, B, C, or D) of the System Control Unit
        containing the real memory page described by this CME.
     pin_counter=N
        number of times to skip eviction of content from this memory
        frame.
     synch_entryp=PC_OFFSET
        Memory frame holds a page of a Data Management synchronized
        segment.  PC_OFFSET is a relative pointer within page_fault
        of entry point transferred to when synchronizing content of
        memory frame with its disk record.

Examples:
List contents of the first 256 pages of real memory, using the
index_set active request to generate the index values so only a single
heading line is displayed.
```

```
azm:  cme -ix [index_set 0 255]

 CME_INDX     FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT ------------  --- FLAGS ---
        0  777777 777777       0      0     U  Seg   4|0       fault_vector    scu(A)
        1  777777 777777       0      0     U  Seg  11|600     iom_mailbox     scu(A)
 ...
      222  777777 777777       0      0        Seg  57|0       idle_dsegs      scu(A)
      223  777777 777777       0      0        Seg  60|0       idle_pdses      scu(A)
      224  175620   1614       0      0        Vacant memory frame.            abs_usable  scu(A)
                                                 [series of 32 vacant frames]
      255    2000   2010       0      0        Vacant memory frame.            abs_usable  scu(A)
```

List contents of the first 256 pages of real memory, with CMEs identified by their octal offset within the core_map segment. Since the FWD and BACK values in CME output lines are octal offsets, identifying each line by its CME offset can aid in tracing threads of CME entries.

The core_map begins with an 8-word header, and CMEs are 4 words in length. index_set normally expects decimal input values, but will accept values with an octal radix indicator; and will generate octal output when -oct or -octal is given. The calc active request generates only decimal outputs.

```
azm:  cme -ofs [ixs 10o [calc 255*4] 4 -oct]

  CME_OFS     FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT ------------  --- FLAGS ---
       10  777777 777777       0      0     U  Seg   4|0       fault_vector    scu(A)
       14  777777 777777       0      0     U  Seg  11|600     iom_mailbox     scu(A)
  ...
     1600  777777 777777       0      0        Seg  57|0       idle_dsegs      scu(A)
     1604  777777 777777       0      0        Seg  60|0       idle_pdses      scu(A)
     1610  175620   1614       0      0        Vacant memory frame.            abs_usable  scu(A)
                                                  [series of 30 vacant frames]
     1774    1770   2000       0      0        Vacant memory frame.            abs_usable  scu(A)
```

Search for all PTWs that reference the memory frame at core_map offset 1604.

```
azm:  cme -search 1604

  CME_OFS     FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT ------------  --- FLAGS ---
     1604       0      0       0      0    W   Seg  60|0       idle_pdses      abs_wired  scu(A)
     1604       0      0       0      0    W   Seg 436|0       free_area_1     abs_wired  scu(A)
```

Search for all segments which include the breakpoint_page.

```
azm:  cme -seg breakpoint_page

CMEs  for      Segment without ASTE: breakpoint_page (Seg 14  in Proc 0),
with Page Table at unpaged_page_tables|36.

  CME_OFS     FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT -------------  --- FLAGS ---
      144       0      0       0      0    W   Seg  14|0       breakpoint_page  abs_wired  scu(A)
                                              CA-MAX 1777

azm:  cme -srh 144

19 PTWs reference memory frame described by core_map|144:

  CME_OFS  FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT ----------------  --- FLAGS ---
      144    0      0       0      0    W   Seg  14|0       breakpoint_page     abs_wired  scu(A)
      144    0      0       0      0    W   Seg  17|6000    ws_linkage          abs_wired  scu(A)
      144    0      0       0      0    W   Seg  34|6000    bound_interceptors  abs_wired  scu(A)
      144    0      0       0      0    W   Seg  35|12000   bound_io_wired      abs_wired  scu(A)
      144    0      0       0      0    W   Seg  37|4000    bound_iom_support   abs_wired  scu(A)
      144    0      0       0      0    W   Seg  42|50000   bound_page_control  abs_wired  scu(A)
      144    0      0       0      0    W   Seg  43|6000    bound_priv_1        abs_wired  scu(A)
      144    0      0       0      0    W   Seg  44|10000   bound_tc_priv       abs_wired  scu(A)
      144    0      0       0      0    W   Seg  45|2000    bound_unencacheable abs_wired  scu(A)
      144    0      0       0      0    W   Seg  46|30000   bound_wired_1       abs_wired  scu(A)
      144    0      0       0      0    W   Seg  55|2000    emergency_shutdown  abs_wired  scu(A)
      144    0      0       0      0    W   Seg  61|2000    init_processor      abs_wired  scu(A)
      144    0      0       0      0    W   Seg  75|2000    restart_fault       abs_wired  scu(A)
      144    0      0       0      0    W   Seg  76|2000    return_to_ring_0_   abs_wired  scu(A)
      144    0      0       0      0    W   Seg 100|2000    signaller           abs_wired  scu(A)
      144    0      0       0      0    W   Seg 406|2000    wi_linkage          abs_wired  scu(A)
      144    0      0       0      0    W   Seg  72|22000   prds                abs_wired  scu(A)
      144    0      0       0      0    W   Seg  41|114000  bound_library_wired abs_wired  scu(A)
      144    0      0       0      0    W   Seg  56|42000   error_table_        abs_wired  scu(A)
```

The first 3 pages of real memory holds contents of 3 special segments. Each is a contiguous region within part of a page, or within several adjacent pages of real memory. Each is defined by an SDW whose sdw.add references the starting point of the contiguous region, and whose sdw.bound gives its length (in 16-word blocks), and sdw.unpaged = "1"b meaning no page table is used to define these special segments.

The following example shows starting point and bounds of these three segments by displaying CME information for the first 4 pages of real memory, giving absolute addresses separated by 128 words. Because the first two pages of real memory hold contents of two different segments, this region cannot be managed by page control. Therefore, the segments must be wired in memory. The cme request must search for an SDW claiming ownership of each absolute address given as input.

```
azm:  cme [ixs 0 10000o 200o -oct]

 ABS_ADDR     FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT -------------  --- FLAGS ---
        0       0      0       0      0    WU  Seg   4|0       fault_vector     abs_wired  scu(A)
      200       0      0       0      0    WU  Seg   4|200     fault_vector     abs_wired  scu(A)
      400       0      0       0      0    WU  Seg   4|400     fault_vector     abs_wired  scu(A)
      600       0      0       0      0    W   Vacant memory.                   abs_wired  scu(A)
     1000       0      0       0      0    W   Vacant memory.                   abs_wired  scu(A)
     1200       0      0       0      0    WU  Seg  11|0       iom_mailbox      abs_wired  scu(A)
     1400       0      0       0      0    WU  Seg  11|200     iom_mailbox      abs_wired  scu(A)
     1600       0      0       0      0    WU  Seg  11|400     iom_mailbox      abs_wired  scu(A)
     2000       0      0       0      0    WU  Seg  11|600     iom_mailbox      abs_wired  scu(A)
     2200       0      0       0      0    WU  Seg  11|1000    iom_mailbox      abs_wired  scu(A)
     2400       0      0       0      0    WU  Seg  11|1200    iom_mailbox      abs_wired  scu(A)
     2600       0      0       0      0    WU  Seg  11|1400    iom_mailbox      abs_wired  scu(A)
     3000       0      0       0      0    WU  Seg  11|1600    iom_mailbox      abs_wired  scu(A)
     3200       0      0       0      0    WU  Seg  11|2000    iom_mailbox      abs_wired  scu(A)
     3400       0      0       0      0    WU  Seg   3|0       dn355_mailbox    abs_wired  scu(A)
     3600       0      0       0      0    WU  Seg   3|200     dn355_mailbox    abs_wired  scu(A)
     4000       0      0       0      0    WU  Seg   3|400     dn355_mailbox    abs_wired  scu(A)
     4200       0      0       0      0    WU  Seg   3|600     dn355_mailbox    abs_wired  scu(A)
     4400       0      0       0      0    WU  Seg   3|1000    dn355_mailbox    abs_wired  scu(A)
     4600       0      0       0      0    WU  Seg   3|1200    dn355_mailbox    abs_wired  scu(A)
     5000       0      0       0      0    WU  Seg   3|1400    dn355_mailbox    abs_wired  scu(A)
     5200       0      0       0      0    WU  Seg   3|1600    dn355_mailbox    abs_wired  scu(A)
     5400       0      0       0      0    WU  Seg   3|2000    dn355_mailbox    abs_wired  scu(A)
     5600       0      0       0      0    WU  Seg   3|2200    dn355_mailbox    abs_wired  scu(A)
     6000       0      0       0      0    WU  Seg   3|2400    dn355_mailbox    abs_wired  scu(A)
     6200       0      0       0      0    WU  Seg   3|2600    dn355_mailbox    abs_wired  scu(A)
     6400       0      0       0      0    W   Vacant memory.                   abs_wired  scu(A)
     6600       0      0       0      0    W   Vacant memory.                   abs_wired  scu(A)
     7000       0      0       0      0    W   Vacant memory.                   abs_wired  scu(A)
     7200       0      0       0      0    W   Vacant memory.                   abs_wired  scu(A)
     7400       0      0       0      0    W   Vacant memory.                   abs_wired  scu(A)
     7600       0      0       0      0    W   Vacant memory.                   abs_wired  scu(A)
    10000       0      0       0      0    W   Seg   1|0       bos_toehold      abs_wired  scu(A)
```

display.info, d.info

```
1984-11-19  display, d

Syntax:  d VIRTUAL-ADDR {EXP} {RANGE} {-ctl_args}

Syntax as an active request:
        [d VIRTUAL-ADDR {EXP} {RANGE} {-ctl_args}]

Function: displays a selected portion of a segment in the FDUMP.
As an active request, the value(s) that would have been displayed are
returned as the active request result.

Arguments:
VIRTUAL-ADDR
   specifies the initial offset of the virtual address space to be
   dumped.  May be a segment number, name, or symbolic address (e.g.,
   64, prds, prds$am_data).    Do a 'help virtual_address' for more
   detailed information on acceptable virtual-address constructs.
EXP
   is an expression, which is either an octal value or a VIRTUAL-ADDR
   construct yielding an octal value.  This value can be positive or
   negative, specified by the plus or minus sign.
RANGE
   specifies the octal number of words to be dumped.  (default: dump
   one word; or if an ITS pointer format is specified, dump two words)

Control arguments (mode specification):
-character, -ch, -ascii
   displays the selected number of stored words as ascii characters.
   Characters that cannot be printed are represented as periods.  As
   an active request, returns the character representation of data
   stored at the requested address.
-instruction, -inst
   displays the selected number of stored words as instructions.
   Usage as an active request is not allowed.
-octal, -oc
   displays the selected number of stored words in octal.  (default)
   When used as an active request, returns data stored at the
   requested address in octal format.

-ptr, -p
   displays the selected number of word pairs as ITS pointers.  When
   used as an active request, returns the stored data in the form
   SEGNO|OFFSET where the components are octal numbers.
-ptrx, -px
   displays the selected number of word pairs as pointers and expands
   the segno|offset to a segment name.  Usage as an active request is
   not allowed.

-pptr, -pp
   displays each of the selected number of words as a packed-pointer.
   When used as an active request, returns the stored data in the form
   SEGNO|OFFSET where the component are octal numbers.
-pptrx, -ppx
```

```
   displays the selected number of words as packed-pointers and expands
   the segno|offset to a segment name.  Usage as an active request is
   not allowed

Control arguments (structure display):
-as STRUCTURE_NAME
   displays stored data as a PL/I structure defined by STRUCTURE_NAME.
   This is usually a structure name from a system-defined hardcore
   include file.  For a list of supported STRUCTURE_NAMEs, type:
      help structure_names

   The address given in the display request is taken as the address of
   the beginning of the structure.  If the whole structure is being
   displayed, that is the address where display begins.  If only
   certain elements are being displayed, that is the starting address
   to which the computed offset of the named element is applied.

   The structure reference following -as must be a single string,
   containing no spaces, and follows the syntax described below.  The
   single string is used to specify structure elements, array indexes,
   and substring matching.  Usage as an active request is not allowed.

-long, -lg
   displays each element of the structure on a separate line.  This
   control argument is only implemented with -as.

Notes on structure names:
The structure reference is made up of two parts: a structure element
reference, and an optional set of match strings.  If no match strings
are supplied, all structure elements are displayed.

The structure element reference syntax consists of one or more element
names, separated by periods, and may contain subscripts following some
of these element names.  The first name in a structure element
reference must be a level one structure reference; partially qualified
top level references are not permitted.  Intermediate levels of
qualification may be omitted as long as there is no ambiguity.

The following examples show various kinds of structure references
which may be given when displaying a location as a structure.

pvt
   The whole structure "pvt".
pvt.n_entries
   The single element "n_entries" in the structure "pvt".

All subscripts must be supplied as decimal integers.  The subscripts
may be cross-section references such as "(1:4)" to reference elements
one through four.  Asterisk bounds may not be used: if a cross-section
is desired, its upper and lower bounds must be given as decimal
constants.  If an element has more subscripts than are supplied, the
complete cross-section is printed for the remaining subscripts.

The parenthesis normally surrounding an array index specification
would trigger an iteration when given in the request line.  To avoid
```

```
quoting, subscripts may be surrounded by braces instead of parentheses.

sst.space
   all elements of "sst.space".
sst.space{3}
   element three of the "sst.space" array.
sst.space{2:4}
   cross-section of elements two, three, and four of the "sst.space".

sst.level{1}
   both elements of the "level" array-of-structures.
sst.level{1}.ausedp, sst.level.ausedp{1}
   the single element "ausedp" of the "level" array of structures.
   As in PL/I, the index may follow the array element, or any
   sub-element of that array.

In order to specify that only certain elements be displayed (such as
all those with names containing the string "time"), a set of match
strings may be given after the structure element reference.  Each match
string begins with a slash and is followed by the string itself.  The
final match string may be followed by a slash, but this is not
required.  If match strings are specified, any element which matches at
least one string will be displayed.

sst/time/,  sst/time
   Any elements in the structure "sst" containing the string "time".
   Note that the final slash is optional.
sst/time/meter/
   Any elements in the structure "sst" containing either the string
   "time" or the string "meter".

Notes on structure output format:
The default output format is a compressed form, which places as many
values on a line as will fit within the line length.  The -long control
argument places one value on a line.  The short form, additionally,
collects all bit(1) flags and displays them at the end of the display
for each substructure or array element, in two groups: one listing all
the flags which were on ("1"b), and one for all the ones which were off
("0"b).

All PL/I datatypes are displayed in the same representations used by
probe.  Additionally, the following special formats are used:

 1) Bit strings are displayed as octal, if the length is divisible by
    three, in hex if divisible by four and not three, and as bit
    strings otherwise.
 2) Character strings are displayed as a string concatenated with a
    repeated constant, if the string is padded on the right with more
    than sixteen nulls, spaces, or octal 777 characters.
 3) Large precision (> 51) fixed binary values are also displayed as
    clock readings if the stored value represents a clock reading
    within ten years of the present.
```

```
Examples:
d 75|560 2
   displays the two words in seg number 75 starting at octal offset
   560.
d pds|560 2
   displays the two words in the segment named pds starting at octal
   offset 560.

d pds$trace
   displays one word in the pds segment beginning at the offset
   specified by $trace.

display 244|260 +20 4
   displays four words of segment number 244 starting at octal offset
   300.

d sp 20
   displays 20 octal words starting with the segment offset defined in
   the azm internal temporary pointer.  For information about temporary
   pointers, type:  help set

d sst$cmp,* +sst$cmesize sst$strsize
   causes the word at sst$cmp to be used as an indirect word (or an
   indirect pointer if the resultant address pair has ITS
   modification) to develop the starting virtual address.  The value
   stored at sst$cmesize will then be added to the starting offset to
   obtain a 'final' starting address.  The range, or number of words
   to be displayed, is specified by the value stored at sst$strsize.

d sst|2 -as apte
   displays the APTE entry at the given offset in the SST as it is
   defined by apte.incl.pl1
```

display_absolute.info, da.info

```
2022-07-22  display_absolute, da

Syntax:  da  ABS-ADDR {range} {-ctl_args}

Syntax as an active request:
        [da  ABS-ADDR {range} {-ctl_args}]

Function: dumps an absolute memory address space in the FDUMP.
As an active request, the value(s) that would have been displayed are
returned as the active request result.

Arguments:
ABS-ADDR
   is the starting absolute memory address, in octal.
RANGE
   specifies the octal number of words to be dumped.  (default: dump
   one word; or if an ITS pointer format is specified, dump two words)

Control arguments (mode specification):
-character, -ch, -ascii
   displays the selected number of stored words as ascii characters.
   Characters that cannot be printed are represented as periods.  As
   an active request, returns the character representation of data
   stored at the requested address.
-instruction, -inst
   displays the selected number of stored words as instructions.
   Usage as an active request is not allowed.
-octal, -oc
   displays the selected number of stored words in octal.  (default)
   When used as an active request, returns data stored at the
   requested address in octal format.

-ptr, -p
   displays the selected number of word pairs as ITS pointers.  When
   used as an active request, returns the stored data in the form
   SEGNO|OFFSET where the components are octal numbers.
-ptrx, -px
   displays the selected number of word pairs as pointers and expands
   the segno|offset to a segment name.  Usage as an active request is
   not allowed.

-pptr, -pp
   displays each of the selected number of words as a packed-pointer.
   When used as an active request, returns the stored data in the form
   SEGNO|OFFSET where the component are octal numbers.
-pptrx, -ppx
   displays the selected number of words as packed-pointers and expands
   the segno|offset to a segment name.  Usage as an active request is
   not allowed
```

list_processes.info, lsp.info

```
1984-01-19  list_processes, lsp

Syntax:
  lsp {proc_indicator} {-control_arg}

Syntax as an active request:
 [lsp {proc_indicator} {-control_arg}]

Function:
Lists all known processes in the selected FDUMP.

As an active request, returns the process_ids meeting the control
argument criteria.  If no process matches the criteria, an empty
string is returned.  If -count is specified, only the total is
returned.

Arguments:
proc_indicator
   specifies an individual processes.  It can take one of three
   forms:
   - The decimal index (starting at zero) of a process in the FDUMP.
   - The octal apte offset of the process.
   - The octal process_id of the process.

Control arguments:
-all, -a
   Lists all processes in the FDUMP (Default).
-blocked, -blk
   Lists processes marked as blocked.
-count, -ct
   specified by the control_args.  With -all, it gives the
   counts of each process state including the overall total.
-current, -cur
    Lists the current process.
-page_tbl_lock, -ptl
    Lists processes marked as page table locking.
-ready, -rdy
    Lists processes marked as ready.

-run
   Lists processes marked as running.
-stopped, -stop
   Lists processes marked as stopped.
-wait
   Lists processes marked as waiting.

Examples:

The following request displays the SDW for DSEG for all processes in
the FDUMP.

  do "select_process &1; sdw 0" ([list_processes]
```

page_control_check.info, pcc.info

```
2022-11-17  page_control_check, pcc

Syntax:  pcc {-control_args}

Function:  Reports any inconsistency between page control data items
(ASTE location, PTW offset within SST, core_map offsets to those
items, core_map disk addresses, etc.).  Checks for inconsistency are
performed using the pc_check_tables_ subroutine with modes:

  report_error_counts:    total inconsistencies found
    report_statistics:    count of each type of inconsistency
        report_errors:    description of individual inconsistencies
         report_state:    description of interrupted operations

When a crash occurs, pc_check_tables_ is called by Emergency Shutdown
(ESD) to repair reported inconsistencies (if possible).  Since ESD
runs after the dump is saved, none of these corrective repairs are
reflected in the dump image.  The pcc request can therefore describe
those earlier problems, perhaps adding more details.  No data in the
dump is changed by running this request.

Control arguments (reported data):
-long, -lg
   display content of inconsistent data items and run diagnostic
   requests to display affected data items.
-brief, -bf
   display location of inconsistent items with brief information
   naming affected data items (when possible).  Diagnostic requests
   are recommended for later use by the user.  (default)

Examples:

azm:  sld >old_dumps>25
Could not find apte for process_idx 0
     dbr = 17543650
This is an early dump.

  ERF 25  in directory >old_dumps dumped at 09/19/21  2120.7 pst Sun.
Proc   0 DBR  17543650 empty     last on cpu    Initializer.SysDaemon.z

azm:  pcc
_____________________________________
Bad counter for ASTE.
 --> ASTE at SST|1240
     ASTE for config_deck (Seg 2  in Proc 0), at SST|1240.
 np = 0, should be 4
             DIAGNOSTIC REQUEST(s): aste -offset 1240;


_____________________________________
CME ptwp ^= ptw address.
 --> PTW at SST|1275
     ASTE for wi_linkage (Seg 406  in Proc 0), at SST|1260.
 --> CME at core_map|144    breakpoint_page
     CME for breakpoint_page (Seg 14|0  in Proc 0), at core_map|144
             EXPECTED INCONSISTENCY:  the breakpoint_page ends many hardcore segs.
              cme.ptwp intentionally set to "000000"b3 cannot match PTW offset.
```

```
_____________________________________
CME ptwp ^= ptw address.
 --> PTW at SST|16165
     ASTE for prds (Seg 72  in Proc 0), at SST|16140.
 --> CME at core_map|144    breakpoint_page
     CME for breakpoint_page (Seg 14|0  in Proc 0), at core_map|144
             EXPECTED INCONSISTENCY:  the breakpoint_page ends many hardcore segs.
              cme.ptwp intentionally set to "000000"b3 cannot match PTW offset.
_____________________________________
CME ptwp ^= ptw address.
 --> PTW at SST|16370
     ASTE for free_area_1 (Seg 436  in Proc 0), at SST|16354.
 --> CME at core_map|1604

2 PTWs reference memory frame described by core_map|1604:

  CME_OFS  FWD   BACK   ASTEP   PTWP  DFWU  --- FRAME CONTENT ----------------  --- FLAGS ---
     1604    0      0       0      0    W   Seg  60|0       idle_pdses          abs_wired  scu(A)
     1604    0      0       0      0    W   Seg 436|0       free_area_1         abs_wired  scu(A)

             DIAGNOSTIC REQUEST(s): aste -ptw 16370 -pt; cme -ptw 16370; cme -search 1604;
_____________________________________
Bad counter for ASTE.
 --> ASTE at SST|16354
     ASTE for free_area_1 (Seg 436  in Proc 0), at SST|16354.
 records = 9, should be 8
 np = 9, should be 8
             DIAGNOSTIC REQUEST(s): aste -offset 16354;
_____________________________________
CME ptwp ^= ptw address.
 --> PTW at SST|26512
     ASTE for bound_library_wired_ (Seg 41  in Proc 0), at SST|26430.
 --> CME at core_map|144    breakpoint_page
     CME for breakpoint_page (Seg 14|0  in Proc 0), at core_map|144
             EXPECTED INCONSISTENCY:  the breakpoint_page ends many hardcore segs.
              cme.ptwp intentionally set to "000000"b3 cannot match PTW offset.
_____________________________________
CME ptwp ^= ptw address.
 --> PTW at SST|26715
     ASTE for error_table_ (Seg 56  in Proc 0), at SST|26660.
 --> CME at core_map|144    breakpoint_page
     CME for breakpoint_page (Seg 14|0  in Proc 0), at core_map|144
             EXPECTED INCONSISTENCY:  the breakpoint_page ends many hardcore segs.
              cme.ptwp intentionally set to "000000"b3 cannot match PTW offset.
_____________________________________
Bad counter for ASTE.
 --> ASTE at SST|35300
     ASTE for disk_mst_seg (Seg 434  in Proc 0), at SST|35300.
 csl = 15, should be 31
 records = 0, should be 31
             DIAGNOSTIC REQUEST(s): aste -offset 35300;
_____________________________________
Statistics:
         3 Bad aste counter
         5 Bad cme ptwp
```

sdw.info

```
2022-11-16  sdw

Syntax:  sdw {segno/name} {segno/name}

Function:  Displays the SDW's in the current process's DSEG.

Arguments:
segno/name
   is the segment number or name of interest.  The first is the
   starting segment number and the second is the ending segment
   number.  If only one is given then only one is displayed if none
   are given then all are displayed.

Notes on sdw display:
The sdw request displays the segment number, name, memory address,
ring brackets, the maximum computed address, the entry bound address
and a bit string REWPUGCDF.

List of display definitions:
ADDRESS
   is the base address of the segment or segment page table.
RNGS
   the ring brackets of the segment.
CA-MAX
   the highest computed address that may be used in referencing the
   segment without causing an out_of_segment_bounds fault.
EBOUND
   is the entry bound or call limiter. Any external call to this
   segment must be to an offset less than the EBOUND if the entry
   bound switch (G) is off.
SEGNO
   segment number.

SEGMENT-NAME
   segment name.

REWPUGCDF
   The letter will show in the sdw display of the segment if the bit
   is on.  The REWPUGCDF string is broken down as follows:
     R  is the read permission bit
     E  is the execute permission bit
     W  is the write permission bit
     P  is the privileged bit
     U  is the unpaged bit, segment is unpaged (has no page table)
        if this bit is on
     G  is the gate indicator or entry bound bit. If off, the entry
        bound is checked by hardware
     C  is the cache enable switch.
     DF is the directed fault bit. If on, the necessary page of the
        segment is in memory.
```

Examples:

```
 ADDRESS RNGS  CA-MAX REWPUGCDF EBOUND SEGNO SEGMENT-NAME
 6262154  000  177777 R W  G DF      0   200 str_seg
       0  000   37777 R W  G         0   300 >udd>Multics>GDixon
    3400  000    2777 R W UG DF      0     3 dn355_mailbox
```

slt_entry.info, slte.info

```
2022-11-22  slt_entry, slte

Syntax:
   slte  {SEGNO | SEGNAME}  {SEGNO | SEGNAME}  {-control_args}

Syntax as an active request:
  [slte  {SEGNO | SEGNAME}  {SEGNO | SEGNAME}  {-control_args}]

Function:  Displays one or more Segment Loading Table entries (SLTEs).
As an active request, returns SEGNO of entries that match attributes
given with the -attributes control argument.

Arguments:
   If no SEGNO or SEGNAME is given, all SLT entries are displayed.
   If one SEGNO and one SEGNAME are given, the SEGNAME is mapped to a
   segment number which is coupled with the given SEGNO to define a
   range of SLTEs to display.
SEGNO
   segment number (octal) whose SLTE is to be displayed.  If two
   numbers are given, they start/end the range of segment numbers
   whose SLTEs are displayed.
SEGNAME
   name of a segment whose SLTE is to be displayed.  If two names
   are given, their respective segment numbers identify start/end of a
   range of segments whose SLTEs are displayed.

Control arguments:
-header, -he
   displays a header before the first SLTE item describing segment
   range for supervisor segments and initializing segments.
   (default if no arguments or control_args are given)
-no_header, -nhe
   omits the header.  (default if any arguments or control_args are
   given.

-all_names
   display all names associated with selected SLTE items. (default)
-primary, -pri
   display only the first name associated with each SLTE item.

-attribute ATTRIBUTEs,
-attr ATTRIBUTEs
   displays SLTE items having one or more of the given ATTRIBUTE
   keywords.  All arguments following -attributes are treated as
   ATTRIBUTE names.  See the "List of attributes" section.

List of attributes:
   operands of the -attribute control_arg may be one or more of the
   ATTRIBUTE keywords below.  Use ^ATTRIBUTE to select items that
   do not have the given ATTRIBUTE keyword.
privileged, priv
   displays items granted access to execute privileged instructions.
```

```
encacheable, cache
   displays items whose data may be read via the hardware cache.
breakpointable, break
   displays items ending with a probe breakpoint_page.  The BCE probe
   command can set breakpoints in such segments.
wired
   displays items which are wired into real memory frames.

^paged
   displays items which are unpaged: segments defined without using an
   ASTE/PageTable combination to define segment contents.  ^paged is
   displayed for such items.  Despite this attribute name, some of
   these segments do have a page table in the unpaged_page_tables
   segment.  Since most items do have ASTE entries, the paged
   attribute is displayed only if the user is selecting items that are
   paged.
firmware
   displays items designated to contain firmware data files.
gate
   displays items with ring brackets designating them as gates leading
   to inner-ring services.

abs_seg, abs
   display items which have no content of their own, but are used by
   supervisor code to overlay contents of other segments.
init_seg, init
   display items used only during system initialization.    Such items
   are not displayable after Multics is fully bootloaded.
temp_seg, temp
   display items used as temporary segments.
per_process, proc
   display items which appear in every process but have contents that
   differs for each process.
branch
   display items instantiated as segments in the >system_library_1
   (>sl1) directory.

Notes on the output:
Line 1   primary name by which the segment is known;
         segment number assigned by the SLTE;
         ring brackets assigned by the SLTE;
         access mode bits (read, execute, write) for Initializer;
         other attributes:
           privileged  encacheable  gate  wired  ^paged

Line 2   next name on segment (unless -primary given);
         initialization type attributes:
           init_seg  temp_seg  per_process  firmware  breakpointable
         (line present only if any of the type attributes are set)
```

```
Line 3   next name on segment (unless -primary given);
         segment length:
           wired words or count of pages;
           max length if any is specified in the SLTE;  or
         abs_seg: a utility segment used to temporarily overlay
           contiguous memory pages having no fixed length.

Line 4   next name on segment (unless -primary given);
         pathname of segment
         (line present only if seg has a branch in file system)

Line N   other names on segment (if any) unless -primary was given.

Notes on changing contents of the slte database:
During bootload operations, data is loaded from collections of the
Multics System Tape (MST), and the in-memory database of the
Segment Loading Table (SLT) grows and shrinks as new segments are
added and/or initialization-related segments are no longer needed and
are discarded.  For example:

   - While collection 1 is running, the int_unpaged_page_tables
     segment contains page tables for initialization-related segments
     at/above segno 400 in the Initialization.SysDaemon.z (proc 0)
     address space.

   - Once collection 1 running ends, these initialization segments
     have been deleted, along with int_unpaged_page_tables and the
     slte entries above segno 225.  The int_unpaged_page_tables
     segment no longer exists.

The slte request provides only information known in the SLTE database
when the selected dump was taken.

When the system is fully booted and running after the entire Multics
System Tape has been loaded, no initialization-related segments remain
in Multics memory, or in the SLTE database.

Examples:
The slte request may give starting and ending names for a range of
segments to be displayed.

 azm:  slte dseg ws_linkage

 dseg                    0  (0, 0, 0) read write wired
                                      per_process
                                      wired length: 1024 words; paged length: 8 pages;

 bos_toehold             1  (0, 0, 0) read write wired ^paged
                                      wired length: 1024 words;

 config_deck             2  (0, 5, 5) read write
                                      abs_seg -- no storage allocated.
                                      path: >system_library_1>config_deck

 dn355_mailbox           3  (0, 0, 0) read write wired ^paged
                                      wired length: 2048 words; max length: 1 pages;
```

```
 fault_vector            4  (0, 0, 0) read write wired ^paged
                                      wired length: 1024 words;

 flagbox                 5  (0, 0, 0) read write wired ^paged
                                      wired length: 1024 words;

 name_table              6  (0, 5, 5) read write
                                      paged length: 12 pages;
                                      path: >system_library_1>name_table

 slt                     7  (0, 5, 5) read write
                                      paged length: 2 pages;
                                      path: >system_library_1>slt

 toehold_data           10  (0, 0, 0) read write wired ^paged
                                      wired length: 2048 words;

 iom_mailbox            11  (0, 0, 0) read write wired ^paged
                                      wired length: 2048 words; max length: 1 pages;

 unpaged_page_tables    12  (0, 0, 0) read write wired ^paged
                                      wired length: 1024 words;

 toehold                13  (0, 0, 0) read write wired ^paged
                                      wired length: 2048 words;

 breakpoint_page        14  (0, 0, 0) read write wired ^paged
                                      wired length: 1024 words;

 lot                    15  (0, 0, 0) read write wired ^paged
                                      wired length: 1024 words;

 as_linkage             16  (0, 0, 0) read execute write
      active_sup_linkage               breakpointable
                                                 paged length: 17 pages;

 ws_linkage             17  (0, 0, 0) read execute write wired ^paged
      wired_sup_linkage                breakpointable
                                       wired length: 3072 words; max length: 1 pages;


If a single segment name or octal segment number is given, only that
SLTE is displayed.

 azm:  slte 113

 bound_date_time_      113  (0, 5, 5) read execute encacheable
      convert_date_to_binary_          breakpointable
      time_defaults_                   paged length: 30 pages;
      time_data_                       path: >system_library_1>bound_date_time_
      time_info_              date_time_           decimal_date_time_      request_id_
      encode_clock_value_     decode_clock_value_  cv_fstime_              date_name_
      set_system_time_zone_

If no arguments are given or -header is given, information from the
SLT header is displayed followed by all entries in the Segment Loading
Table.  This header gives first and last segment numbers assigned for
segments in the Multics supervisor.

In a dump taken early in the bootload process (aka, an "early dump"),
segments used only while booting may be given higher segment numbers
```

```
and moved to higher memory locations.  Such segment numbers are shown
in the header as Initializing Segments.  Each segment in this range
has the "init seg" attribute.

 azm:  slte

                          Supervisor Segments      Initializing Segments
                            First:    0              First:  400
                             Last:  115               Last:  465

 dseg                      0  (0, 0, 0) read write wired
                                        per_process
                                        wired length: 1024 words; paged length: 8 pages;

 bos_toehold               1  (0, 0, 0) read write wired ^paged
                                        wired length: 1024 words;

 config_deck               2  (0, 5, 5) read write
                                        abs_seg -- no storage allocated.
                                        path: >system_library_1>config_deck
 ...

 bound_qedx_             115  (0, 5, 5) read execute encacheable
      search_file_                      breakpointable
      qedx_                             paged length: 15 pages;
      check_entryname_                  path: >system_library_1>bound_qedx_ß
      qedx                    qx

 bound_bootload_0        400  (0, 0, 0) read execute write privileged
                                        init_seg
                                        abs_seg -- no storage allocated.

 physical_record_buffer  401  (0, 0, 0) read write
                                        init_seg
                                        paged length: 3 pages;

 abs_seg0                402  (0, 0, 0) read execute write privileged ^paged
                                        init_seg
                                        abs_seg -- no storage allocated.
```

Version History

Date	Revision	Author	Comment
2022-12-27	1.0	Gary Dixon	Initial draft of the MCR.
2022-12-29	1.1	Gary Dixon	Fix typos in original draft.
2022-12-30	1.2	Eric Swenson, Gary Dixon	Address preview comments from the auditor. Significant terminology revisions improved readability.
2022-12-31	1.3	Gary Dixon	More terminology revisions.
2022-12-31	1.4	Eric Swenson, Gary Dixon	More terminology revisions.
2022-12-31	1.5	Gary Dixon	Correct typos.
2023-01-01	1.6	Gary Dixon	Fix incorrect URLs linking to Multics Tickets.
2023-01-03	1.7	Gary Dixon	Correct typos found during preview of the MCR.
2023-01-03	1.8	Eric Swenson, Gary Dixon	Simplify/correct wording.