Subject: MCR-10035, Enhance probe's format_pointer_ Subroutine

Author: Gary Dixon

Date: February 23, 2017

The probe (pb) command calls interpret_ptr_ in specific situations to display more meaningful information about pointers. See probe_print_handlers_.pl1 and probe_display_data_.pl1 for such existing uses.

However, probe's general pointer display routine, format_pointer_.pl1 does not make use of interpret_ptr_. format_pointer_ is also called by display_data_, which is used by analyze_multics and dump_segment commands. Thus, improved pointer descriptions from format_pointer_ could enhance probe, and many other tools.

Currently, the most neglected case is mapping a pointer into a bound segment to its segname$entryname symbolic form. Such mapping of numbers to named entrypoints can greatly assist in debug and investigation efforts.

For example, dump_segment (ds) calls format_pointer_ to display information about pointer elements in a structure. However, the information returned by format_pointer_ only identifies the segment referenced by the pointer; it gives no further information.

```
dump_segment 234 -as stack_header.sct_ptr
000000
  sct_ptr = 234|1000   [pd]>stack_4
r 05:38 0.150 79

dump_segment 234 1000 -as "sct_ptr(21)" -in [hd]>lib>gdStructures
001000
  sct_ptr (21) = 243|3734   >sss>bound_process_init_
r 05:34 0.176 55


ds w>tests>hello 114 -as pt.pointer -in [hd]>lib>gdStructures
000114
  pointer = 777766(0),f2 [invalid]
r 07:02 0.019 2
```

- Reference Multics Ticket: http://multics-trac.swenson.org/ticket/47

Proposed Changes

Change >ldd>sss>source>bound_probe_.1.s::format_pointer_.pl1 as follows:

- In the edit_its_pointer internal procedure which validates the incoming aligned pointer, accept a pointer with a FT2 (Fault Tag 2) modifier, in addition to ITS and ITP modifiers currently allowed.
    - For a pointer with FT2 modifier, call interpret_ptr_ to evaluate the potential unsnapped link.

- In the edit_packed_pointer internal procedure which validates the incoming unaligned pointer, treat a packed pointer whose storage word is all zero bits as an invalid pointer. It is currently treated as a reference to 0|0 (segno 0 is the Descriptor Segment, dseg).
- In the add_segment_name internal procedure, try calling interpret_ptr_ first to evaluate the pointer. If it returns segment_name and entryname information, add that information to the pointer representation string, and return; otherwise, let the existing code in add_segment_name attempt an evaluation. It will return pathname of the segment pointed to, applying alias names, etc.
- When format_pointer_ calls its internal add_segment_name procedure, pass its incoming ITS pointer parameter directly, rather than using a copy. This can be important when using interpret_ptr_ to evaluate the pointer.
- Include the object_link_dcls.incl.pl1 file, which defines the FAULT_TAG_2 modifier constant.
- Include the interpret_ptr_struc.incl.pl1 file, which defines the structure returned by the interpret_ptr_ subroutine.

With these changes, more descriptive information is returned by interpret_ptr_, as shown in the examples below.

```
dump_segment 234 1000 -as "sct_ptr(21)" -in [hd]>lib>gdStructures
001000
  sct_ptr (21) = 243|3734   process_overseer_$mme2_fault_handler_


ds w>tests>hello 114 -as pt.pointer -in [hd]>lib>gdStructures
000114
  pointer = -12|27  ioa_$nnl [unsnapped link]
```

In >ldd>sss>source>bound_probe_.1.s::display_data_.pl1, dis-entangle references to parameters in code paths shared by its two entrypoints: display_data_$for_probe and $for_azm.

```
display_data_$for_azm:
  entry (P_iocb_ptr, P_display_format, P_match_names, P_match_name_count,
          P_amu_info_ptr,
          P_data_ptr, P_data_size, P_start_ptr, P_symbol_ptr,
          P_subscripts, P_n_subscripts, P_code);


display_data_$for_probe:
  entry (P_probe_info_ptr, P_reference, P_code);
```

These two entrypoints accept differently-named parameters, having differing data types, passed in differing order. This includes the following changes:

- Move global.probe_options.probe_sw and global.probe_options.language_type to the top_level of the global structure (e.g., global.probe_sw). These two fields are shared by both entrypoints, and should reside in the shared portion of the global structure.

- Move calls to set_globals_for_azm and set_globals_for_probe to be the first calls made by their respective entrypoints. Setting the global structure first will help prevent entangled parameter references in any future changes.
- Eliminate the azm_entry parameter from the common_c_setup internal procedure. Since that internal procedure is now called after the set_globals_for_xxx routines, it should reference global.probe_sw to determine which entrypoint was called.
- In common_c_setup, move code that references the $for_probe(…, Preference) parameter structure to fall after the test for C_lang_type. The $for_azm entrypoint does not have a Preference parameter; but it always sets global.language_type to PL1_lang_type, and will therefore no longer attempt to access the missing Preference parameter.
- Eliminate references to probe_info.language_type and .output_switch throughout display_data_ code. They are replaced by references to global.language_type and .output_switch (or eliminated when superfluous). These global elements are set in the setup_for_XXX routines.
- Wrap code dealing with C array dimensions with a protective if global.probe_sw then… clause.
- When calling the print_symbol internal procedure, make separate calls in $for_probe invocations (passing probe-specific parameters) versus $for_azm invocations (which pass azm-type parameters). [It surprises me that these calls ever worked for the $for_azm entrypoint; they essentially passed garbage data.]
- Fix spelling errors in several introductory comments.

Documentation

Both display_data_ and interpret_ptr_ are undocumented routines, deemed internal support procedures by their various callers. So there is not subroutine writeup or info segments to review or amend. The external calling sequences for these subroutines are not changed by the MCR.

Testing

display_data_$for_probe is called only by probe_expr_requests_. display_data_$for_azm is called by azm_requests_1_, and by dump_segment (the ds and rzd commands, and associated subroutines). Changes were tested using dump_segment … -as (and rzd … -as) commands, and using probe commands to display data values (scalars, arrays, and structures).

interpret_ptr_ is called by various trace_stack and probe routines. Its basic functioning was not changed; new code for Fault_Tag_2 modifiers was tested with the new pointer_info command, and with dump_segment … -as.

Version History

Date	Revision	Author	Comment
2017-02-22	1.0	Gary Dixon	Initial draft of this MCR.
2017-02-23	1.1	Gary Dixon	Correct examples given in MCR.