Published: 02/16/68
(Supersedes: BL.7.02, 04/04/67)

Identification

Multics pre-linker
pre_link_2
N. I. Morris

Purpose

During Multics initialization and process creation, segments must be pre-linked. At these times, the operating environment will not support a dynamic linker. It is the purpose of pre_link_2 to scan through a given linkage section, "snapping" as many links as possible. pre_link_2 consists of two procedures, scan_linkage and force_link. scan_linkage searches through a given linkage section for linkage pairs and calls force_link to "snap" a given link pair. pre_link_2 is called by pre_link_1 during Multics initialization (see Section BL.7.01) and by pre_linker_driver during process creation (see Section BJ.9.02). Since it must be able to be called in an environment in which no links have been made, scan_linkage may be entered by calling <pre_link_2>|0, and force_link may be entered by calling <pre_link_2>|1. force_link is also called by estbl_ptr (see Section BJ.9.08) during process creation.

scan_linkage

scan_linkage is as follows:

```
call scan_linkage (lp, table_manager);
```

lp is a pointer to the beginning of the linkage section to be scanned.

table_manager is a pointer to a procedure which will return a pointer to a segment and its linkage section, given a segment name. In Multics initialization, it is a pointer to slt_manager$get_text_link_ptr (see Section BL.2.02), and in process creation, it is a pointer to the pre-linker driver table manager (see Section BJ.9.02). The table manager is called in the following manner:

```
call table_manager (segment_name, text_pointer,
                    linkage_pointer, error_switch);
```

scan_linkage operates in the following steps:

1. Compute length of linkage section from word 6 of the linkage header (see Section BD.7.01 for linkage section formats).

2. Examine each word pair, beginning at location 10(8) of the linkage section. If bits 18-29 of the first word of the pair are zero and bits 30-35 contain a fault tag 2 (46(8)), then the words probably constitute a linkage pair. Otherwise, go to Step 5.

3. Validate the linkage pair by testing the back pointer to the header. If the address of the first word of the pair plus the offset of the word pair from the linkage section header is equal to zero, then the pair is a valid linkage pair. Otherwise, go to Step 5.

4. Call force_link to "snap" the link.

5. If offset of next word pair is less than length of linkage section, go to Step 2 and process next word pair.

6. Return.

force_link

force_link operates in a manner similar to the Multics dynamic linker (see Section BD.7.04). It does not, however, process type 2 links (ITB) or handle traps before definition. force_link is called as follows:

```
call force_link (link_pair_ptr, table_manager, trap_switch);
```

link_pair_ptr is a pointer to the linkage pair to be linked.

table_manager is the same argument as passed to scan_linkage.

trap_switch is 1 if force_link is expected to make a trap call if it detects a trap_before_link while evaluating the linkage pair. If trap_switch is 0, a trap-before-link will be ignored. When force_link is called recursively by dbi, the data base initializer (see Section BL.7.03), trap_switch will always be 0 in order to prevent infinite recursion.

If force_link is unable to make a link (i.e., if the segment or symbol to which the link pair points is undefined), force_link will take no error action. It will simply return. Its caller can test for success in "snapping" the link by examining the tag of the first word of the link pair. If it is still a fault tag 2, force_link was unsuccessful.

force_link performs the following steps in execution:

1. Pick up the tag and external expression word pointer from the second word of the link pair.

2. Using the back_pointer to the linkage header and the definitions pointer, generate pointers to the linkage header and the definitions.

3. Pick up the external expression value and the type pointer from the external expression word.

4. Examine the trap pointer in the type pair block. If zero, proceed to Step 6. Test trap_switch. If zero, proceed to Step 6.

5. Process trap before linking:

   a. Using the arg pointer in the trap word, call force_link recursively to generate a link to the EPL-compiled argument list (see Section BN.7.08).

   b. Using the call pointer in the trap word, call force_link recursively to generate a link to the trap procedure (see Section BL.7.03).

   c. Call the trap procedure using the links just generated.

6. Using the linkage type from the type pair block, get pointers to the text and linkage pointed to by the link. If the link is type 3 or type 4, this is done by generating a specifier and dope vector for the segment name pointed to by seg pointer in the type pair block and calling the table manager routine to search for the segment name. If the segment is not found, return. If the link is a type 1 or type 5, force_link already has the linkage section pointer, and the text segment number can be found in word 7 of the linkage header.

7. If a type 4 or 5 link, search the definitions of the external segment for the symbol pointed to by sym pointer in the type pair block. Extract the value and type from the external definition. If the symbol is not found, return.

8. Using the values and pointers generated in the previous steps, make a pointer and replace the original linkage fault pair with the pointer.

9. Return.