Published: 10/14/68

Identification

Avoiding Unbindable Segments
R. H. Thomas

Purpose

Although in principle the binder will eventually be able to bind all EPL and EPLBSA code, it is currently possible to inadvertantly write code which cannot be bound by the present implementation of the binder. The purpose of this document is to explain in some detail what the post_binder does to links and to begin to document code constructs that are known to be unbindable by the current binder, in order that programmers will avoid using these constructs. It is hoped that whenever binding problems occur they will be reported so that a list of problems may be compiled.

Discussion

The post_binder is driven by relocation code. The relocation code for the bound text segment is scanned. Whenever the code for the left half of a word is lp-15 ("10100"b - see BD.2.01), the modifier field indicates indirection (bits 30-35 are "010000"b) and the fifteen bit address points to a link, the word under consideration may contain a resolvable intrasegment reference.

The post_binder proceeds as follows:

(1) The internal expression word is examined (see BD.7.01). The expression (exp) is saved (bits 18-35).

(2) The type pair is examined.

a. If the trap pointer is not zero the link is not resolved.

b. If the type is 1: (the reference is *|exp, m)

(i) If the segment pointer is 0 then the reference is to the text segment. The left 18 bits of the word are replaced by exp, bit 29 is zeroed, and the modifier is replaced by the modifier in the fault-2 pair.

(ii) If the segment pointer is 1 then the reference is to the linkage section. The 15 bit address is replaced by exp, bit 29 is left on, and the modifier is replaced by the modifier in the fault-2 pair.

(iii) If the segment pointer is neither 0 or 1 the link is not resolved.

c. If the type is 2: (the reference is base|exp + [ext], m) The link is not resolved.

d. If the link is type 3: (reference is <seg>|exp, m) The name pointed to by the segment pointer is examined to see if it is the name of one of the components of the bound segment. If it is, the left 18 bits of the word are replaced by exp + seg_begin (where seg_begin is the origin of the component segment), bit 29 is set to 0 and the modifier is replaced by the modifier in the fault-2 pair.

If the name is not the name of a component segment the link is not resolved.

e. If the reference is type 4 or 5:

(<seg>|exp + [ext], m is type 4)

(*|exp + [ext], m is type 5)

If <seg> is not a component segment the link cannot be resolved. The value of ext is found.

(i) If ext is a class 0 symbol the left 18 bits of the word is replaced by ext + exp, bit 29 is set to zero and the modifier is replaced by the modifier in the fault-2 pair.

(ii) If ext is a class 1 symbol (this is the case when one component segment calls another component segment) the 15 bit address of the word is replaced by ext + exp, bit 29 is left on and the modifier is replaced by the modifier in the fault-2 pair.

To prevent the pre-linker from snapping an unnecessary link, the fault tag in a resolved fault-2 pair is set to zero.

In cases (2)b(ii) and (2)e(ii) where a 15 bit address must remain the computed address is checked to insure that it does not overflow the 15 bits. If it does the link is not resolved.

Unbindable constructs

What follows is an open-ended list of code constructs that are known to be currently unbindable. (In many cases the binder or post_binder could be altered to make the construct bindable; however such modifications will not be undertaken in the immediate future because of the associated expense in time and resources.)

1. Sequences of code such as the following are unbindable:

```
        tra       lp|k,*
         .
         .
         .
       eapbp      lp|k
        tra       bp|0,*
```

The first "tra" results in the removal of the fault tag at lp|k.

Alternative that is bindable:

```
        tra       lp|k,*
         .
         .
         .
       eapbp      lp|k,*
        tra       bp|0
```