Identification

The Binding Procedure

G. S. Stoller

Purpose

This document describes a binder which achieves the goals set forth in BD.2.00.

Glossary

See Glossary in BD.2.00.

Restrictive Assumptions

1. In all object segments, the linkage-info appears in the same section (i.e. always in the text-section or always in the linkage-section). It may even be assumed that this section is the text-section.

2. All logical-segments given as input to the binder are assumed to be in object-format.

3. Segments containing gates and doors will not be bound.

4. A "trap before link" occurring on a link between two of the component segments will not be allowed. When such an occurrence is seen, binding will be terminated.

5. Only the latest EPLBSA entry-sequence, the binder entry-sequence, and the PL/1 entry-sequence will be accepted. All other entry-sequences are declared unbindable and will be flagged by the binder when seen. If many instances of other entry-sequences are discovered, the set of acceptable entry-sequences may be expanded.

Efficiency Considerations

For reasons of speed it is probably best to unpack several units of the relocation-bits at once via EPLBSA code. An array will be prepared containing the relocation-information for approximately 500 words (1000 half-words) per call to this EPLBSA-procedure. A similar technique will be used for packing relocation-bits.

probably should move + relocate in EPLBSA

Simplifying Assumptions

1. The linkage-info can be parsed and properly decoded without the aid of relocation-information.

2. Each definition specifying a place within the linkage-block that looks like the beginning of an entry-sequence is specifying an entry.

Strategy

Consider the binder to be a translator (thus translator-terms can be used in this description); as such it is a two-pass translator on the components (as atoms) of the bound-segment but only a one-pass translator on the half-words (as atoms) of the bound-segment.

The first pass is the standard assembler-type pass of name-definition and location-counter base-value setting. Post-processing of this pass consists of producing all of the linkage-info (link-snap-info and link-block) for the bound-segment by disassembling the components' linkage-info and reassembling the final collection.

Now the second pass begins. Here the actual relocation is done and all references to a component's linkage-block are converted to intra-segment references if possible; otherwise they are converted to references to the bound-segment's linkage-block. Postprocessing consists of placing the linkage-info in the bound-segment.

Detailed strategy is given in the remainder of this section of the MSPM.

A. User-Interface

  1. Command-type segment-handling.

  2. Initialization of structures and variables directing the basic binder.

  3. Set user-options.

B. Definitional pass over components.

  1. Fix attention on one component.

    a. From the symbol-section get the text-length and link-length. Check against values from object-format.

    b. Extract information from linkage-section.

      i. Extract def-pointer; if defs are in linkage-section set flag to abort binding.

      ii. Set size of ego-text-section in binder-array.

      iii. Obtain size of internal-static and place it in binder-array.

    c. Obtain size of ego-symbol-section (it is the offset of [rel_text] minus the offset of [symbol_table]) and place it in binder-array.

d. Unravel the linkage-info and coalesce information and links where possible. This information will be kept in a symbol-table-like store, hence coalescence and conversion (where possible) to intra-segment references will be automatic.

i. Go through all definitions (including associated entry-sequence if relevant) extracting the basic information. Delete the definitions of "symbol_table", "rel_text", "rel_link", and "rel_symbol".

Possible errors include:

multiple definition of entry-symbol
(e.g. "c" in "a$c" "b$c")

undefined symbol (e.g. binding segment <a> but there is no definition of "a$c" although "a$c" is referenced).

ii. Pick up each link from the linkage-block and gather its link-snap-info placing this in an appropriate structure. The "undefined symbol" error message mentioned earlier is possible here, as is an "out-of-bounds" error message.

e. Compute base value (i.e. offset in bound-segment's section of component section's loc zero) for each section. Pay attention to "0 mod 2" restrictions and "0 mod 8" restrictions (if any).

2. Fix attention on bound-segment (after passing over each component).

   a. Delete definitions specified by user, error-messages if def not found.

   b. Insert additional names for remaining entries according to user specifications; error messages possible.

   c. Act on some user-specified-options.

      i. Print out bind-map.

      ii. Save partial results and stop binding (in case of unusual messages or a user-specified "halt" here).

   d. Reassemble linkage-info.

   e. Make conversion-map for linkage-blocks. Thus a component's reference to its linkage-block can be converted to a reference to the bound-segment's linkage-block.

   f. Create symbol-section header for bound-segment.

      i. Text-size is the size of the entire text-section, link-snap-info included.

      ii. Link-size is the size of the entire linkage-section (i.e. header, internal static, linkage-block).

      iii. Translator name is "binder".

      iv. Next-header points to header for first component; component-level is set to zero.

Have we lost individual Translator info?

C. Relocation pass over components

(fix attention on one component, go through each ego-section).

1. Simple and obvious procedures apply to all half-words not related to linkage-info.
2. Each reference to the component's linkage-block is converted to an intra-segment reference if possible, otherwise it is converted to a reference to the bound-segment's linkage-block.
3. Process the symbol-section header.
   a. Increment (by one) the component-level (formerly called "binding-indicator").
   b. Set the next-header pointer (if it is currently zero and another component follows).
   c. Set text-size and link-size (to reflect ego-section sizes).
4. As this relocation is going on, repack the relocation-information for each section. Note that the values of [rel_link] and [rel_symbol] are currently unknown.

D. Finalize binding.

1. Set up linkage-header.
   a. def-pointer.
   b. size of internal-static .
   c. size of section.
2. Place linkage-info in the bound-segment.