Published: 11/03/67

Identification

create_linker_segs
R. L. Rappaport

Purpose

Subroutine create_linker_segs is one of the procedures invoked at process creation time. This procedure creates copies of various linkage sections and places these copied segments into the new process directory of the created process. Create_linker_segs also produces a table, the pre-linker driving table, in which these created segments (and others as well) are listed. This table is used at process initialization time to pre-link the linker in the new address space. It is intended that several versions of create_linker_segs will be available, each capable of establishing a particular version of the linker and its needed subroutines. That is, to create a process with a particular linker would require calling a particular version of create_linker _segs. This document provides an outline of the structure into which all such versions must fit. Section BJ.8.06 describes the initial version of create_linker_segs which will be implemented in initial Multics.

Introduction

In order to be able to handle dynamic linkage faults, a process must have a pre-linked linker in its address space that will be invoked upon recognition of a linkage fault. When we say pre-linked this does <u>not</u> imply that the linker need be pre-linked to <u>every</u> procedure it calls and that these in turn need be pre-linked. Rather it implies that a <u>minimum</u> path through the linker be pre-linked and that the linkage faults that the linker itself may get can be handled by the subroutines on the minimum path. Before going on let us review the events that occur at the time of a linkage fault.

Let us consider the case where procedure <a> with linkage section <a.link> calls procedure <b> with linkage section <b.link>. At the time of the fault, control is immediately passed to the fault interceptor module (FIM, see Section BK.3.03). Upon determining that the fault is a linkage fault, the FIM decides to call the linker. In order to call the linker, the FIM must call indirectly through a pointer in the process definition segment (pdf. see BJ.1.06). The reason this call must be done indirectly is that the FIM and its linkage section are shared segments pre-linked at system initialization. Since the linker need not have the same segment number in each process, we cannot place a valid pointer to the linker in FIM.link. However, pdf does have the same segment number in each process and hence its segment number can be placed in FIM.link.

The purpose of the linker is to produce, from the symbolic information available in <a> and <a.link>, a valid ITS pointer that points to the entry point of <b> located in <b.link>. This ITS pointer will then be stored directly into the word pair in which the original fault was discovered. In order to develop the needed ITS pointer, the linker must first obtain segment numbers for <b> and <b.link>. This is done by a call to the segment managment module (SMM, see Section BD.3.00) which returns the needed segment numbers. With these segment numbers it is not difficult to see how the linker might develop the needed pointer that will replace the original fault tag. Since it is not our purpose to review the algorithms coded into the linker but to get an overview of the whole strategy, let us instead look at the way the SMM develops the segment numbers that it returns to the linker. What follows is a simplified overview of the SMM which only points out things relevant to the discussion at hand.

The SMM uses a data base known as the Segment Name Table (SNT, see BD.3.01). The SNT is a table which lists correspondences between

1. call names of segments (i.e. the names by which they are called.)

2. path names in the file system.

3. segment numbers.

Conceptually, the SNT is a set of 3-tuples. The elements of each particular 3-tuple are the call name of a segment, its path name in the hierarchy, and the segment number, if any, that has been assigned to the segment. (The "if any" in the preceeding sentence refers to the fact that the segment number element in a particular 3-tuple might be blank signifying that no segment number has as <u>yet</u> been assigned to this segment.)

The SMM is basically faced with the following task. It is called and passed the call name of a segment and it wishes to return the segment number of the segment. The SMM accomplishes the task in the following way. First the SMM looks into the SNT to find if a given call name is listed in an existing 3-tuple. Suppose for example that it is listed in an existing 3-tuple. (We will discuss later how this 3-tuple came into existence.) If the segment number element is also listed in the 3-tuple, the job is done. However, if the segment number is not listed, it must be determined before we can proceed. In order to determine this number we must call the basic file system primitive estblseg (see Section BG.8.04) and pass to estblseg the path name found in the 3-tuple. Estblseg returns the desired segment number. If on the other hand we cannot find an existing 3-tuple which contains the desired call name, our problem is to establish such a 3-tuple. The search module (see BD.4.00) is the procedure to be called in this case. The task of the search module is simple. The search module is called by the SMM, passed a call name and it returns a path name. The SMM takes this path name and the call name and establishes a new 3-tuple which as yet contains no segment number. SMM then calls estblseg to complete the 3-tuple. On subsequent calls for this call name, an existing 3-tuple will be found.

The introduction of one more concept will allow us to complete this overview. In the hierarchy there exists a certain class of segments known as "relationship segments". These segments are lists of 3-tuples in which the segment numbers are left out. These segments play an important role in the SMM.

SMM calls estblseg in order to obtain a segment number for a segment located in the file system hierarchy by a given path name. However, the segment named by the path name might not be the segment in which the SMM is primarily interested; it might in fact be a relationship segment associated with the desired segment. The relationship segment of a segment serves to establish the association between call names that the segment uses and path names that the human author of the segment wished to make explicit. For example, if we have a procedure named z that calls a routine named "cosine" and we wish this call diverted to the segment with path name >a>b>x, we need merely establish a relationship segment for z that lists this desired correspondence. When we first encounter z, the SMM will obtain its relationship segment and SMM will incorporate the contents (i.e. the 3-tuples) of its relationship segment into the SNT. When z subsequently experiences a linkage fault for cosine, the 3-tuple associating cosine with a>b>x will already exist.

Let us now review the path followed on a linkage fault. The FIM calls the linker indirectly through pdf. The linker calls SMM in order to obtain segment numbers. The SMM refers to its SNT and either calls, (1) nothing because a complete 3-tuple exists, (2) estblseg because an incomplete 3-tuple exists or (3) the search module to get a 3-tuple. When the SMM has obtained the desired segment number it returns to the linker which sets the desired link and returns to the FIM. Now we are faced with the question of what to pre-link.

The indirect call from FIM to pdf need not be pre-linked at process initialization time since all processes use a shared copy of FIM and its linkage section and this shared copy is linked to pdf at system initialization time.

The pointer in pdf to the linker must be set at process initialization time. Likewise the call from the linker to the SMM must be pre-linked.

The SMM however makes two calls and one external reference. The reference is to the SNT and it of course must be pre-linked. However, only one of the calls, the call to estblseg, must be pre-linked. The call to search will not be pre-linked. However, in order to be able to handle the fault that will result from the call to search, a 3-tuple that defines an association between the name "search" and a pathname must be placed in the initial SNT of the new process. This pathname will in fact be for a relationship segment for search which will list all the call names used by search and <u>their</u> associated pathnames. Let us see how this will work.

At the time of the first linkage fault in the new process (a fault for call name x) the FIM will call the linker which will call the SMM. (So far so good.) The SMM will find no 3-tuple defining x and will therefore call search at which time we will get a recursive linkage fault. We will again travel down the path arriving at the SMM again this time looking for a 3-tuple defining "search". This time we will find one and call estblseg to get a segment number. Estblseg will inform the SMM that the given pathname was in fact a relationship segment and the SMM will then incorporate the contents of the relationship segment directly into the SNT before obtaining the segment number of the search module itself. After obtaining the number, the SMM will return and the linker will set the fault in the SMM's linkage section and return to the FIM. The FIM will restore the conditions as they were before the second fault and the SMM will complete its call to search. Search may get linkage faults but the call names will all be defined because of the relationship segment described above. Eventually, search will return and the 3-tuple defining x will be established by the SMM. In this way the original linkage fault for x can be satisfied.

Discussion

Create_linker_segs is called from create_proc (see BJ.8.01) and the calling sequence is:

```
call create_linker_segs (dir_pathname);
```

where dir_pathname is the pathname of the new process directory.

The purpose of create_linker_segs is to establish the needed pieces of data that will be used in pre-linking the linker in the new process address space. In particular, in order to pre-link the linker, the new process will have to have available a copy of the linker's linkage section, a copy of SMM's linkage section, etc. Therefore, create_linker_segs must first create copies of several segments and place them into the new process directory. In particular, create_linker_segs must make copies of the linkage sections of the segments that will be pre-linked. These are:

1. The linker's linkage section.

2. The SMM's linkage section.

Also the initial SNT which lists a 3-tuple defining search must be copied into the new directory.

Secondly, the pre-linker needs to know which segments to pre-link. The pre-linker's principal piece of data is the pre-linker driving table which must be created by create_linker_segs. This table, whose format is given below has an entry for each segment which either is to be pre-linked (e.g., the linker) or is referred to by a segment that is to be pre-linked (e.g., estblseg which is called from SMM). The segments listed in the pre-linker driving table are:

1. linker
2. linker.link
3. SMM
4. SMM.link
5. SNT
6. estblseg (actually hcs_1$estblseg. See BD.6.03)
7. estblseg.link (actually hcs_1.link)

The pre-linker driving table is placed into the new process directory. The table will be accessed by the new process itself once it begins its self initialization.

Format of the pre-linker driving table

The pre-linker driving table is implemented in two segments. The first segment <pre_link_dt> contains a fixed length entry per listed segment. The second segment <pre-link_nametable> contains variable length information (i.e., character strings) about each of the segments. The fixed length entries contain relative pointers to their respective entries in the name table. The PL/I declaration of <pre_link_dt> is:

```
dcl  1  pre_link_dt based (p)

        2  count fixed

        2  entry (p→pre_link_dt.count),

           3  call_name_ptr bit (18),     /* Relative ptr to call
                                             name of seg */

           3  path_name_ptr bit (18),     /* Relative ptr to
                                             directory path name */

           3  entry_name_ptr bit (18),    /* Relative ptr to entry
                                             name */

           3  linkage_section_sw bit (1),/* "1"b if segment is
                                             linkage section "0"b
                                             if text segment */

           3  pre_link_sw bit (1),        /* "1"b if segment should
                                             be pre_linked */

           3  assoc_seg_ptr bit (18),     /* Relative ptr to entry
                                             of associated text
                                             or linkage section */

           3  segptr ptr;                 /* Pointer to segment */
```

The relative pointers to the call names, path names, and entry names are pointers to structures allocated in the name table. The PL/I declaration of the respective structures is:

```
dcl  1   name_struc based (p),

         2   count fixed,

         2   char(p→name_struc.count);
```

The assoc_seg_ptr in an entry is a relative pointer to the fixed length entry of the associated text (linkage) segment if the current entry is one for a linkage (text) segment. That is, this relative pointer points into <pre_link_dt> itself.

The segptr is an ITS pointer to the segment that will be established at pre-linking time. Create_linker_segs initially leaves it empty.

The pre_link_sw is one of several things that will be discussed below.

For the advanced reader

Several points have been ignored until this point. First several other segments not yet mentioned will have to appear in the pre_linker driving table. Of particular interest among these is a "datmk" type segment that will be needed to pre_link "trap before link" type references that will be encountered. The system initialization program, dbi (data base initializer, see BL.7.03), a hardcore segment pre-linked at system initialization, is available and so is its shared linkage section. All that need be done is list the two segments in the table. However, without special consideration several of the pre-linking procedures are liable to attempt to write into the linkage section of dbi. This would be disastrous since the segment has been made read only. Therefore for dbi.link's entry in the pre-linker driving table the pre_linker_sw is set to "0"b in order to prevent the attempted writing.

Other segments that have to be listed include the various EPL routines called be SMM and the actual segments into which the SMM is really broken. That is, we have been considering the SMM as a single segment when in reality it is a collection of several segments.

Finally, one more point should be made. When the new process gets through pre-linking it will make its first call to a procedure named "init_admin". This call will cause a linkage fault. In order for a creator process to be sure that the correct segment is established for "init_admin" an additional 3-tuple is placed in the SNT by create_linker_segs. This 3-tuple relates the name init_admin to a path name in the hierarchy. In this way, the creator has control over the initial path which the new process will follow.