Published: 05/04/67

Identification

Segment Name Table (SNT)
S. L. Rosenbaum

Purpose

The Segment Name Table (SNT) is a per-process table maintained exclusively by the Segment Management Module (SMM). The SNT is itself a segment within the process to which it refers and contains information about each segment and each call name which is currently known to the process.

When a process initiates a call name for a segment, the segment known to the process by that name, it causes the SMM to enter the appropriate information into the SNT. When a process terminates a call name for a segment, rendering the segment unknown to the process by that name, it causes the SMM to remove all information from the SNT pertaining to that name for that segment.

Segments may be multiply-defined, e.g., two procedures may each use the same segment for its data base - one knows the data base as "sin.table", the other knows the data base as "cos.table". Call names also may be multiply-defined, e.g., two procedures may each use a data base known as "sin.table" - one uses a segment from directory ">radian", the other uses a segment from directory ">degrees". A mechanism for "relating" the SNT's information handles the problem of multiply-defined call names. (See the discussion of "Related SNB's" later in this section.)

A known name is a call name which appears in the SNT; a known name may or may not be associated with a segment; An initiated name is a call name which is associated with at least one segment; a terminated name is a call name which is known for no segment.

An initiated segment is a segment which is known by at least one call name; a terminated segment is a segment which is known by no call name. When a user or procedure terminates a segment (terminates all names for a segment), the SMM removes all information pertaining to the segment from the SNT.

Organization of the SNT

The SMM records in the SNT the environment existing at the time of each initiation. That is, it records the call name, the segment pointer (ITS pair pointing to the base of the segment), the segment's path name in the file system hierarchy, an indication if the call name is multiply-defined and an indication if the segment itself needs specific segments for names it calls. Basically, the information in the SNT is organized around units called Segment Name Blocks (SNB's) and Segment Headers. Each initiated segment has one Segment Header and one or more SNB's. The Segment Header contains information about the particular initiated segment; each of its SNB's contains information about the particular initiation for the segment such as the name for which the segment was initiated. (Remember: a segment may be known to the process by more than one name.) All the SNB's for the same segment are double-threaded to each other; the Segment Header points to the SNB most recently created for the segment (see Figure I). Note that the SMM can reach the SNB's from their Segment Header and, alternately, the SMM can reach the Segment Header from any of its SNB's. As its name implies, each SNB represents a specific segment initiated for a specific call name. As stated earlier, all the SNB's for the same segment are double-threaded to each other. Similarly the SNB's are double-threaded by call name. The SMM can easily access an SNB via these two main avenues of approach: by segment and by name. An SNB can exist in the SNT in two forms: an initiated form and a terminated form. An initiated SNB is completely threaded - that is, it points to a segment and a name. A terminated SNB does not point to a segment; it does point to a name. (See figure II.) An SNB is known if it appears in the SNT; an SNB is unknown if it does not appear in the SNT. In light of these revelations:

> Initiation is the act of threading an SNB with respect to a segment, i.e., associating the SNB with a specific Segment Header. Termination is the act of unthreading an SNB with respect to a segment, i.e., disassociating the SNB from any Segment Header. Besides referencing an SNB by segment, via a Segment Header, the SMM can reference an SNB by its name. Analagous to Segment Headers, one Name Header exists for each known call name and contains the symbolic call name which is independent of segments.

(Reminder: the name can be initiated for several segments.) A Name Header points to the SNB most recently created for the name. Both the Segment Headers and the Name Headers simplify the SNT by reducing the amount of duplicate information stored. For example, the character string which represents a particular symbolic call name is stored only once - in its Name Header - and each known SNB for that name points to that Name Header (see figure III).

In order for the SMM to get an SNB or to a set of SNB's it goes through one of two tables, the Segment Table or the Name Table. Each entry in the Segment Table points to one Segment Header which in turn points to the last SNB initiated for the segment. Each entry in the Name Table points to one Name Header which in turn points to the last SNB created for the name. The immediate goal of the SMM determines its approach to the SNT. For example, if the SMM wants to find all the call names for a specific segment given its segment number (or segment pointer), it enters via the appropriate entry in the Segment Table. When the SMM wants to find all the segments for a specific call name, it enters via the Name Table.

To review:

All the SNT's information is easily accessible by segment or call name. One level higher in the SNT organization are the Segment Headers and the Name Headers. One Segment Header exists for each initiated segment; it contains information about the segment which is independent of name information, e.g., its segment number; it also points to the SNB most recently initiated for the segment. Analogously, one Name Header exists for each known call name and contains the symbolic call name which is independent of the segment information. The two doorways of the SNT are the Segment Table - a table of Segment Headers and the Name Table - a table of Name Headers.

Hence, given a segment pointer, the SMM goes through the Segment Table to a Segment Header to an SNB. Given a call name, the SMM goes through the Name Table to a Name Header to an SNB. (See figure IV.)

Related SNB's

In addition to the two methods of cross-sectioning the SNT discussed above (by segment and by name), the SNT contains threads which relate SNB's to each other. The SMM overview (section BD.3.00) and the relate command (section BX.8.13) discuss the subject of related SNB's in detail but a capsule description here seems pertinent and necessary for understanding the SNT's structure. The following example highlights the main features of "related" SNB's.

A user writes a procedure called "alpha" which invokes the procedures called "beta" and "delta". Another user hears about this great and wondrous procedure "alpha" and wants to use it in his process. Meanwhile back at the drawing board, "alpha"'s author decides to guarantee that "alpha" always will get the right versions of "beta" and "delta", i.e., the author's "beta" and "delta", even if the borrower has other segments by the same names. In addition, "alpha"'s author

1. does not want to re-program either "alpha", "beta" or "delta" - e.g., he could make "beta" internal to "alpha" but does not want to.

2. does not want to leave it up to the borrower to get the correct versions - in fact, the author doesn't want the borrower to even have to know about them.

By using the relate command, "alpha"'s author relates "beta" and "delta" to "alpha" so that whenever "alpha", and only "alpha", calls for a "beta" or "delta", it gets the author's "beta" or "delta".

The relate command establishes a segment in the file system hierarchy where the user expects to find "alpha". This segment, called "alpha"'s relationship segment contains the actual path name of "alpha", "alpha"'s related names, i.e., "beta" and "delta", and directions for obtaining the specific segments for the related names. The author of "alpha" can now take a coffee break -- he need do no more. When the borrower invokes "alpha", directory control informs the SMM that, instead of a segment for "alpha",

it has a relationship segment for "alpha". The SMM uses the relationship segment to get the actual segment and create a complete SNB for "alpha". In addition, the SMM sets up a skeletal SNB (known but not initiated), for each related name - in this example, "beta" SNB and the "delta". The SMM looks upon "alpha" as a "mother" SNB whose "daughters" are the "beta" SNB and the "delta" SNB. To complete the family tree, "beta" and "delta" are "sisters". Various switches and pointers in the SNB's, described later in this section, keep the family relationship intact. (Note: skeletal SNB's are always created for "daughter" SNB's before their "mother" actually needs them.)

Now when "alpha" is running and invokes "beta", the SMM goes into action again. At this point the SMM knows the segment number of the caller (the segment number assigned to "alpha") and the symbolic call name of the desired segment, (the character string "beta"), the SMM does not know if the caller wants a particular "beta" or not. The SMM looks for an SNB whose "mother" is the calling segment and in this case finds the desired SNB for "beta". Hence when "mother" calls, "daughter" responds. (Figure V shows the threading for "alpha" and its daughters, "beta" and "delta".)

If the SMM had not found a "daughter" SNB for the calling segment, it assumes no relationship was intended and either uses the "beta" SNB most recently created (if one exists) which has its global usage switch set "on" or attempts to create an SNB for "beta" in the normal way. The global usage switch set "on" indicates that the SNB may be used by any caller looking for this name even if the SNB itself is some other segment's "daughter".

If the user desires more information on the topic of related SNB's he should consult the following sections:

1. for the motivation for relating SNB's - BD.3.00 - The SMM Overview

2. for using related SNB's in the SMM and the format of relationship segments - BD.3.02 - the SMM primitives using call names.

3. for relating segments by the user - BX.8.13 - relate command.

4. for referencing relationships from the Search Module, - BX.13.01 - Search Module Vocabulary - (in particular the keywords "caller" and "caller_not")

Contents of the SNT

The SNT contains:

1. A header which points to two variable length sub-tables - the Name Table and the Segment Table.

2. A Name Table which points to the Name Headers containing information for each known call name.

3. A Segment Table which points to the Segment Headers containing information for each initiated segment, i.e., segment for which at least one SNB is initiated.

4. A Name Header for each known name.

5. A Segment Header for each initiated segment.

6. One Segment Name Block (SNB) for each unique combination of segment number, symbolic call name and ring of availability. (E.g., if a segment can be referenced from both ring one and ring two, there are at least two entries (SNB's) for it in the SNT.)

Contents of the SNT Header

A header of fixed length appears at the beginning of the SNT. The header points to two sub-tables, each of which represents a different avenue of approach to the SNT's information. The header consists of the following items:

1. Pointer to the beginning of the Name Table - ntrp - The Name Table is used when the user refers to a segment by a symbolic call name.

2. Size of the Name Table - ntsize

3. Pointer to the beginning of the Segment Table - strp - The Segment Table is used when the user refers to a segment by a segment pointer.

4. Size of the Segment Table - stsize.

Contents of the Name Table

The Name Table contains one entry for each unique symbolic call name currently known to the process. The Name Table is a variable length table of pointers, hash-coded with respect to symbolic call name. It provides the normal path for referring to segments by a symbolic call name. Each entry in the Name Table, nhrp, points to a Name Header which is a block of information about a single call name. When a call name is made known for the first time, i.e., no SNB is currently known by the name, the SMM creates a Name Header and inserts a pointer to it into the Name Table.

The SMM hash codes the call name to determine the position of the Name Header pointer in the Name Table. The call name is known now. When a name becomes unknown, i.e., there is no longer any SNB for the name, the SMM deletes all the information associated with the name from the SNT, including its Name Header and the Name Header pointer in the Name Table. Now, as far as the SMM is concerned, the name is no longer known to the current process i.e., the name is terminated.

Contents of the Segment Table

The Segment Table contains one entry for each segment currently known to the process. The Segment Table is a variable length table of pointers, indexed by segment number. It provides the most direct route to a physical segment. Each entry in the Segment Table, shrp, points to a block of information for a single segment, called a Segment Header. That is, shrp (i) points to the Segment Header for segment number i.

When the SMM initiates a segment for the first time, i.e., the segment is not currently known to the process by any calling name, the SMM creates a Segment Header and inserts a pointer to it into the Segment Table. If the segment number for the new segment exceeds the current size of the Segment Table, stsize, the SMM expands the length of the Segment Table and increases the value of stsize accordingly. At this stage, the segment is initiated.

When the SMM terminates a segment, it removes all the information connected with the segment from the SNT, including its Segment Header and the Segment Header pointer in the Segment Table.

Contents of a Name Header

Each known symbolic call name is represented by one block of information called a Name Header. A Name Header points to the detailed data associated with its symbolic call name. Each Name Header consists of:

1. name - a character string which is the known symbolic call name,

2. nblockn - the number of known SNB's for this name,

3. nbrp - a pointer to the beginning of the SNB most recently created for this symbolic call name.

Contents of a Segment Header

Each initiated segment is represented by one block of information called a Segment Header. This block of information contains the data associated with a specific initiated segment and in effect defines a segment irrespective of its call names. The Segment Header points to the SNB's created for this segment.

Each Segment Header consists of the following items:

1. segment number (from which the segment pointer can be constructed)

2. directory path name

3. entry name

4. unique identification

5. slotlist - locates the segment in the file system hierarchy

6. delete switch

7. usage lock

8. number of SNB's for this segment

9. pointer to last SNB for this segment

10. pointer to last daughter SNB related to this segment

1. segment number - segno - which can be converted into a pointer to the base of the specific initiated segment. (The SMM obtains the segment number from the basic file system by calling the directory supervisor's entry estblseg - BG.8.02.)

2. directory path name - dpath - path name of the directory (relative to the root directory) in which this segment resides

3. entry name - ename - the entry name of this segment in directory dpath. (If the segment is in directory ">a>b" with the entry name "c", its path name is: ">a>b>c". Its linkage section segment must be ">a>b>c.link"; its symbol table segment must be ">a>b>c.symbol".)

4. unique identification - uid - the unique identification assigned to this segment by the file system

5. slotlist - an ordered list of the segment's slot numbers. (Note: a slot number defines an entry's position in a single directory - a slotlist describes its position in the file system hierarchy.) The slotlist is useful for interprocess communication since it is not name dependent. Section BQ.6 discussed communication between processes.

The following example illustrates the relationship between a segment's slotlist and its slotname.

if:

slotlist (1) = 2
slotlist (2) = 8
slotlist (3) = 7

then the segment's slotname is represented by the character string = >>2>>8>>7

6. delete switch - delsw - determines whether or not the segment should be deleted from the file system hierarchy when the segment is terminated.

delsw = 0 indicates no deletion

= 1 indicates deletion

After the SMM terminates the last initiated name for the segment, i.e., there are no other initiated SNB's for the segment, it checks the value of delsw. If the delete switch is "on" (delsw = 1), the SMM deletes the segment from the file system hierarchy.

7. usage lock - lock - where

lock = 0 means segment unlocked

= 1 means segment read-locked

= 2 means segment write-locked

= 3 means segment data-share-locked

To read the segment, the process read-locks the segment (sets lock = 1) preventing another user's procedure from writing in or data-sharing the segment. Any number of processes can read-lock the same segment at the same time. The user must have read access to the segment in order to read-lock it. Section BG.8.00 describes access rights. To write in a segment, the process write-locks the segment (sets lock = 2) preventing other authors from making modifications at the same time. Only one procedure can write-lock a specific segment at any one time. At the time a user attempts to write-lock a segment the following two conditions must be true:

1. the user is allowed to write in the segment by having write-access and/or append-access to the segment.

2. the segment currently is unlocked (lock = 0).

Read-locking and write-locking are relatively conventional concepts. To facilitate shared data bases, interprocess communication and more sophisticated segment usage, the SMM allows data-share-locking of a segment. A data-share locked segment means that the user agrees to share the segment with other users who also have this segment data-share-locked. These users establish usage conventions for the segment among themselves. For example, they mutually agree to allow each other to monitor each other's modification, i.e., any one of them can read a segment into which one of them is currently writing.

8. number of SNB's - nblocks - the number of unique combinations of initiated names and rings of availability known to the process for this segment.

9. pointer to the last SNB - sbrp - pointer to the beginning of the SNB most recently created for this segment.

10. pointer to the last daughter SNB - drp - pointer to the beginning of the last SNB created for the call names related to this segment. Section BD.3.00 discusses the environment of related call names - however as a refresher: When a user wants to guarantee that a procedure uses a certain version of a sub-procedure (e.g., whenever this segment calls for a procedure name "sin", it gets the "sin" procedure in the system library), the user "relates" the sub-procedure to this segment via the relate command. The procedure is called the "mother" segment, the sub-procedure in the above example, "sin") is called a "daughter" segment. Sub-procedures having the same "mother" are called "sister" segments.

Contents of a Segment Name Block

An SNB is a crossroads for segment and name information; it contains roadsigns (in the form of three pairs of pointers) which connect it to the other SNB's with the same segment number, the same call name and the same "mother" segment respectively. One pointer in each pair of pointers leads to the previous SNB in the chain. Hence, it is possible to start with any SNB and

1. following the segment thread - find all the SNB's having the same segment number.

2. following the name thread - find all the SNB's having the same symbolic call name

3. following the "sister" thread - find all the SNB's having the same "mother" segment.

Each SNB consists of the following items:

1. global usage switch - indicates if this SNB can only be used if the calling segment is its mother or if it may be used by any calling segment which wants a segment with this SNB's call name

2. related segment switch - indicates if this SNB is a daughter SNB

3. initiation switch - indicates whether this SNB is in skeletal or complete form, i.e., whether a segment has been found for it or not

4. create switch - indicates whether its segment is in the file system hierarchy or that its segment must be created

5. search switch - indicates whether the Search Module may be invoked to find the segment for this SNB in the file system hierarchy

6. directory name - the path name of the directory where the segment was or is to be found

7. entry name - the entry name of the segment that was or is to be found in the directory defined by 7. (Note: the path name of the segment is constructed by concatenating item 7, the graphic ">" and item 8.)

8. unique name - the entry name of the segment copy in the Process Directory.

9. mother segment's segment number

10. ring number - calling procedure's validation ring number (the SNB can only be used by segments calling from this ring)

11. name header pointer for this SNB

12. name thread

13. sister thread

14. segment number for this SNB

15. segment thread

If the SNB is a daughter SNB, items 1-13 are set initially when its mother SNB is initiated - the remaining items are set if and when the SNB is itself initiated, e.g., when mother wants to use it.

1. global usage switch - gsw - a binary switch where

   gsw = 0

   means this SNB is a "daughter" block which may be used only by its "mother" block.

   gsw = 1

   means this SNB may be used by any procedure looking for a segment with this SNB's call name.

2. related segment switch - rsw - a binary switch where

   rsw = 0

means this SNB is a "daughter block" and items 4 and 5 are meaningful. rsw always equals 1 if gsw = 0.

The only legal combinations for gsw and rsw are:

gsw	rsw
0	1
1	0
1	1

3. initiation switch - isw - where

   isw = 0

   means this SNB is known but not initiated

   isw = 1

   means this SNB is initiated.

4. a create switch - csw - where

   csw = 0

   means do not create a segment - the segment is in the file system hierarchy.

   csw = 1

   means do create a segment for this SNB and attach it to the current process's Process Directory.

5. a search switch - ssw - where

   ssw = 0

   means do not search for the segment

   ssw = 1

   means do search, i.e., invoke the Search Module to find the segment if it is not found in directory dpath.

6. directory name - dpath - a path name which symbolically locates the directory of this SNB segment in the file system hierarchy. The discussion of the relate command (section BX.8.13) describes the method for specifying the path name of a daughter relative to

   a. root directory

   b. the calling directory of the "mother" SNB, i.e., item 6 in the "mother" SNB

   c. various other directories.

7. entry name - ename - the entry name of the segment that was or is to be found in directory dpath.

   Assume that the segment with the path name ">a>b>c" is used for the symbolic call name "x", then

   dpath = ">a>b"

   and

   ename = "c"

8. unique name - uname - the unique name created for the segment copy (uname is null if no copy was made, i.e., the segment "dpath>ename" is used).

9. "mother" segment's segment number - mseqno - the segment number of this SNB "mother".

10. ring number - ringno - validation ring at the time this SNB was created.

11. name pointer - nhrp - pointer to the name header for this SNB,

12. pair of name pointers - pnrp and nnrp -

    pnrp points to the previous SNB for the same symbolic call as this SFNB, i.e., has the same name pointer, nnrp. (pnrp = 0 means this is the first SNB.)

nnrp points to the next SNB for nhrp. (nnrp = 0 means this is the last SNB.)

13. pair of "sister" pointers - prrp and nrrp

prrp points to the previous "sister" SNB, i.e., the previous SNB whose "mother" is also segment number msegno (prrp = 0 means this is the first SNB).

nrrp points to the next "sister" SNB (nrrp = 0 means this is the last SNB).

14. segment number - segno - the number of the segment (extracted by the SMM from the segment pointer assigned by the basic file system) for which this SNB was created.

15. pair of segment pointers - psrp and nsrp -

psrp points to the previous SNB for the same segment as this SNB i.e., segment number segno. (psrp = 0 means this is the first SNB for segno.)

nsrp points to the next SNB for segment segno. (nsrp = 0 means this is the last SNB for segno.)

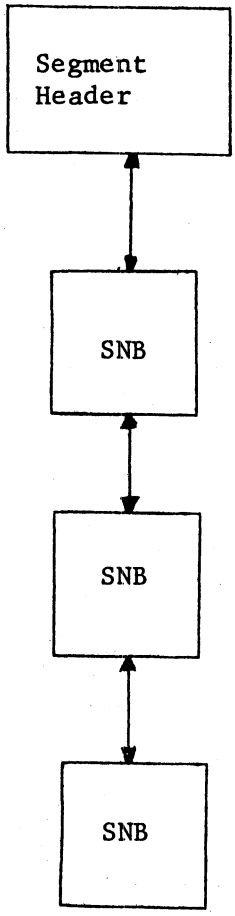


Figure I - Representation of an Initiated Segment

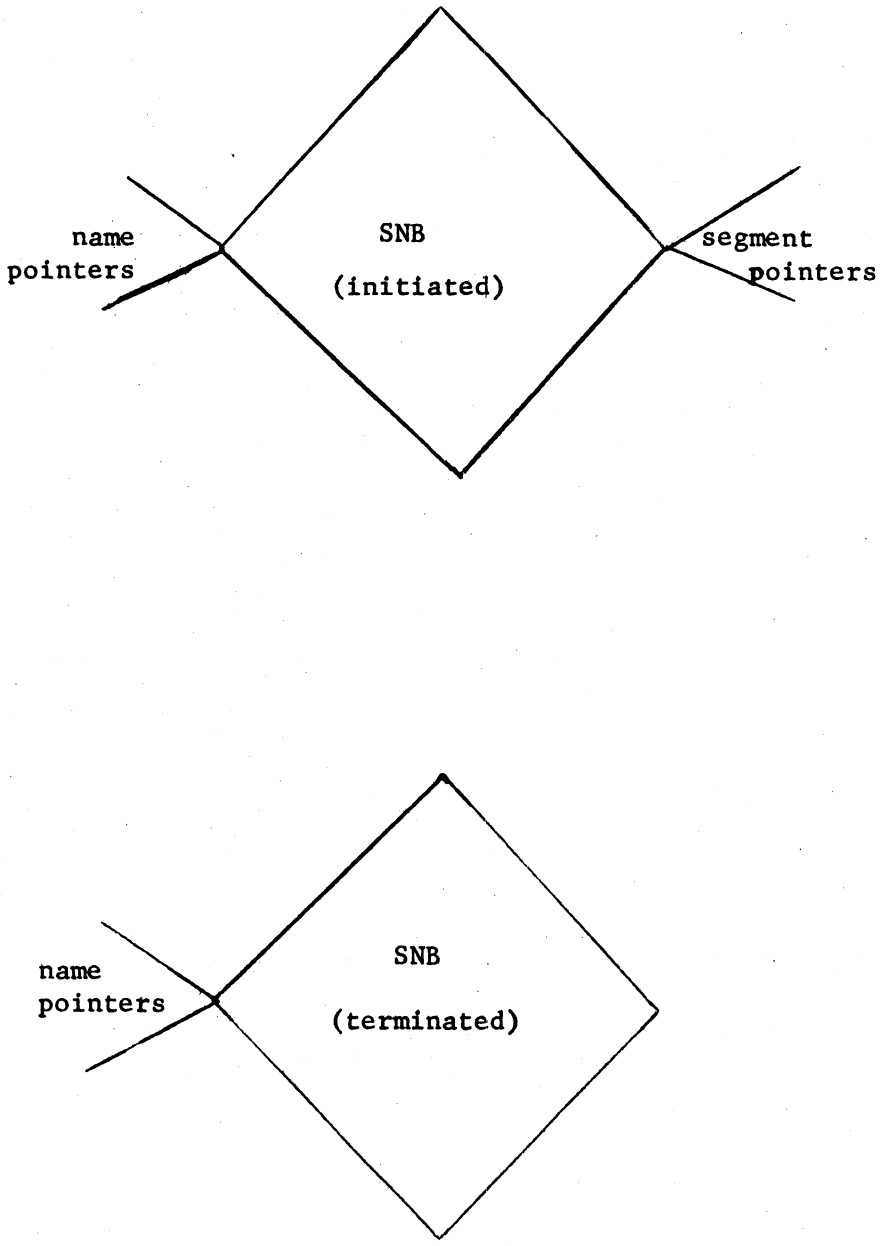


Figure II - Representation of a Known SNB -
A Known SNB can be initiated or terminated

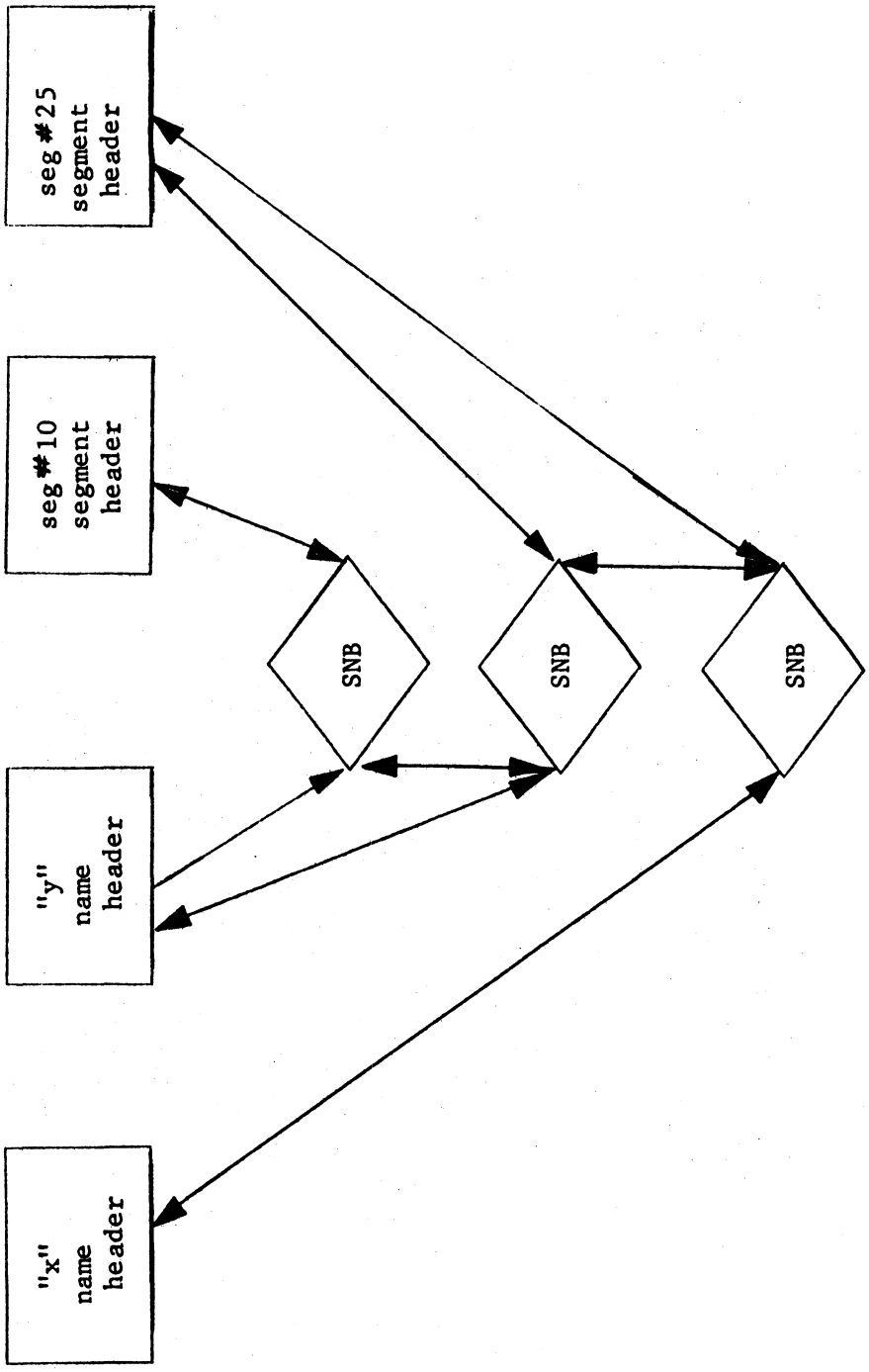


Figure III - Segment #10 is initiated for (known to the process by) only the name "y"
Segment #25 is initiated for both the name "x" and the name "y"

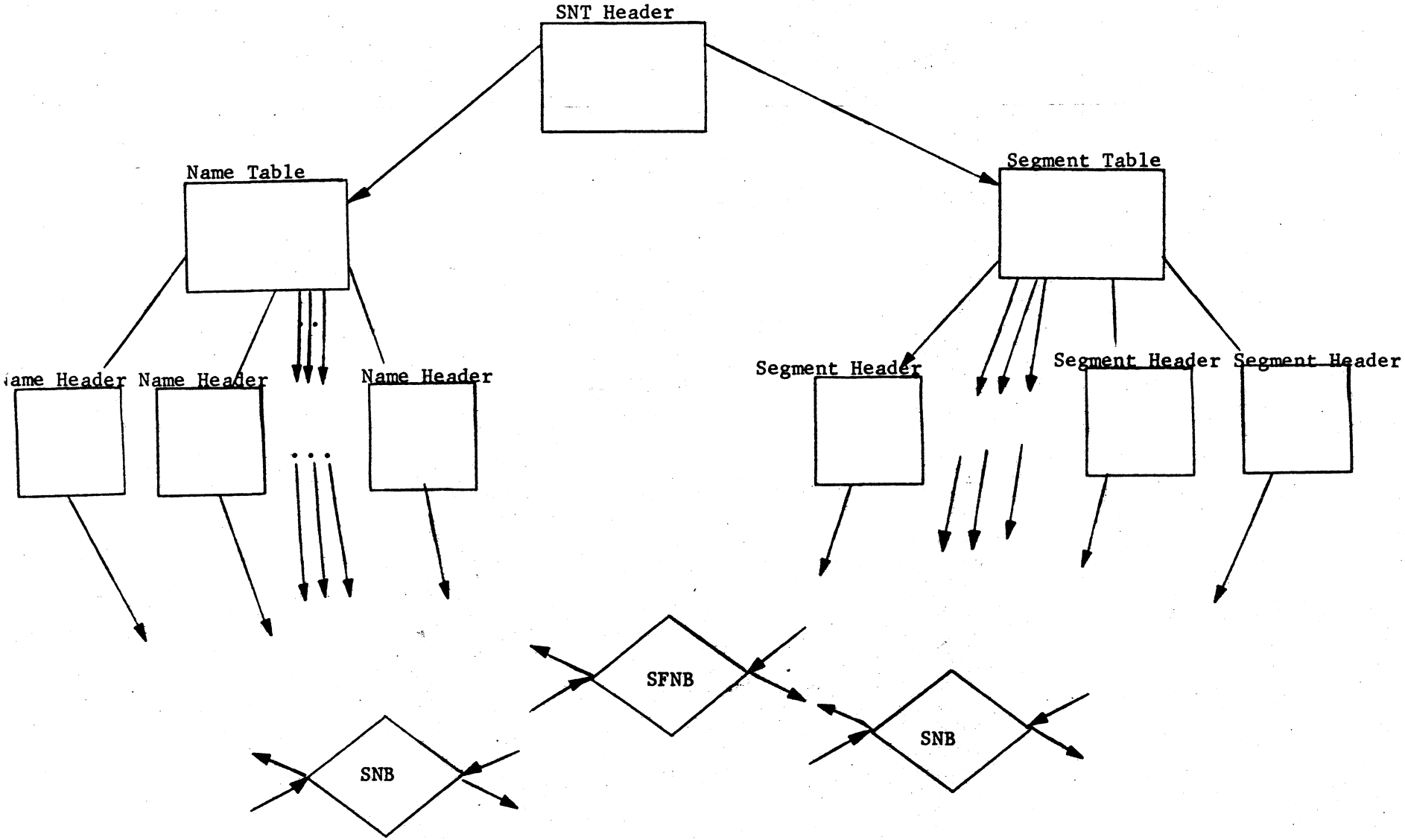
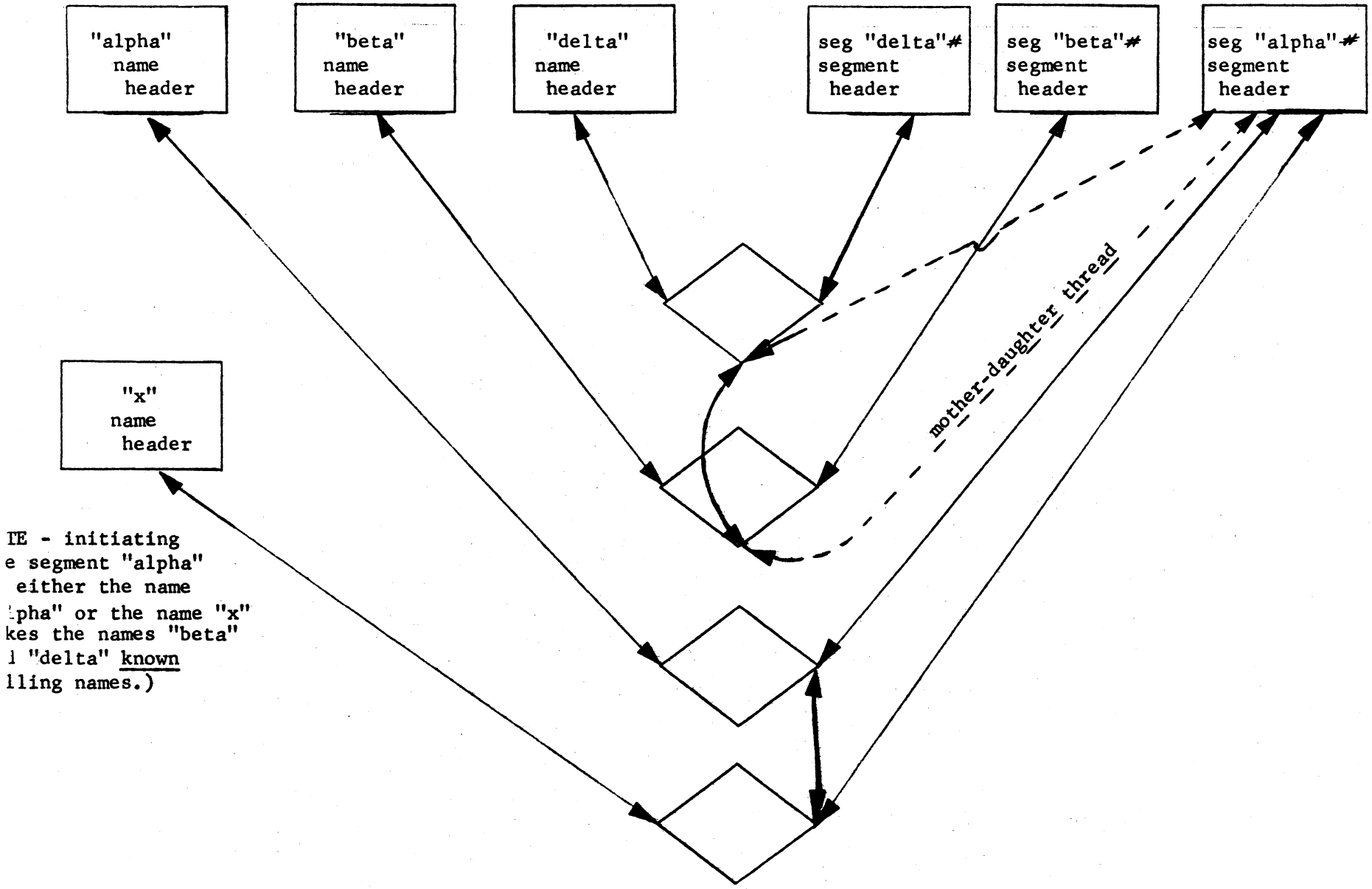


Figure IV - Representation of Segments and names in the SNT



TE - initiating
e segment "alpha"
either the name
lpha" or the name "x"
kes the names "beta"
l "delta" known
lling names.)

Figure V - Representation of Related SNB's and their Mother Segment
Mother is "alpha"# Segment (named "alpha" and "x")
Daughters (sisters) are "beta" and "delta" SNB's

Implementation

All pointers (except the pointer to the segment for the SNT) are relative to the beginning of the SNT segment. As such, these relative pointers are implemented as 18 bit numbers.

The SMM allocates storage for the SNT structures in the same way as the basic file system.

```
/* declarations for the segment name table */

dcl sntp ptr static;

/* declaration for segment name table header */

dcl  1 snt ctl (sntp),
     2 ntrp bit (18), /* pointer to name table */
     2 strp bit (18), /* pointer to segment table */
     2 ntsize fixed bin(17), /* size of name table */
     2 maxsegno bit(18), /* largest known segment number */
     2 nfilled fixed bin(17), /* number of entries in
       name table */
     2 ndelete fixed bin(17), /* number of deletions from
       the name table */
     2 stsize fixed bin(17), /* size of segment table */
     2 ntsizemin fixed bin(17),
     2 stsizemin fixed bin(17),
     2 sntarea (1950) fixed bin(35); /* area for table */

/* declaration for Name Table */

dcl  1 ntable (sntp -> snt.ntsize) ctl(ntp),
     2 vacant bit(1), /* vacant bit */
     2 delete bit(1), /* delete bit */
     2 nchar fixed bin(17),          /* length of name */
     2 nhrp bit (18); /* relative pointer to name header */

/* declaration for segment table */

dcl  1 stable (sntp -> snt.stsize) ctl (stp),
     2 vacant bit(1), /* vacant bit */
     2 shrp bit(18); /* relative pointer to segment header */
```

```
/* declaration for name header */

dcl  1 nhead ctl (nhp), /* one for each name */
     2 nchar fixed bin(17), /* number of chars */
     2 nblock fixed bin(17), /* number of sfnb's-for name */
     2 nbrp bit(18), /* rel pointer to last sfnb for name */
     2 name char(nhp-> nhead.nchar); symbolic call name */

/* declaration for a segment header */

dcl  1 shead ctl (shp), /* one for each segment */
     2 segno fixed bin(17), /* segment number */
     2 slotsize bit(17), /* number of slots */
     2 slotlist (shp -> shead.slotsize) bit(17),
/*   list of slot numbers */
     2 ncdpath fixed bin(17), /* chars in directory path */
     2 dpath char (shp -> shead.ncdpath),
/*   directory path name */
     2 ncentry fixed bin(17), /* chars in entry name */
     2 entry char (shp -> shead.ncentry), /* entry name */
     2 delsw bit(1), /* segment delete switch */
     2 lock bit(2), /* segment usage lock */
     2 nblocks fixed bin(17), /* number of sfnbs for segment -
     2 sbrp bit(18), /* rel pointer to last sfnb for segment */
     2 drp bit(18), /* rel pointer to last daughter for segment */
     2 uid bit(70); /* unique identification of segment */

/* declaration for an snb */

dcl  1 snb ctl (snbp),
     2 gsw bit(1), /* global usage switch */
     2 rsw bit(1), /* related switch */
     2 isw bit(1), /* initiation switch */
     2 csw bit(1), /* create switch */
     2 ssw bit(1), /* search switch */
     2 ncd fixed bin(17), /* chars in path */
     2 dpath char (snbp->snb.ncd), /* pathname of directory */
     2 nce fixed bin(17), /* chars in entry name */
     2 entry char (snbp->snb.nce), /* entry name */
     2 ncu fixed bin(17), /* chars in unique name */
     2 uname char (snbp->snb.ncu), /* unique name */
     2 msegno bit(18), /* mother's segment number */
     2 ringno bit(18), /* ring of availability */
     2 nhrp bit(18), /* rel pointer to name header */
     2 segno bit(18), /* segment number */
     2 (prrp,nrrp) bit(18), /* related pointers */
     2 (psrp,nsrp) bit(18), /* segment pointers */
     2 (pnrp,nnrp) bit(18); /* name pointers */
```