Published 01/17/67
(Supersedes: BG.1, 5/9/66)

## Identification

The Known Segment Table (KST)
R.C. Daley, D.M. Ritchie

## Purpose

The Known Segment Table (KST) is a paged segment within each process. It contains information sufficient to allow segment control to service segment faults for each segment with an entry in the table. This information consists essentially of the mapping between a segment number - all that is supplied by the hardware at the time of a segment fault - and the unique identifier and location in the directory hierarchy of the associated segment. A KST entry also contains less vital information gleaned from the directory branch pointing to the segment at the time of the creation of the entry.

## Introduction

Segments are entered in the KST primarily as a result of linkage faults while attempting to access the segment. The hardcore supervisor is prelinked, however; linkage faults are not allowed while executing in the hardcore ring. Thus a segment in this ring might never have an entry in the KST. To allow segment faults to be processed for these segments, there is a system-wide table, the Hardcore Segment Table (HST), which serves in lieu of a KST for these segments.

A segment is known to a process if

1. It has been assigned a segment number in the process, and
2. A missing-segment fault on the segment can be handled properly.

Operationally, then, a segment is known to a process if it has an entry in the KST or HST.

There is a KST for each Multics process; the branch pointing to this segment is kept in a special directory associated with each process called the Process Directory (BD.6.09) which contains segments which, though part of the process, are normally kept hidden from the user. The KST is maintained exclusively by segment control.

## Structure of the KST

The KST is organized into the following sections.

1. The KST entries themselves.

2. An entry table, with one entry for each known segment as well as some vacant entries corresponding to segments that have been removed from the table. Entries in the entry table contain a pointer to the actual entry for a segment, if it corresponds to a known segment, or link information threading together the vacant entries otherwise,

3. A name and an ID hash table, present for reasons discussed below.

4. A header section, which contains information pertinent to the entire table, such as the sizes of various subtables and their locations.

## KST Entries

The following is a list of items which are stored for each entry in the KST and is followed by a detailed description of each item.

1. List of symbolic names for this segment

2. Unique identifier of this segment

3. Effective mode

4. Protection list

5. Directory-segment switch

6. Directory-segment-hold switch

7. Transparent-usage switch

8. Segment number of directory segment which is immediately superior to this segment

9. Index relative to immediately superior directory of branch defining this segment

10. Date and time branch was last modified

11. Count of the number of segments which are currently known to this process and to which this directory segment is immediately superior

Entries in the KST are indexed by segment number through the entry table. As a result, segment numbers are not explicitly stored with each entry in the KST. The above items are now described in detail.

1. List of symbolic segment names--these names are stored for directory segments only; they are used by segment control in communicating with directory control. When a new name is used by the process for a directory segment, the new name is appended to the list of names already existing for this segment.

2. Unique identifier for this segment -- this item uniquely identifies the precise copy of the segment to which the process is currently committed. This identifier is constructed by directory control so that no two segments within any version of Multics will ever have the same identifier. When segment control attempts to reactivate an inactive segment, this identifier is used to insure that only the correct segment is used.

3. Effective mode -- this item contains four switches corresponding to the four access attributes (REWA), and is used to control access to the segment by the current process. This item is kept up to date by comparing the date-and-time-branch-last-modified item in the AST (Active Segment Table; BG.2) entry for the segment, and refreshing when necessary from the branch pointing to the segment.

4. Protection list -- this section contains a list of items extracted from the segment's access control list. The protection list allows computation of the access attributes of the segment according to the ring mechanism (cf.BD.9) The first item is the lower access ring, then an optional upper access ring number, then an optional call access ring number, finally an optional list of segment offsets corresponding to entry points for this (procedure) segment. This item, like the last, is kept up-to-date by use of the date-and-time-branch-last-modified item.

5. Directory-segment switch--if the segment corresponding to this KST entry is a directory segment, this switch is set ON.

6. Directory-segment-hold switch--for reasons described below, segment control normally removes entries in the KST for directory segments as soon as they are no longer needed by inferior segments. However, if a user wishes to keep a directory (such as a working directory) known to his process, he may do so by specifying that this switch be turned ON. When this switch is in ON, segment control will not remove the corresponding KST entry until the user explicitly requests its removal. Proper use of this switch can save the time associated with deactivating and re-activating frequently used directories such as the user's working directory. If the directory-segment switch is OFF, the directory-segment-hold switch is ignored by segment control.

7. Transparent-usage switch-- this switch is turned ON to prevent the file system, on behalf of the current process, from changing the time-last-used or the time-last-modified in the branch for the segment. For example, the backup system might use this switch to dump a file on magnetic tape without changing the time-last-used of the file. This switch is also set ON for all directory segments, since only directory control can determine when a directory has been "modified".

8. Segment number of directory segment immediately superior to this segment--when a process references an inactive segment, a segment fault occurs, and segment control is called to activate the segment. (See section BG.0). To do this information must be retrieved from the branch in the immediately superior directory which points to the segment. This item identifies the needed directory so that it can be read by directory control for segment control.

9. Index of segment branch in superior directory-- this item, together with the previous item, form a complete pointer to the branch defining the segment.

10. Date and time branch last modified--this item is used to assure that the access attributes and protection list (items 3 and 4) are up to date. If the current process has the most recent information, as determined by the equality of this item and the corresponding item in the Active Segment Table (AST) entry for the segment, then the access control information is up to date. If these entries are not equal, then some process may have changed the access control situation and the attributes and protection list must be recomputed. The comparison takes place when a segment fault occurs.

11. Count of the number of segments which are currently known to the process and to which this directory segment is immediately superior--each time a new segment is added to the KST which is immediately inferior to the directory segment represented by this entry, this count is incremented by one. Whenever an immediately inferior segment is removed from the KST, the count is decremented by one. If this count is reduced from one to zero, and the directory-segment-hold switch is OFF, then this entry is removed from the KST. The removal of this segment may cause the entry for the next superior directory segment to be removed and so on. In a KST entry for a non-directory segment, this count is always zero.

This automatic removal of directory segment entries from the KST prevents the KST (and the associated descriptor segment) from being overloaded with segments no longer required by the process. Segments numbers freed by removal of KST entries are reused as new segments become known to the process thus keeping the KST and its associated descriptor segment tightly packed.

The Entry Table

All the KST entries are kept in a common area in no particular order; but there is an entry table, indexed by segment number, which contains points to the actual KST entries. Each entry in the entry table contains

1. vacant switch

2. entry pointer or next-vacant-entry pointer

3. last-vacant-entry pointer

If the vacant entry switch is OFF, item 2 points to the KST entry for this segment number. If the switch is ON, there is no such entry, and items 2 and 3 link this entry into a doubly-threaded list of vacant entries.

Since the number of segments known to a process may grow or shrink, the entry table must grow or shrink accordingly. When it becomes necessary to change the size of the entry table, the new table is allocated, the existing information is copied into it, and the old table is discarded. To save time the original size is made large enough so that the table will not usually have to grow.

The Hash Tables

As one might expect, the KST is frequently accessed by segment number or by symbolic segment name. The KST is also accessed by unique identifier for a not-so-obvious reason. Before entering a new entry in the KST, segment control searches the KST for an entry having the same unique identifier as the new entry about to be entered. If an entry already in the KST has the same identifier as the new entry, the name of the new entry is merely appended to the list of symbolic names of the existing segment and no new KST entry is made. This situation is brought about when a process uses a new name to refer to a directory segment which is already known to the process under one or more different names.

Access to a KST entry by segment number is simple because the KST is indexed by segment number. For example, if "i" is a segment number and "a" is an array of entries, then one could logically refer to the entry for segment "i" as "a (i)". However, when it becomes necessary to find an entry in the KST which has a specified name or identifier, the KST must be searched.

To speed searching for entries, a hash-coded table-lookup technique is used. There is a hash table for unique identifiers and one for names. Each hash table entry contains the following information.

1. vacant switch

2. deleted switch

3. segment number

When a name or unique identifier is to be searched for, it is hash coded to produce a pseudo-random number which is used as an index in the corresponding hash table. If the vacant switch is ON, the desired entry does not exist; if it is OFF, the KST entry for the segment number obtained from the hash table entry is examined for a match with the desired name or ID. If no match is found, the next higher entry in the hash table is treated in the same way until a match is found or a vacant entry is reached. If the deleted switch is ON in any of the hash table entries that entry is simply passed over; it does not terminate the search.

When an entry in the KST is added, its name and ID are hashed and used to find an entry in the proper hash table as above. If this entry is vacant or deleted, both the vacant switch and deleted switch are set OFF and the segment number of the new entry is put in place. If the hash table entry is not vacant or deleted, consecutive hash table entries are tested until a vacant or deleted one is found.

If the number of non-vacant entries in the hash tables exceeds a preset percentage of the table, larger hash tables must be constructed by rehashing the names and ID's of each KST entry. Similarly if the number of non-vacant entries falls below a preset percentage, a smaller table is constructed. In either case, the old table is discarded.

## The KST Header

The fixed-length header portion contains the following information:

1. size of the entry table
2. pointer to entry table
3. size of, number of used entries in, and pointer to name hash table
4. size of, number of used entries in, and pointer to ID hash table
5. highest assigned segment number
6. pointer to first member of vacant entry list

These items are used as follows:

1. entry table size - when a segment number about to be assigned exceeds this item, a new table must be created

2. entry table pointer - used to access the entry table

3. hash table information - to manipulate the hash tables

4. same

5. highest assigned segment number

6. pointer to first member of vacant entry list

Whenever an entry is added to the KST and item 6 is null, the entry is assigned the segment number next higher than item 5, and item 5 is updated. If item 6 is not null, the entry pointed to by item 6 is used and the vacant entry list is rethreaded. In either case the corresponding entry in the entry table is marked nonvacant and entries for the segment are added to both hash tables.

Whenever an entry is deleted from the KST, its hash table entry is marked "deleted" if the next consecutive hash table entry is non-vacant or "vacant", if the next higher hash table entry is also vacant. The entry table entry for the segment is then marked "vacant" and is added to the vacant-entry list.

The Hardcore Segment Table

The hardcore segment table (HST) is a system wide table constructed during Multics system initialization. This table contains an entry for each segment of the hardcore supervisor. Each entry in the HST contains the segment number of the segment, its unique identifier, the descriptor control information, and the status of the segment (defined below). Information collected in the HST allows segment control to manage the descriptor segment for the hardcore ring in the absence of a KST. The main advantage of this scheme is that much of the work involved in process initialization can be done by the process being initialized. This means that a process may be created with an empty KST and that segments required for normal execution may be filled in by the process itself using the normal file system primitives.

A secondary advantage is that segments listed in the HST need not have corresponding entries in the KST of each Multics process. This results in a significant saving of space since most of the segments of the hardcore supervisor are not normally referenced by user programs.

The HST is used in two distinct ways. When a segment fault occurs for one of the segments in the hardcore ring, the HST is consulted. The entry corresponding to the segment number on which the fault occurred yields a unique identifier which may be used to access the AST in order to load the segment (see BG.2 for a discussion of the AST and BG.3.01 for a discussion of seg fault). The access control information in the HST is placed in the segment descriptor word (SDW) for the segment, completing the treatment of the fault.

If a segment fault occurs in the hardcore supervisor, the HST is always consulted first. Since the fault can be serviced from information in the HST, it is now possible to service missing segment faults for the hardcore stack and the KST of a loaded process.

The HST is also used when a segment is made known (see section BG.3.01 for a discussion of makeknown). Although hardcore segments can refer to each other without linkage faults, a linkage fault will occur when a procedure in an outer ring references a hardcore ring segment - for example, to call a supervisor routine. This fault will ultimately result in an attempt to make the hardcore segment known, that is, to create a KST entry for it. Originally, there are no KST entries for hardcore segments, but whenever a KST entry is created for a hardcore segment, the segment number assigned to this segment in the KST must be the same as that used within the hardcore ring. Thus makeknown searches the HST for a unique identifier the same as that given in its calling sequence; if this unique ID is found, the corresponding segment number will be assigned. Otherwise, the first available number will be used.

Each hardcore segment table entry contains the following information. Entries are indexed by segment number.

1. unique ID- this item is used as described above

2. PST index- there are a few segments which belong to the hardcore ring but are different segments for each process. These include the KST itself, the hardcore stack, and the process data segment. The unique ID's of these segments cannot appear in the HST, which is a system-wide table. Instead, this item, when nonzero, gives an index in the PST (see BG. 2) entry of the current process, where the unique ID of these segments is kept.

3. Status- this item specifies whether the corresponding segment is always active, loaded, or wired down. It simplifies the handling of segment faults for hardcore segments.

4. SDW access field - this item corresponds to the access control information kept in the KST of each known segment. It consists of the bits to be placed in the access field of the hardcore ring SDW of the segment.

5. Enforced access switch- see item 6

6. Enforced access mode- certain segments must never cause a segment fault when accessed in any ring. These include, for example, the fault and interrupt handlers. When item 5 is ON the SDW for this segment in any but the hardcore ring will contain the access bits specified by this item, never a segment fault.

7. Current segment length- used in preparing the boundary field of an SDW.

The HST hash table and header

During makeknown the HST is searched by unique identifier. To speed this searching a hash table is provided. The unique identifier is the item which is hashed. The structure and use of this hash table are the same as those in the KST.

The HST also has a header section containing information pertaining to the whole table; it contains the following information:

1. number of entries

2. highest always-accessible segment

3. size of hash table

1. Number of entries - segments in the HST are assigned numbers sequentially. Thus, a segment with a number greater than this item cannot be in the HST.

2. Highest always-accessible segment- this is the highest segment number in the HST with the Enforced access switch ON.

3. Hash table size- this item is used in the same way as the sizes of the ID and name hash tables for the KST.

## PL/1 Implementation of the KST

### The KST Header

The entire KST is declared as controlled storage containing a fixed number of pointers to tables of variable length. The header, or fixed-length, portion of the KST is declared by the following statement:

```
dcl 1 kst ctl (kstp),

    2 ec fixed bin (17),      /* entry count */

    2 etp ptr,      /* pointer to entry table */

    2 hcname fixed bin (17), /* size of name hash table */

    2 huname fixed bin (17), /* used entries in name hash table*/

    2 hpname ptr,            /* pointer to name hash table */

    2 hcid fixed bin (17),   /* size of ID hash table */

    2 huid fixed bin (17),   /* used entries in ID hash tables */

    2 hpid ptr,              /* pointer to ID hash table */

    2 highseg fixed bin(18), /* highest segment number */

    2 freeseg fixed bin(18), /* first free segment number */

    2 kstarea area ((N));    /* KST allocation area */
```

The identifier kstarea defines an area of storage within the KST segment in which all additional information is allocated. (In the context of this writeup, the capital letter "N" is used to denote an integer constant which must be decided upon before compilation.)

### The Entry Table

The data layout for the entry table is indicated by the following statement:

```
dcl 1 et (kstp->kst.ec) ctl (etp),

    2 vs bit (1),            /* vacant switch */

    2 ep bit (18),           /* entry pointer or forward
                                vacant-list pointer */
```

```
    2 bp bit (18);          /* backward vacant-list pointer */
```

The Hash Tables

The data layout for both hash tables is given in the following statements:

```
dcl 1 htname (kstp→kst.hcname) ctl (hnp),
    2 vs bit (1),           /* vacant switch */
    2 ds bit (1),           /* deleted switch */
    2 segno bit (18);       /* segment number */

dcl 1 htid (kstp→kst.hcid) ctl (hip),
    2 vs bit (1),           /* vacant switch */
    2 ds bit (1),           /* deleted switch */
    2 segno bit (18);       /* segment number */
```

The KST Entries

Each KST entry is divided into two parts, a fixed-length part and a variable-length part. Since a segment may have several names of variable length, the list of segment names may grow and shrink while the remainder of the KST entry is unchanged. As a result, the names are stored separately from the rest of the KST entry. The fixed length portion of a KST entry is defined by the following statement:

```
dcl 1 kste ctl (kstep),
    2 namep bit (18),       /* pointer to list of names */
    2 id bit (70/*uidsize*/), /* unique ID */
    2 mode bit (5), /* effective mode */
    2 psize bit (17),       /* size of protection list */
    2 plrp bit (18),        /* pointer to protection list */
    2 dirsw bit (1),        /* directory - segment switch */
    2 dshsw bit (1),        /* directory - hold switch */
```

```
     2 tusw bit (1),          /* transparent-usage switch */

     2 xsegno bit (18),       /* number of immediately superior
                                 directory */

     2 xbranch bit (18),      /* slot number of this segment
                                 in xsegno */

     2 dtbm bit (52/*dtsize*/),
                              /* date and time branch last
                                 modified */

     2 infcount bit (17);     /* count of known inferior
                                 segments */
```

The following declaration specifies the storage layout of a single segment name in a linked list of names:

```
dcl  1 namst ctl (np),

     2 nlen bit (17),         /* length of name */

     2 name char (np→ namst.nlen),
                              /* the name itself */

     2 nrp bit (18);          /* pointer to next name*/
```

The following declaration specifies the storage layout of a protection list:

```
dcl plist (kstep→ kste.psize) bit (18) ctl (plp);
```

PL/I Declaration for the HST

The following defines the storage layout of the HST:

```
dcl  1 hss ctl (hssp),        /* HST structure */

     2 hscnt fixed bin (17),  /* number of hardcore segments*/

     2 nmm fixed bin (17),    /* number of always accessible
                                 segments */

     2 htcnt fixed bin (17),  /* hash table size */


     2 ht (hssp→hss.htcnt),   /* hash table */

     3 vs bit (1),            /* vacant switch */

     3 ds bit (1),            /* deleted sw. (not used; for
                                 compatability) */
```

```
    3 segno bit (18),          /* the subscript to hst */

  2 hst (hssp→ hss.hscnt),     /* the table itself */
    3 uid bit (70),            /* unique ID /*
    3 pste_index bit (3),
    3 status bit (3),
    3 ds_acc bit (6),          /* access bits */
    3 easw bit (1),            /* enforced access switch */
    3 ea_acc bit (6),          /* access bits for enforced access*/
    3 csl bit (12);            /* current segment length */
```