To: Distribution

From: Steve Webber

Subject: Known Segment Table (KST) Change Proposal

Date: 06/18/74

Introduction

This MTB describes some modifications to the format of the Known Segment Table (KST) that have been proposed over the last few months. The changes are primarily intended to 1) allow faster computation of effective access and 2) provide a more convenient structuring for the pre-linking mechanism if and when it is ever implemented.

Access Information in the KSTE

The first proposed change is to extend the KST entry (KSTE) to include the following items:

1) effective mode (R, E, W, R1, R2, R3)

2) date-time-branch-modified (DTBM)

3) extended access (e.g. ADROS for message segments or SMA for directories)

The intent here is to calculate the effective mode much less frequently (in the most common case only at initiate time). The effective mode will be placed in the KSTE along with the time the branch was last modified (which is an upper bound on the time the access was last changed) at initiate time. Each time the effective mode is wanted it can easily be obtained directly from the KSTE unless the DTBM in the branch is later than the DTBM in the KSTE which indicates that the mode should be recomputed. (In the rest of this writeup the term "effective mode" is used to mean the actual bits placed in the SDW, including the ring brackets. The mode returned by entries such as hcs_$fs_get_mode, often called effective mode, has the ring bracket computation performed. This computation is very easy and for all practical purposes the final needed mode exists in some form in the KSTE.) If the branch has been modified, the effective mode is recalculated and saved anew in the KSTE along with the updated value of DTBM.

There are several consequences of the above change but the primary one is that mode computation will be much faster. One interesting problem is that in order to look at the DTBM in the branch it would appear to be necessary to lock the directory in which the branch resides. (The current scheme requires the directory be locked during the entire mode computation.) This is, in fact, not necessary if the following rules are followed:

1) the online salvager can not move the location of an entry in a directory,

2) the DTBM in a branch should be changed before an ACL list is changed (for maximum effectiveness), and

3) Whenever the DTBM is to be updated in a branch it must be changed, i.e. a mechanism must be provided for waiting until the time has passed, etc. A simple way to insure this works without too much overhead is to add one to the time value (if it didn't change from the last value) until the time goes, say, 250 milliseconds in the future. Only then need we wait for the time to actually "catch up" to our value.

With these rules and a slight restructuring of the entry structure in a directory to put the unique ID (UID) and the DTBM in a single machine word pair, it is possible to check the validity of the DTBM very quickly (i.e. with a CMPAQ machine instruction). Note that there are enough unused bits in the KSTE to add this mechanism today without changing the size of a KSTE.

The effective mode saved in the KSTE would be the actual bits returned by access_mode and would contain any necessary security controls factored in. The security controls will have the effect of preventing read or write permission to the given process for the given segment. This information will be reflected in the actual read and write bits saved in the mode of the KSTE. Since the mode saved in the KSTE would contain ring brackets, it is a simple task to calculate the mode with respect to a given validation level.

A further advantage to placing the mode in the KSTE is that a ring 1 initiate primitive could call a special supervisor entry (through a ring 1 gate) to initiate message segments. This eliminates the need for a bit in the branch specifying the segment is a message segment (something proposed to solve a problem with the security system interface to message segments).

A third component of the access to a segment, the extended access, should also be placed in the KSTE for the same reasons mentioned above. This makes message segment primitives, etc. more efficient but will probably force the size of the KSTE to be extended.

Note that the effective mode for directories consists of the SMA bits and R1 and R2. The inclusion of this information in the KSTE saves a good deal of access calculation done to determine if a user has the right to perform some directory control operation.

Initiated Mode

The second proposed change is to extend the KSTE to include "initiated mode" bits. The intent here is to allow a user to initiate a segment in such a way that he can not accidently modify a segment that he has write permission to. For example, the source segment initiated by a compiler could probably be initiated with R access only, thereby avoiding accidental modification. Similarly, backup could use this feature to protect user files.

The actual implementation would probably be simply placing the AND of the effective mode and the initiated mode in the SDW at segfault time. The standard initiate calls of today (hcs_$initiate, hcs_$initiate_count and hcs_$makeseg) could be compatibly extended to allow specifying the initiated mode. For hcs_$make_seg, unused bits in the mode parameter could be used. For the other entries unused bits in the copysw parameter could be used. (These parameters are fixed bin (n) where n can safely be extended.)

An important reason for the initiated mode feature is that it enables us to make better use of the cache hardware. The design of the cache (software) implementation specifies that if a segment is being used by more than one process and at least one of the processes has write permission (in the actual SDW) then the segment can not be cached. This means that references to a segment by backup (which generally has RW access to files) will cause that segment to be made uncacheable, needlessly. If backup initiated segments to be dumped with R access only the SDW would not (necessarily, see below) have the W bit on and hence not force the segment to be uncached.

The issue of what to do if a segment is initiated with a given mode and it has already been initiated with another mode can only be solved by the following rule:

> The initiated mode can only be changed from a state of less privilege to a state of greater privilege.

That is, it is not possible for one program to "turn off" an SDW bit needed by some other program in the process. This rule is necessary because segments can be used by several separate subsystems within a process that are working asynchronously and it should not be possible for one subsystem to prevent another from operating in its normal manner (e.g. quits followed by work followed by "start"). This rule does not prevent the mechanism

from being useful in the large majority of cases.

Bit Count in KSTE

A third proposal is to extend the KSTE to include the bit count of a segment. This is a trivial change but makes status_$mins and initiate$initiate_count calls more efficient and hence makes linking (as well as user-ring programs) more efficent. The disadvantage here is that the addition of the bit count to the KSTE would force an increase in the size of an entry.

The Reference Name Table

A fourth proposed change is to split the part of the KST needed by segfault and the part of the KST needed by the linker (and find_) into two separate regions. This is of interest both for prelinking as well as removing the linker from ring 0. The proposed split would move all name management items (reference names and initiated pathnames used by kstsrch on behalf of find_) into a separate region that could eventually be removed from the supervisor. This region, the Reference Name Table (RNT), would contain reference names, pathnames and hash tables to facilitate searches. Associated with each name is:

1) the segment number,

2) a bit array saying which rings the name is initiated in,

3) a pointer to the (a) parent pathname,

4) a pointer to the next reference name (pathname) for the segment, and

5) the reference name itself (which may be a pathname).

The KSTE would retain:

1) the unique ID of the segment,

2) The DTBM of the segment,

3) the effective, initiated and extended access modes of the segment,

4) the inferior count (for directories only),

5) flags (TMS, TUS, TPD, RSW, DIRSW, USED, etc.),

6) the entry pointer (a pointer to the branch for the segment in the parent directory) and

7) the bit count of the segment.

PL/I Declarations

The following PL/I declarations show the detailed proposed format of the KST and RNT (for the first implementation the RNT will still be located in the same segment as the KST, but only to conserve AST entries):

```
dcl 1 kste based (kstep) aligned,

    2 uid bit (36),

    2 dtbm bit (36),

    2 entry_ptr ptr unaligned,

    2 rsw bit (1) unaligned,
    2 dirsw bit (1) unaligned,
    2 used bit (1) unaligned,
    2 infcnt bit (9) unaligned,
    2 bit_count bit (24) unaligned,

    2 tms bit (1) unaligned,
    2 tus bit (1) unaligned,
    2 tpd bit (1) unaligned,
    2 access unaligned,
      3 (R, E, W) bit (1),
      3 (R1, R2, R3) bit (3),
      3 (IR, IE, IW) bit (1),
    2 hash_table_relp bit (18) unaligned,

    2 extended_access bit (36);
```

The extended_access field will contain ring brackets if appropriate.

```
dcl 1 rnte based (rntep) aligned,

    2 iring (0:7) bit (1) unaligned,
    2 segno fixed bin (15) unaligned,
    2 pad bit (12) unaligned,

    2 rel_pointers,
      3 next_name bit (18) unaligned,
      3 parent bit (18) unaligned,
      3 hash_thread bit (18) unaligned,
```

```
      3 null_name_count fixed bin (17) unaligned,

    2 len fixed bin,

    2 name char (1 refer (rnte.len));


dcl 1 kst_seg$ aligned ext,

    2 first_seg fixed bin (15),
    2 last_seg fixed bin (15),
    2 entry_size fixed bin init (6),
    2 free_thread bit (18),
    2 uid_ht_size fixed bin,
    2 uid_ht_relp bit (18),
    2 rnt_relp bit (18),
    2 ksta_relp bit (18),

    2 uid_hash_table,
      3 buckets (0:uid_ht_size-1)) bit (18) unaligned,

    2 ksta (0:N) aligned like kste,

    2 rnt_header,
      3 allocation_area_relp bit (18),
      3 name_ht_relp bit (18),
      3 name_ht_size fixed bin,
      3 seqno_ht_relp bit (18),
      3 seqno_ht_size fixed bin,
      3 pad (3) fixed bin,

      3 srules (0:7) bit (18),

      3 name_hash_table,
        4 buckets (0:name_ht_size-1)) bit (18) unaligned,

      3 seqno_hash_table,
        4 buckets (0:seqno_ht_size-1)) bit (18) unaligned,

      3 allocation_area fixed bin (35);
```

Note that the hash table sizes will be variable and set by a value in the PIT for the process. Similarly, the value that rnt_relp assumes will be variable so that it will be possible to control N (again from the PIT) so that a 1K KST/RNT is possible for small processes.

The hashing scheme proposed consists of a relatively small hash table indexed by hash index generated from the item being hashed. The indexed cell of the hash table points to a list of

all entries in the KST or RNT with the same hash index. This design allows the storage used by the hash function to start out small and grow as more entires are added to the given table. If the hash table is initially allocated with a reasonably sufficient size (a PIT variable) there should be little loss in efficiency, especially if a page fault is avoided.