Published: 05/24/67

Identification

Segment Control, The Segment Utility Module
R.C. Daley, D.M. Ritchie

Purpose

The segment utility module of segment control contains several primitives for use in manipulating the active segment table (AST) and the known segment table (KST). All of these primitives are privileged to modules of the hardcore supervisor and are provided for the exclusive use of segment control modules.

Primitives

The following is a list of the primitives of the segment utility module and is followed by a detailed discussion of each primitive.

1. sum$searchast
2. getastentry
3. getastentry$delastentry
4. maketrailer
5. sum$nsrchkst
6. sum$idsrchkst
7. alloc_sst
8. alloc_sst$free_sst

1. searchast

To find an entry in the AST with a specified unique identifier the following call is provided.

```
call sum$searchast(id,found,hsi,astep,errcode);
```

In this call, id is the unique identifier of the desired AST entry and found is a switch to be set upon return to the calling program indicating whether or not the specified entry is in the AST. If the desired AST entry is found, found is set ON, the index within the hash table (see BG.2) at which the entry was found is returned as the value of hsi and a pointer to the AST entry is returned as the value of astep. If the desired entry is not found in the AST, found is set OFF and an index within the hash table at which the entry may subsequently be created is returned as the value of hsi.

If searchast finds the AST entry, it leaves it locked upon return. If any use is to be made of hsi (for example, to create or delete the entry) the hash table must be locked before calling searchast and not unlocked until hsi is no longer needed.

2. getastentry

To find an AST entry with a specified unique identifier or create the entry if not found, the following call is provided.

```
astep = getastentry (kstep, did);
```

In this call, kstep is a pointer to the KST entry defining the desired segment and did is the identification of a device to which the segment is to be moved. If did is zero, no segment moving is specifically requested by the caller. A pointer to the desired AST entry is returned to the caller as the value of astep.

Upon receiving this call, another utility routine (searchast) is called to search the AST for the desired entry. If the desired entry is found, a check is made to see if the entry is currently included on the list of candidates for removal and if it is, the entry is removed from the list.

If the desired AST entry is not found in the AST, getastentry must activate the directory segment superior to the segment specified by kstep. A pointer to this directory is found in the KST entry and getastentry calls itself with this pointer and a device identifier of zero. Upon normal return from this call, a new AST entry is created for the originally specified segment. The necessary information is obtained from the branch of the segment by calling a directory control primitive (activinfo$rdbranch).

Whether or not an AST entry had to be created, did is checked; if it is non-zero, and the segment is not already being moved, a call is made to a primitive of multilevel move control (move_advice) to determine whether the move is wise. If not, no moving is done, but if so, an additional active file trailer (AFT) is appended to the AST entry to prepare the segment to be moved.

An AST created by this procedure represents an active but unloaded segment. The decision as to whether or not the segment should subsequently be loaded is left to the caller of getastentry.

Upon return, the AST entry discovered or created by getastentry is locked.

3. delastentry

To deactivate a segment by deleting its corresponding entry from the AST, the following call is provided.

```
call getastentry$delastentry(uid, errcode);
```

In this call, uid is the unique identifier of the segment whose AST entry is to be deleted.

The count of the number of inferior active segments must be zero in the specified AST entry, i.e., the segment must be a terminal node in the hierarchy.

When this call is received, the segment may be loaded as well as being active. If this is the case, the segment must first be unloaded by calling a primitive of page control (pcfreecore) to remove any pages of the segment which are currently in core. Upon return from this call, the segment is unloaded since page control always unloads a segment when removing the last page from core.

Once the segment is unloaded, the branch associated with the segment is updated with information in the AST entry by calling a directory control primitive (activinfo$wrbranch). Upon return from this call, the AST entry is deleted by delastentry and control is returned to the calling program.

When delastentry is called, the AST entry of the segment being deleted and the AST entry of its immediate superior in the hierarchy must be locked. On return from delastentry, the parent AST entry is still locked.

4. maketrailer

To create a process trailer for an AST entry, the following call is provided.

```
call maketrailer (segno, astep);
```

Here segno is the segment number in the current process of the segment for which the trailer is being made, and astep is a pointer to the AST entry for the segment.

This routine calls alloc_sst to get space in which to create the trailer, places segno and astep in the proper spots, and establishes in the trailer a pointer to the PST entry for this process. The trailer is then threaded into the linked list of trailers attached to this AST entry, and into the list of trailers attached to this PST entry (see BG.2).

The AST entry must be locked when maketrailer is called, and it will remain locked on return.

5. sum$nsrchkst

To search the KST for an entry containing a specified symbolic segment name, the following call is provided.

```
call sum$nsrchkst(name, found, hsi, segno, kstep, errcode);
```

In this call, name is the segment name of the desired KST entry and found is a switch to be set upon return to the calling program indicating whether or not the desired entry is in the KST. If the desired entry is found, found is set ON, the index within the hash table (see BG.2) at which the entry was found is returned as the value of hsi, the segment number corresponding to the KST entry is returned as the value of segno and a pointer to the KST entry is returned as the value of kstep. If the desired entry is not found in the KST, found is set OFF and an index within the hash table at which the entry may subsequently be created is returned as the value of hsi.

6. sum$idsrchkst

To search the KST for an entry containing a specified unique identifier, the following call is provided.

```
call sum$idsrchkst(id, found, hsi, segno, kstep, errcode);
```

In this call, id is the unique identifier of the desired KST entry. The remaining parameters are as defined for the previous call (nsrchkst).

Note so that the KST is a per process table, so that locks are not necessary in accessing it; in particular, the care needed in managing the hash table lock in calling searchast need not be taken for idsrchkst and nsrchkst.

7. alloc_sst

In order to obtain space in which to create an SST entry, the following call is provided.

```
call alloc_sst (code, ptr, errcode);
```

Here code is a number indicating for what type of entry space is desired:

if code =	1	space wanted for AST entry
	2	AFT trailer
	3	process trailer
	4	DST entry
	5	PST entry
	6	special segment list

Upon return, ptr contains a pointer to the base of the allocated space.

This routine tries first to obtain the needed space from the free storage area of the SST; if this is insufficient, it finds the AST entry at the top of the removal list, calls delastentry to return it to free storage, and tries again to allocate the desired space.

Except for a special segment list, the size of each structure that may be specified is fixed. The size of a special segment list is contained in the PST entry for the process doing the allocating, so the PST entry must be created first.

8. alloc_sst$free_sst

To return an SST structure to the free storage area, the following call is provided.

```
call alloc_sst$free_sst(code, ptr, errcode);
```

Here code specifies a type of SST entry, exactly as in the discussion of alloc_sst, and ptr points to the base of the area being freed.