THE MITRE CORPORATION

BEDFORD, MASS.

MEMORANDUM

TO:	E. L. Burke	DATE:	27 February 1975
FROM:	W. L. Schiller	MEMO NO.:	D73-M798
SUBJECT:	Multics Design Note #12 - Removing Pages from the User Environment		
COPIES TO:	Distribution		

Multics Design Note #12, Removing Pages from the User Environment, is attached.

[illegible]

W. L. Schiller
Intelligence and
Information Systems

WLS:jkl

Attachments

Distribution:

S. R. Ames, Jr.
D. E. Bell
E. H. Bensley
K. J. Biba
E. L. Burke
C. S. Chandersekaran
M. Gasser
C. D. Jordan
L. J. LaPadula
S. B. Lipner
J. K. Millen
R. D. Rhode
W. L. Schiller
D. F. Stork
J. C. C. White

Attachment 1

Text of Design Note #12

In the current Multics there is no attempt to hide from the user the fact that segments are implemented with pages. In fact, "records used" (the terms page and record are used interchangeably) is the basic measure of a segment's size, and it is used for storage allocation and accounting. To further complicate matters, pages of zeros are not included in records used. Thus, an area of zeros in a segment may or may not be reflected in records used depending on its size, the page size, and its location relative to page boundaries.

Although the use of paging as an implementation mechanism is clearly justified, not hiding paging from the user adds considerable bulk, if not complexity, to the description of the user interface. Inspection of the current top level storage system specification (Multics Design Note #11) supports this claim. Since excessive complexity in the top level specification may adversely affect the lower level specifications and the abstract implementation of the kernel, as well as hinder certification, we should consider removing records used from the user environment. The result should be a simpler, but less compatible, specification.

An Alternative Specification

The attached specification is the result of removing pages from the Design Note #11 specification. Since quota is based on pages, the entire quota mechanism has also been eliminated. An alternative mechanism for accounting and controlling resource allocation will be discussed at the end of this note.

The alternative specification is clearly much more compact than the original - physically it is about half as big. The following segment attributes (primitive V-functions) have been removed:

Branch_records_used

Branch_page_exists

Branch_TRP

Branch_last_time

Branch_quota

Branch_quota_used

Since quota is gone the following hidden V-functions are no longer necessary:

Terminal_seg

Quota

Quota_used

User visible V-functions that have been eliminated are:

Dir_quota

Dir_quota_used

Seg_records_used

Seg_current_length

Seg_TRP

The removed O-functions are:

Give_quota

Remove_quota

Create_upgraded_segment

O-functions that have been simplified because records used is no longer maintained are:

Write

Add_ACL_element

Remove_ACL_element

Create_segment

Delete

And finally, the following implementation V-functions are gone:

NPFA

RPFA

NPFB

RPFB

Note that the implementation V-functions ACLE_offset and Branch_offset are still used.

Accounting

The supervisor must compensate for the removal of the quota mechanism from the kernel. As long as the kernel prevents segments from growing beyond their "max. length" attribute (as currently specified) and the kernel functions are only available to the supervisor (not currently specified), the supervisor can control a user's ability to consume storage resources by monitoring his requests to create segments and set/change their max. length attribute. The user can be charged for the aggregrate max. length of his segments, not for their records used.

This won't fly! [illegible]

1. Compatibility

2. [illegible] control

3. Pressing wrong [illegible]

This new approach to accounting changes the interpretation (transparent to the kernel) of max. length and segment overflow. Currently segment overflow (an attempt to read or write past max. length) results in an error message at the user's terminal - the supervisor makes no attempt at corrective action (unless it is the stack segment that overflowed). With the new approach segment overflow is simply a signal to the supervisor to increment the segment's max. length by some amount, assuming the resource allocation policy permits it, and to update accounting information. The segment overflow is transparent to the user unless his storage allocation is exhausted or his segment's max. length has reached a (possibly user defined) upper bound on max. length (max. max. length?). There is a tradeoff involving the cost of the supervisor's handling of a segment overflow exception and the amount by which it increments max. length.

This accounting scheme does not recognize the fact that areas of zeros in a segment may not require physical storage space, and it penalizes users who do not keep a segment's max. length down near its current length - the highest address containing meaningful non-zero data. Also, it does not permit a segment to be shortened (with respect to accounting charges) simply by storing zeros into it, an explicit call to the kernel's Set_max_length function is required.

Since the kernel is no longer controlling storage allocation it can not guarantee that the system will not run out of storage. As we all remember from the early days of the PDP-11/45 kernel design, if users can determine that the system is out of disk space a one bit communication channel that violates the *-property exists. I do not think this is much of a problem because the kernel will be able to audit disk space usage and notify the SSO or SA when the situation starts to look bad.

One aspect of removing pages from the user environment that has not been thought out is how the supervisor can allocate space to users. I do not think it will be possible to move storage allocations across security level changes in the hierarchy the way the move quota function can currently increase an upgraded directory's quota.

Why not make [illegible] work on an accounting basis rather than [illegible] [illegible] basis?

Attachment 2

Figures of Design Note

```
BRANCH_(dir_uid, entry) Hidden_V_functions:

BRANCH_INUSE: "true" or "false"
BRANCH_UID: unique identifier
BRANCH_TYPE: "data" or "directory" or "msg_segment"
BRANCH_SECURITY_LEVEL: security level
BRANCH_RING_W: ring number
BRANCH_RING_R
BRANCH_RING_E
BRANCH_RING_MA
BRANCH_RING_S
BRANCH_MAX_LENGTH: length
BRANCH_MAX_ACLE: integer
BRANCH_INFERIOR_COUNT: integer


ACL_(dir_uid, entry, acle) Hidden_V_functions:

ACL_USER: user-id
ACL_PROJECT: project-id
ACL_TAG: instance tag
ACL_MODE: set of access modes


PST_(seg) Hidden_V_functions:

PST_INUSE: "true" or "false"
PST_DIR_SEG: seg
PST_ENTRY: entry
PST_INFERIOR_COUNT: integer
PST_UID: BRANCH_UID
PST_TYPE: BRANCH_TYPE
PST_MODIFIED: "true" or "false"
```

```
O-Function: INITIATE(dir_seg, entry, seg);
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
parent_not_initiated! ¬PST_INUSE(dir_seg);
seg_inuse! 'PST_INUSE'(seg);

Effect:
PST_INUSE(seg) = "true";
PST_DIR_SEG(seg) = dir_seg;
PST_ENTRY(seg) = entry;
PST_INFERIOR_COUNT(seg) = 0;
PST_MODIFIED(seg) = "false";
PST_INFERIOR_COUNT(dir_seg) = 'PST_INFERIOR_COUNT'(dir_seg) + 1;
IF (dir_uid ≠ 0) &
   (PST_TYPE(dir_seg) = "directory") &
   BRANCH_INUSE(dir_uid, entry);
   Then PST_UID(seg) = BRANCH_UID(dir_uid, entry);
      PST_TYPE(seg) = BRANCH_TYPE(dir_uid, entry);
   Else PST_UID(seg) = 0;
End;


O-Function: TERMINATE(seg);
Let:
dir_seg = PST_DIR_SEG(seg);

Exception_if:
seg_not_inuse! ¬'PST_INUSE'(seg);
siblings_still_initiated! PST_INFERIOR_COUNT(seg) ≠ 0;

Effect:
PST_INFERIOR_COUNT(dir_seg) = 'PST_INFERIOR_COUNT'(dir_seg) - 1;
PST_INUSE(seg) = "false";
```

```
V_function: [illegible](seg, access_mode);
Let:
uid = PST_UID(seg);
dir_seg = PST_DIR_SEG(seg);
dir_uid = PST_UID(dir_seg);
entry = PST_ENTRY(seg);
dir_dir_uid = PST_UID(PST_DIR_SEG(dir_seg));
dir_entry = PST_ENTRY(dir_seg);
seg_security_level = BRANCH_SECURITY_LEVEL(dir_uid, entry);
seg_type = BRANCH_TYPE(dir_uid, entry);
dir_seg_security_level = BRANCH_SECURITY_LEVEL(dir_dir_uid, dir_entry);

Value:
If PST_INUSE(seg) &
   UID_VALID(uid) &
   [If access_mode = "search";
       Then [ (CUR_SECURITY_LEVEL ≥ seg_security_level) &
              (seg_type = "directory") ];
       Else ACCESS_PERMISSION(dir_uid, entry, access_mode) &
          [Caseof access_mode;
             "read"! [ (CUR_RING ≤ BRANCH_RING_R(dir_uid, entry)) &
                    (CUR_SECURITY_LEVEL ≥ seg_security_level) &
                    (seg_type = "data") ];
             "write"! [ (CUR_RING ≤ BRANCH_RING_W(dir_uid, entry)) &
                    (CUR_SECURITY_LEVEL = seg_security_level) &
                    (seg_type = "data") ];
             "execute"! [ (CUR_RING ≤ BRANCH_RING_E(dir_uid, entry)) &
                    (CUR_SECURITY_LEVEL ≥ seg_security_level) &
                    (seg_type = "data") ];
             "status"! [ (CUR_RING ≤ BRANCH_RING_S(dir_uid, entry)) &
                    (CUR_SECURITY_LEVEL ≥ seg_security_level) &
                    (seg_type = "directory") ];
             "modify"! [ (CUR_RING ≤ BRANCH_RING_MA(dir_uid, entry)) &
                    (CUR_SECURITY_LEVEL = seg_security_level) &
                    (seg_type = "directory") ];
             "append"! [ (CUR_RING ≤ BRANCH_RING_MA(dir_uid, entry)) &
                    (CUR_SECURITY_LEVEL = seg_security_level) &
                    (seg_type = "directory") ];
             "enter"! [ (dir_seg_security_level ≤ CUR_SECURITY_LEVEL ≤ seg_security_level) &
                    (seg_type = "msg_segment") ];
             "examine"! [ (dir_seg_security_level ≤ CUR_SECURITY_LEVEL) &
                    (seg_type = "msg_segment") ];
             "remove"! [ (dir_seg_security_level ≤ CUR_SECURITY_LEVEL ≤ seg_security_level) &
                    (seg_type = "msg_segment") ];
          End;]
   End;]
   Then "true";
   Else "false";
End;
```

```
Hidden_V_function: ACCESS_PERMISSION(dir_uid, entry, access_mode)
Possible_values: "true" or "false"

Value:
If (∃iacle)(ACLE_APPLY(dir_uid, entry, iacle));
   Then Let acle = MIN{iacle|ACLE_APPLY(dir_uid,entry,iacle)};
      If access_mode ∈ ACL_MODE(dir_uid, entry, acle);
         Then "true";
         Else "false";
      End;
   Else "false";
End;


Hidden_V_function: ACLE_APPLY(dir_uid, entry, acle)
Possible_values: "true" or "false"

Value:
If ((ACL_USER(dir_uid, entry, acle) = CUR_USER) |
      (ACL_USER(dir_uid, entry, acle) = "*")) &
   ((ACL_PROJECT(dir_uid, entry, acle) = CUR_PROJECT) |
      (ACL_PROJECT(dir_uid, entry, acle) = "*")) &
   ((ACL_TAG(dir_uid, entry, acle) = CUR_TAG) |
      (ACL_TAG(dir_uid, entry, acle) = "*"));
   Then "true";
   Else "false";
End;
```

```
O_function: WRITE(seg, offset, bit_pattern)
Let:
uid = PST_UID(seg);
dir_uid = PST_UID(PST_DIR_SEG(seg));
entry = PST_ENTRY(seg);

Exception_if:
not_in_as! ¬INAS(seg, "write");
segment_overflow! offset > BRANCH_MAX_LENGTH(dir_uid, entry);

Effect:
UID_READ(uid, offset) = bit_pattern:
PST_MODIFIED(seg) = "true";


V_function: READ(seg, offset);
Possible_values: bit_pattern;
Let:
uid = PST_UID(seg);
dir_uid = PST_UID(PST_DIR_SEG(seg));
entry = PST_ENTRY(seg);

Exception_if:
not_in_as! ¬INAS(seg, "read");
segment_overflow! offset > BRANCH_MAX_LENGTH(dir_uid, entry);

Value:
UID_READ(uid, offset);
```

```
O_function: WRITE(seg, offset, bit_pattern)
Let:
uid = PST_UID(seg);
dir_uid = PST_UID(PST_DIR_SEG(seg));
entry = PST_ENTRY(seg);

Exception_if:
not_in_as! ¬INAS(seg, "write");
segment_overflow! offset > BRANCH_MAX_LENGTH(dir_uid, entry);

Effect:
UID_READ(uid, offset) = bit_pattern;
PST_MODIFIED(seg) = "true";


V_function: READ(seg, offset);
Possible_values: bit_pattern;
Let:
uid = PST_UID(seg);
dir_uid = PST_UID(PST_DIR_SEG(seg));
entry = PST_ENTRY(seg);

Exception_if:
not_in_as! ¬INAS(seg, "read");
segment_overflow! offset > BRANCH_MAX_LENGTH(dir_uid, entry);

Value:
UID_READ(uid, offset);
```

```
O_function: WRITE(seg, offset, bit_pattern)
Let:
uid = PST_UID(seg);
dir_uid = PST_UID(PST_DIR_SEG(seg));
entry = PST_ENTRY(seg);

Exception_if:
not_in_as! ¬INAS(seg, "write");
segment_overflow! offset > BRANCH_MAX_LENGTH(dir_uid, entry);

Effect:
UID_READ(uid, offset) = bit_pattern;
PST_MODIFIED(seg) = "true";


V_function: READ(seg, offset);
Possible_values: bit_pattern;
Let:
uid = PST_UID(seg);
dir_uid = PST_UID(PST_DIR_SEG(seg));
entry = PST_ENTRY(seg);

Exception_if:
not_in_as! ¬INAS(seg, "read");
segment_overflow! offset > BRANCH_MAX_LENGTH(dir_uid, entry);

Value:
UID_READ(uid, offset);
```

```
O_function: WRITE(seg, offset, bit_pattern)
Let:
uid = PST_UID(seg);
dir_uid = PST_UID(PST_DIR_SEG(seg));
entry = PST_ENTRY(seg);

Exception_if:
not_in_as! ¬INAS(seg, "write");
segment_overflow! offset > BRANCH_MAX_LENGTH(dir_uid, entry);

Effect:
UID_READ(uid, offset) = bit_pattern;
PST_MODIFIED(seg) = "true";


V_function: READ(seg, offset);
Possible_values: bit_pattern;
Let:
uid = PST_UID(seg);
dir_uid = PST_UID(PST_DIR_SEG(seg));
entry = PST_ENTRY(seg);

Exception_if:
not_in_as! ¬INAS(seg, "read");
segment_overflow! offset > BRANCH_MAX_LENGTH(dir_uid, entry);

Value:
UID_READ(uid, offset);
```

```
O_function: ADD_ACL_ELEMENT(dir_seg, entry, acle,
                                            user_id, project_id, tag, mode)
Let:
dir_uid = PST_UID(dir_seg);
dir_dir_uid = PST_UID(PST_DIR_SEG(dir_seg));
dir_entry = PST_ENTRY(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "modify");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);
duplicate_acle! 'ACLE_EXISTS'(dir_uid, entry, user_id, project_id, tag, mode);
bad_acle! ¬(0 ≤ acle ≤ 'BRANCH_MAX_ACLE'(dir_uid, entry));
segment_overflow! 'ACLE_OFFSET'(dir_uid,entry,acle,user_id,project_id,tag,mode)
   > BRANCH_MAX_LENGTH(dir_dir_uid, dir_entry);

Effect:
∀iacle(acle ≤ iacle ≤ 'BRANCH_MAX_ACLE'(dir_uid, entry);
   ACL_USER(dir_uid, entry, iacle+1) = 'ACL_USER'(dir_uid, entry, iacle);
   ACL_PROJECT(dir_uid, entry, iacle+1) = 'ACL_PROJECT'(dir_uid, entry, iacle);
   ACL_TAG(dir_uid, entry, iacle+1) = 'ACL_TAG'(dir_uid, entry, iacle);
   ACL_MODE(dir_uid, entry, iacle+1) = 'ACL_MODE'(dir_uid, entry, iacle);
End;
ACL_USER(dir_uid, entry, acle)= user_id;
ACL_PROJECT(dir_uid, entry, acle)= project_id;
ACL_TAG(dir_uid, entry, acle)= tag;
ACL_MODE(dir_uid, entry, acle)= mode;
BRANCH_MAX_ACLE(dir_uid, entry) = 'BRANCH_MAX_ACLE'(dir_uid, entry)+ 1;


O_function: REMOVE_ACL_ELEMENT(dir_seg, entry, user_id, project_id, tag)
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "modify");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);
no_acle! ¬'ACLE_EXISTS'(dir_uid, entry, user_id, project_id, tag, mode);

Effect:
Let acle = ACLE_POSITION(dir_uid, entry, user_id, project_id, tag);
∀iacle(acle < iacle ≤ 'BRANCH_MAX_ACLE'(dir_uid, entry);
   ACL_USER(dir_uid, entry, iacle-1) = 'ACL_USER'(dir_uid, entry, iacle);
   ACL_PROJECT(dir_uid, entry, iacle-1) = 'ACL_PROJECT'(dir_uid, entry, iacle);
   ACL_TAG(dir_uid, entry, iacle-1) = 'ACL_TAG'(dir_uid, entry, iacle);
   ACL_MODE(dir_uid, entry, iacle-1) = 'ACL_MODE'(dir_uid, entry, iacle);
End;
BRANCH_MAX_ACLE(dir_uid, entry) = 'BRANCH_MAX_ACLE'(dir_uid, entry)- 1;
```

```
O_function: ADD_ACL_ELEMENT(dir_seg, entry, acle,
                            user_id, project_id, tag, mode)
Let:
dir_uid = PST_UID(dir_seg);
dir_dir_uid = PST_UID(PST_DIR_SEG(dir_seg));
dir_entry = PST_ENTRY(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "modify");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);
duplicate_acle! 'ACLE_EXISTS'(dir_uid, entry, user_id, project_id, tag, mode);
bad_acle! ¬(0 ≤ acle ≤ 'BRANCH_MAX_ACLE'(dir_uid, entry));
segment_overflow! 'ACLE_OFFSET'(dir_uid,entry,acle,user_id,project_id,tag,mode)
   > BRANCH_MAX_LENGTH(dir_dir_uid, dir_entry);

Effect:
∀iacle(acle ≤ iacle ≤ 'BRANCH_MAX_ACLE'(dir_uid, entry);
   ACL_USER(dir_uid, entry, iacle+1) = 'ACL_USER'(dir_uid, entry, iacle);
   ACL_PROJECT(dir_uid, entry, iacle+1) = 'ACL_PROJECT'(dir_uid, entry, iacle);
   ACL_TAG(dir_uid, entry, iacle+1) = 'ACL_TAG'(dir_uid, entry, iacle);
   ACL_MODE(dir_uid, entry, iacle+1) = 'ACL_MODE'(dir_uid, entry, iacle);
End;
ACL_USER(dir_uid, entry, acle)= user_id;
ACL_PROJECT(dir_uid, entry, acle)= project_id;
ACL_TAG(dir_uid, entry, acle)= tag;
ACL_MODE(dir_uid, entry, acle)= mode;
BRANCH_MAX_ACLE(dir_uid, entry) = 'BRANCH_MAX_ACLE'(dir_uid, entry)+ 1;


O_function: REMOVE_ACL_ELEMENT(dir_seg, entry, user_id, project_id, tag)
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "modify");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);
no_acle! ¬'ACLE_EXISTS'(dir_uid, entry, user_id, project_id, tag, mode);

Effect:
Let acle = ACLE_POSITION(dir_uid, entry, user_id, project_id, tag);
∀iacle(acle < iacle ≤ 'BRANCH_MAX_ACLE'(dir_uid, entry);
   ACL_USER(dir_uid, entry, iacle-1) = 'ACL_USER'(dir_uid, entry, iacle);
   ACL_PROJECT(dir_uid, entry, iacle-1) = 'ACL_PROJECT'(dir_uid, entry, iacle);
   ACL_TAG(dir_uid, entry, iacle-1) = 'ACL_TAG'(dir_uid, entry, iacle);
   ACL_MODE(dir_uid, entry, iacle-1) = 'ACL_MODE'(dir_uid, entry, iacle);
End;
BRANCH_MAX_ACLE(dir_uid, entry) = 'BRANCH_MAX_ACLE'(dir_uid, entry)- 1;
```

```
Hidden_V_function: ACLE_EXISTS(dir_uid, entry, user_id, project_id, tag)
Possible_values: "true" or "false"

Value:
If (∃acle)
   ((ACL_USER(dir_uid, entry, acle) = user_id) &
   (ACL_PROJECT(dir_uid, entry, acle) = project_id) &
   (ACL_TAG(dir_uid, entry, acle) = tag));
   Then "true";
   Else "false";
End;


Hidden_V_function: ACLE_POSITION(dir_uid, entry, user_id, project_id, tag)
Possible_values: acle

Value:
{acle|(ACL_USER(dir_uid, entry, acle) = user_id) &
   (ACL_PROJECT(dir_uid, entry, acle) = project_id) &
   (ACL_TAG(dir_uid, entry, acle) = tag)}
```

```
O_function: CREATE(dir_seg, entry, length, type, security_level)
Let:
dir_uid = PST_UID(dir_seg);
dir_dir_uid = PST_UID(PST_DIR_SEG(dir_seg));
dir_entry = PST_ENTRY(dir_seg);
uid = UNIQUE_NAME;

Exception_if:
not_in_as! ¬INAS(dir_seg, "append");
entry_inuse! 'BRANCH_INUSE'(dir_uid, entry);
invalid_type! (type ≠ "data") | (type ≠ "directory") |
          (type ≠ "msg_segment");
bad_length! (length > MAX_POSSIBLE_LENGTH) |
   (length MODULO LENGTH_INCREMENT ≠ 0);
not_compatible! BRANCH_SECURITY_LEVEL(dir_dir_uid, dir_entry) > security_level;
segment_overflow! BRANCH_OFFSET(dir_uid, entry, "directory")
   > BRANCH_MAX_LENGTH(dir_dir_uid, dir_entry);

Effect:
BRANCH_INUSE(dir_uid, entry) = "true";
BRANCH_UID(dir_uid, entry) = uid;
BRANCH_TYPE(dir_uid, entry) = type;
BRANCH_SECURITY_LEVEL(dir_uid, entry) = security_level;
 If (type = "directory") | (type = "msg_segment");
   Then BRANCH_RING_W(dir_uid, entry) = KERNEL_RING;
       BRANCH_RING_R(dir_uid, entry) = KERNEL_RING;
       BRANCH_RING_E(dir_uid, entry) = KERNEL_RING;
       BRANCH_RING_MA(dir_uid, entry) = CUR_RING;
       BRANCH_RING_S(dir_uid, entry) = CUR_RING;
   Else BRANCH_RING_W(dir_uid, entry) = CUR_RING;
       BRANCH_RING_R(dir_uid, entry) = CUR_RING;
       BRANCH_RING_E(dir_uid, entry) = CUR_RING;
       BRANCH_RING_MA(dir_uid, entry) = 0;
       BRANCH_RING_S(dir_uid, entry) = 0;
BRANCH_MAX_LENGTH(dir_uid, entry) = length;
BRANCH_MAX_ACLE(dir_uid, entry) = 0;
BRANCH_INFERIOR_COUNT(dir_uid, entry) = 0;
BRANCH_INFERIOR_COUNT(dir_dir_uid, dir_entry) =
   'BRANCH_INFERIOR_COUNT'(dir_dir_uid, dir_entry) + 1;
UID_VALID(uid) = "true";
```

```
O_function: DELETE(dir_seg, entry)
Let:
dir_uid = PST_UID(dir_seg);
dir_dir_uid = PST_UID(PST_DIR_SEG(dir_seg));
dir_entry = PST_ENTRY(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "modify");
entry_not_inuse! ¬'BRANCH_INUSE'(dir_uid, entry);
no_delete! (('BRANCH_TYPE'(dir_uid, entry) = "data") &
       CUR_RING > BRANCH_RING_W(dir_uid, entry))) |
   (('BRANCH_TYPE'(dir_uid, entry) = "directory") &
       CUR_RING > BRANCH_RING_MA(dir_uid, entry)));
upgrade_err! (BRANCH_SECURITY_LEVEL(dir_uid, entry) ≠
       BRANCH_SECURITY_LEVEL(dir_dir_uid, dir_entry);
not_empty! ('BRANCH_TYPE'(dir_uid, entry) = "directory") &
   BRANCH_INFERIOR_COUNT(dir_uid, entry) ≠ 0;

Effect:
BRANCH_INUSE(dir_uid, entry) = "false";
BRANCH_INFERIOR_COUNT(dir_dir_uid, dir_entry) =
  'BRANCH_INFERIOR_COUNT'(dir_dir_uid, dir_entry) - 1;
UID_VALID(BRANCH_UID(dir_uid, entry)) = "false";
```

```
O_function: SET_RING_BRACKETS(dir_seg, entry, r1, r2, r3)
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "modify");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);
msg_segment! BRANCH_TYPE(dir_uid, entry) = "msg_segment";
no_change! ((BRANCH_TYPE(dir_uid, entry) = "data") &
   (CUR_RING > BRANCH_RING_W(dir_uid, entry))) |
   ((BRANCH_TYPE(dir_uid, entry) = "directory") &
   (CUR_RING > BRANCH_RING_MA(dir_uid, entry)));
bad_rings! (r1 > r2) | (r2 > r3);
too_low! r1 ≤ CUR_RING;

Effect:
If BRANCH_TYPE(dir_uid, entry) = "directory";
   Then BRANCH_RING_MA(dir_uid, entry) = r1;
      BRANCH_RING_S(dir_uid, entry) = r2;
   Else BRANCH_RING_W(dir_uid, entry) = r1;
      BRANCH_RING_R(dir_uid, entry) = r2;
      BRANCH_RING_E(dir_uid, entry) = r3;
End;


O_function: SET_MAX_LENGTH(dir_seg, entry, length)
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "modify");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);
bad_length! (length > MAX_POSSIBLE_LENGTH) |
   (length MODULO LENGTH_INCREMENT ≠ 0);

Effect:
BRANCH_MAX_LENGTH(dir_uid, entry) = length;
```

```
V_function: SEG_INUSE(dir_seg, entry)
Possible_values: "true" or "false"
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "status");

Value:
BRANCH_INUSE(dir_uid, entry)


V_function: SEG_TYPE(dir_seg, entry)
Possible_values: "directory" or "data"
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "status");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);

Value:
BRANCH_TYPE(dir_uid, entry);


V_function: SEG_SECURITY_LEVEL(dir_seg, entry)
Possible_values: class
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "status");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);

Value:
BRANCH_SECURITY_LEVEL(dir_uid, entry);
```

```
V_function: SEG_RINGS(dir_seg, entry)
Possible_values: rings
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "status");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);
msg_segment! BRANCH_TYPE(dir_uid, entry) = "msg_segment";

Value:
 If BRANCH_TYPE(dir_uid, entry) = "directory";
   Then BRANCH_RING_MA(dir_uid, entry),
        BRANCH_RING_S(dir_uid, entry);
   Else BRANCH_RING_W(dir_uid, entry),
        BRANCH_RING_R(dir_uid, entry),
        BRANCH_RING_E(dir_uid, entry);
End;


V_function: SEG_MAX_LENGTH(dir_seg, entry)
Possible_values: length
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "status");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);

Value:
BRANCH_MAX_LENGTH(dir_uid, entry);
```

```
V_function: SEG_MAX_ACLE(dir_seg, entry)
Possible_values: integer
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "status");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);

Value:
BRANCH_MAX_ACLE(dir_uid, entry);


V_function: SEG_ACLE_EXISTS(dir_seg, entry, user_id, project_id, tag, mode)
Possible_values: "true" or "false"
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "status");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);

Value:
If (∃acle)
   ((ACL_USER(dir_uid, entry, acle) = user_id) &
   (ACL_PROJECT(dir_uid, entry, acle) = project_id) &
   (ACL_TAG(dir_uid, entry, acle) = tag) &
   (ACL_MODE(dir_uid, entry, acle) = mode));
   Then "true";
   Else "false";
End;
```

```
V_function: DIR_ACLE(dir_seg, entry, acle)
Possible_values: access control list element
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "status");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);
no_acle: acle > BRANCH_MAX_ACLE(dir_uid, entry);

Value:
(ACL_USER(dir_uid, entry, acle),
ACL_PROJECT(dir_uid, entry, acle),
ACL_TAG(dir_uid, entry, acle),
ACL_MODE(dir_uid, entry, acle));


V_function: DIR_INFERIOR_COUNT(dir_seg, entry)
Possible_values: integer
Let:
dir_uid = PST_UID(dir_seg);

Exception_if:
not_in_as! ¬INAS(dir_seg, "status");
entry_not_inuse! ¬BRANCH_INUSE(dir_uid, entry);
not_dir! BRANCH_TYPE(dir_uid, entry) ≠ "directory";

Value:
BRANCH_INFERIOR_COUNT(dir_uid, entry);


V_function: SEG_MODIFIED(seg)
Possible_values: "true" or "false"

Value:
PST_MODIFIED(seg);



Implementation_Hidden_V_functions:
ACLE_OFFSET(dir_uid, entry, acle, user_id, project_id, tag, mode);
BRANCH_OFFSET(dir_uid, entry, type);
Value:
"Value is implementation dependent, but must be a function of the parameters
of the V_function, including (in particular) the contents of the segment identified
by the first parameter."
```