Identification

File System Flowcharts
A. Bensoussan

This document contains a series of flowcharts of the file system procedures (as is at May 1968), made following the EPL code. Although not complete and not error proof, it may be used as a guide for having a precise idea about what is the function performed by a procedure and how it is performed.

Data bases are not described but references to them are made using the name which appears in the EPL declaration.

The following notation is used for locks:

Try to lock X and go to
1 if successful
2 if X was locked by another process.
3 if X was already locked by the process.

Unlock X

The difference between the various entries to ilock is as follows:

loop lock
lock
mod lock
read lock

No special notation for try lock. It is explicitly indicated in the flowchart.
calls =

Call $X(a, b)$ where $a$ is input argument  
$b$ is returned argument  

1. If no error code go to 1  
2. If error code go to 2
CORE CONTROL

CORE-MAN $ ASSIGN
$ UNASSIGN
$ WIRE
$ UNWIRE
$ GET.TYPE
ASSIGN (atoc, atype, asps, apageno, asstp, areplen)

sp = asps
ctype = atype + 2
core found = 0

if sp = 1
    (1K)

lo = i
li = type
vl = 2

if lo = 9
    li = type + 8
    vl = 10

ASSIGN

i = first entry in free list lo

<table>
<thead>
<tr>
<th>List#</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>unused</td>
</tr>
<tr>
<td>1</td>
<td>free</td>
</tr>
<tr>
<td>2</td>
<td>Normal</td>
</tr>
<tr>
<td>3</td>
<td>wired down</td>
</tr>
<tr>
<td>4</td>
<td>PERM.</td>
</tr>
<tr>
<td>5</td>
<td>temp.</td>
</tr>
<tr>
<td>6</td>
<td>unused type</td>
</tr>
<tr>
<td>7</td>
<td>split 128/64</td>
</tr>
<tr>
<td>8</td>
<td>unused</td>
</tr>
<tr>
<td>9</td>
<td>free</td>
</tr>
<tr>
<td>10</td>
<td>Normal</td>
</tr>
<tr>
<td>11</td>
<td>wired down</td>
</tr>
<tr>
<td>12</td>
<td>Perm.</td>
</tr>
<tr>
<td>13</td>
<td>temp.</td>
</tr>
<tr>
<td>14</td>
<td>unused type</td>
</tr>
<tr>
<td>15</td>
<td>unused</td>
</tr>
</tbody>
</table>
UNASSIGN (aloc)

- type = 1 (FREE)
  - $LE$ Bind
    - Loc = aloc
    - $i = k\# + 1$
    - $ml(i).type = split$
      - YES
      - $i = ml(i).point + 4$ right bits of Loc
    - NO
      - Bind list = type
      - Unbind list = $ml(i).type$ 
      - UNBIND (unbind list, $i$)
      - BIND (bind list, $i$)
      - $ml(i).type = type$
      - RTN
  - Bind list = type + 8
    - Unbind list = $ml(i).type + 8$
  - RTN

UNWIRE (aloc)

- type = 2 (NORMAL)
  - RTN

WIRE (aloc)

- type = 3 (WD)
  - RTN
GET_TYPE (aloc, atype)

Loc = aloc

i := K + 1

ml(i).type = split

YES

NO

i := ml(i).point + 4 right bits of loc

atyp := ml(i).type - 2

RTN
PAGE CONTROL

PAGE $FAULT
    $ IN
    $ DONE
    $ OUT
    $ TABLEIN
    $ TABLEOUT

PC $CHECK ENTRY
    $ CLEAN-UP
    $ FREE-LOC
    $ READ-SEG
    $ TRUNCATE
    $ UNWIRE

SETFAULTS

UPDATES
PAGE & FAULT (scu.ptr, db.ptr, cr.code)

- Save

- cr.code = 0
  - Lock SST
    - ps = page size of ps (tor.r)
    - address = cu, ds.ptr
    - pptp = ptr to ds.ptr

- dssw = 1 \( \land \) no PF in ds.ptr
  OR dssw = 0 \( \land \) PF in ds.ptr

- THEN
  - ds
    - THEN
      - SF in ps
        - ps = 1 or le
        - pptp = ptr to ptw
      - THEN
        - no PF in ptw

- THEN
  - Get ssstep from ptw

- THEN
  - ptw.os
    - THEN
      - ok
      - THEN
        - cr.code = 1
        - ptw.cr
        - THEN
          - dssw = ptw, ds.ptr
          - THEN
            - page $ in (ssstep, pptp, dssw, 1)

- THEN
  - ptw.os
    - THEN
      - ok
      - THEN
        - pwn & addesent (ptrns, rel(ssstep)11ret(pptp), ind)

- UNLOCK SST

- UNLOCK SST

- pwn & wait (ind)

- RTW
PAGE & OUT (cstep, plp, dssw, relarg, repl)

page = ref (plp) - 1
Loc = ptw.add
plw.asc = "0"b
plw.add = rel (cstep)

CLEAR ASS.MEM

dssw = 0
AND
There is a movefile and
page has NOT been moved
OR
page has been modified

THEN
plw.os = 1
plw.pmz = 1
+1 = add. loc out

THEN
There is a move file

THEN
fmp = ashe.moup
did = ashe.unfdid

THEN
fmp = ashe.moup
did = ashe.unfdid

THEN
str. devadd = fmp.ree (page) = free. store & with draw (did)

str. memadd = loc
- opcmd = "1"b
- op = "1"b
- pagano = fixed (page, 8)
- ashtep = rel (ssstep)

device.control & write (did, addr (str))
updates (ssstep, 1, 1)

relarg = 2

relarg = 1

dssw

forcemove (ssstep, repl)

RTW
PAGE $\Rightarrow$ TABLEIN (aslep, dssw)

$esl = \text{aste. rel (units of 6u)}$

$mbl = \text{aste. mbl (units of 1u)}$

$\text{Y}$

$mbs > 256$

$\text{N}$

$mbl \leq 64$

$\text{Y}$

$sps = 1$

$sps = 0$

$core\text{.man} \Rightarrow \text{assign (loc, 1, sps, 0, aslep, 1)}$

$\text{asle. pid = loc}$

$\text{asle. sls = 1}$

$size = 1$ if $64$

$16$ if $1u$

$\text{ZERO (loc, null, size)}$

for $i = 1$ to $msl - 1$ do:

$pt(i).\text{add = vel (aslep)}$

$pt(i).\text{display dssw}$

$\text{dceu = 0 and } i \times 16 < ccel$

Then

$\text{pt}(i).\text{pnr} = 1$

$\text{RTN}$
pč = CLEAN_UP (astep)

SETFAULTS (astep, 1)

astep.hpn = 0

= 0 some pages in core

FOR j = 1 TO 2 DO

FOR i = 0 TO (astep.esl - 1) / 16 DO

ptwp = add (ptw(i))

page # i in core

THEN

page & IN (astep, ptwp, 0, 1)

ELSE

page & OUT (astep, ptwp, 0, return, 1)

1 return

2 UNASSIGN (Loc)

THEN

1 ptwp ANDREXIT (ptwp, astep, pageck, i, code)

UNASSIGN (Loc)

Else

page & IN (astep, ptwp, 0, 1)

\[
\text{AST ENTRY is already locked}
\]

Then

SST.PTL

Else

Panic

SST.PTL

ATN

Then

astep.hpn = 0

or astep.hpl = 0

astep.ptc = 0

record = 0

astep.sts = 1

Page & TABLEOUT (astep, 0, record)

Panic

SST.PTL

Panic

SST.PTL

ATN

\[
\text{Page & WRITE (code)}
\]
Called by \texttt{SETUP RING} to unwire the first page of the ring \texttt{n DS} when a process leaves ring \texttt{n}.

pc \# UNWIRE (astep)

\texttt{SST, PTL}

\texttt{astc, ptdc} := 0

\texttt{astc, stc} := 0 \quad \texttt{SEG is Loaded}

= 0

\texttt{FOR i=0 TO (astc, csl-1)/16 DO}

\texttt{page \# i in core}

\texttt{Loc = ptw(i), add}

\texttt{type = csc, man \# GET.TYPE (loc)}

\texttt{if (normal) = 0 then}

\texttt{type = 1 (wd)}

\texttt{Core, man \# UNWIRE (loc)}

\texttt{SST, PTL}

\texttt{RTN}
## SEGMENT CONTROL

### SEGMENT UTILITY MODULE

- **ALLOC_SST**
  - ALLOC_SST
  - FREE_SST
- **ALLOCATE_SSTVAR**
  - ALLOCATE_SSTVAR
  - FREE_SSTVAR
- **CMD-_ACC**
- **DST-_SEARCH**
- **DST-_THREAD**
- **GETASENTRY**
  - GETASENTRY
  - DELASENTRY
- **HASH_INDEX**
  - ID_INDEX
  - NAME_INDEX
- **MAKE_TRAILER**
- **REMOVAL_LIST_UTIL**
  - SEARCH
  - TEST
  - UNTREADH
- **SUM**
  - ID_SRCH_KST
  - N_SRCH_KST
  - SEARCHAST
  - SEARCH_HST
- **THREAD_TRAILER**

### SYSTEM INTERFACE MODULE

- **BOUND_FAULT**
- **GETRING**
- **INITIALIZE_KST**
- **MAKE_KNOWN**
- **MAKE_UNKNOWN**
- **SEGFAULT**
  - SIM 1
    - BRANCHMOD
    - DELETE_SEG
    - DIROM
    - TRANSUSE
    - UNLOAD_SEG
    - UPDATE_E
  - SIM 2
    - GETDIR_SEG
    - MOVE_SEG

### USER INTERFACE MODULE

- **UI M**
  - CHECK_ACCESS
    - CHECK_RING
    - CORE_TEST
    - FREE_CORE
    - REFDE SEG
    - TRUNCATE_SEG
    - WRITE_SEG

### RING REGISTER SIMUL MODULE

- **SETUP_RING**
  - LOAD
  - SETUP_RING
Alloc_SST $ FREE_SST (ptr)

allocate_sstvar $ free_sstvar (ptr)

Rtn
Allocate sstvar \& ALLOCATE_SSTVAR (sstpt\(r\), \(f\), fm.length).

\(f = 2\)

Y (Filemap)

\(b = \text{fm.length}\)

(b is the length which appears in file_map decl)

\(/\ast\ \text{MASK}\ \text{PROCESSOR} \ast/\)

\(\text{Panic}\)

\(\text{SST.FSL}\)

\(f = \)

4 2 3 4 5 6

\(\text{ASTE}\) \(\text{fm}\) \(\text{str}\) \(\text{DSTE}\) \(\text{PSTE}\) \(\text{PSTR}\)

Allocate ASTE in (sst. sstvar) set (p)
Allocate FILEMAP in (sst. sstvar) set (p)
Allocate STR in (sst. sstvar) set (p)
Allocate DSTE in (sst. sstvar) set (p)
Allocate PST in (sst. sstvar) set (p)
Allocate PSTSS in (sst. sstvar) set (p)

\(\text{Cut postop from list}\)

\(/\ast\ \text{RESTORE}\ \text{PROCESSOR}\ \text{MASK} \ast/\)

sst\(\text{pt}\)\(r\) = p

RTN
Allocate sstvar & FREE_SSTVAR (sst.ptr).

/\ MASK PROCESSOR */

MY

SST.FSL

Panic.

FREEN_ (sst.ptr)

/\ RESTORE PROCESSOR MASK /*

RTN
DST_SEARCH (ring\text{no}, psteplr)

\begin{align*}
\text{dstep} &= \text{pste}.\text{dstep} \\
\text{do while} \; \text{dstep} \neq 0 \\
\text{dstep} &= \text{ptr} (\text{sst}.\text{ptr}, \text{dstep}) \\
\text{dste}.\text{ring}\text{no} &= \text{ringno} \\
\text{if} \; \text{dste}.\text{ring}\text{no} = \text{ringno} \; \text{then} \\
\text{dstep} &= \text{dste}.\text{dstep} \\
\text{next dste} \\
\text{not found} \\
\text{dstep} &= \text{NULL} \\
\text{found} \\
\text{pst}\text{e} \\
\text{RTN} \text{dste} \text{ptr}
\end{align*}
DST_THREAD (dsteptr, psteptr)

Insert the new dste after Ringo dste

Before

After
GETRING (gateinfopt, oldring, newring, type, validsw, typeptr, err)

```
1. Check if err is 0.
2. If not, proceed.
3. If type is 0, go to end.
4. Open a window in RDS to look at oldring.
5. If oldring == R-F, use SDW.
6. If oldring == 0, go to panic.
7. Use KST.
8. Restore window.
9. If newring == rb2, go to end.
10. Use KST.
11. Execute KST.
12. If newring > KST.highseq or et(seqno).vrs == 1, go to end.
13. If err == invalid seqno, go to end.
14. Execute data.
15. If newring < oldring, go to end.
16. If gate mop is 00010000, go to end.
17. If oldring == ginfo.rb2, go to end.
18. If type == ginfo.type, go to end.
19. If validsw == ginfo.argval, go to end.
20. If typeptr == NULL, go to end.
```
hash_index $\neq$ ID_INDEX (id, n_bucket)  

\[ i = \text{right 35 bit of } \text{id}, \text{ converted into fixed (35)} \]

\[ \text{index} = \text{mod} (i, \text{n_bucket}) \]

Return (index)

\[ \text{RTN} \]

hash_index $\neq$ NAME_INDEX (name, n_bucket)  

\[ i_1 = \text{first 2 char of name, converted into bit (27)} \]
\[ i_2 = \text{middle 3 char of name, converted into bit (17)} \]
\[ i_3 = \text{last 3 char of name, converted into bit (17)} \]
\[ i = i_1 + i_2 + i_3 \text{ fixed (35)} \]

\[ \text{index} = \text{mod} (i, \text{n_bucket}) \]

Return (index)

\[ \text{RTN} \]
MAKE TRAILER (segno, astep)

1. psstep = pels + pstep
2. ilock & try/lock (gsl:lockptr, my, fact)
3. Maken cura I had already locked ASTe
   - Y: aste. 2
   - N: Look if ASTe has already a trailer for the process
4. call ornext (3, strp)
5. str. segno = segno
   str. pstep = psstep
   str. astep = ptr + 1
6. insert the new trailer at top of list of processes using the segment
7. Thread.trailer (psstep, strp)
   * Lock pstep (temp)
   * Insert the new trailer at top of list of segments, active for the process
   * Unlock pstep
8. RTN
MAKE, MOVE, TRAILER

new.id = aid

MOVE ADVICE
(briefp, arp, impval, new.id)

new.id

if mk = briefp.mk

Associate (imp, mk)

SET_MOVE (aica, imp, new.id)

RTN
REMOVAL_LIST_UTIL $
old.ring = pds & wired.ring

n = old.ring

IF esw = 1 THEN

old.step = DST_SEARCH (old.ring, psteptr)

pce & UNWIRE (old.step)

pds & wired.ring = n

esw = 1

IF esw = 0 THEN

pds & wired.ebr = m selw(dsecpno)

dbr = m selw(dsecpno)

/* Now the process can again be unloaded */

Restore the window opened
in Ro descriptor sequence

RMV

Question: How do we delete a LIte
Called by DC to reflect changes in the access control and/or protection list.
sim1 $ DELETSEG (seqptr, err)

err = 0

seqno = ptr $ baseno (seqptr)

get kstep using seqno

as tep = GETASTENTRY (kstep, 0, err)

pc $ TRONCATE (astep, /* add = */ 0)

sim1 $ UNLOADSEG (kste. id, 1, err)

RTN
SIM 1 & DIRMOD (dp)

seqno = prf & baseno(dp)

get kstep using seqno

astep = GETASTENTRY (kstep, 0, err)

UPDATES (astep, /a mod sw = #/1)

pe & CHECKENTRY (astep, 0)

RTN
\texttt{sim1 \& TRANSUSE (segptr, tus)}

- \texttt{segno = ptr \& base.no (segptr)}
- \texttt{get Kstep from segno}
- \texttt{tusw = kste.tusw}
- \texttt{kste.tusw = tus}
- \texttt{tus = tusw}
- \texttt{RIN}
$\text{sim1} \& \text{UNLOADSEG}(id, \text{deactsw}, \text{err})$

- $\text{err} = 0$
- $\text{not found}$
  - $\text{not found}$
    - found
      - $\text{deactsw}$
        - yes
          - $\text{pc} \& \text{CLEANUP}(\text{astep})$
          - $\text{DELASTENTRY}(\text{astep}, \text{err})$
        - no
          - $\text{pc} \& \text{CHECKENTRY}(\text{astep}, 0)$
    - $\text{not found}$
      - $\text{RTN}$


sim1 $ UPDATEB (id, dp)

not found

sum $ SEARCHAST (id, found, hst, astep)

asl, exfeldid = 0

if (not found)

asl, exfeldid

did = asl, exfeldid

fmp = ptr (sslp, asl, exfp)

asl, mordid

movdid = 0

movdid = asl, mordid

movdid = 0

br.copy, esl = asl, esl

br.copy, did = did

msl

dblm

dblu

dlm

dlt

dtind

dtime

slot = asl, x branch

actsw = t

br.copyP = add (br.copy)

bfm = fmp $ file.map

bfmp = add (bfm)

p $ CHECKENTRY (astep, 0)

actinfo $ WRBRANCH (dp, slot, id, actsw, br.copyP, bmp, err)

RTN

panic

err

Branch info

Argument for actinfo
lint $1$ GET DIR SEG (dirname, segptr, name, rnrn)

D.C.

\text{not found}

\text{find dir} (dirname, segnr, rnrn)

\text{segno} = \text{ptr} \& \text{baseptr}(\text{segno})

\text{segno} = \text{ptr} \& \text{baseptr}(\text{segptr})

\text{kstep} = \text{et}(\text{segno}) \cdot e^p$

\text{valring} = \text{getval}$

\text{valring} \leq \text{kste} \cdot \text{rb} 1 \\
\text{valring} \leq \text{kste} \cdot \text{rb} 2 \\
\text{valring} > \text{kste} \cdot \text{rb} 3$

\text{mod} = \text{kste} \cdot \text{mode}

\text{mod} = \text{kste} \cdot \text{mode}

\text{mod} = \text{nullmode}

\text{setptr} = \text{null}

\text{mode} = \text{nullmode}

\text{RTN}
called by MULTISVEL to move a file specified by "source" to drive "dir".

The source to be moved is supposed to be known by the system before
the call is issued.
HASH_SEARCH (compare, eindv, err)

err = 0
trycnt = 0
freeFound = 0

x = hsh x
hsh x = index set by caller

ht(x), vs

ht(x), ds

freeFound = 0
free x = x

freeFound = 1
free x = x

if freeFound = 1

x = x + 1

Not Found

+1 → tryent

tryent < n

x = x + 1

x ≥ n

x = 1

found = 0
hs i = free x

err = 1

RTN

RTN

RTN

err = 0

found

compare (ev)

ev = go || ht(x), entry

found = 1
hs i = x
eindv = ex
vim & CHECKACCESS (scmple, zingno, mode, err)

err = 0
Get kulep from scmple
Kulep = NULL
err = invalid sce

kulep = GETASTENTRY (kulep, 0, err)

then
dp = baseph (kulep, xsegm)
slot = kulet x branch
id = kulep, id

else

Kulep, dlbm < asle, dlbm

err = RFindk (dp, slot, id, dlbm, mode, ringbresk, err)

Kulep, mode = mode
   . dlbm = dlbm
   . rb1 = ringbresk(s)
   . rb2 = ringbresk(s)
   . rb3 = ringbresk(s)

PE & CHECKENTRY (asle, 0)

PE & CHECKENTRY (asle, 0)

ringno > kulep, rb2

mode = NULLMODE
mode = kulep, mode
mode = NULLMODE
mode = NULLMODE

RTN
RTN
RTN
RTN
uiui FREE CORE (addrptr, mwords, err)

err = 0
Cut kstep from addrptr
kstep = NULL

sum SEARCHAST (kstep.id, found, hsi, astep)

addr = ret (addrptr)

pc FREE CORE (astep, addr, mwords, o, err)

de CHECK ENTRY (astep, o)

RTN
ui3 $ READSEG (addptr, mwords, err)

err = 0

Get kstep from addptr

kstep = NULL

err = invalid seq

a$tep = GET AENTRY (kstep, o, err)

addrs = rd (addptr)

RTN

pe $ READSEG (astep, addrs, mwords, 0, err)

pe $ CHECKENTRY (astep, o)

RTN
uiim $TRUNCATE~SEG~(addr,~err)

err = 0

Get kstep from addr

kstep = NULL

asle = GETAENTRY (kstep, 0, err)

addr = rel (addr)

pe $TRUNCATE (asle, addr, err)

pe $CHECKENTRY (asle, 0)

RTN
## Directory Control

<table>
<thead>
<tr>
<th><strong>Directory Maintainer</strong></th>
<th><strong>System Interface</strong></th>
<th><strong>User Interface</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Findbranch</td>
<td>Activate $AD$ Branch</td>
<td>Append $b$</td>
</tr>
<tr>
<td>Find entry</td>
<td>$WR$ Branch</td>
<td>Append $bx$</td>
</tr>
<tr>
<td>Hash &amp; In</td>
<td>ESTBLSG</td>
<td>Append $lx$</td>
</tr>
<tr>
<td>$OUT$</td>
<td>FINDDIR</td>
<td>CUNAME</td>
</tr>
<tr>
<td>$SEARCH$</td>
<td>RETINDIR</td>
<td>DILINRY</td>
</tr>
<tr>
<td>PACKER 1</td>
<td>SET BASE DIR</td>
<td>LISTDIR</td>
</tr>
<tr>
<td>PACKER 2</td>
<td>SET USAGE</td>
<td>MOVEFILE</td>
</tr>
<tr>
<td>REHASH</td>
<td></td>
<td>READAACL</td>
</tr>
<tr>
<td>REMOVE B</td>
<td></td>
<td>SET &amp; $bc$</td>
</tr>
<tr>
<td>REMOVE C</td>
<td></td>
<td>$@CONSIST$</td>
</tr>
</tbody>
</table>

## Access Control

<table>
<thead>
<tr>
<th><strong>Special User Interface</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Entry</td>
</tr>
<tr>
<td>Put Entry</td>
</tr>
<tr>
<td>SET LPD</td>
</tr>
<tr>
<td>SET LIMITS</td>
</tr>
<tr>
<td>SET RETRIEVE</td>
</tr>
<tr>
<td>SET SYSTEM TRAP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Access Control</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>APPMDE &amp; APPMDE</td>
</tr>
<tr>
<td>&amp; APPMDE ENTRY</td>
</tr>
<tr>
<td>EPM Mode</td>
</tr>
</tbody>
</table>
APPEND B (dirname, entry, dirw users ids, optw, mode, code)

\[ \text{curl} = 0 \]
\[ \text{bitend} = 0 \]

LEVE: & GET (cur, ring)

\[ \text{ringbracx(1)} = \text{fixed (cur, ring)} \]
\[ \text{(2)} = \]
\[ \text{(3)} = \]

\[ \text{one} = \text{pdf & proc - group id} \]
\[ \text{two} = \text{pdf & proc - group id, proj} \]
\[ \text{usercode} = \text{one} \parallel \text{two} \parallel * \]

APPENDX (dirname, entry, dirw, users ids, ringback usercode, optw, mode, curl, bitend, etc.)

RTN
APPENDL error handlers.

- ALLOC ERR3
- ALLOC ERR1
- ALLOC ERR1
- ALLOC ERR1
- HASH & OUT (dp, name, err)
- DIR (n)
- GET err

- Node = NULL mode
  - err = mode, err AND E & mode
  - err = err
  - code = no access
- RTN

Free np->name

was there a vacant link slot

N

FREE np->LINK

Y

PANIC
ESTBLSEG (parent, entry, raw, seqp, vid, optsw, slotlist, code)

FLOW CHART:

- Code = err = 0
- VSW = 0
- Y = 1, seqp = NULL
- Mode = NULLMODE
- if mode of parent
- FINDBRANCH (parent, entry, slot, mode, ep, err)
  - branch, vacsw = Y
  - branch.dirsw = 0
    - E & mode
      - err = NOENTRY
    - n
  - branch.dirsw = 1
    - E & mode
      - err = NULLENTRY
    - n
  - EFFMODE (ep, dp, slot, estblseg, vacautsw, emode, ringback, err)
    - vacautsw
      - Y
        - err = NOENTRY
      - N
        - branch, dirsw = Y
          - E & mode
            - err = mode err
          - N
            - optsw = branch, optsw
            - vid = branch, vid

      seqlname = fixpath (parent) || || entry

      MAKE KNOWN (seqlname, vid, emode, ringback, branch, dirsw, branch, dlbm, dp, slot, branch, dirsw, raw, seqp, slotlist, err)

ENTRY

- Mode = NULLMODE
- if err = modeerr
  - E & mode
    - err = err
    - Code = err
FIND BRANCH (dirname, entry, slot, mode, ep, err)

numlinks = 0  
err = 0  
mode = NULLMODE  

FIND ENTRY (dirname, entry, slot, mode, ep, err)

branch + slot  

branch + slot  

mode = NULLMODE  

LINK

err = linkmeter  

err = too many links

entryuid = link.uid

+1 to numlinks

numlinks = maxlinks

E in mode

ENTRY

err = linkmeter

entryuid = branch.uid

linkettel = value

linkname = ssvpath (link, pathname)

ENTRY

LINK

split linkname into dir > ent

mode = NULLMODE

slot = 0

FIND ENTRY (dir, ent, slot, mode, ep, err)

RTN
HASHT IN (dp, entry, slot used)

Card = 0
dir, htsize = 0
Y
ALLOCATE a HT with minimum size
ptc = next
ran = random
RTN

htsize = dir, htsize

hsi = HASH_INDEX & NAME_INDEX (entry, htsize)

entries = 0

+ 1 \rightarrow entries

ht (hsi), slot = 0

N

Y
FREE

ht (hsi), slot = 0

N

Y

used

compare entry with all the names

= ?

Y

slot = htsize

N

+ 1 \rightarrow hsi, insert

entries = htsize

Y

Panic

Y

+1 \rightarrow dir, ht used

RTN

Don't follow the code
HASH & OUT (dp, entry, code)

```
1. code = 0
2. dir.htsize = 0
   Y/N

3. Y: code = rehash (dp, -1, code)
4. N: htsiz = dir.htsize
5. li = hash.index & name.index (entry, htsiz)
6. ntries = 0
7. ntries = ntries + 1
8. Y: ht(htsi), slotno = 0
9. N: ht(htsi), vacated = 0
10. Y: FREE
    N: USED

   Y: Compare entry with all the names
   N: code = noentry

   Y: ht(htsi), vacated = 1
   N: -1 -> dir. htused

   Y: htsi = htsiz
   N: next = hasi + 1

   Y: htsi = htsiz
   N: slotno = 0

   Y: Scan hash table backwards until a non vacant is found.
   Free each vacant entry & scanned:
   { ht(a) = slotno = 0
     | vacated = 0

   N: htsi = htsiz

   Y: +1 -> htsi
   N: ntries = htsiz

   RTN
```
HASH & SEARCH (dp, entry, found, nsi, slot, ep, code)

1. code = 0
2. dir.size = 0
   - code = 0, hashtable
     - RTN
3. hsize = dir.size

hs = HASH (INDEX & NAME, entry, hsize)

4. ntries = 0
5. +1 → ntries

6. Y: FREE

7. N: h (hs) ≠ slot

8. Y: user
   - compare entry with each name
   - hsi = hsi
   - slot = h (hs) = slot
   - ep = slot.ep (± slot)
   - found = 1

9. N: variant
   - hsi = hsize
   - +1 → hsi | hsi ≥ 1
   - ntries = ntries
   - RTN
LIST DIR (dirname, uscrarea, branchp, brancht, linkp, linkt, code)

1. \text{dirname} = \text{pathname (dirname)}
2. \text{Sim.2 \& GETDIRSEG (dirname, clp, mode, err)}
3. \text{RE mode} \rightarrow \text{GET ERR}
4. \text{Panic}
5. DIR \rightarrow \text{wait}
6. \text{branchct} = \text{dir. bcount}
7. \text{branchct} = 0 \rightarrow \text{LINKS}
8. \text{ALLOCATE branchct \& IN (uscrarea)}
9. \text{SET (branchp)}
10. i = 1; j = 0
11. \text{branchct} = \text{dir. bcount}
12. \text{branchct} > \text{dir. bcount} \text{ OR } \text{branchct} = \text{dir. bcount}
13. \text{alloc rp(i) \& empty}
14. \text{branchct} = j
15. \text{branchct} = \text{branchvacant}
16. \text{ENTRYVID = branch. uid}
17. i = i + 1
18. \text{PACKER1 \& packb}
19. \text{(j, branchp, uscrarea, i, ep)}
20. \text{i = i + 1}

See error handlers on next page.
**REMOVE B** (ep, slot)

1. \( \text{err} = 0 \)
2. \( dp = \text{ptr}(ep, 0) \)
3. \( \text{dir.uid} = dp \rightarrow \text{dir.uid} \)
4. \( \text{DIR (w)} \)
5. For each name DO
6.   \( \text{HASH & OUT} (dp, \text{name.name}, \text{name.err}) \)
7. Free storage for name
8. Free storage for all ACLs
9. \( \text{ent.uid} = \text{branch.uid} \)
10. \( \text{branch.dtlbm} = \text{rdclsk} \)
   - \( \cdot \text{vacant} = 1 \)
   - \( \cdot \text{wid} = 0 \)
   - \( \cdot \text{usage} = \text{dir.vbsn} \)
11. \( \text{dir.vbsn} = \text{slot} \)
   - \( \cdot b\text{count} \leftarrow -1 \)
   - \( \cdot v\text{bcount} \leftarrow +1 \)

Free vacant branches until there are no more than "max.vacant" in the DIR.

Remember: A branch cannot be freed unless the dumper has noted the fact that it has been vacated.

**REMOVE L** (ep, slot)

1. \( \text{err} = 0 \)
2. \( dp = \text{ptr}(ep, 0) \)
3. \( \text{dir.uid} = dp \rightarrow \text{dir.uid} \)
4. \( \text{DIR (w)} \)
5. For each NAME DO
6.   \( \text{HASH & OUT} (dp, \text{name.name}, \text{name.err}) \)
7. Free storage for NAME
8. Free storage for PATHNAME
9. \( \text{ent.uid} = \text{link.uid} \)
10. \( \text{link.dtlbm} = \text{rdclsk} \)
   - \( \cdot \text{vacant} = 1 \)
   - \( \cdot \text{wid} = 0 \)
11. \( \text{dir.vlnn} = \text{slot} \)
   - \( \cdot l\text{count} \leftarrow -1 \)
   - \( \cdot v\text{lcoun} \leftarrow +1 \)

Free vacant links until there are no more than "max.vacant" in the DIR.

Remember: A link cannot be freed unless the dumper has noted the fact that it has been vacated.
SET_BASE_DIR (pathname, code)

```plaintext

<table>
<thead>
<tr>
<th>Length (pathname) &lt; 65?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

- If `Length (pathname) < 65?` is `Y`, then
  - `pwd` & `case dir = pathname`
  - `pwd` & `case dir = base (pathname)`

- If `Length (pathname) < 65?` is `N`, then
  - `code = none today`
SETWL (present, only, next code)

ENTRY

ENTRY

ENTRY