TO: MSPM Distribution
FROM: C. A. Cushing
SUBJ: BG.8.01-BG.8.04
DATE: 07/14/67

The following major revisions have been made to sections BG.8.01, BG.8.02, BG.8.03 and BG.8.04 to reflect the changes made to the primitives they describe.

1. Alternate returns on error conditions will no longer be taken. If an error occurs, a non-zero error code will be returned.

2. PL/I declarations for the arguments of each primitive have been included in each write-up. (Beware! Some have changed.)

Published: 07/14/67
(Supersedes: BG.8.01, 05/17/66;
BG.8.01, 01/30/67)

Identification

Directory Maintainer
C. A. Cushing

Purpose

The directory maintainer is responsible for finding, deleting, and itemizing the entries in a directory. It is essentially a utility package used by directory supervisor. All primitives of the directory maintainer are privileged i.e., they reside in the hard-core ring and are callable only by hard-core procedures, and are actually used only by directory supervisor.

Primitives

1. packer
2. findentry
3. findbranch
4. hash
5. rehash
6. removeb
7. removel

1. packer

The primitive packer itemizes the contents of an entry in a directory. All of the items listed in section BG.8.00 are taken from the entry and packed into an area specified by the caller. In this context, the caller refers to the caller of directory supervisor. The mode item for a branch entry is obtained by calling the access control module to find the apparent mode of the user with respect to the branch from the access control list (ACL) of the branch and the common access control list (CACL) of the directory.

```
call packer$packb (j,branchp,user_area,i,ep);
call packer$packl (j,linkp,user_area,i,ep);

dcl j fixed bin(17), /*index into an array of structures
                       (given by directory supervisor)*/

(branchp,linkp) ptr, /*pointer to the base of the array of
                       structures (returned)*/

user_area area((*)), /*pointer to an area in which the array
                       of structures is stored (given by
                       caller)*/
```

```
i fixed bin(17), /*signed slot number of the branch or
                   link whose contents are to be packed
                   into the jth element in the array
                   (given by directory supervisor)*/

ep ptr;          /*pointer to the ith branch or link in the
                   directory whose contents are to be stored
                   into the jth element of the array (given
                   by directory supervisor)*/
```

The entries packb and packl in packer store the contents of a given branch or link into a structure for the caller. The structure is part of an array of structures which was allocated in the area given by the caller. See the discussions of list_dir and status (BG.8.02) for the actual declarations of these arrays.

2. findentry

The primitive findentry reads a directory to search (by hashing as defined in Section BG.8.00) for a specified entry. The effective mode of the user with respect to this directory is checked against the mode needed by this user. If the user has the needed permission, directory maintainer searches the directory for the given entry. When the entry is found, it is locked and the pointer to the entry and the signed slot number for the entry are returned to directory supervisor.

There is a convention whereby the caller may refer to an entry by slot number rather than by name. If the slot number of an entry is 10, the caller may specify this entry by the character string ">10". The use of ">" in an entry name is illegal bacause of its special meaning in path names (see BX.8.00). Hence if ">" appears as the first character of an entry-name argument, it indicates that the argument contains a slot number.

If entry then contains a slot number rather than a name, the directory need not be searched (by hashing) because the requested entry can be found immediately through its pointer in the slot table (see BG.7.00).

```
call findentry (dir,entry,slot,mode,ep,code);

dcl dir char(*), /*character string containing the path name
                   of the directory (given by caller)*/

    entry char(*), /*character string containg the name of
                     an entry in dir (given by caller)*/
```

```
slot fixed bin(17), /*signed number pointing to a location
                      in the branch or link slot table
                      (returned)

                          if sign = +, entry is a branch
                          if sign = -, entry is a link */

mode bit(5), /*five bit flag representing trap,read,
               execute,write and append. If a bit is 1,
               the corresponding attribute is ON. This
               is both an input and output argument.
               It specifies

               (a) mode needed by user to find entry
                   (given by directory supervisor)

               (b) actual mode of user with respect to
                   dir (returned) */

ep ptr, /*pointer to entry (returned)*/

code fixed bin(17); /*if non-zero, it represents the code
                      of an error detected by the file
                      system*/
```

The primitive findentry first calls getdirseg in segment control to get the pointer to the base of dir and to get the effective mode of the user with respect to dir. If the mode of the user does not contain the permission specified in mode, then mode is set equal to the mode returned by getdirseg and an error is reflected to the caller. If the user's mode does contain the needed permissions, then mode is set equal to the mode returned by getdirseg and findentry then goes about finding entry in dir. The segment dir must be locked for reading. If entry is a symbolic name it must be hashed in order to find the entry in dir to which it refers (see hash primitive in this section). If entry is a slot number, it is converted to a binary number and the entry to which it points is determined immediately through the proper slot table. In either case slot and ep are set, the entry is locked, the directory is unlocked and control is returned to the caller.

3. findbranch

The primitive findbranch finds the branch to which a given entry effectively points.

```
call findbranch (dir,entry,slot,mode,ep,code);
        arguments are defined as for findentry
```

The primitive findbranch reads directory dir to search for entry as described for findentry. When entry is found, it is tested to see if it is a branch or a link. If entry is a branch, the effective mode of the user with respect to the directory is checked against the needed mode supplied by directory supervisor in mode. If the user has the needed permission, then the directory maintainer locks the branch and returns the pointer to the branch, the positive slot number of the branch and the actual mode of the user with respect to dir to directory supervisor.

If entry is a link the mode needed in this directory is not the mode specified by directory supervisor. The mode specified by directory supervisor is only needed in the directory containing the branch. If the effective mode of the user with respect to the directory containing this link indicates execute permission (or if mode given by directory supervisor has all bits 0, then no permission is needed) then the path name of the entry to which the link points is saved and the date and time last used of this link is updated. The last entry name is extracted from the saved path name dividing the path name into two parts; the path name of the directory containing the entry to which the link points and the name of the entry. The directory maintainer then calls segment control at getdirseg to get the segment number for this new directory and the effective mode of the user with respect to it. If the effective mode indicates the execute permission (or if mode given by directory supervisor has all bits 0, then no permission is needed) then the directory will be searched for the entry. When the entry is found, the same tests are applied as stated above. This process is repeated for each link found.

In order to prevent a loop, the number of links used to find a branch will be limited (possibly to 20). If a branch cannot be found in this limited number of tries, an error will be reflected to the caller.

4. hash

The procedure hash contains three entry points, search, in, and out. The purpose of these primitives is, respectively, to find, add and delete hash table entries. The search routine is not only called externally but is also called by in to find an empty hash location for a given name in order to fill it and by out to find the hash location used by a given name in order to delete it. (The structure of a directory hash table is described in the MSPM section BG.7.00).

```
call hash$search (dp,name,found,hloc,slot,ep,code);
call hash$in (dp,name,slot,code);
call hash$out (dp,name,code);

dcl dp ptr, /*pointer to base of directory containing the hash
              table (given by directory supervisor)*/

    name char(*), /*character string containing the name to
                    be hashed (given by caller)

    found fixed bin(1), /*a switch when 1 means a hash table
                          location is being used for name
                          when 0 means a hash table location
                          isn't being used for name*/

    hloc fixed bin(17), /*index of the location in the hash
                          table found for name (returned)*/

    slot fixed bin(17), /*signed slot number taken from hloc
                          and returned by hash$search and
                          (possibly subsequently) put into
                          hloc by hash$in*/

    ep ptr; /*pointer to the entry with the name name
              (returned)*/
```

The directory dir must be read-locked or write-locked before a call to hash$search and write-locked before a call to hash$in or hash$out. The status of the directory lock is not changed by hash.

The argument name is hashed and the resulting location (primary location) and possibly successive locations (secondary locations) in the hash table are checked. Each location is checked to see if it contains a slot number which points to an entry with the name, name.

In the case of hash$search, if a location containing such a slot number is found, the found switch is set on, hloc, ep and slot are filled and control is returned to the caller. In the case of hash$in, an error is reflected to the caller and in the case of hash$out, this location is vacated (see below for the distinction between empty and vacated hash table locations).

If when checking the primary location and secondary locations a never-used empty location (see explanation below) is encountered and no slot number was found which pointed to an entry with the name, name, then for hash$search

the found switch is set off, hloc is set to the index of this empty location and control is returned to the caller. In the case of hash$in, the empty location is filled in with slot and control is returned to the caller. In the case of hash$out, an error is reflected to the caller.

A general description of hashing techniques is discussed in section BG.8.00, however, the following explanation of the distinction between two types of empty hash table locations is necessary to clarify the above discussion of the hash primitives. There are two types of empty locations in a hash table. A hash table location from which a signed slot number has been deleted, called a vacated location, is distinguishable from an empty location which has never been used. This distinction is needed in order to find a slot number which is in a secondary location after the contents of the primary location have been deleted.

Whenever adding or deleting hash table entries, hash must check to see if the hash table needs to be rehashed. If the number of locations currently in use is greate (less) than a set fraction upt (lowt) of the total number of locations in the table, then the size of the table must be increased (decreased) and the names of the directory entries must be rehashed. If the number of locations which have been used (currently used locations plus vacated locations) but not the number of locations currently in use is greater than the set fraction upt of the total number of locations in the table, then the names of the directory entries must be rehashed, but the size of the table does not have to be changed.

5. rehash

The primitive rehash rehashes the names of the directory entries after possibly changing the size of the hash table. The hash$in or hash$out primitive calls rehash after a check has shown that the hash table is either too full or too empty for efficient use.

The call is as follows:

```
call rehash (hp,size);

 dcl hp ptr, /*pointer to the hash table*/
```

```
size fixed bin(17); /*= -1, hash table size to be
                        decreased
                    = +1, hash table size to be
                        increased
                    = 0, hash table size not to be
                        changed*/
```

The names of each entry in the directory are hashed and the signed slot numbers of these entries are placed in the resulting locations in a newly created hash table. The old table is then deleted.

6. removeb

The primitive removeb removes a branch which was found by findentry or findbranch. The directory containing the branch must be write-locked before calling removeb. This primitive first calls hash$out to vacate the locations in the hash table used by the names of this branch. The date-and-time-branch-last-modified item is updated, the vacant entry switch is set ON, and this branch is threaded into the list of vacant branches.

If the number of vacant branches is greater than a certain fixed number, then an attempt is made to free the storage used by one of the vacant branches.

The date and time this chosen branch was last modified must be less than the date and time it was last dumped, i.e., the fact that this branch had been vacated must be known to the backup system before the branch can be freed.

The directory and branch are unlocked before control is returned to directory supervisor.

```
call removeb (ep, slot, code);
        arguments are defined as for hash
```

7. removel

The primitive removel removes a link which was found by findentry. The link is removed, and the hash table is modified etc., as stated for removeb, substituting link for branch.

```
call removel (ep, slot, code);
        arguments are defined as for hash
```