Published: 01/27/67
(Supersedes: BX.8.00, 01/14/66)

Identification

Overview of File System Commands

C. A. Cushing, C. Garman, E. Q. Bjorkman

Purpose

The file system commands and procedures are designed as an interface between the Multics user and the Basic File System (Section BG): they provide frequently-used sequences of manipulations upon segments which reside in the file system hierarchy, as well as service functions such as the formatting and printing of segment status information.

Similarly, the file system procedures provide a more flexible interface between a user's programs and the "primitives" of the Basic File System, as well as error handling by means of the standard error procedures (BY.11). For further information see BY.2.00, Overview of File System Library Procedures.

Introduction

A file is a linear sequence of data elements; an element may be a machine word, an ASCII character, or a bit, or multiples of each of these, depending upon the context of a particular reference. In Multics, a file is generally found as a segment somewhere in the hierarchy of directories maintained, manipulated, and massaged by the Basic File System. A segment may be known to a user or his procedures by its segment name, which may include information on its location in the directory hierarchy, or by its (hardware) segment number, which provides a shorthand method for accessing the data of the segment in core memory.

A Multics user may create, modify, or delete segments only through the use of the Basic File System: directly, by programmed calls to user-accessible entries in the Basic File System and hardware segment addressing of data, and indirectly, by using the file system commands and subroutines. The first group is covered in MSPM section BG, The Basic File System, and BD.3, Segment Housekeeping Module, while the latter group is the subject of sections BX.8, File System Commands, and BY.2, File System Library Procedures.

Files, Branches and Entries

A directory is a special segment maintained by the Basic File System, which contains a list of entries. To the user, an entry appears to be a segment which is accessed in terms of one or more symbolic entry names. The names are composed of strings of ASCII characters, whose syntax is discussed below. An entry name need be unique only in the directory in which it occurs.

There are two types of entries each containing a "pointer":

If the entry is a branch, the pointer defines the location of the contents of the segment somewhere in the secondary storage hierarchy; this segment may or may not be a directory.

If the entry is a link, the pointer describes, symbolically, another entry in the same or another directory. A link may point to a link, etc., to a reasonable depth of nesting, but once a link points to a branch, the segment pointed to by that branch is accessible as if the original entry had been the branch pointing to the segment.

Each branch contains a description of the way the segment it points to may be used or referenced, which is termed its access control information; a link does not contain this information, but instead derives its access privileges from the branch to which it ultimately points.

File System Hierarchy

The Basic File System works with a basic tree hierarchy of segments across which links may be added to facilitate simple access to segments elsewhere in the hierarchy. With one exception each segment (e.g., directory) finds itself directly pointed to by a branch in exactly one directory; the exception is the root directory at the root of the tree, whose location is known to the Basic File System, but which does not appear in any directory.

A segment pointed to by a branch in some directory is immediately inferior to that directory and the directory is immediately superior to the segment. The master directory has level zero, and segments immediately inferior to it have level one. By extension, inferiority (or superiority) is defined for any number of levels of separation via a chain of immediately inferior (superior) segments.

Links are considered superimposed upon, but independent of, the tree hierarchy.

At any one time, a user is considered to be operating in some one directory, called his working directory (wdir). He may access a segment simply by specifying the entry name in his working directory which effectively points to that segment. An entry name is meaningful only with respect to the directory in which it occurs, and may or may not be unique outside that directory. In order to refer to an entry which is not in the working directory, it is necessary to have a symbolic name which unambiguously defines an entry in the hierarchy as a whole. Such a name is a path name. It consists of the chain of entries (branches or links) required to reach the desired entry from the root directory or from the working directory. A number of abbreviations may be used for the special directories at each Multics installation, e.g. the Multics Command and Subroutine Library, the Local Command and Subroutine Library, or the Process Directory (see BD.6 System Skeleton, and BD.4, the Search Module).

Names: Formation and Syntax

A file-system-name is a string of ASCII graphic characters which plays a role corresponding to that of an identifier in a programming language. Names are further classed as entry names, path names, and access-control names (user names).

Five punctuation characters (".", ">", "<", "*", and "=") are reserved and receive special interpretation when encountered in names in the context of the Basic File System and the file-system commands and subroutines. While the other punctuation characters and the ASCII control characters are not specifically excluded, most names will in fact consist of only the upper- and lower-case alphabetic characters, the digits, the underscore character "_", and appropriate usages of the five special characters named above.

An entry name is composed of one or more graphic components (not containing the reserved characters), separated by "."; the components may be referred to (in order, left-to-right) as the primary, secondary, etc., components of an entry name. Certain commands attach particular significance to various components of an entry name; thus the command

```
     pl1 my_seg
```

directs the PL/I compiler to attempt the compilation of the (ASCII) file,

```
     my_seg.pl1
```

(See also BB.5.01, Reserved Segment Name Suffixes.)

A successful compilation would produce the following new segments in the user's working directory:

my_seg	[note no secondary component]
my_seg.link	[linkage segment]
my_seg.symbol	[symbol table and binding information]

and if the "list" option was "on"

my_seg.list	[listing segment].

A component may be null or empty, if necessary to preserve the sequence of components, or for other reasons. The following are examples of null components:

a., a..c	[null secondary component]
.b	[null primary component]
.	[primary and secondary components both null]

To provide a shorthand notation for referring to several segments at once in a command invocation, special meanings are attached to the character strings "*", "**", and "=".

In the discussion below <u>entry-name argument</u> refers to the character string used as an argument to one of the file system commands. The entry-name argument defines the entry name(s) in a particular directory with which the command is to concern itself.

In general, when an entry-name argument is supplied in the argument list of a command, a comparison is implied between the argument and one or more entries in a particular directory, to find which entries <u>match</u> the given entry-name argument. For a simple entry-name argument (one without the character strings `*' and `**'), the test for a match is simply a character-by-character comparison of the entry-name argument with individual entries in a directory.

If a single asterisk (*) appears as the sole accupant of a component position in an entry-name argument, it permits a match against any character string (possibly null), in that component position of an entry name in a directory.

If the character string `**' appears as the contents of a component position in an entry-name argument, it permits a match against any integral number of components of an entry name in a directory. Non-** components of an entry-name argument must match exactly with corresponding components of an entry name, e.g., components to the right of the last `**' in an entry-name argument must match exactly with those components on the end of an entry name.

Note that certain commands (e.g., link - BX.8.04) do not observe these conventions; they do, however, check for entry-name arguments with `*' and `**' and ask suitable questions before attempting to perform their duties.

Thus, using the names from the previous example,

```
     my_seg
```

as an entry-name argument matches the single segment by that name;

```
     my_seg.**
```

matches the five segments whose first component is "my_seg":

```
     my_seg.pl1
     my_seg
     my_seg.link
     my_seg.symbol
     my_seg.list
```

while

```
     **
```

stands for "all files", including the files named above.

Note, however, that

```
     my_seg.*
```

does not include

```
     my_seg
```

since the match is to a two-component name.

The single character "=" as a component denotes "the component occurring in the corresponding position of the preceding entry name"; thus, the command

```
     delete (my_seg.link  =.symbol)
```

would remove the entries and segments named

```
     my_seg.link
```

and

```
     my_seg.symbol
```

The "=" as a component is interpreted by each command individually; the command descriptions discuss particular aspects or restrictions.

An access control name (or user name) is constructed similarly to an entry name; however, it always contains a fixed number of components (this number may vary from installation to installation) separated again by ".". The character strings `*' and `**' are defined as for entry-name arguments; since the number of components is fixed, however, missing components on the right are assumed to be `*'. The single-character user name `*' denotes "all users of this Multics installation".

A path name is written symbolically as a chain of entry names, each name separated by ">". If the first character of the path name is ">", the path name is an absolute path name; that is, it is fixed with respect to the root directory. If the first character is not ">", then the path name is relative to the working directory or one of the special system directories. For example, the path name of the directory corresponding to the box marked 1 in Figure 1 is >Z>H. If >Z>H is now the working directory, then a path name of the directory corresponding to the box marked 2 is L. The symbol `<' is the shorthand notation for the path name of the directory which contains the link or branch that was used to access the current working directory. Using this notation and considering the path taken to directory 2 as >Z>H>L, a path name for directory 3 relative to directory 2 is <<A. Considering another path to directory 2 as >Z>A>B>C, a path name for directory 3 relative to directory 2 is <<.

Access Control

In attempting to access a segment a user may or may not be successful depending upon his implicit intentions and his permissions with respect to the segment. The set of permissions with which a given user may access the segment pointed to by a particular branch is called the mode of the branch for that user. The permissions given the user of a particular branch are specified by an access control list for that branch. This list is a list of users (i.e., of sets of users) along with the corresponding mode associated with each user. The access control list is ordered according to the weights of the user names. The weight of a user name is equal to the sum of the weights of its components where the weight of the ith component (reading from left-to-right) is 2**i. The components designated by `*' have weight 0. The ordering is from the highest to the lowest weighted user name. User names with the same weight have no ordering with respect to one another. For example, the following user names

```
     M1416.Smith.*
         *
     T104.*
     *.*.aa
     M1416.*.aa
```

would be ordered

```
     M1416.*.aa
     *.*.aa
     M1416.Smith.*
     T104.*
     *                    [equivalent to *.*.*]
```

The list is scanned from the top to discover the mode of a user. If all access control information required for the use of every segment in a particular directory is the same for certain users, this access control information may be put in the <u>common access control list</u> of the directory. If a user's name or class is not on the access control list of the branch pointing to the segment he wishes to use, then the common access control list is searched for this user's name or class, to determine his mode of access to the segment.

The mode consists of five <u>attributes</u>, named Trap, Read, Execute, Write, and Append, (sometimes abbreviated TREWA) each of which is either on or off. Collectively they define the <u>apparent mode</u> of the segment. The trap attribute is examined first. It has the power to override the other four attributes called <u>usage attributes</u>. The usage attributes indicate permission to perform the given ativity only if the attribute is on.

<u>TRAP ATTRIBUTE</u>. When a branch has the Trap attribute on for a given user, a trap occurs on any reference using that branch by that user. That is, the procedure whose name is given as the <u>trap procedure</u> is called. A list of parameters may be defined with this procedure name and are passed as arguments to the called procedure. The return from the trap procedure specifies the effective values of the four usage attributes, which may override the original values. A user can inhibit the trap mechanism, in which case all references to a branch by the user with the Trap attribute on will cause an error return to the calling procedure. (For further information see BX.8.02, <u>Access Control Commands</u>, BG.8, <u>Directory Control</u>, and BG.9, <u>Access Control</u>.)

<u>USAGE ATTRIBUTES</u>. Every operation on a given segment implies one of four intents, namely, read, execute write or append. The interpretation of the intent depends upon

whether the accessed branch points to a directory or a non-directory segment.

<u>Attribute</u>	<u>Directory</u>	<u>Non-directory branch</u>
Read	can "read" a directory to get information about any or all of the entries, including access control lists	can read the segment
Write	can delete or rename specifically named entries and change access control lists of specifically named entries.	can truncate or rewrite existing contents of the segment without adding to its length.
Append	can add entries without changing existing entries	can add to segment without changing original contents of file.
Execute	can search for specifically named entries in the directory in order to use them or to get information about them, excluding access control lists.	can execute the contents of the segment as a procedure.

<u>EXAMPLE</u>: Consider the request to delete all entries whose names contain the secondary component GAMMA in a particular directory. The user issuing this request must have the Write attribute on in the access control list of the branch pointing to the directory in order to delete each of the entries in the set described. The user must also be able to Read the directory in order to find all the entries with this secondary component, GAMMA.

<u>Organization of the File System Commands</u>

The file system commands are described in the section which follow; the calling sequence for these commands (with examples of the kind of arguments expected) is generally "command entry args". (See also BX.1.00, <u>The Command Language</u>, and BX.2.00, <u>The Shell</u>.)

<u>command</u> is simply the name of a command

entry	is an entry-name argument as discussed earlier, and defines the group of entries to be considered by command; entry may be a name defining a group of entries in the current working directory; or entry may be a path name with an entry name appended to it defining a set of entries in the directory pointed to by the preceding path, e.g.

<A	[entry A in the directory immediately superior to the working directory]
A>*.epl bsa	[all 2 component entries with secondary component "eplbsa" in directory A which is immediately inferior to the working directory]
>B>**	[all entries in directory B which is inferior to the root]
<<T>U>TEST.pl1	[This one is left as an exercise to the reader]

are all correct entry arguments. When the entry argument can be a list of these arguments, (entries) appears in the calling sequence. The arguments are separated by blanks and the list is enclosed in parentheses.

args	are arguments which vary from command to command. Two types which are common to many file system command calling sequence are path and (list).
path	is a series of entry names separated by ">" or "<" which defines a directory different from the working directory. (See the previous discussion of path names in this section.)
(list)	is simply a list of items separated by blanks and surrounded by parentheses.

In addition, one system option is specifically provided for use with the file system commands, where applicable: "omit spec" may be set with an interjected command (see BX.1.00, The Command Language), to affect the operation of a particular command. For example, the command sequence

```
list [omit dir]
```

would direct the list command (BX.8.01) to itemize and display the contents of the user's working directory, excluding directory entries.

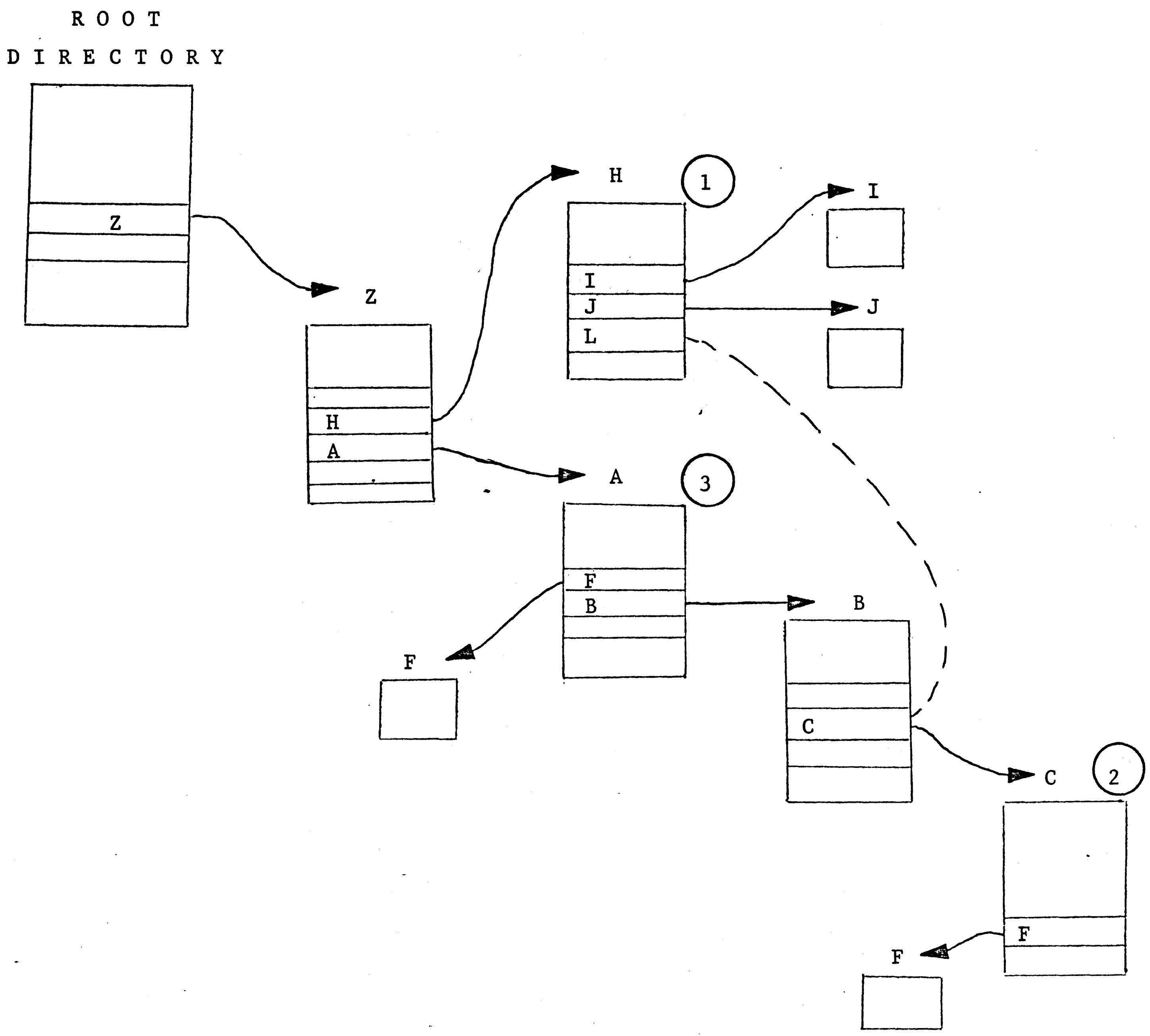

Figure 1

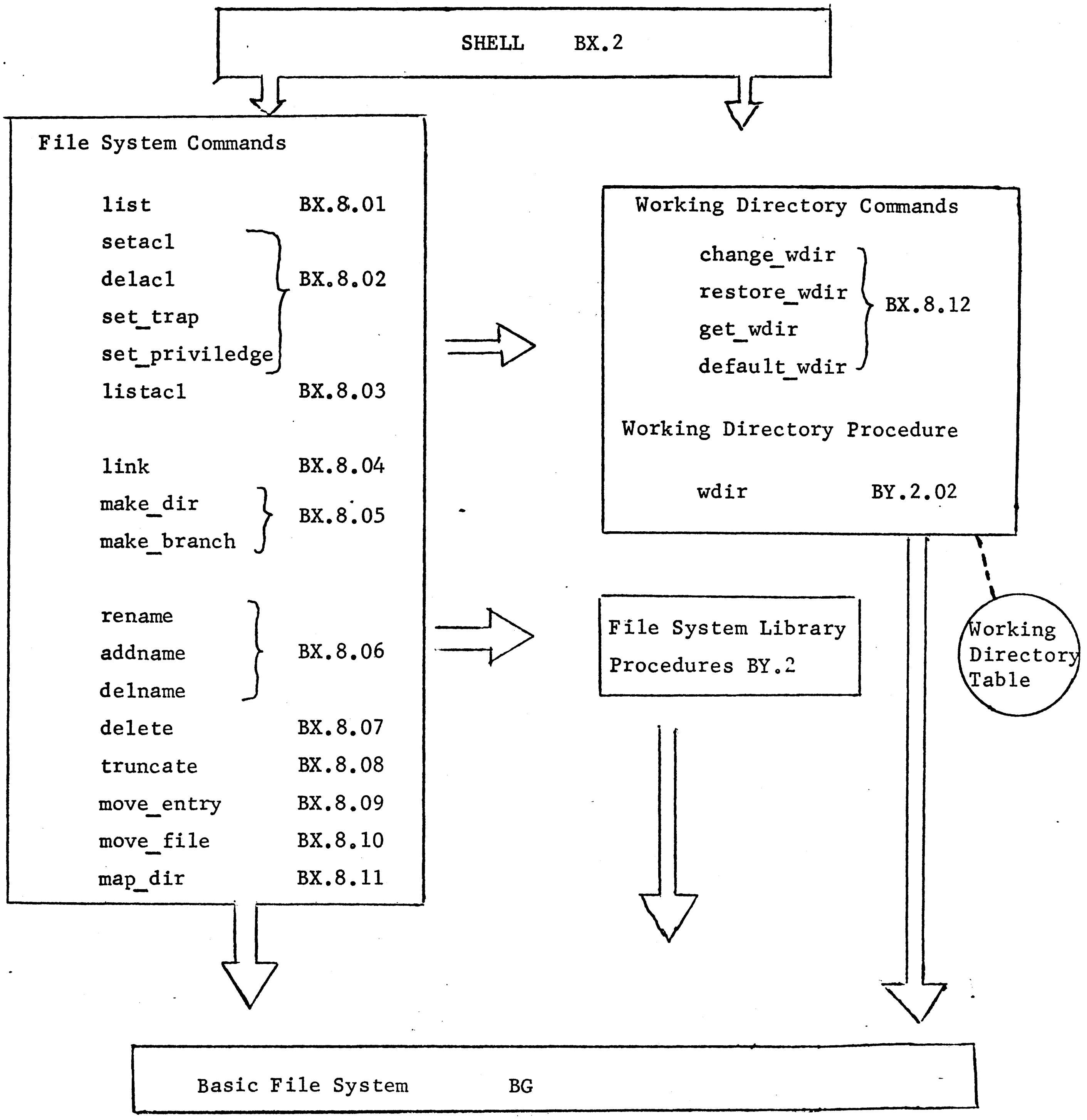

Figure 2: Schematic Diagram of Interactions Between File System Commands and Library Procedures, and the Basic File System