ALGORITHM 568
PDS—A Portable Directory System

DAVID R. HANSON
The University of Arizona

Key Words and Phrases: file system, UNIX, RATFOR
CR Categories: 4.19, 4.33, 4.35, 4.41
Language: RATFOR (FORTRAN)

1. DESCRIPTION

PDS is a set of procedures that provides a machine-independent method of file specification. PDS provides capabilities beyond those provided by many vendor-supplied systems. In addition, because PDS is portable, additional capabilities, such as protection schemes, file usage statistics, or some of the functions of a source code control system [5], can be added easily.

The basic function of PDS is to maintain a useful directory structure and provide a set of primitives for manipulating that structure. The PDS directory structure is identical to the tree structure of the UNIX [4] file system, and many of the PDS primitives are identical to UNIX primitives. PDS is, in large part, a portable implementation of the UNIX directory system. Besides PDS's machine-independence, the major differences are the extensibility of PDS and, as described in the next section, its i/o independence.

In the simplest terms, PDS provides a directory structure and a *mapping* from machine-independent file names to machine-dependent names. It deals only with the information *describing* a file; it does not use or manipulate actual files in any way. The importance of this approach is that PDS is used to specify a file but does not participate in the actual i/o to that file. Consequently, there is no impact on i/o efficiency when PDS is used.

PDS manipulates a rooted tree structure in which the leaves are files or directories and the nodes are directories. A directory is simply a list of files and directories. An example is shown in Figure 1, in which circles indicate directories and squares indicate files. The root of the tree is denoted by "/", and files and directories are denoted by their "path," which specifies their absolute position in the tree. A path is composed of the names of the nodes on the path from the root

Received December 1979; revised September 1980; accepted November 1980

This work was supported by the National Science Foundation under grant MCS78-02545.
Author's address: Department of Computer Science, The University of Arizona, Tucson, AZ 85721.

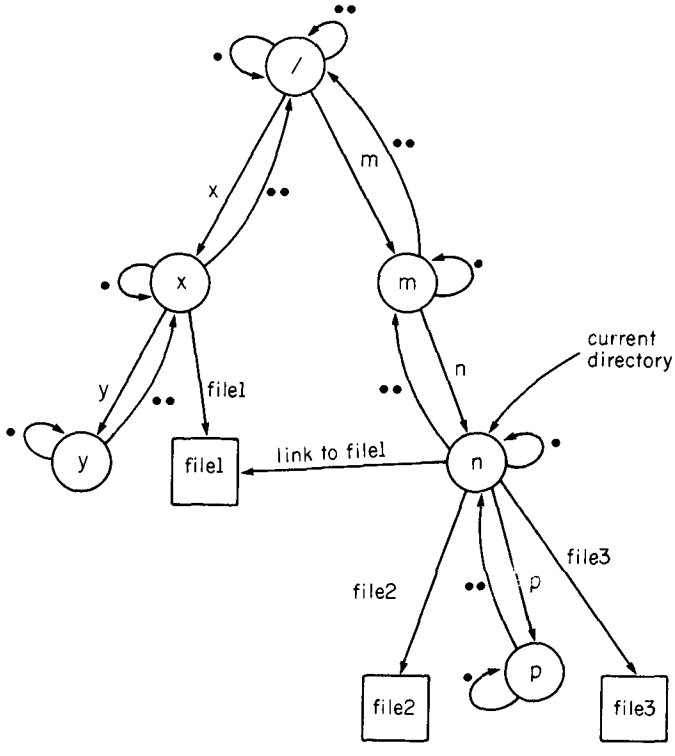

Fig. 1. A directory system.

to the desired file or directory. The path components are separated by slashes; for example, the path for file "file2" in Figure 1 is "/m/n/file2." The directory entries "." and ".." refer, respectively, to the directory itself and to the immediate ancestor. These names may be used as path components, providing an explicit means of using the structural properties of the tree. If a path does not begin with "/", it is taken to be rooted at the "current directory." For example, with the current directory at "n" in Figure 1, the path "file3" is equivalent to "/m/n/file3."

Files and directories are equivalent with the exception that directories are manipulated by PDS and files are not. Files simply "contain" the machine-dependent names of the actual files. These machine-dependent names are referred to as *host names*. The basic function of PDS is to map paths into host names.

There are ten PDS primitives; they are summarized in Table I. Detailed descriptions of each primitive are given in [1] and in the machine-readable comments. The most important primitives are *openf* and *creatf*, which map a path to a host name and perform machine-dependent operations concerned with preparing files for i/o. The rest are concerned primarily with manipulating the directory structure.

openf obtains the host name corresponding to *path* and calls a machine-dependent routine to actually open the file. *mode* indicates how the file is to be opened (e.g., "read", "write", "append", etc.). *mode* is not modified by *openf*, so it can be whatever is most appropriate on the host system. *openf* returns what

Table I. PDS Primitives

chdir(*path*)	change current directory to *path*
chds(*root, ilist*)	change to another directory system
creatf(*path, mode*)	create *path* and open it for i/o
link(*path*1, *path*2)	make a link to *path*1 named *path*2
mkdir(*path*)	make a directory named *path*
mkds(*root, ilist*)	make a directory system
mkfile(*path, hname*)	make a file *path* with host name *hname*
openf(*path, mode*)	open *path* for i/o according to *mode*
stat(*path, array*)	return information about *path*
unlink(*path*)	unlink *path*

the machine-dependent open routine returns, which is whatever is most useful as an argument in calling machine-dependent i/o routines that perform the actual i/o (e.g., **read** and **write**). Typically, a channel number or FORTRAN unit number is returned.

creatf is similar to *openf*, except that *path* is created along with a host name and file. The generation of the host name is performed automatically. After the file is created, it is opened according to *mode* as if *openf* had been called.

openf and *creatf* are insensitive to the values of *mode*. It is, however, useful to introduce standard values representing common file usage, such as reading and writing. This approach promotes machine-independence in programs that use PDS and is used, for example, with the programs described in [3]. Such *mode* values are scrutinized by the machine-dependent i/o interface routines, however, permitting PDS to be used in almost any environment—not just those in which i/o capabilities conform to preconceived notions of "common" file usage.

PDS has *no* other i/o primitives; it is completely independent of the actual i/o. Thus, additional overhead is incurred in opening files, but none is incurred in accessing them. The effect of the additional overhead is minimal since the former is performed much less frequently than the latter. The contribution of PDS is a portable hierarchical directory system with absolutely no time impact on i/o efficiency.

2. USAGE

PDS is packaged as a set of RATFOR [2, 3] (and hence FORTRAN) functions and subroutines, which is loaded with the program or system that uses it. PDS is useful in programs or systems that use named files extensively, or where a machine-independent means of naming files and a flexible directory structure would enhance utility. Another use would be in a set of tools, such as those described in [3], implemented on computers with limited file systems.

The following program, saveall, illustrates a typical use of PDS. It copies all of the files in the current directory to files of the same name in the directory "../backup". saveall is written in RATFOR in the style of [3].

```
# saveall - save all files in ../backup
  character dline(MAXLINE), bname(MAXLINE)
  integer fd, fdi, fdo
  integer openf, creatf, getlin
  string dot "."
```

```
string backup "../backup/"
fd = openf(dot, READ)
while (getlin(dline, fd) ~= EOF) {
  fdi = openf(dline(5), READ)
  call strcat(backup, dline(5), bname)
  fdo = creatf(bname, WRITE)
  call fcopy(fdi, fdo)
  call close(fdi)
  call close(fdo)
  }
call close(fd)
end
```

Uppercase names denote defined constants; for example, a typical value for *MAXLINE* might be 80. The **string** statement declares a character array large enough to accommodate the indicated character string. The program begins by opening the current directory for reading using *openf*. The **while** loop reads the directory, line-by-line, until end-of-file, placing each line in the character array *dline*. As described in the next section, each line in a directory contains a file name beginning in column 5. Thus the body of the **while** loop opens each file for reading (*openf*), constructs the path name of the backup copy (*strcat*), creates the backup file and opens it for writing (*creatf*), copies the file (*fcopy*), and finally closes the original and backup files (*close*).

This program makes use of the i/o routines described in [3] (*getlin* and *close*), but any other set of routines—including standard FORTRAN **read** and **write** statements—could be used.

Another example of the use of PDS is as a simple command preprocessor that translates machine-independent system commands to the appropriate host command sequence. In commands, files are referred to by their PDS names and are mapped to the corresponding host names during the generation of host commands. For example, the command

list *files* . . .

lists the named files on the line printer. The preprocessor translates the **list** into whatever is appropriate on the host system. For example,

list ../backup/saveall.r saveall.r

results in the DEC-10 command sequence

r queue
ilist.120, save.rat

where "ilist.120" and "save.rat" are the host names for "../backup/saveall.r" and "saveall.r", respectively.

3. IMPLEMENTATION

PDS is designed to use only simple sequential i/o (i.e., FORTRAN i/o). It is implemented in RATFOR for portability reasons and because FORTRAN is the only widely available high-level language to which calls from other languages can be made.

The implementation is similar to the implementation of the UNIX file system [7]. The basic technique is to separate the information about a file from the

presence of that file in a directory. An *i-node* is associated with each file and directory, and it contains all of the information, including the host name, concerning the file. All of the i-nodes are stored in a single file in which i-node n is line n. This file, referred to as the *i-list*, is stored in character format so that it can be read easily with FORTRAN i/o or its equivalent.

The i-list maps i-node number, or i-number, to a host name. Directories provide a map of PDS name to i-number, thereby completing the mapping from PDS name to host name. A directory is a file of lines, each line containing an i-number and the corresponding PDS name.

Further implementation details are given in [1].

4. INSTALLATION

PDS consists of about 1000 lines of RATFOR including comments. Of this total, however, 300 lines are utility subroutines commonly available in a RATFOR environment. After preprocessing by RATFOR, the resulting FORTRAN code is about 1200 lines in length. The resulting FORTRAN conforms to the portable subset of ANSI standard FORTRAN as defined by PFORT [6].

On a DEC-10 (36-bit words, 512-word pages), the code area for the entire system occupies 6 pages and the data area occupies 17 pages. Small adjustments in the size of the data area can be made by changing various parameters, although the tendency seems to be to *increase* those parameters.

As mentioned in the previous section, PDS operates using sequential i/o only. It is therefore possible to use FORTRAN i/o, although in practice modifications are necessary to avoid having to use FORTRAN unit numbers and to make use of named host files. PDS is written to use the i/o interface described in [3]; a FORTRAN version of these routines is provided along with suggested modifications.

Installation of the system using the FORTRAN version of the i/o interface can be accomplished in 1–2 man-days. The implementation of more sophisticated i/o systems requires substantial time investment, typically 3–6 man-months depending on the target system. This is unnecessary unless heavy use of the RATFOR i/o interface is anticipated. System-specific i/o systems are supplied for the DEC-10 and Cyber 175.

PDS can be modified to use standard FORTRAN i/o facilities directly (i.e., **read** and **write** statements). The modification requires replacing parts of the RATFOR i/o interface—the 300 lines of code mentioned above—with the appropriate standard FORTRAN i/o statements. While these modifications have *not* been made, similar experience suggests that 1–2 man-weeks is a conservative estimate of the effort required.

PDS is distributed with the following components:

PDS written in RATFOR
PDS written in FORTRAN (RATFOR output)
RATFOR and RATFOR i/o system written in RATFOR
RATFOR and RATFOR i/o system written in FORTRAN
DEC-10 i/o system
Cyber 175 i/o system

REFERENCES

1. HANSON, D.R. A portable file directory system. *Softw. Pract. Exper. 10,* 8 (Aug. 1980), 623–634.
2. KERNIGHAN, B.W. Ratfor—A preprocessor for a rational Fortran. *Softw. Pract. Exper. 5,* 4 (Dec. 1975), 396–406.
3. KERNIGHAN, B.W., AND PLAUGER, P.J. *Software Tools.* Addison-Wesley, Reading, Mass., 1976.
4. RITCHIE, D.M., AND THOMPSON, K. The UNIX timesharing system. *Commun. ACM 17,* 7 (July 1974), 365–375.
5. ROCHKIND, M.J. The source code control system. *IEEE Trans. Softw. Eng. SE-1,* 4 (Dec. 1975), 364–370.
6. RYDER, B.G. The PFORT verifier. *Softw. Pract. Exper. 4,* 4 (Dec. 1974), 359–377.
7. THOMPSON, K. UNIX implementation. *Bell Syst. Tech. J. 57,* 6 (July 1978), 1931–1946.

ALGORITHM

[The complete algorithm is available from the ACM Algorithms Distribution Service (see page 209 for order form)].