Operating Systems (CS F372)

Tutorial – 9

You will learn about Unix File systems in this tutorial.

File Systems

Files systems form the basic identity of data in any kind of storage device. The linux file system is special because it supports a large number of files systems ranging from journaling to clustering to cryptographic. Linux is a wonderful platform for using standard and more exotic file systems and also for developing file systems. The Linux file system architecture is an interesting example of abstracting complexity. Using a common set of API functions, a large variety of file systems can be supported on a large variety of storage devices. Take, for example, the read function call, which allows some number of bytes to be read from a given file descriptor. The read function is unaware of file system types, such as ext3 or NFS. It is also unaware of the particular storage medium upon which the file system is mounted. Yet, when the read function is called for a file, the data is returned as expected.

Everything in Linux can be reduced to a file. Partitions are associated with files such as /dev/hda1. Hardware components are associated with files such as /dev/modem. Detected devices are documented as files in the /proc directory. The Filesystem Hierarchy Standard (FHS) is the official way to organize files in Unix and Linux directories.

Several major directories are associated with all modern Unix/Linux operating systems. These directories organize user files, drivers, kernels, logs, programs, utilities, and more into different categories. The standardization of the FHS makes it easier for users of other Unix-based operating systems to understand the basics of Linux. Every FHS starts with the root directory, also known by its label, the single forward slash (/). All of the other directories shown below are subdirectories of the root directory. Unless they are mounted separately, you can also find their files on the same partition as the root directory.

/	The root directory, the top-level directory in the FHS. All other directories are subdirectories of root, which is always mounted on some partition. All directories that are not mounted on a separate partition are included in the root directory's partition.
/bin	Essential command line utilities. Should not be mounted separately; otherwise, it could be difficult to get to these utilities when using a rescue disk.

/boot	Includes Linux startup files, including the Linux kernel. Can be small; 16MB is usually adequate for a typical modular kernel. If you use multiple kernels, such as for testing a kernel upgrade, increase the size of this partition accordingly.
/etc	Most basic configuration files.
/dev	Hardware and software device drivers for everything from floppy drives to terminals. Do not mount this directory on a separate partition.
/home	Home directories for almost every user.
/lib	Program libraries for the kernel and various command line utilities. Do not mount this directory on a separate partition.
/mnt	The mount point for removable media, including floppy drives, CD-ROMs, and Zip disks.
/opt	Applications such as WordPerfect or StarOffice.
/proc	Currently running kernel-related processes, including device assignments such as IRQ ports, I/O addresses, and DMA channels.
/root	The home directory of the root user.
/sbin	System administration commands. Don't mount this directory separately.
/tmp	Temporary files. By default, Red Hat Linux deletes all files in this directory periodically.
/usr	Small programs accessible to all users. Includes many system administration commands and utilities.
/var	Variable data, including log files and printer spools.

A file system is an organization of data and metadata on a storage device. There are many types of file systems and media. With all of this variation, you can expect that the Linux file system interface is implemented as a layered architecture, separating the user interface layer from the file system implementation from the drivers that manipulate the storage devices. The **inode** is a very basic concept related to Unix file system. Each and every object/element in the file system is associated with an inode. Just like these identification numbers for people, there is a unique identity of every member of a Linux file system which is known as inode number and it uniquely exists for each and every individual file on Unix file systems. In any Linux/Unix file system, inodes occupy only up to 1% of the available disk space. The inode space is helpful to "track" the files saved in the hard disk memory. The inode entries store metadata about each object of the file system (file or directory) that just points to these structures and does not store any kind of data.

Accessing inode numbers :

1) `ls -i` : displays the inode numbers associated with each file.

2) `df -i` : displays the inode information.

3) `stat [filename]` : displays the file statistics including the inode number.

4) You can locate a file based on its inode number using:

```
find . -inum [inode-number]
```

The `stat()` system call : stat() is a system call that is used to determine the information about a file based on its file path.

```
int stat(const char *filename, struct stat *buf);
```

Related functions are **`fstat()`** and **`lstat()`**.

```
int lstat(const char *filename, struct stat *buf);
```

```
int fstat(int filedesc, struct stat *buf);
```

The functions take a `struct stat` buffer argument, which is used to return the file attributes. On success, the functions return zero, and on error, −1. The `stat()` and `lstat()` functions take a filename argument. If the file is a symbolic link, `stat()` returns attributes of the eventual target of the link, while `lstat()` returns attributes of the link itself. The `fstat()` function takes a file descriptor argument instead, and returns attributes of the file that it identifies.

The stat structure is defined in sys/stat.h header file as follows.

```
struct stat {
      mode_t st_mode;
      ino_t st_ino;
      dev_t st_dev;
      dev_t st_rdev;
      nlink_t st_nlink;
      uid_t st_uid;
      gid_t st_gid;
      off_t st_size;
      struct timespec st_atim;
      struct timespe    st_mtim;
      struct timespec st_ctim;
      blksize_t   st_blksize;
      blkcnt_t st_blocks;
};
```

An example code to use stat() is as follows :

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
int main(int argc, char **argv)
{
      if(argc != 2)
          return 1;
      struct stat fileStat;
      if(stat(argv[1],&fileStat) < 0)
          return 1;
      printf("Information for %s\n",argv[1]);
      printf("---------------------------\n");
      printf("File Size: \t\t%d bytes\n",fileStat.st_size);
      printf("Number of Links: \t%d\n",fileStat.st_nlink);
      printf("File inode: \t\t%d\n",fileStat.st_ino);

      printf("File Permissions: \t");
      printf( (S_ISDIR(fileStat.st_mode)) ? "d" : "-");
      printf( (fileStat.st_mode & S_IRUSR) ? "r" : "-");
      printf( (fileStat.st_mode & S_IWUSR) ? "w" : "-");
      printf( (fileStat.st_mode & S_IXUSR) ? "x" : "-");
      printf( (fileStat.st_mode & S_IRGRP) ? "r" : "-");
      printf( (fileStat.st_mode & S_IWGRP) ? "w" : "-");
      printf( (fileStat.st_mode & S_IXGRP) ? "x" : "-");
      printf( (fileStat.st_mode & S_IROTH) ? "r" : "-");
      printf( (fileStat.st_mode & S_IWOTH) ? "w" : "-");
      printf( (fileStat.st_mode & S_IXOTH) ? "x" : "-");
      printf("\n\n");
      printf("The file %s a symbolic link\n", (S_ISLNK(fileStat.st_mode)) ?
                                                               "is" : "is not");
      return 0;
}
```

The following program calls **stat**() and displays selected fields in the returned *stat* structure.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
 struct stat sb;
 if (argc != 2) {
     fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
     exit(EXIT_FAILURE);
 }
 if (stat(argv[1], &sb) == -1) {
    perror("stat");
    exit(EXIT_FAILURE);
 }

 printf("File type: ");
 switch (sb.st_mode & S_IFMT){
        case S_IFBLK:  printf("block device\n");
       break;
        case S_IFCHR:  printf("character device\n");
       break;
        case S_IFDIR:  printf("directory\n");
       break;
        case S_IFIFO:  printf("FIFO/pipe\n");
       break;
        case S_IFLNK:  printf("symlink\n");
       break;
        case S_IFREG:  printf("regular file\n");
       break;
        case S_IFSOCK: printf("socket\n");
       break;
        default:       printf("unknown?\n");
  }

  printf("I-node number: %ld\n", (long) sb.st_ino);
  printf("Mode: %lo (octal)\n",(unsigned long) sb.st_mode);
  printf("Link count: %ld\n", (long) sb.st_nlink);
  printf("Ownership: UID=%ld   GID=%ld\n",(long) sb.st_uid,(long) sb.st_gid);
  printf("Preferred I/O block size: %ld bytes\n", (long) sb.st_blksize);
  printf("File size: %lld bytes\n",(long long) sb.st_size);
  printf("Blocks allocated:%lld\n",(long long) sb.st_blocks);

  exit(EXIT_SUCCESS);
}
```

Challenge problem : Take input of an inode number from user and locate the file and print all its contents using a C program.