CS 452 Lab 11
Files and File Systems - Information Access

Overview

The purpose of this lab assignment is to investigate characteristics of modern file systems. Specifically, the goal is to improve your understanding of file storage, file access, file information management, directory structure, and file system traversal.

Activities

- Work your way through the following exercises, demonstrating your knowledge of the material by answering the numbered questions.
- Submit a detailed lab report. Include the answers to the numbered questions and all of your source code and lab scripts. Pay special attention to the requests for verification. Be prepared to demonstrate your programs.

Files

Every file in the UNIX file system, including directories, has an associated inode. An inode is a data structure that contains all of the information the system maintains on every file. For example, an inode may contain information on file access permissions, file creation time, block size and disk block addresses. This information is used by the file system to find files on secondary storage, and to perform auditing and administrative functions. The "location" information helps implement the mapping of logical file addresses to physical disk blocks (this is how filesystems locate specific bytes in a file - reminiscent of paged virtual memory management). The administrative information is what will be investigated in this lab.

Some of the information contained in an inode can be viewed using the **`stat()`** function. The mechanism should be familiar - the system call is given a filename and provided with a user-supplied structure. It works by filling the appropriate fields of the predefined structure with the current values from the specified inode; users then access this structure to obtain information about the file the inode describes.

Take a moment to read the man pages for **`fstat()`** and **`stat()`** (be sure to read both the **`stat(1)`** and **`stat(2)`** pages). Additional information on the data structures and associated macros can be found in the `/usr/include/sys/stat.h` and `/usr/include/bits/stat.h` include files.

Then carefully examine the following program that accesses an inode data structure:

Sample Program 1

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

int main(int argc, char *argv[])
{
   struct stat statBuf;

   if (argc < 2) {
      printf ("Usage: filename required\n");
      exit(1);
   }

   if (stat (argv[1], &statBuf) < 0) {
      perror ("huh?  there is ");
      exit(1);
   }

   printf ("value is: %u\n", statBuf.st_mode);
   return 0;
}
```

1. Answer the questions and perform the following operations:

- what is the difference between **`stat(2)`** and **`fstat(2)`**?
- compile and test Sample Program 1
  - use a source code file and an executable file as test inputs
- what *exactly* does Sample Program 1 do?
- modify the Sample Program so that it does the following:
  - tests whether a file is a directory or not
- start a script file and verify that your program works
  - use Sample Program 1 and your current directory as your test inputs
  - *verify*/demonstrate correctness by testing your program against the **`stat(1)`** utility

Directories

File access proceeds via directories. A directory is itself a file, and simply contains a list of tuples consisting of filenames and their corresponding inode numbers. Programs that need to open files or report information about files (e.g. "cat", "ls") begin by searching the directory for the specified filename. Upon finding the filename, they use the associated inode number to access the inode and from there, the file.

Each directory entry consists of a <filename : inode #> tuple. Being a file, the contents of a directory can be examined. The relevant system calls are:

- **`opendir()`** - opens the named directory and associates a stream with it
- **`readdir()`** - returns a pointer to the next directory entry
- **`closedir()`** - closes the named directory stream

Time to peruse the man pages again to understand the mechanics of the above functions and the data structures they utilize. Then study the following program.

Sample Program 2

```
#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <errno.h>

int main()
{
   DIR *dirPtr;
   struct dirent *entryPtr;

   dirPtr = opendir (".");

   while ((entryPtr = readdir (dirPtr)))
      printf ("%-20s", entryPtr->d_name);

   closedir (dirPtr);
   return 0;
}
```

2. Complete the following operations:

- compile and test Sample Program 2
- what *exactly* does Sample Program 2 do?
- modify Sample Program 2 so that it cleans up the output, and also reports the size of each file
- start a script file and verify that your program works
    - *verify*/demonstrate correctness by testing your program against "**`ls -l`**"

File Systems

The UNIX file system is hierarchical, and implements a general graph structure that includes hard and symbolic links. Because of its tree-like organization, tree traversal algorithms (such as the recursive depth-first-search and breadth-first-search routines) are generally used to traverse the file system to access files and directories. Note: this process is slightly complicated by the presence of symbolic links and the subsequent possibility of cycles in the graph.

The utility **du** (disk usage) is an example of a file system traversal program. It provides a summary of the amount

of disk space currently occupied by a user's files - this information is totaled by directory. The program operates by recursively descending into a user's subdirectories to report statistics on the entire subtree.

Read the man pages for the **du** utility and experiment with it. While you are doing this, remember the definitions of depth-first-search (dfs) and breadth-first-search (bfs), as covered in the data structures class.

3. Answer the following questions:

- based on the order of information provided, which of the two traversal algorithms does **du** use?
- use **du** to report the usage of all the files in the current directory
    - what is the default block size used by **du**?
    - why is the usage reported in blocks, instead of bytes?

Programming Assignment (Directory Listing)

Using what you have learned while experimenting with the Sample Programs given in the lab, write a program that implements some of the functionality of the "**`ls`**" command. In particular, your program should use the file and directory system calls presented in the lab to list:

- filename
- inode number

Your program should accept as input the name of *any* directory, whose contents will then be listed.
Verify/demonstrate the correctness of your program by testing it against the '**ls -i**' command