CS 452 Lab Week 12

File System Interface: Information

# 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 output samples.
  + **Note**: pay special attention to the requests for verification.
  + In particular, *submit recorded script files or screenshots* that demonstrate the correct operation of your programs.
* As usual, be prepared to demonstrate your final program.

### 

# 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 a filesystem locates 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 in the appropriate fields of the predefined structure with the current values from the inode of the specified file; 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 all of the **stat(1-3)** pages (user program, system call, library routine).  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.   Perform the following operations and answer the associated questions:**

* access the man pages

1. what is the difference between **stat(1)** and **stat(3)**?

* compile and test Sample Program 1
  + use a source code file and then an executable file as test inputs

1. what *exactly* does Sample Program 1 do?

* modify Sample Program 1 so that it does the following:
  + reports whether a file is a directory or not

1. verify that your program works. Submit your modified program (or the relevant lines of modified source code), and a script file or screenshot showing its execution.
   * use Sample Program 1 (source code) and your current directory as your test inputs.  *Verify* and demonstrate the correctness of your program by testing its output against the output of the **stat(1)** utility, e.g.:
     + **stat sample1.c**
     + **./a.out sample1.c**

### 

# 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\n", entryPtr->d\_name);

   closedir (dirPtr);

   return 0;

}

**2.  Perform the following operations and answer the associated questions:**

* compile and test Sample Program 2

1. what *exactly* does Sample Program 2 do?
   * TODO

* modify Sample Program 2 so that it also reports the size of each file (bytes) in the directory
  + ensure your output is human-friendly (i.e. readable)

1. verify that your program works.  Submit your modified program (or the relevant lines of modified source code), and a script file or screenshot showing its execution.
   * use your source code as a test case*.  Verify* and demonstrate the correctness of your program by testing it against the **ls** program in the current directory, e.g.:
     + **ls -l**
     + **./a.out**

### 

# 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 filesystem to access files and directories.  Note: this process is slightly complicated by the presence of symbolic links and the subsequent possibility of cycles.

The system 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 usage statistics on each entire subtree.

Read the man pages for the **du** utility and experiment with it.  While you are doing this, recall the definitions of depth-first-search (DFS) and breadth-first-search (BFS), as covered in your data structures and/or discrete math class(es).

**3.  Answer the following questions:**

* use **du** to report the usage of all the files in some of your directories (be sure to choose some with subdirectories)

1. based on the *order* of information provided, which of the two tree traversal algorithms does **du** use?
2. what is the default block size used by **du**?
3. speculate: given the intended purpose of **du**, why is the usage reported in blocks, instead of bytes?

# Programming Assignment (ls - Directory Listing)

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

* file size in bytes
  + **ls -l**  // gives the file size in bytes (and much other info)
* file inode #
  + **ls -i**  // gives the file inode #

Your program should accept as input the name of *any* directory, whose contents will then be listed, along with any specified information (just like '**ls**').  Verify that your program works.  Submit your source code and a script file or screenshot demonstrating your program in execution.

* ***Verify* and demonstrate the correctness of your program by testing it against the 'ls' command for the given directory with appropriate options.  For example:**
  + **[eos22:~]$ ls -i /lab/web/eosLabs**
  + **[eos22:~]$ ./a.out -i /lab/web/eosLabs**

# Extra Credit

Read the man pages for '**ls**'.  Incorporate additional functionality into your version of the utility (i.e. provide more of the options of the '**ls**' command).  Be sure to submit sample output that demonstrates the enhanced functionality of your program.