CIS 452

Lab 12 Report

Ashley Hendrickson

Muna Gigowski

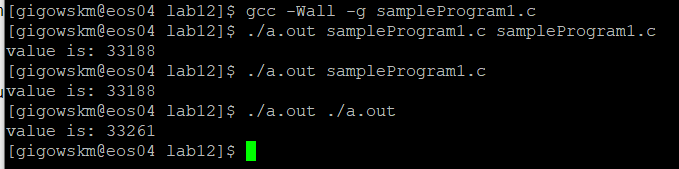
Fall 2019

Files

1. What is the difference between stat(1) and stat(2)?

stat(1) is a Linux command that takes in a file name and specified argument(s) then, based on the provided arguments(s), prints various statistics of that file or files system to the command window. stat(2) on the other hand is a Linux system call, which takes in a file name and a user-defined structure of type stat, then fills that structure’s fields with the current values from the inode of the specified file.

1. What exactly does Sample Program 1 do?



This program first creates a struct of type stat, then checks to ensure that the number or arguments passed into the program is adequate (a file name needs to be provided to the program). The program then calls stat() on the filename that was provided via the program arguments, also passing it the stat struct that was created earlier to hold the retuned file data. Assuming the stat call completed successfully, the program then prints to the screen the value of “statBuf.st\_mode” - which displays the file type and mode - before returning.

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.

REVISED SAMPLE PROGRAM 1 CODE:

#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);

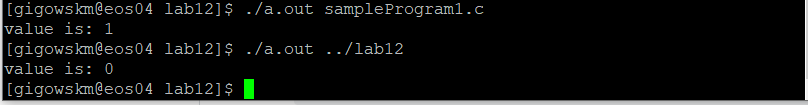
   }

   printf ("value is: %u\n", S\_ISREG(statBuf.st\_mode));

   return 0;

}

SCREENSHOT OF EXECUTION:



Directories

1. What exactly does Sample Program 2 do?

The program begins by creating a directory pointer and struct pointer of type dirent; it then assigns the directory pointer to an opendir() call that takes in the current directory as its argument. The program then assigns the dirent struct pointer to a readdir() call that takes in the directory pointer (pointing now to the current directory) as its argument, and loops until the dirent struct pointer returns null - printing “entryPtr->d\_name” on each iteration. Finally, the program calls closedir() on the directory pointer to close the stream before returning.

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.

REVISED SAMPLE PROGRAM 2 CODE:

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <errno.h>

int main()

{

   DIR \*dirPtr;

   struct dirent \*entryPtr;

   struct stat statBuf;

   dirPtr = opendir (".");

   while ((entryPtr = readdir (dirPtr))) {

      printf ("%-20s\n", entryPtr->d\_name);

      if (stat (entryPtr->d\_name, &statBuf) < 0) {

        perror ("huh?  there is ");

        exit(1);

     } else {

       printf ("Size in bytes is: %li\n", statBuf.st\_size);

     }

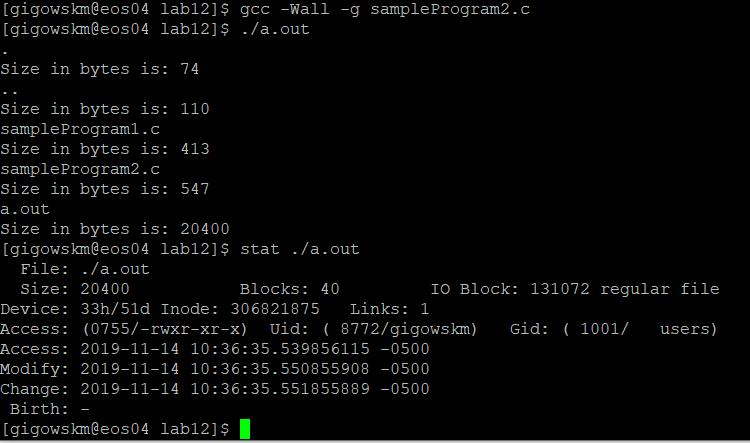
    }

   closedir (dirPtr);

   return 0;

}

SCREENSHOT OF EXECUTION:



File Systems

1. Based on the order of information provided, which of the two tree traversal algorithms does du use?

Du uses depth-first-search (DFS)

1. What is the default block size used by du?

1024 bytes per block

1. Speculate: given the intended purpose of du, why is the usage reported in blocks, instead of bytes?

A block size isn’t just some random memory. A block is a group of sectors that the operating system can address. There is a limit to the number of blocks an operating system can address. By defining a block as several sectors, an OS can work with bigger hard drives without increasing the number of block address. When you increment the block size reading and writing from the disk is slower and **more space is wasted** but when you decrease the block size then reading and writing is faster and **less memory is wasted**. When more memory is allocated to a block the chance of internal fragmentation is increased

Programming Assignment (ls - Directory Listing)

lsProgram.c Source Code:

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <errno.h>

#include <dirent.h>

#include <string.h>

#include <unistd.h>

int main(int argc, char \*argv[])

{

   //declare variables

   struct stat statBuf;

   DIR \*dirPtr;

   struct dirent \*entryPtr;

   char \*filePathPtr;

   char actualpath [4096+1];

    if (argc < 2) {

      printf ("Usage: directory name required\n");

      exit(1);

    }

//Take user-specified directory and create an absolute path to it

if((filePathPtr = realpath(argv[1], actualpath)) == NULL) {

  printf ("Usage: realpath failed\n");

      exit(1);

}

   //open stream in the specified directory for reading in data

   dirPtr = opendir (filePathPtr);

   //loop through every file in the specified directory, and print out its statistics

   char \*currentFilePath = malloc(150);

   while ((entryPtr = readdir (dirPtr))) {

  strcpy(currentFilePath, filePathPtr);

  strcat(currentFilePath, "/");

  strcat(currentFilePath, entryPtr->d\_name);

     if (lstat (currentFilePath, &statBuf) < 0) {

        printf ("Path that errored out: %s ", currentFilePath);

        perror ("huh?  there is ");

        exit(1);

     } else {

  //print file statistics

  printf ("User ID: %u ", statBuf.st\_uid);

  printf ("Group ID: %u ", statBuf.st\_gid);

       printf ("Inode number: %li ", entryPtr->d\_ino);

  printf ("%-20s\n", entryPtr->d\_name);

     }

    }

   //close data stream

   closedir (dirPtr);

   free(currentFilePath);

   return 0;

}

SCREENSHOT OF EXECUTION:

