

# 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.

2. What exactly does Sample Program 1 do?

```
[gigowskm@eos04 lab12]$ gcc -Wall -g sampleProgram1.c
[gigowskm@eos04 lab12]$ ./a.out sampleProgram1.c sampleProgram1.c
value is: 33188
[gigowskm@eos04 lab12]$ ./a.out sampleProgram1.c
value is: 33188
[gigowskm@eos04 lab12]$ ./a.out ./a.out
value is: 33261
[gigowskm@eos04 lab12]$
```

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 returned 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.

3. 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);
    }
}
```

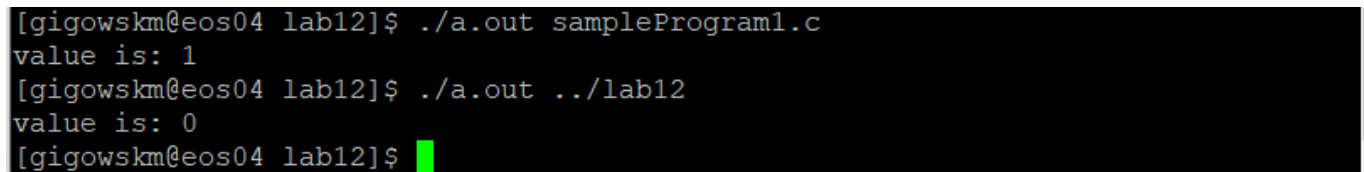
```

    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:



```

[gigowskm@eos04 lab12]$ ./a.out sampleProgram1.c
value is: 1
[gigowskm@eos04 lab12]$ ./a.out ../lab12
value is: 0
[gigowskm@eos04 lab12]$ █

```

## Directories

### 4. 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.

### 5. 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;

```

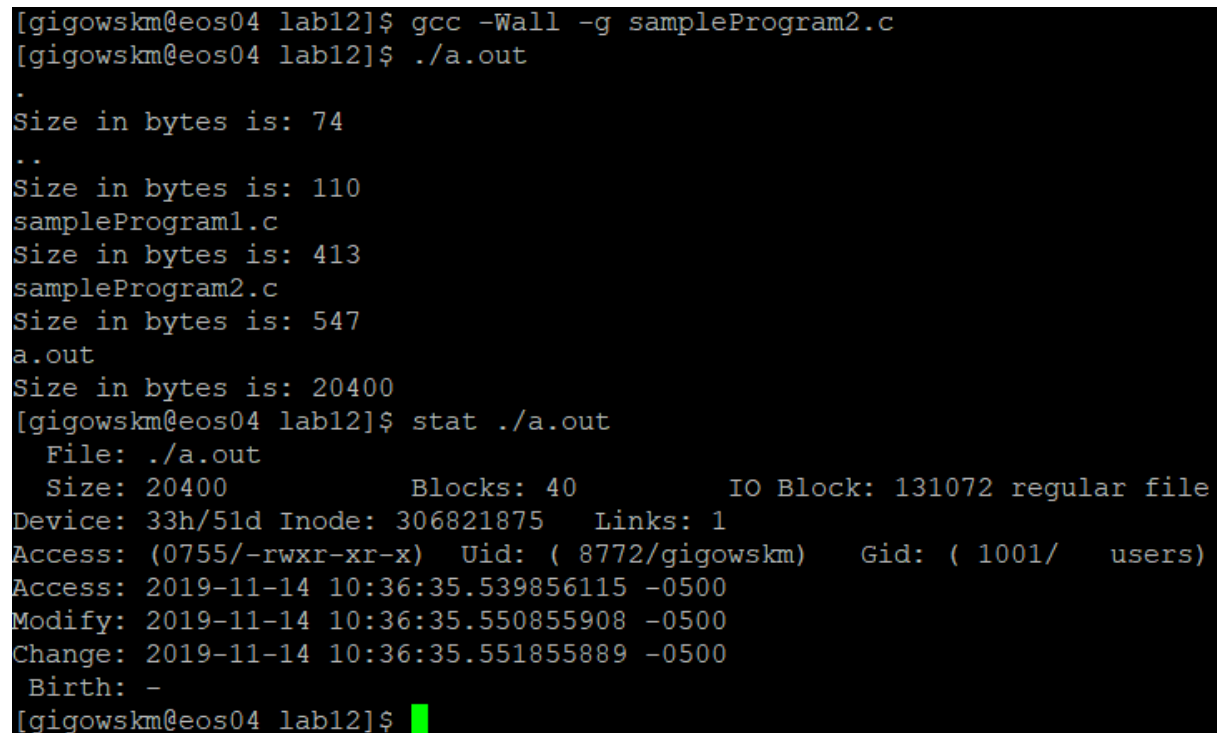
```

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:



```

[gigowskm@eos04 lab12]$ gcc -Wall -g sampleProgram2.c
[gigowskm@eos04 lab12]$ ./a.out
.
Size in bytes is: 74
..
Size in bytes is: 110
sampleProgram1.c
Size in bytes is: 413
sampleProgram2.c
Size in bytes is: 547
a.out
Size in bytes is: 20400
[gigowskm@eos04 lab12]$ stat ./a.out
  File: ./a.out
  Size: 20400          Blocks: 40          IO Block: 131072 regular file
Device: 33h/51d Inode: 306821875   Links: 1
Access: (0755/-rwxr-xr-x)  Uid: ( 8772/gigowskm)   Gid: ( 1001/  users)
Access: 2019-11-14 10:36:35.539856115 -0500
Modify: 2019-11-14 10:36:35.550855908 -0500
Change: 2019-11-14 10:36:35.551855889 -0500
 Birth: -
[gigowskm@eos04 lab12]$ █

```

## File Systems

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

Du uses depth-first-search (DFS)

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

1024 bytes per block

8. 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 – it 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 ("%s\n", entryPtr->d_name);
    }
}

//close data stream
closedir (dirPtr);
free(currentFilePath);
return 0;
}

```

## SCREENSHOT OF EXECUTION:

```
[gigowskm@eos04 lab12]$  
[gigowskm@eos04 lab12]$ gcc -Wall -g lsProgram.c  
[gigowskm@eos04 lab12]$ ./a.out /lab/bin  
User ID: 0 Group ID: 985 Inode number: 16647480 copyWindows10VM  
User ID: 0 Group ID: 985 Inode number: 16646149 goBladeCIS  
User ID: 0 Group ID: 985 Inode number: 16646150 copyWindows2012r2VM  
User ID: 0 Group ID: 985 Inode number: 16647454 copyUbuntuVM  
User ID: 0 Group ID: 0 Inode number: 16646633 PackLSD.64.x  
User ID: 0 Group ID: 0 Inode number: 16646470 sim-fast  
User ID: 0 Group ID: 985 Inode number: 16646151 copySeedVM  
User ID: 0 Group ID: 0 Inode number: 16646468 sim-bpred  
User ID: 0 Group ID: 985 Inode number: 16646146 .  
User ID: 0 Group ID: 0 Inode number: 16646472 sim-profile  
User ID: 0 Group ID: 0 Inode number: 16646634 PackLSD.Electric.64.x  
User ID: 0 Group ID: 0 Inode number: 16646534 dlxview  
User ID: 0 Group ID: 985 Inode number: 16646152 copyWindows2019VM  
User ID: 0 Group ID: 0 Inode number: 16646471 sim-outorder  
User ID: 0 Group ID: 985 Inode number: 16646148 copyCentOS7VM  
User ID: 0 Group ID: 0 Inode number: 16646467 sim-cache  
User ID: 0 Group ID: 0 Inode number: 16646474 sysprobe  
User ID: 0 Group ID: 985 Inode number: 16646145 ..  
User ID: 0 Group ID: 985 Inode number: 16646147 copyMininetVM  
User ID: 0 Group ID: 0 Inode number: 16646469 sim-eio  
User ID: 0 Group ID: 0 Inode number: 16646473 sim-safe  
[gigowskm@eos04 lab12]$ ls -li -n /lab/bin  
total 32  
16646633 lrwxrwxrwx 1 0 0 30 Aug 26 08:42 PackLSD.64.x -> /lab/apps/PackLSD/PackLSD.64.x  
16646634 lrwxrwxrwx 1 0 0 39 Aug 26 08:42 PackLSD.Electric.64.x -> /lab/apps/PackLSD/PackLSD.Electric.64.x  
16646148 -rwxr-xr-x 1 0 985 768 Aug 26 07:56 copyCentOS7VM  
16646147 -rwxr-xr-x 1 0 985 768 Aug 26 07:56 copyMininetVM  
16646151 -rwxr-xr-x 1 0 985 765 Aug 26 07:57 copySeedVM  
16647454 -rwxr-xr-x 1 0 985 767 Aug 28 11:08 copyUbuntuVM  
16647480 -rwxr-xr-x 1 0 985 774 Sep 18 12:40 copyWindows10VM  
16646150 -rwxr-xr-x 1 0 985 804 Aug 26 07:57 copyWindows2012r2VM  
16646152 -rwxr-xr-x 1 0 985 812 Oct 30 19:35 copyWindows2019VM  
16646534 lrwxrwxrwx 1 0 0 28 Aug 26 08:31 dlxview -> /lab/apps/dlxview0.9/dlxview  
16646149 -rwxr-xr-x 1 0 985 496 Aug 26 07:56 goBladeCIS  
16646468 lrwxrwxrwx 1 0 0 33 Aug 26 08:25 sim-bpred -> /lab/apps/simplesim-3.0/sim-bpred  
16646467 lrwxrwxrwx 1 0 0 33 Aug 26 08:25 sim-cache -> /lab/apps/simplesim-3.0/sim-cache  
16646469 lrwxrwxrwx 1 0 0 31 Aug 26 08:25 sim-eio -> /lab/apps/simplesim-3.0/sim-eio  
16646470 lrwxrwxrwx 1 0 0 32 Aug 26 08:25 sim-fast -> /lab/apps/simplesim-3.0/sim-fast  
16646471 lrwxrwxrwx 1 0 0 36 Aug 26 08:25 sim-outorder -> /lab/apps/simplesim-3.0/sim-outorder  
16646472 lrwxrwxrwx 1 0 0 35 Aug 26 08:25 sim-profile -> /lab/apps/simplesim-3.0/sim-profile  
16646473 lrwxrwxrwx 1 0 0 32 Aug 26 08:26 sim-safe -> /lab/apps/simplesim-3.0/sim-safe  
16646474 lrwxrwxrwx 1 0 0 32 Aug 26 08:26 sysprobe -> /lab/apps/simplesim-3.0/sysprobe  
[gigowskm@eos04 lab12]$
```