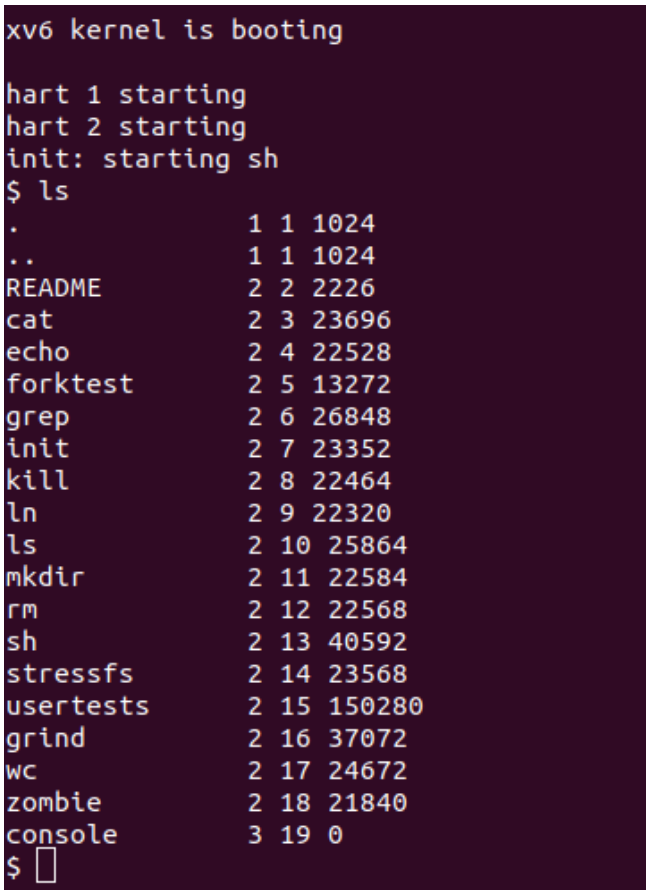
Dr. Shirley Moore
12/06/2023
Alberto Delgado
aodelgado@miners.utep.edu

Homework 7 Report

The ls command in the Linux operating system is a command that is used to display a list of all the files in the current working directory. When working with ls in xv6 and QEMU it works similarly. The ls command is an important function that helps user know what files are in the directory they are currently in. When running ls in QEMU we get the following output:



When running the ls command in the terminal we can see various rows and columns outputted in the terminal. The first column with the words is the column that shows the name of the file in the directory. The second column describes the characteristic of the first column name. The number 1 represents a directory, number 2 represents a file and the number 3 represents other file types. The third column shows the inode number of the file or directory. The fourth and final column shows the byte size for each row type.

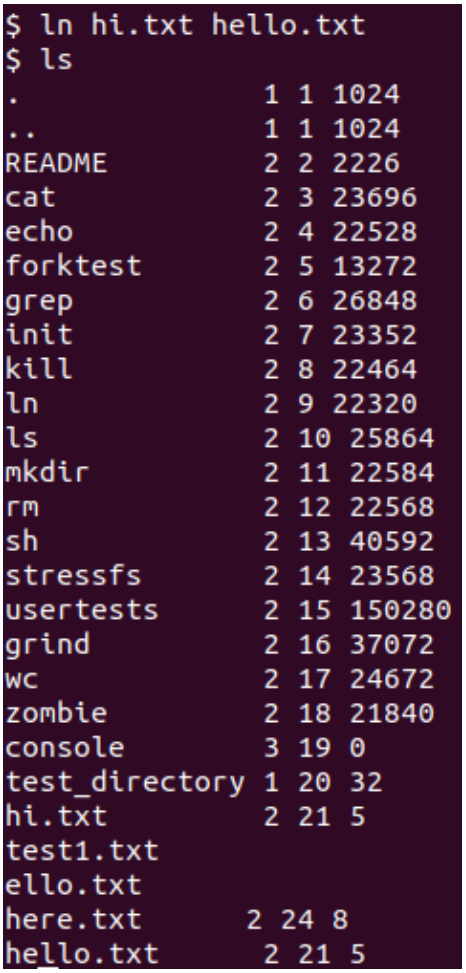
In the code we begin with the header files for other files within the xv6 working directory. Inside the ls function there is a system call, “open” which is used to open specified files. The other portions are used to get the information of specified files such as fstat in the ls function. This gets the information of the directory or file and displays onto the terminal screen. After the file has been opened once all the desired actions are taken, then the function calls close() which will close the files that had been opened.

The command mkdir is a command used in linux to create a new directory that can be used for storing any desired files. Created a new directory separate from others is important for any user who may need to have specific files that are specific for that directory. The files in the new directory can help in organization of files for any project. Below is a screenshot of the output from the ls once I ran the mkdir command.

As shown in the screenshot, there is a new directory once running the ls command. We can verify that the test_directory, is a directory since the second column shows a number 1 which specifies the type which is directory.

Once we delve into the code we can make sense of how this works. Just like the previous file, the header files are calling specific functions and components that will be needed to execute the mkdir command. In the code that is given, we see that the mkdir system call is used. The rest of the code within the file is used to output to the user the success of failure of a directory creation.

The command ln in Linux is generally used to create links, specifically links between files and directories. The ln command is good to use if a user wants to link a current file into a new file, this is useful whenever a user needs to have multiple instances of data. It can also cause issue if there are removals of item that are original links as all the files linked to the original file will lose the data. Below is a screenshot showing the ls output prior and post ln entry.



As we can see prior to using ln, I only had two text files that I echoed to be used for this example. For ln to function correctly we need an original file which in this case was my hi.txt and an empty file to create with the ln which I named “hello.txt”. The screenshot on the right shows the output after I run the ln command. As we can see, there is now a new file named “hello.txt” which was linked to the “hi.txt” file. For some reason when I was creating files to be used I got an entry of test1.txt and ello.txt which I do not know how they were entered without information but regardless the other test files functioned properly with the ln command. I believe that an incorrect implementation of echo was the cause other extra file in the screenshots.

Just as I stated in the first section of this assignment, fstat() is used to get information for a specific file and display that information in the terminal. To get a better understanding how the fstat() function operated we need to go the helpoer function which is located in the kernel directory. The full name of the function is filestat() and does exactly what the name suggests. So the function takes in a pointer and an address as arguments in the function to be used for determining the file that will use the fstat() functionality. In the if statement there is a condition to determine the file type of whatever file is being examined. Within the if statement the portion that obtains the information is the stati() function which gets all statistics for the desire file. Before and after the stati() function there are functions which lock and unlock inodes, this is to ensure that the inode is locked on desired file and once the stat retrieval is complete then the inode is unlocked. The second if statement is used to copy the statistics to the user space and is deemed a successful retrieval, if the retrieval is not successful then a “-1” is returned. Below is a screenshot of the code that I just discussed.

```
// Get metadata about file f.
// addr is a user virtual address, pointing to a struct stat.
int
filestat(struct file *f, uint64 addr)
{
  struct proc *p = myproc();
  struct stat st;

  if(f->type == FD_INODE || f->type == FD_DEVICE){
    ilock(f->ip);
    stati(f->ip, &st);
    iunlock(f->ip);
    if(copyout(p->pagetable, addr, (char *)&st, sizeof(st)) < 0)
      return -1;
    return 0;
  }
  return -1;
}
```

The stat command in Linux is used similarly to the fstat() function previously discussed. Stat obtains the statistics of the file that is entered along with the command. To successfully obtain the information then we need to enter the command into the terminal as follows: stat <filename>. In this portion of the assignment we are tasked to also use stat with a directory which will be the same but instead of a file name, we use the name of a directory. Below is a screenshot for both the directory and file.

```
(base) aodelgado@aodelgadolinux:~/Desktop$ stat Delgado_Alberto_HW4_Report.pdf
  File: Delgado_Alberto_HW4_Report.pdf
  Size: 208166          Blocks: 408        IO Block: 4096   regular file
Device: 10307h/66311d   Inode: 52050994    Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/aodelgado)   Gid: ( 1000/aodelgado)
Access: 2023-11-10 23:29:20.541318185 -0700
Modify: 2023-11-10 23:29:11.120897597 -0700
Change: 2023-11-10 23:29:11.120897597 -0700
 Birth: 2023-11-10 23:29:11.120897597 -0700
(base) aodelgado@aodelgadolinux:~/Desktop$ stat xv6-riscv-labs-riscv-HW5
  File: xv6-riscv-labs-riscv-HW5
  Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: 10307h/66311d   Inode: 55479252    Links: 5
Access: (0775/drwxrwxr-x)  Uid: ( 1000/aodelgado)   Gid: ( 1000/aodelgado)
Access: 2023-12-06 00:34:40.604688368 -0700
Modify: 2023-12-01 21:46:36.599013014 -0700
Change: 2023-12-01 21:46:36.599013014 -0700
 Birth: 2023-12-01 15:25:03.340069041 -0700
```

For the final task of this assignment we are tasked to create an fstat command in QEMU that will work just like the stat command in Linux. To get the functionality of the fstat command to work I started by creating a code file that will be used by the user to call the command and get the information of a desired file. All of the code that I implemented into the file is named “fstat.c” and is in the user directory of the assignment. I begin by implementing the necessary header files from the kernel directory as well as the user directory. After this the first if statement that I implement is used to check if the command entered is a valid input, if the input does not meet the requirement such as entering a file to be analyzed, then the system will output a statement stating what is necessary for the fstat to function properly. After this I declare two variables to be used when obtaining and displaying the information of the file. The char *path is used to get the specified file that was taken as an argument at the command line following the fstat command. The struct stat st is used to declare a structure that will be used to store the data that is taken from the file. The second if statement is used to retrieve the information from the specified file. I call fstat() since it was a default function given to us from the assignment. This is what will retrieve all the desire information to de displayed to the user. If the fstat is unsuccessful then the if statement will output an error message stating the file can not be analyzed properly. Otherwise the program will output all the information is a readable manner so the user can know all statistical information of the desired file. Once I competed the code I made sure to enter the “$U/_fstat\” into the make file to ensure that the user can use the command in the terminal. Once I completed this, I tested out the command. Below is a screenshot of the output for the fstat command.

```
hart 1 starting
hart 2 starting
init: starting sh
$ fstat
Usage: fstat <file or directory>
$ ls
.              1 1 1024
..             1 1 1024
README         2 2 2226
cat            2 3 23696
echo           2 4 22528
forktest       2 5 13272
grep           2 6 26848
init           2 7 23352
kill           2 8 22464
ln             2 9 22320
ls             2 10 25864
mkdir          2 11 22584
rm             2 12 22568
sh             2 13 40592
stressfs       2 14 23568
usertests      2 15 150280
grind          2 16 37072
wc             2 17 24672
zombie         2 18 21840
fstat          2 19 22840
console        3 20 0
$ fstat cat
File: cat
Type: 2
Size: 23696
Inode: 3
Links: 1
```

As we can see the fstat command was able display the information for the cat file. Also we can see that when fstat is entered without a file or directory then the error is projected.

In conclusion, I was able to better understand how some very major and key commands in the Linux operating system work. Although we work in the QEMU environment, the command function similarly to that of the Linux operating system. Commands that I have become accustomed to using frequently I now better understand thanks in part to this assignment. It is a great experience to see how the functions are implemented and how they function together to get a efficient and fluid user experience. I never knew that such command could require such intricate detail and complexity to function properly such as the ls function which seems like such a simple process to create. After going through this assignment I now know that the commands and functions that are important to us require much attention to operate properly. The more I see the work that happens behind the scenes of an operating system the more I appreciate the operating system and the developers who have created them. This was truly a great learning experience and I do believe that this is barely scratching the surface but I look forward to obtaining more knowledge in this area in the future.