CIS 415 Operating Systems

Project <1> Report Collection

Submitted to:

Prof. Allen Malony

Author:

<Andrew Chan>

<UO ID - achan10>

<Duck ID>

Report

Introduction

In this project, we are making a CLI (command line interface) / pseudo shell that has basic functionalities of UNIX commands. These commands feature: ls, pwd, mkdir, cd, cp, mv, rm, and cat. There are two different modes that function; file mode and interactive mode. We can access file mode by passing the -f flag when executing the program, which is then followed by the input file. Doing so will result in the output file labeled "output.txt". The input file should be a list of commands which will be done through this pseudo-shell. Alternatively you can also use interactive mode by not passing in any flags and this will result in a CLI where users can manually type in their commands. This is the same as using a command line interface normally. This project looked at the idea of improving our general understanding of not only how linux commands work, but more importantly, how system calls are done in C and how they interact with the operating system. Also, it developed our understanding of how to manipulate files and directories at a lower level as opposed to using high level libraries like stdio.

Background

This project uses system calls such as: open, close, read, write, mkdir, chdir, rename, remove, etc. to perform the file and directory actions. These calls allow us to directly interact with the file system at the kernel level. Since the project specifically called for only the use of system calls, this made it a little more difficult when handling the operations. One place in particular I found more difficult than I originally intended was the cp and mv commands. This is because the input can consist of not just files, but also directories. This is where I learned about the stat function. This is a function that checks if the destination is a file or a directory. This was very important and ended up being one of the more important parts of those commands since we can perform different actions accordingly. This is because if a file were to be moved or copied into a directory we will need to require additional path manipulation.

Also another thing I decided to do was to create a helper function that writes a given message. This seems counterintuitive but the reason why I had to use this roundabout way, was because we weren't allowed to print anything, only using the system call, "write". I created the helper function "write_message" to do a repeated tasks easier. Since when you call write(), it takes in 3 arguments, fd (file descriptor), buffer, and count (buffer length), that means that when I want to write a message I either have to create a separate char buf[] or pass in the strlen(buf). To alleviate this process, I wrote a helper function where you can just straight up pass in the string and it will call the write function without having to do the process each time.

Implementation

When it came to actually implementing the project the biggest issue I had was with the rm command but it turns out that the test script was just wrong. Thankfully I was able to catch the error in the test_script.sh but other than that I think all of the actual implementation was good. One thing that was a little tricky was the mv and cp commands having to use stat and the S_ISDIR was a little weird since it wasn't like the functions previous to them. The reason why it was different was because we had to check to see if the destination was either a file or directory. In the end, it was a simple call of stat(destinationPath, &destStat). And we then compared that to the S_IRDIR that looked like this:

```
struct stat destStat;

if (stat(destinationPath, &destStat) == 0 && S_ISDIR(destStat.st_mode)) {
  char *baseName = strrchr(sourcePath, '/');
  baseName = baseName ? baseName + 1 : sourcePath;

  char newDestPath[1024];
  strcpy(newDestPath, destinationPath);
  strcat(newDestPath, "/");
  strcat(newDestPath, baseName);
  destinationPath = newDestPath;
}
```

Another implementation that I thought was pretty "nifty" was using the errno.h library to dynamically display the error messages with some of the errors provided. For example, I used errno in changeDir to look at different reasons why it would not be able to change to the provided directory. Whether it was because the directory doesn't exist, or the specified argument wasn't a directory, it handled it well using a switch statement.

```
void changeDir(char *dirName){
  if(chdir(dirName) != 0){
    switch(errno){
      case ENOENT:
        write_message("Error! Directory does not exist\n");
        break;
      case ENOTDIR:
        write_message("Error! ");
        write_message(dirName);
        write_message(" is not a directory\n");
        break;
      default:
        write_message("Error! Unable to change to directory: ");
        write_message(dirName);
        write(1, "\n", 1);
        break;
    }
  }
} /*for the cd command*/
```

Performance Results and Discussion

I believe my code runs well. After extensive testing I don't believe that there are any memory leaks or errors in the code. That being said however, I believe that all code has mistakes or bugs to some extent so I

wouldn't be surprised if I wasn't smart or extensive enough with my testing to catch any discrepancies. But either way here is my own personal testing of this code through interactive mode.

```
andrewsushi@debian:~/Coding/uo/415/projects/1$ make clean
rm -f pseudo-shell *.o
andrewsushi@debian:~/Coding/uo/415/projects/1$ make all
gcc -c command.c
gcc -c main.c
gcc -c string_parser.c
gcc -o pseudo-shell command.o main.o string_parser.o
andrewsushi@debian:~/Coding/uo/415/projects/1$ valgrind ./pseudo-shell
==19729== Memcheck, a memory error detector
==19729== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==19729== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==19729== Command: ./pseudo-shell
==19729==
>>> ls
command.c .. pseudo-shell main.o test_script.sh README.txt string_parser.o string_parser.h . command.h Makefile main.c command.o string_parser.c
>>> ls test
Error! Unsupported parameters for command: ls
>>> ls ; ls test; ls
command.c .. pseudo-shell main.o test_script.sh README.txt string_parser.o string_parser.h . command.h Makefile main.c command.o string_parser.c
Error! Unsupported parameters for command: ls
command.c .. pseudo-shell main.o test_script.sh README.txt string_parser.o string_parser.h . command.h Makefile main.c command.o string_parser.c
>>> pwd
/home/andrewsushi/Coding/uo/415/projects/1
>>> pwd test
Error! Unsupported parameters for command: pwd
>>> mkdir test
>>> mkdir test
Directory already exists!
>>> cd test
>>> pwd
/home/andrewsushi/Coding/uo/415/projects/1/test
>>> cd ..
>>> ls test
Error! Unsupported parameters for command: ls
>>> cp README.txt test.txt
>>> cat test.txt
# Project 1 for CS422>>> mv test.txt 1.txt
>>> cat 1.txt
# Project 1 for CS422>>> mv 1.txt test
>>> ls
command.c .. pseudo-shell main.o test_script.sh README.txt test string_parser.o string_parser.h . command.h Makefile main.c command.o string_pars
er.c
```

```
>>> ls
command.c .. pseudo-shell main.o test_script.sh README.txt test string_parser.o string_parser.h . command.h Makefile main.c command.o string_pars
er.c
>>> cd test
>>> ls
1.txt .. .
>>> cat 1.txt
# Project 1 for CS422>>> rm 1.txt
>>> cd ..
>>> ls
command.c .. pseudo-shell main.o test_script.sh README.txt test string_parser.o string_parser.h . command.h Makefile main.c command.o string_pars
er.c
>>> cp README.txt test.txt
>>> mv test.txt
Error! Needs both source and destination arguments
>>> mv test.txt test
>>> cd test
>>> ls
.. test.txt .
>>> cd ..
>>> rm test
Error! Unable to delete the file or directory
>>> cd test
>>> ls
.. test.txt .
>>> rm test.txt
>>> ls
.. .
>>> cd ..
>>> rm test
>>> ls
command.c .. pseudo-shell main.o test_script.sh README.txt string_parser.o string_parser.h . command.h Makefile main.c command.o string_parser.c
>>> exit
==19729==
==19729== HEAP SUMMARY:
==19729==     in use at exit: 0 bytes in 0 blocks
==19729==   total heap usage: 350 allocs, 350 frees, 333,667 bytes allocated
==19729==
==19729== All heap blocks were freed -- no leaks are possible
==19729==
==19729== For lists of detected and suppressed errors, rerun with: -s
==19729== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

I also chose not to showcase all the different types of mv and cp parameters since they were already tested in the test_script.sh, but I just wanted to show a little insight of my own personal testing that were also some things that weren't on the test script.

Conclusion

In conclusion I liked this project and learned a lot. One thing specifically is that I didn't know that you can even do system calls in C so that was really cool to learn. I also do think that the test_script isn't extensive enough as there are a few ways that things can get past. But I guess that was what was mentioned in the assignment information pdf saying that there will be more extensive tests given when grading. But more importantly, I learned how these shell functions work and how to implement them on my own. On top of that the extensive use of valgrind also helped me understand methods and ways to further test code for memory leaks.