CIS 415 Operating Systems

Project 1 Report Collection

Submitted to:

Prof. Allen Malony

Author:

Alex Brown

Report

Introduction

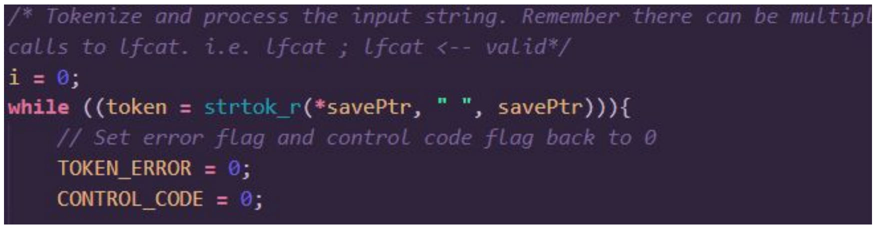
This project is an implementation of a Pseudo Linux Shell with limited functionality. While a typical Linux shell takes in user input from a Command Line Interface (CLI) and can run a number of programs and processes, this implementation (pseudo-shell) only allows the user to perform a limited set of functions. These functions are centered entirely around file system navigation and manipulation. Commands available in the pseudo-shell are exclusive to ls, pwd, mkdir, cd, cp, mv, rm, and cat. The main.c file processes user-input from either the CLI or an input file depending on whether or not the user runs the pseudo-shell in file mode via the command: **./pseudo-shell -f <filename>**, where **<filename>** is the name of a file containing commands which the user desires to be executed. Multiple arguments can be specified for the mkdir, cd, cp, mv, rm, and cat commands, and thus extensive error checking is included in this implementation.

Background

Before starting this project, I was utterly unaware of the usage of system calls in standard C code. My C experience up to this point had only utilized library functions such as moveDir, fopen, etc. At first, the jump from using library functions to system calls wasn't that difficult. For instance, my implementation of mkdir and cd required the use of a single system call to "mkdir" and "chdir", respectively. However, I encountered great difficulty implementing commands like cp and mv which required me to open files via the "open" system call. Unlike its C-library sibling, fopen, "open" required me to utilize flags corresponding to access modes like O_RDONLY, O_WRONLY, and O_RDWR. Though challenging at first, I was able to gain a much better understanding of how to implement I/O functionality using system calls after reading a Geeks for Geeks article on the subject:
https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/

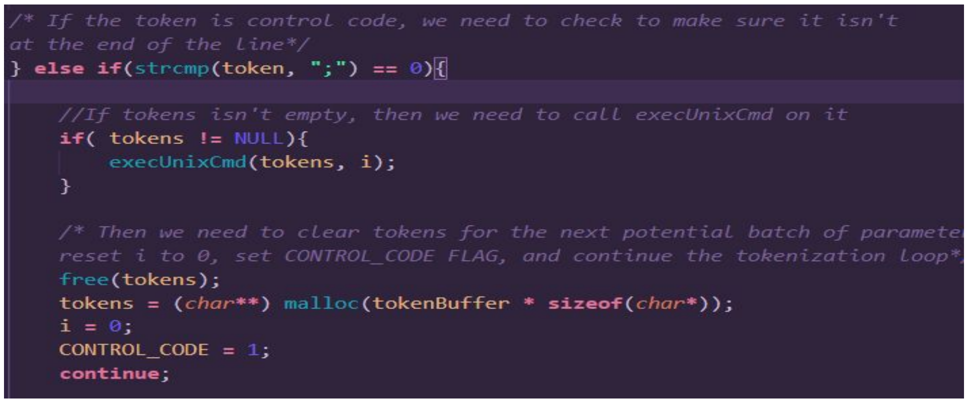
Implementation

If implementing I/O via system calls in command.c was difficult, getting pseudo-shell to function **exactly** as described in the project1 pdf was a nightmare. I achieved desired functionality by implementing a while loop which called strtok_r on each iteration to tokenize the user's input by each space separation. I then defined some flags which would help me to process the user's input exactly how I wanted to:



```
/* Tokenize and process the input string. Remember there can be multipl
calls to lfcat. i.e. lfcat ; lfcat <-- valid*/
i = 0;
while ((token = strtok_r(*savePtr, " ", savePtr))){
    // Set error flag and control code flag back to 0
    TOKEN_ERROR = 0;
    CONTROL_CODE = 0;
```

Next, I checked if the user had entered "exit", and on receiving it exited the current loop and set my CONTINUE flag to 0 to end the program. Immediately following this, I checked if the concerned token was a ";". On receiving this, if there was any input previous to it, I passed the concerned tokens to my execUnixCmd helper function and freed/reallocated my tokens array for any possible commands to follow:



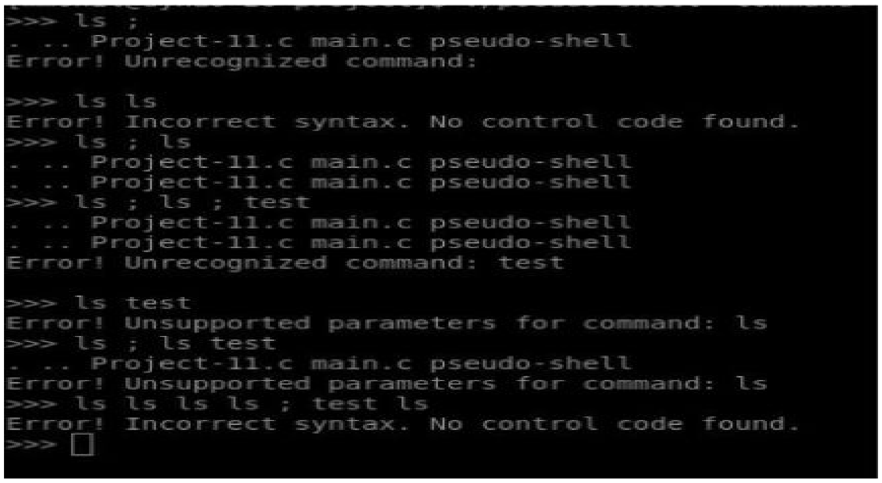
```
/* If the token is control code, we need to check to make sure it isn't
at the end of the line*/
} else if(strcmp(token, ";") == 0){

    //If tokens isn't empty, then we need to call execUnixCmd on it
    if( tokens != NULL){
        execUnixCmd(tokens, i);
    }

    /* Then we need to clear tokens for the next potential batch of paramete
    reset i to 0, set CONTROL_CODE FLAG, and continue the tokenization loop*
    free(tokens);
    tokens = (char**) malloc(tokenBuffer * sizeof(char*));
    i = 0;
    CONTROL_CODE = 1;
    continue;
```

Performance Results and Discussion

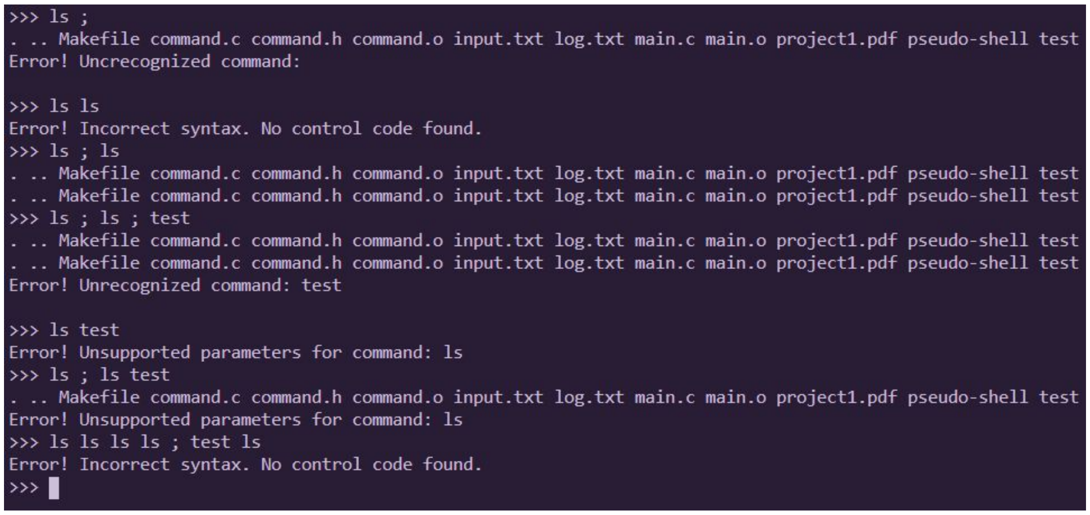
In a last minute push, I was finally able to get my pseudo-shell implementation up to what was specified in the project description. If you look below, you will see the screenshot provided in the project description immediately followed by a screenshot of my implementation performing the exact same commands:



```
>>> ls ;
. .. Project-11.c main.c pseudo-shell
Error! Unrecognized command:

>>> ls ls
Error! Incorrect syntax. No control code found.
>>> ls ; ls
. .. Project-11.c main.c pseudo-shell
. .. Project-11.c main.c pseudo-shell
>>> ls ; ls ; test
. .. Project-11.c main.c pseudo-shell
. .. Project-11.c main.c pseudo-shell
Error! Unrecognized command: test

>>> ls test
Error! Unsupported parameters for command: ls
>>> ls ; ls test
. .. Project-11.c main.c pseudo-shell
Error! Unsupported parameters for command: ls
>>> ls ls ls ls ; test ls
Error! Incorrect syntax. No control code found.
>>> 
```

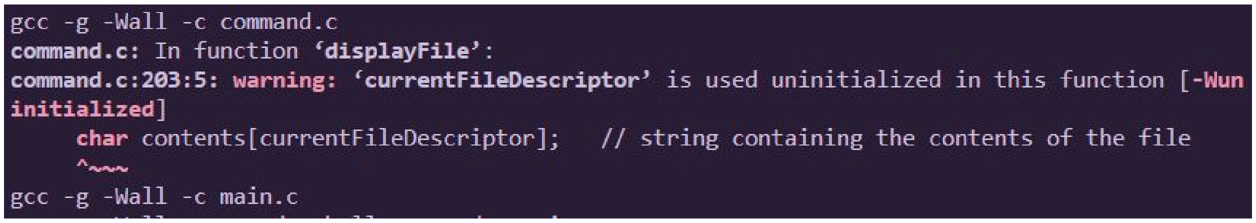


```
>>> ls ;
. .. Makefile command.c command.h command.o input.txt log.txt main.c main.o project1.pdf pseudo-shell test
Error! Uncrecognized command:

>>> ls ls
Error! Incorrect syntax. No control code found.
>>> ls ; ls
. .. Makefile command.c command.h command.o input.txt log.txt main.c main.o project1.pdf pseudo-shell test
. .. Makefile command.c command.h command.o input.txt log.txt main.c main.o project1.pdf pseudo-shell test
>>> ls ; ls ; test
. .. Makefile command.c command.h command.o input.txt log.txt main.c main.o project1.pdf pseudo-shell test
. .. Makefile command.c command.h command.o input.txt log.txt main.c main.o project1.pdf pseudo-shell test
Error! Unrecognized command: test

>>> ls test
Error! Unsupported parameters for command: ls
>>> ls ; ls test
. .. Makefile command.c command.h command.o input.txt log.txt main.c main.o project1.pdf pseudo-shell test
Error! Unsupported parameters for command: ls
>>> ls ls ls ls ; test ls
Error! Incorrect syntax. No control code found.
>>>
```

Additionally, I ran into a strange set of errors in valgrind when executing the cat command. When running valgrind (and during compilation) I am warned of using an uninitialized value in my displayFile function in command.c:



```
gcc -g -Wall -c command.c
command.c: In function ‘displayFile’:
command.c:203:5: warning: ‘currentFileDescriptor’ is used uninitialized in this function [-Wun
initialized]
     char contents[currentFileDescriptor];   // string containing the contents of the file
     ^~~~
gcc -g -Wall -c main.c
```

However, for the life of me, I was unable to implement the function in any other way without getting unexpected/incorrect results. The project specification stated that no **memory leaks** were allowed, so for the sake of time I let it be.

I also should mention that I worked closely with Stephanie Schofield toward the end of this project. We worked a bit together to make her initial do-while loop asking for user input, talked extensively on how best to tokenize and process input, and utilized similar helper functions.

Conclusion

If I learned one thing during this project is was to take extra care to program in small, incremental, pushes. I have a tendency to complete a problem and jump immediately to the next without making a git commit. Since I started this project early and made specific pushes toward particular problems, I was able to break the project into manageable pieces. The result was a well-flushed out project which I had a strong understanding of and enjoyed making.