

Assignment 3 – Simple Shell

Description:

This assignment is to write a simple shell for Linux in C. This shell should run on top of the regular command-line interpreter for Linux. It reads lines of user input, parses it and executes commands by forking new processes. Once forked, the parent process waits for the child process to complete and “die.” In the meantime, the child process executes the command via `execvp()`. Once the child dies, the parent stops waiting and prints the child process id and its exit status.

Approach / What I Did:

I began with the simplest part of the program: providing a prompt to the user and taking in input from the user via `fgets()`. Once completed, I moved onto printing out (to the console) the input buffer to ensure that the input was being captured properly. The next step involved tokenizing the input into an argument array and then once again printing the array to ensure the tokenization was occurring as expected. At this point, I implemented the methods for exiting the command shell by first checking the first argument in the argument vector against the “exit” command and then doing a check of `fgets()` against NULL for EOL as specified in the man page for `fgets()`. Along with this implementation, I included the check of the first element of the argument vector against NULL to determine if the line was empty to reprompt the user for input. Finally, I wrote the portion of code with regards to the `fork()` and `execvp()`. This was probably the easiest portion of the code to implement as I just needed to check whether the current process was the parent or child (as specified in lecture) and provide the proper error if the `fork()` failed or the `execvp()` failed. Within the child process, I used `execvp()` to run the program specified by the user (with all its arguments). In the parent process, `wait()` was invoked and once the child process completed, the child status is updated and the child process and child exit status is printed.

Issues and Resolutions:

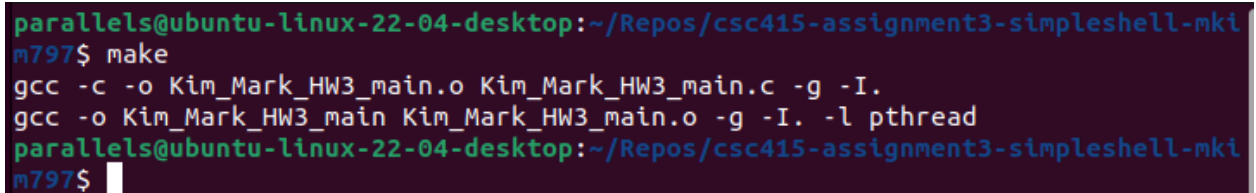
My first issue came from just understanding the process flow and how to handle the three fork outcomes (an error, the parent process, and the child process). After attending office hours, Prof Bierman clarified the steps involved which allowed me to continue with the logic. The code nearly wrote itself at this point.

Most of my time was spent dealing input beyond the specified buffer size. After getting input via `fgets()`, I found that when I entered input beyond the buffer size, the rest of the input was being retrieved and execution was being attempted. Once again, after attending office hours, the cause was identified and a solution was made clear. In this case, `fgetc()` is used to iterate through the rest of the buffer to retrieve and discard the contents of the buffer until it reached the end of the line (`'\n'`).

Related to this, was an issue that required the user to press enter twice to execute a command that was shorter than the buffer. This was resolved by putting in a conditional statement to only iterate through the remaining buffer only if input exceeded it.

Analysis: (If required for the assignment). NOT REQUIRED

Screen shot of compilation:

A terminal window with a dark purple background and light blue/green text. The prompt is 'parallels@ubuntu-linux-22-04-desktop:~/Repos/csc415-assignment3-simpleshell-mkim797\$'. The user enters 'make'. The output shows two gcc commands: 'gcc -c -o Kim_Mark_HW3_main.o Kim_Mark_HW3_main.c -g -I.' and 'gcc -o Kim_Mark_HW3_main Kim_Mark_HW3_main.o -g -I. -l pthread'. The prompt returns to 'parallels@ubuntu-linux-22-04-desktop:~/Repos/csc415-assignment3-simpleshell-mkim797\$' with a cursor.

```
parallels@ubuntu-linux-22-04-desktop:~/Repos/csc415-assignment3-simpleshell-mkim797$ make
gcc -c -o Kim_Mark_HW3_main.o Kim_Mark_HW3_main.c -g -I.
gcc -o Kim_Mark_HW3_main Kim_Mark_HW3_main.o -g -I. -l pthread
parallels@ubuntu-linux-22-04-desktop:~/Repos/csc415-assignment3-simpleshell-mkim797$
```

SCREEN SHOT OF EXECUTION ON NEXT PAGE

Screen shot(s) of the execution of the program:

```
parallels@ubuntu-linux-22-04-desktop:~/Repos/csc415-assignment3-simpleshell-mki
m797$ ./Kim_Mark_HW3_main "prompt$ "
prompt$ ls -l
total 60
-rwxrwxr-x 1 parallels parallels 17056 Sep 22 22:09 Kim_Mark_HW3_main
-rw-rw-r-- 1 parallels parallels 4444 Sep 22 22:09 Kim_Mark_HW3_main.c
-rw-rw-r-- 1 parallels parallels 10128 Sep 22 22:09 Kim_Mark_HW3_main.o
-rw-rw-r-- 1 parallels parallels 1861 Sep 19 19:03 Makefile
-rw-rw-r-- 1 parallels parallels 5058 Sep 19 19:02 README.md
-rw-rw-r-- 1 parallels parallels 2293 Sep 19 19:03 Requirements.md
-rw-rw-r-- 1 parallels parallels 42 Sep 19 19:02 commands.txt
Child 76350, exited with 0
prompt$ ls foo
ls: cannot access 'foo': No such file or directory
Child 76363, exited with 2
prompt$ echo "hello world"
hello world
Child 76397, exited with 0
prompt$ exit

Thank you for using my command shell.

parallels@ubuntu-linux-22-04-desktop:~/Repos/csc415-assignment3-simpleshell-mki
m797$ ls -l
total 60
-rwxrwxr-x 1 parallels parallels 17056 Sep 22 22:09 Kim_Mark_HW3_main
-rw-rw-r-- 1 parallels parallels 4444 Sep 22 22:09 Kim_Mark_HW3_main.c
-rw-rw-r-- 1 parallels parallels 10128 Sep 22 22:09 Kim_Mark_HW3_main.o
-rw-rw-r-- 1 parallels parallels 1861 Sep 19 19:03 Makefile
-rw-rw-r-- 1 parallels parallels 5058 Sep 19 19:02 README.md
-rw-rw-r-- 1 parallels parallels 2293 Sep 19 19:03 Requirements.md
-rw-rw-r-- 1 parallels parallels 42 Sep 19 19:02 commands.txt
parallels@ubuntu-linux-22-04-desktop:~/Repos/csc415-assignment3-simpleshell-mki
m797$ ls foo
ls: cannot access 'foo': No such file or directory
parallels@ubuntu-linux-22-04-desktop:~/Repos/csc415-assignment3-simpleshell-mki
m797$ echo "hello world"
hello world
parallels@ubuntu-linux-22-04-desktop:~/Repos/csc415-assignment3-simpleshell-mki
m797$ □
```