Lab II

C Refresher Module

December 30, ~~2016~~ 2017

1 Ultra Simple Shell

1.1 Background

A command line interpreter, or **shell**, program is a mechanism with which each interactive user can issue commands to the OS and by which the OS can respond directly to the user. Whenever a user has successfully logged into the computer (*a process will have been assigned to the user when they begin the login procedure*), the OS causes the user process to execute a shell. The OS does not ordinarily have a built-in window interface – instead the OS assumes a simple character-oriented interface in which the user types a string of characters (terminated with the **return** key), and in which the OS responds by typing lines of characters back to the screen. The character-oriented shell assumes a screen display with a fixed number of lines (usually 25) and a fixed number of characters (usually 80) per line.

1.2 Design

The shell relies on an important convention to accomplish its task: The command for the command line is usually the name of a file that contains an executable program. For example, ***ls*** and ***gcc*** are the names of files (stored in ***'/bin'*** on most Unix-style machines). In a few cases, the command is not a file name, but is actually a command that is implemented within the shell **builtin**; for example **cd** ("change directory") is usually implemented within the shell rather than in a file.
Since the vast majority of the commands are implemented in files, think of the command as actually being the file name in some directory on the machine. This means that the shell's job is to find the file, to prepare the list of parameters for the command, then to cause the command to be executed using the parameters.

There is a long line of shell programs used with Unix variants, including the original Bourne shell (**sh**), the C shell (**csh**) with its additional features over sh, the Korn shell (**ksh**), and so on, to the standard Linux shell (bash – meaning **B**ourne-**A**gain **sh**ell.) All these shells have followed a similar set of rules for command line syntax, though each has its own special features. The **cmd.exe**

shell for WindowsNT uses its own similar, but distinct, command language.
A shell could use many different strategies to execute a user's computation. ***The basic approach used in modern shells is to create a new process (or thread) to execute any new computation.***

For example, if a user decides to compile a program, the process interacting with the user creates a new child process. The first process then directs the new child process to execute the compiler program.

This idea of creating a new process to execute a computation may seem like overkill, but it has a very important characteristic. When the original process decides to execute a new computation, it protects itself from any fatal errors that might arise during that execution. If it did not use a child process to execute the command, a chain of fatal errors could cause the initial process to fail, thus crashing[†] the entire machine.

Your task is to:

- Build an ultra simple shell (*ufs*) using the ***fork/exec/wait*** system calls.
- It should take commands from *stdin* and the output should be on *stdout*, in an *infinite-while* loop.
- Prefrably it should have an *exit* string, which breaks out of the while loop and exits.

Submission[‡]: Upload on backpack:
A zipped file inculding:

- The **source code**[§] of the shell.
- The **makefile** for compilation.
- A small **write-up** of the approach used.

2 Example

```
mfrw@kp:-$ ./ush
ush> ls
makefile ush.c ush
ush> uptime
22:19:47 up 17:59,  2 users,  load average: 0.36, 0.23, 0.19
ush> pwd
/home/mfrw
ush> exit
mfrw@kp:-$
```

[†]If the shell was the only process.
[‡]We will use a plagiarism detector.
[§]Please ***'indent -linux myshell.c'*** before submitting.

3 Hints

```
int main() {
    /* main loop */
    while (true) {
        /* print prompt string */
        ...
        /* read command from stdin */
        ...
        /* parse command -- exit if quit */
        ...
        if (fork() == 0) {
            /* this is the child */
            ...
            /* use exec to run the command */
            ...
            /* exit child */
        } else {
            /* this is the parent */
            ...
            wait(status)
        }
    }
}
```

4 For those who want more

Any of these or all would make you eligible for a bonus:

- Try to include ***redirection & piping***. (<,>,>>,|)
- Try to include ***job-control***[†].

5 I still want more

Pull this repo and complete it[‡]. It is from CSAPP[§].
https://github.com/mfrw/shlab.git

[†]CTRL-Z, fg, bg, jobs, &.
[‡]Feel free to ask for help.
[§]Computer Systems A Programmer's Prespective