Joanna Klukowska
joannakl@cs.nyu.edu

Project 4: My Own Shell.

Due date: May 1, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but **all work for all assignments must be entirely your own**. Any sharing or copying of assignments will be considered cheating.

Project Description

Your fellow CS201 student attempted to implement her own shell program. It works, but the users discovered several bugs. Your task for this project is to fix the bugs that were reported. The code is not written according to all of the standards that we discussed (well, not all of the students follow instructions). You are free to modify the existing code to eliminate (some) use of globals, but you are not required to do so.

List of Known Bugs

Bug 1: Only the first command works properly

```
$ ./proj4
sh201>  ls
codebloacks_proj4  procline.o  proj4_draft.zip  runcommand.o  test_run.c       userin.c
Makefile           proj4       proj4.h          shell.txt     test_run_loop    userin.o
procline.c         proj4.c     runcommand.c     test_run      test_run_loop.c
sh201>  ls
ls: cannot access ls: No such file or directory
sh201>  ls
ls: cannot access ls: No such file or directory
ls: cannot access ls: No such file or directory
sh201>  who
who: extra operand   w h o
Try 'who --help' for more information.
sh201>
$
```

After we start the shell (running `proj4` the output from the first command is correct, but all later attempts at running anything fail.

HINT: this is a very simple fix, but you need to understand the code for the shell, before you can fix it.

Fixing this bug, will make it much easier to observe the behavior of the other bugs and test if the fixes to them are correct.

Bug 2: `cd` command does not behave as expected

The `cd` command is not supported at all.

```
$ ./proj4
sh201>  cd /home
cd: No such file or directory
sh201>
$
```

The reason for that is that passing `cd` as the name of the program to `execvp` system call fails (this command requires change of the environment and hence it is handled in a different fahion than most other commands that you encountered this semester). The program should, instead, use `chdir` system call to change the current working directory. (Use the man pages to find out how to use `chdir`.)

You need to fix this problem by implementing the code that will allow the user to use the `cd` command in the following three ways:

Joanna Klukowska
joannakl@cs.nyu.edu

`cd` - changes the current working directory to the user's home directory (you will need to use `getenv()` function to obtain the name of the user's home directory

`cd PATH_NAME` - changes the current working directory to the one specified by the `PATH_NAME` (if such directory does not exist, the shell should print an error message and return to the prompt)

`cd ..` - changes the current working directory to the one that is the parent of the current working directory

Bug 3: `exit` command does not terminate the shell

This is really a feature request. The `exit` command is not implemented. We would like to be able to type `exit` with no parameters to terminate the shell program. Currently the only way to terminate the shell is by typing the end of like (EOF) character on the keyboard (this is done by pressing Ctrl+D).

```
$ ./proj4
sh201>  exit
exit: No such file or directory
sh201>
$
```

Bug 4: Pressing Ctrl+C should not kill the shell

It is customary that the shell itself should be immune to `SIGINT` signal (Ctrl+C from the keyboard). Pressing the key combination should terminate a process running in the foreground mode, but not the shell itself. Currently the shell terminates when the user presses Ctrl+C.

```
$ ./proj4
sh201>  ls
codebloacks_proj4  procline.c  proj4    proj4_draft.zip  runcommand.c  shell.txt
test_run.c     test_run_loop.c  userin.o    Makefile           procline.o
proj4.c  proj4.h          runcommand.o  test_run   test_run_loop  userin.c
sh201>  ^C
$
```

Bug 5: Background processes are not reaped

The shell supports running processes in the background, but it does not reap such processes properly. All background processes should be reaped (as opposed to becoming zombie processes).

```
$ ./proj4
sh201>  ./test_run &
[Process id 3865]
sh201>  program name: ./test_run

sh201>  ps
  PID TTY          TIME CMD
 3249 pts/13   00:00:00 bash
 3862 pts/13   00:00:00 proj4
 3865 pts/13   00:00:00 test_run <defunct>
 3870 pts/13   00:00:00 ps
sh201>
$
```

Working on this Project

Your repository contains the following files

- `proj4.c` - the actual program that starts the shell

Joanna Klukowska
joannakl@cs.nyu.edu

- `proj4.h` - the header file containing function declarations, definitions of constant, define statements, etc. (if you need to define additional structures, functions or constants you should add them to this file)
- `procline.c`, `runcommand.c`, `userin.c` - functions that implement the shell;
- `test_run.c`, `test_run_loop.c`, `test_run_sleep.c`, `test_run_pause.c` - the simple self contained programs (use them to test if you can run a program in a foreground / background, if a process can be killed by Ctrl+C, if the process is reaped after termination, etc; feel free to add some of your own program like these ones)
- `Makefile` - the makefile for this project

Compiling and Building the Code

You can compile and build the code by running

```
$ make clean
$ make
```

(`make clean` is optional, but this makes sure that you are recompiling all the files that are required for the project).

This produces `proj4` program that runs the shell.

The bugs are fairly independent. We think they are listed in the order of difficulty, but you may decide otherwise. Fixing bug 1 should make the behavior of the shell more useful for testing of the behavior of the other bugs and bug fixes.

How to fix a bug?

For each bug, you need to add/modify the code that causes the problem. You should clearly indicate in inline comments which bug is being fixed by the change.

For each bug, your `proj4.c` file should contain a paragraph with the brief description of how you fixed the bug and where the fix is (name of the file and line numbers, or multiple such locations if the fix is distributed).

If you add any functions, you should document those function. You are not obligated to document the code that is given.

How will we grade this?

Each bug fix is worth 20 points.

Questions

Post any questions you have regarding this assignment to Piazza under the ”homeworks” topic.

Committing and Pushing Your Work

You should commit your code regularly to the repository on GitHub (this is your backup and a proof to us that you are working on it). At the least, you should commit your work after every single problem. You are **required** to make at least 5 commits (including the push to the remote repository) before the due date for the project.

Please, run `make clean` before committing the changes so that the object files are not uploaded together with your sources.