CS 392: Homework Assignment 5
Due: April 9, 11:55pm

Philippos Mordohai
Department of Computer Science
Stevens Institute of Technology
Philippos.Mordohai@stevens.edu

Collaboration Policy. Homeworks will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems. Use of the Internet is allowed, but should not include searching for previous solutions or answers to the specific questions of the assignment.

Late Policy. No late submissions will be allowed without consent from the instructor. If urgent or unusual circumstances prevent you from submitting a homework assignment in time, please e-mail me explaining the situation.

Objective

The goal of this assignment is to implement a mini shell with a few built-in commands. This assignment requires fork(), the exec() family of functions and signal handling.

Functionality 1. Shell-Within-Shell (100 points)

The goal is to create a simple shell that prints a prompt (see below) and accepts a string from the keyboard. Your program should process the string and decide if it is (i) a built-in command, (ii) a process that should be launched in the foreground, or (iii) a process that should be launched in the background. You do not need to check the correctness of the inputs to your shell. The user's commands should be launched and the OS will handle any errors.

Your program should be called `msh` and should take no arguments. When your program runs, the commands you provide may look like the following:

```
msh:student:/home/student/shared$ ls ..
msh:student:/home/student/shared$ sudo apt-get update &
msh:student:/home/student/shared$ wget http://norvig.com/big.txt
```

Requirements

(1) The prompt should have the form `msh:username:directory$` (note the space at the end). `msh` is a fixed identifier to distinguish your shell from the bash shell. You should retrieve the username from the password file using the user's uid. (You can use `getuid()` and then `getpwuid()` for this. See the 7^{th} set of notes.) Also, see the 7^{th} set of notes for how to retrieve the current directory. The prompt should look like:

```
msh:student:/home/student/shared$
```

(2) Use `strtok()` to tokenize the input string. Assume that commands, options and arguments are separated by spaces.

(3) Your program should recognize two built-in commands: `cd` and `exit`. They should be executed without forking a new process. See the 7^{th} set of notes for how to change the current directory.

(4) `exit` takes no arguments and cannot be followed by `&`. An appropriate error should be generated if this is not the case.

(5) If `cd` is called with no arguments or $\sim$ as the argument, it should change the working directory to the user's home directory. You will need to get the user's password entry for this. See `getpwuid()` above. (This feature is worth 5 points.)

(6) Note that $\sim$ may only appear as part of `cd` $\sim$. Complex paths including $\sim$ must be given in multiple commands. This applies to all commands, not just `cd`.

(7) If the user adds a `&` at the end of an input string for a non-built-in command, the process should be executed in the background. There can only be up to one `&` in a line and must be the last character. Make sure that there are no zombies. Before a process starts in the background, print the process ID number and the string, as in the following example:

```
pid: 1234 cmd: ls ..
```

After the child process is reaped, print its pid again, in a message like:

```
pid 1234 done
```

(8) If the user command is to be executed in the foreground, a child process should be forked and the parent process should wait for it. No printouts like the above should appear.

(9) Install signal handlers for `SIGINT` and `SIGCHLD` using `signal()`. `SIGINT` should be disabled in the parent process. A message specifying that the shell cannot be terminated like this should be printed when `SIGINT` is received, and the shell should

not be terminated. `SIGINT` handling is restored after `exec()` is called, so you only need to worry about the parent process. `SIGCHLD` should be handled as shown at the end of the 9^{th} part of the notes. Note that `printf()` will be considered sufficiently safe for this assignment. You can/should ignore `if (errno != ECHILD)` in the provided code.

Hints

(1) You should include `unistd.h`, `singal.h` and `sys/signal.h`.

(2) If you are having trouble retrieving the username, printing uid instead will earn half of the relevant points.

(3) Remember to remove the `&` from the arguments passed to the command that will be launched.

(4) Use `waitpid()` to wait for children running in the foreground.

(5) Use `waitpid()` with `WNOHANG` to reap child processes running in the background.

(6) Start with either the foreground or background option and make sure that it works before moving on.

(7) When developing your code, keep in mind that you can type exit, but not CTRL-C to exit the shell.

Error messages that should be considered include the following. Error messages should printed to `stderr`. The last %s in each line below is for `strerror(errno)`.

(1) "Error: Cannot get passwd entry. %s.\n"

(2) "Error: Cannot change directory to %s. %s.\n"

(3) "Error: Too many arguments to cd.\n"

(4) "Error: Failed to read from stdin. %s.\n"

(5) "Error: fork() failed. %s.\n"

(6) "Error: exec() failed. %s.\n"

(7) "Error: wait() failed. %s.\n"

(8) "Error: Cannot get current working directory. %s.\n"

(9) "Error: Cannot register signal handler. %s.\n"

Deliverables

In a zip file, submit the following:

(1) Source files, including at least one header file.

(2) A makefile. (You may choose to factor out functions into separate .c/.h files. Any structure is acceptable as long as your makefile can build the project.)

(3) In a pdf file, please answer the following questions:

   (a) Why are `exit` and `cd` handled as built-in commands and no child processes are forked? What would happen if you forked children for these commands?

   (b) Why should you use `killpg(getpid(), SIGTERM);` before `main()` exits?