COMP 3005 & GA/09 Operating Systems
Coursework 1: Unix Command Shell

Please read these instructions fully and carefully.

The aim of this coursework is to write a simplified Unix command interpreter in C, similar to *bash* or *csh* in purpose, but trimmed down to the bare minimum.

1. Upon start-up, the shell should get the current working directory, read the file `profile` from that directory and configure the environment variables `HOME` and `PATH` as specified in that file. For example,

   ```
   PATH=/bin:/usr/bin:/usr/local/bin
   HOME=/home/os
   ```

   sets the home directory to be `/home/os` and sets the search path to comprise the sequence `/bin`, `/usr/bin` and `/usr/local/bin` of directories. You should report an error if `profile` does not exist or if either variable is not assigned. It is permissible for the variables to be assigned in either order and for their values to be replaced by subsequent assignments. You are **not** required to implement variable expansion.

2. As a command prompt to the user, the shell should display the full path of the current working directory followed by `>`. When the user types a command and hits enter, the shell should read the input. The first word on the line should be interpreted as the name of the program to run and the rest of the line should be interpreted as the arguments to give to the program. For example, if the user types `ls –l /tmp` , then the shell should run the program `ls` and give it the arguments `-l` and `/tmp` . The output should be the same as if the user had typed this into a regular Unix shell. To achieve this, your shell (note – *your* shell, not another system) should search through the directories listed in `PATH` until it finds the named program. It should then fork and execute the program. Note that, by convention, the name of the program should also be supplied to `exec()`as the first element in the list of arguments. You may also use a variant of `exec()`if needed.

3. When the program completes, the user should be presented with the prompt again, so he or she can enter another command.

4. For full marks, you must also build into your shell the capability to handle assignment to `HOME` and `PATH` from the command line (e.g., `$HOME=/home/os2`), and to act upon the `cd` command by changing the working directory as specified (or to the `HOME` directory by default).

You are expected to submit the C source files and a profile file, and you must produce a makefile to go with these. **Your code will be compiled by typing 'make'; if this doesn't work, your code is deemed not to compile.**

The following functions might be useful to you:

- `fopen()` to open a file and `fgets()` to read a line from it.
- `opendir()` to open a directory and `readdir()` to read an entry from it. You can use these to figure out which directory the executable program is in. Alternatively you might use `stat()` to test each directory in turn to see which one contains the program.
- `strcmp()` for comparing strings and `strtok()` for finding tokens in a string.
- `fork()` to fork a new process.
- `execv()` to replace your shell's child process with the program to run.
- `wait()` or `waitpid()` for the shell to wait until the child process has completed.
- `getcwd()` returns the current working directory and `chdir()`changes it.

Use the Linux `man` command to find out about these functions. Most of them are in Section 2 of the manual: eg. `man 2 wait` gives you the C man page (without the 2, you get the man page for the bash wait command).

Marking

Your programs will be both run and read. As a consequence, you will be awarded marks for two things: (i) the ability of your program to compile and execute the commands it should and (ii) the structure and generality of the code, and the quality of documentation provided.

Your code must compile and run on frontal.cs.ucl.ac.uk. It will be tested on this machine, and if it fails to compile or run, then you will not be awarded marks for any of the first part. The code will be tested by running the same sequence of commands for everyone and you will be marked on the outcome of that test. You will not be told what this sequence is (other than that standard unix syntax will be used), so you are strongly advised to test your program, and even to get others to attempt to find its faults before you hand it in.

Your code will also be read. It should be professional: neatly laid out, well commented and come with explanations of what each function is trying to achieve and how. You are recommended to use doxygen formatting or something similar. Even if it doesn't run, it may still gain some marks for attempting to solve the problem in a reasonable way. Your code should generalise: in other words *to the extent possible* you should not build in functions, you should do things in such a way that the full set of shell commands can be used.

The coursework is worth 5% of the marks for the course, but also will help exercise your C programming skills. Not submitting a reasonable attempt at the coursework will render you incomplete for the course, rather than just losing 5%. While you can discuss general ideas with your friends, your code must be your own. Electronic submission makes it simple to run automated plagiarism detection software.

Your program must be submitted electronically on the Moodle page. To do this, you should zip together a directory named `firstName_lastName` containing the source, makefile, profile file and anything else you want to submit. You should include a 'README' file that describes each file included in that directory.

You can resubmit your coursework more than once if you need to, up until the deadline – only the last version will be marked.