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

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, the shell 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).

Your code must be capable of running on the Linux machines in the lab.

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).

Marks

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