CS2106 Operating Systems
Assignment 1
Processes

In CS2106 the assignments are designed not just for you to try out API calls. The assignments are designed to motivate you to do your own research to solve the problems. As such the assignments are very brief, only providing you with the requirements, and some small hints.

You should complete these assignments on your own laptops if you have some flavor of UNIX installed (e.g. Linux), or on Sunfire if you don't.

Do not that OSX, while being a direct descendent of UNIX, is ***NOT*** POSIX compliant. While all the POSIX API calls are provided, many are either not implemented at all, or work in unexpected and often buggy ways. This is because OSX has its own structures (e.g. Grand Central Dispatch or GCD) that integrate with its Cocoa framework, and therefore prefer programmers to use those.

Bottom line is that if you do these assignments on OSX, they will compile, but won't work.

1. **Introduction**

   Since it is the Lunar New Year period, this first assignment will start off slow and easy. Your task in this first assignment is to write a simple shell that takes a command from the user and executes it.

   Through this assignment you will learn the various versions of exec, and how to use them to launch new processes.

   Instructions

   Part 1.

   Create a file called "shell.c" and write the following program:

   a. Run in an infinite loop, printing a ">" prompt and reading in user input.
   b. After the user enters a command (consisting of a program name and possible arguments):
      i. Create a new process, printing out the new process's pid.
      ii. In the child, print the pid of the parent, and load and execute the command entered by the user.
      iii. Your shell should be able to execute commands entered by the user as long as the command can be found in the system's PATH environment variable.

iv. Print an error message if the command cannot be found or some other problem occurs.

Again a command consists of a program name and arguments (e.g. "ls –l"). You may assume that none of the commands are built into your shell.

The screen captures below illustrate what you need to achieve:

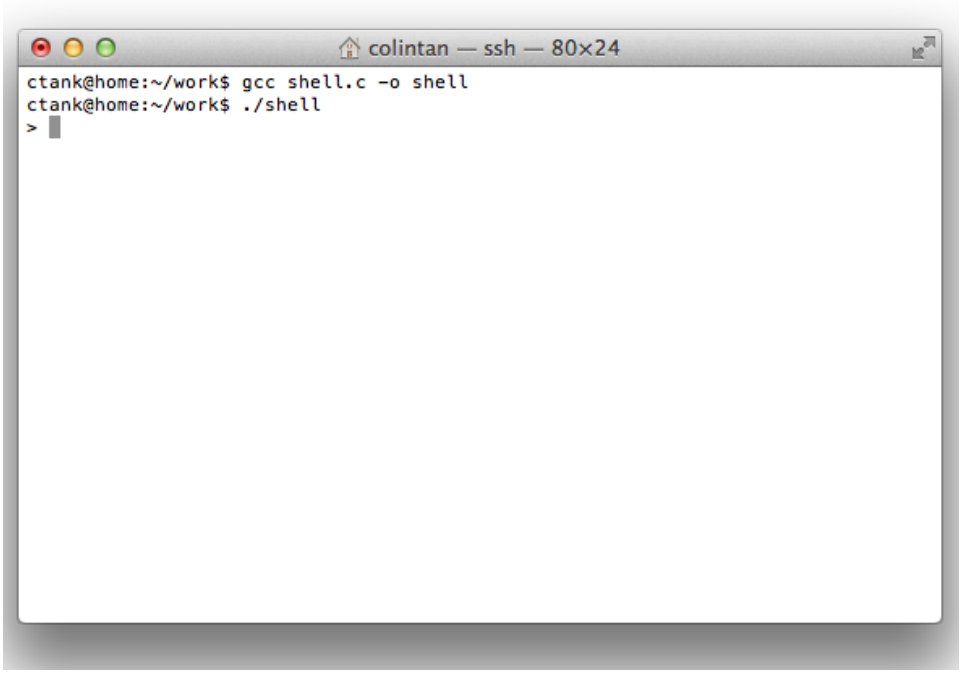

Figure 1. Starting shell

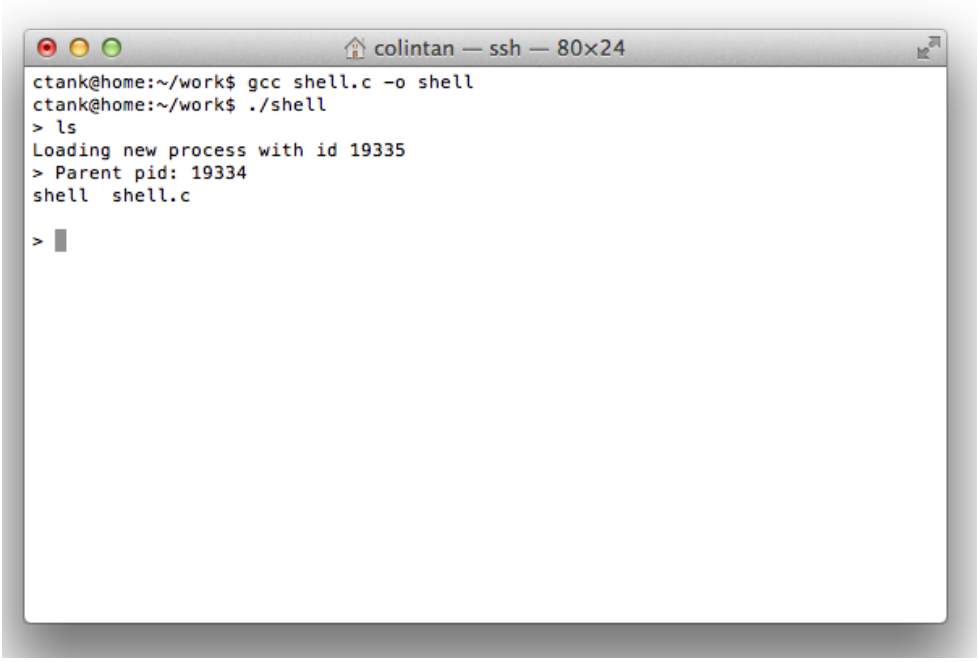

Figure 2. Executing the "ls" command in the shell. Note how the new process id and parent pid are printed.

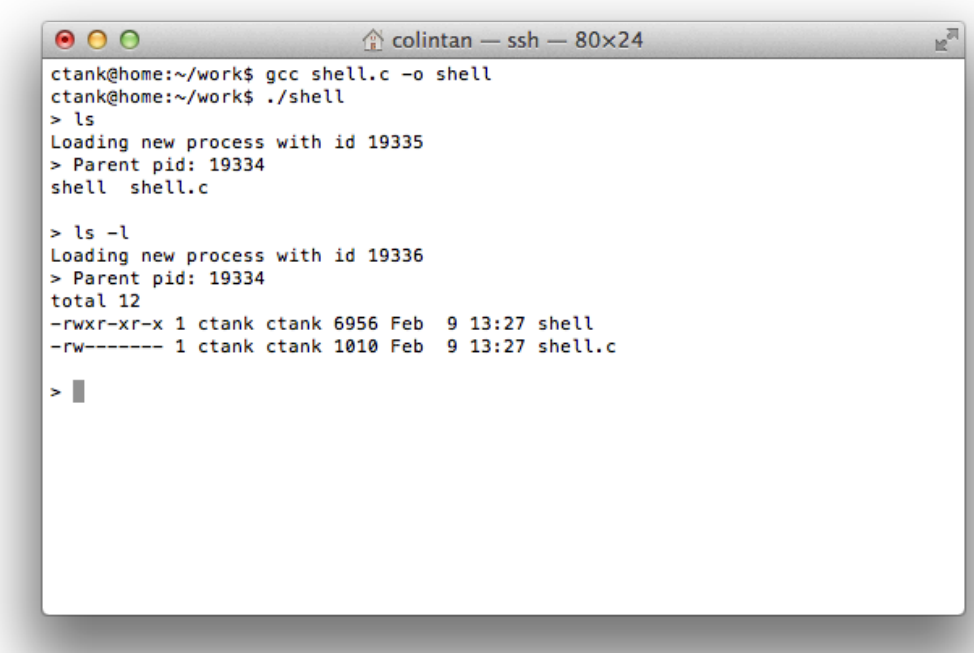

Figure 3. Executing the "ls -l" command in the shell. Note how the new process id and parent pid are printed.

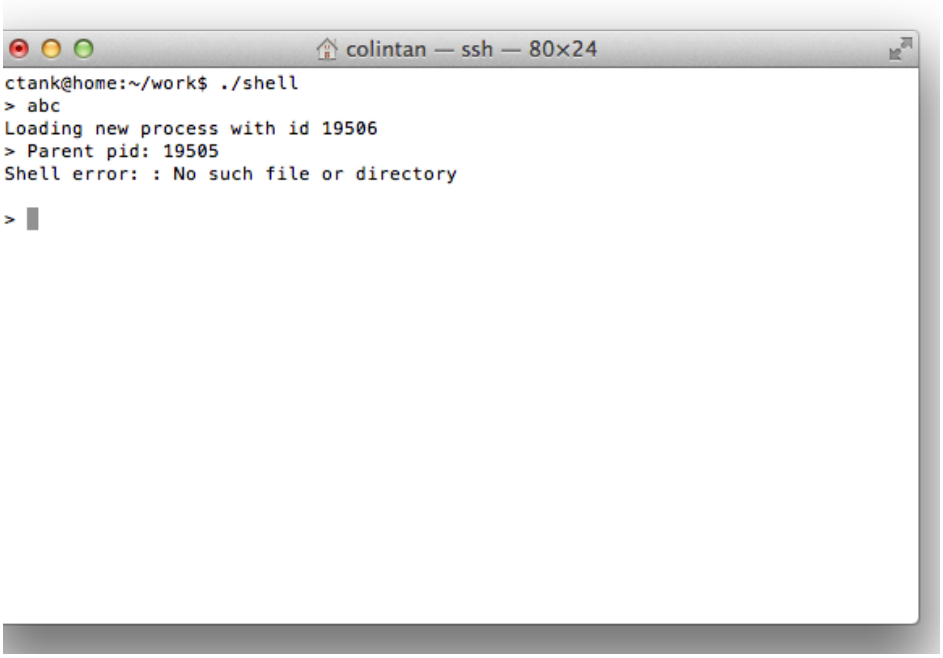

Figure 4. Sample Error Message

Some hints:

i) If you are using fgets to read user input, be sure to replace the '\n' character with '\0'.

ii) This is a good reference for the exec family of commands: http://linux.die.net/man/3/execl . Choosing the correct version of exec will make this assignment trivial.

iii) You might find the C "strtok" function very useful for extracting arguments from the user's command. See http://www.cplusplus.com/reference/cstring/strtok/ for how to use it.

iv) To compile your shell program, use `gcc shell -o shell`, and to execute, run `./shell`

v) Check out perror: http://man7.org/linux/man-pages/man3/perror.3.html

vi) You will need to include stdio.h, string.h, unistd.h and stdlib.h

Part 2.

We will now write a program that prints all the environment variables in when run.

a. Create a new program called prog.c
b. Implement code that will print out ALL of the environment variables.
c. After printing out the environment variables, your program should look for the variable "SHELL_PATH" and print out its value, or "UNKNOWN" if this variable cannot be found.
d. Run this program on the standard UNIX shell, NOT the shell you wrote in Part 1.

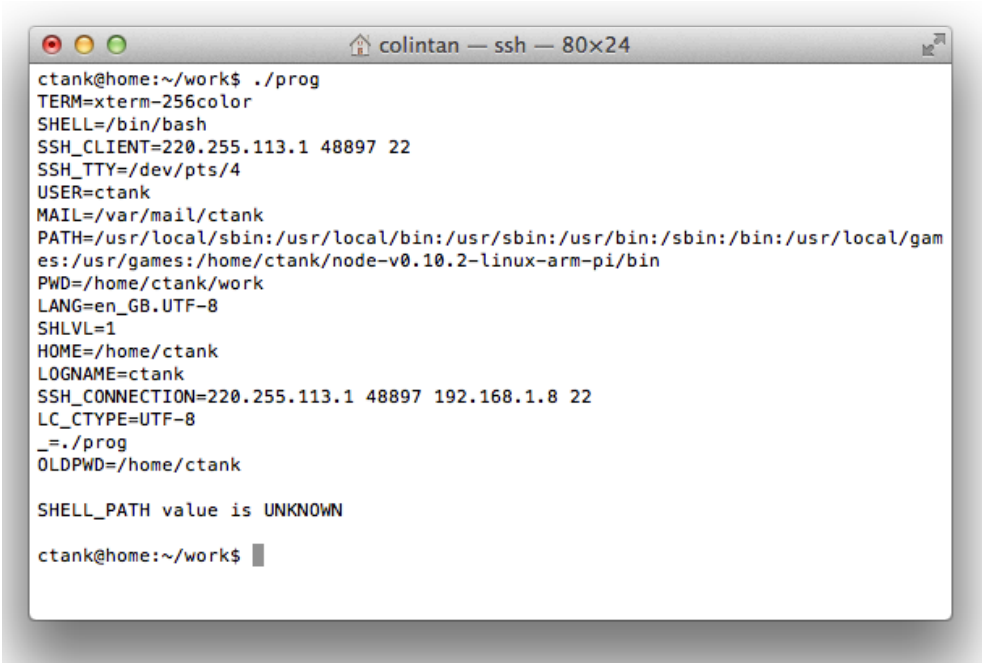

Figure 5: The prog programme, running with SHELL_PATH undefined.

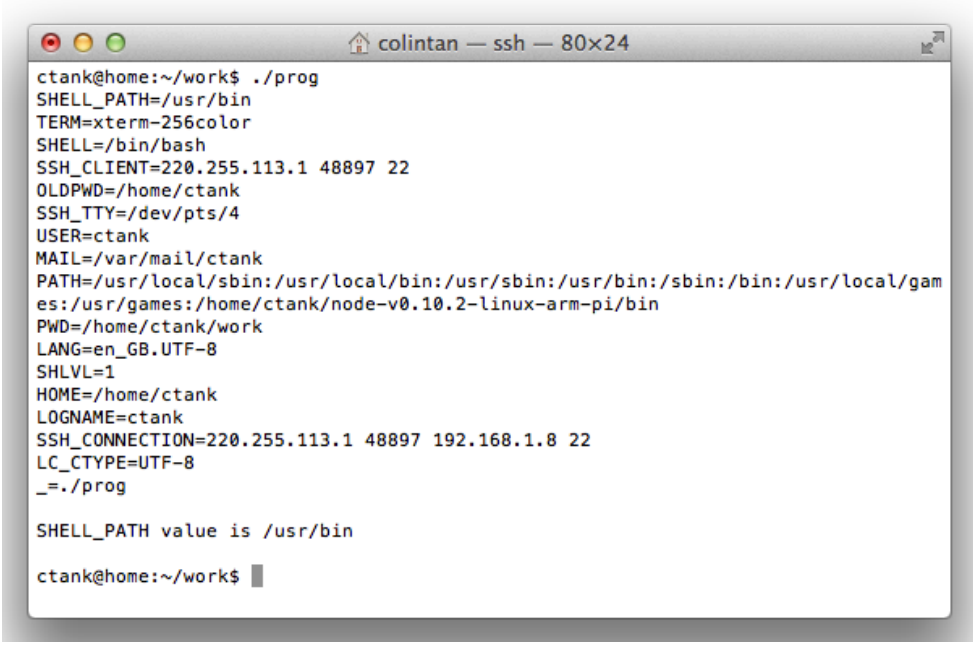

Figure 6. Prog running with SHELL_PATH defined to be /usr/bin

Some hints:

a. This program is very, very easy to write; there's no need to use any POSIX calls like getenv or other similar calls at all. The lecture notes already tell you how to access environment variables from within a C program.
b. You can use the strstr function to test for the SHELL_PATH variable. Read up more about strstr at: http://man7.org/linux/man-pages/man3/strstr.3.html

Part 3.

We will now modify shell.c:

a. Copy shell.c to shell2.c. In UNIX you can do this by typing "cp shell.c shell2.c" without the quotes.
b. Modify shell2.c so that whenever it calls a program, it sets the SHELL_PATH environment variable to its own COMPLETE path, and passes over all of the existing environment variables.
c. Compile using gcc shell2.c –o shell2
d. Execute the "prog" program you wrote in Part 2.

See the following screen shots for what you have to do.

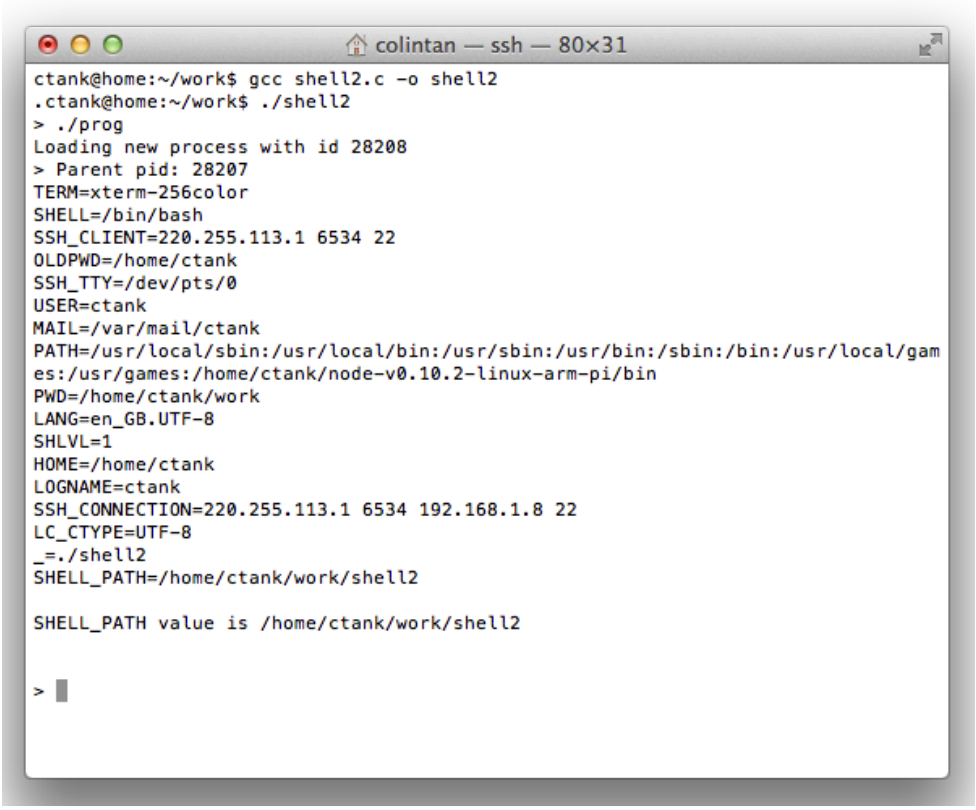

Figure 7. Executing "prog" in shell2

Some hints:

i) Using argv[0] alone only gives you the name of the program. You will need to use get_current_dir_name to get the working directory that shell2 is being run in. See http://man7.org/linux/man-pages/man2/getcwd.2.html

ii) You will need to build a list of environment variables AND add in SHELL_PATH.

iii) You may need to use strtok to extract the name of the shell from argv[0].

SUBMISSION INSTRUCTIONS

You have two weeks to complete this assignment, and it is due on 26 February 2016 at 5 pm.

To submit:

i) Ensure that the name and matric number of BOTH team members is in every program in a comment section **AT THE START OF EVERY PROGRAM**. E.g.

```
/*
   Name: Tan Ah Kow
   Matric: A0103948U

   Name: Lau Ah Beng
   Matric: A0110394B

*/

.. Rest of program
```

ii) Appoint a leader, and ZIP up all files, naming the zip file using the matric number of the leader. E.g. A0103948U.zip

iii) Upload to the submission directory which will be created on IVLE nearer the due date.