# COMP3511 Operating System (Fall 2019)

## Project 1 – A Simplified Shell Program

**Submission due date: 21/10/2019 23:59 via CASS**

## Introduction

This aim of this program is to help students understand **process management** and **inter-process communication**. Upon completion of the project, students should be familiar with a number of related Linux system calls.

## Shell Program Design

A shell program serves as an interface between the user and the underlying operating system. A command prompt allows the user to type in commands. The shell program parses the commands, invokes the corresponding programs, and then displays the output.

In general, the shell program executes in an infinite loop until an exit command is received. The flow of the shell program is illustrated below:

1. Print the shell program and its process id (`pid`)
2. Print a prompt (e.g. `$>`) to let user to type in commands
3. Read a command line
4. Parse a command line
5. Execute a command
6. Repeat step 2 until the `exit` command is received.
7. Print the process id (`pid`) of the shell program before the termination of the shell program

In this assignment, you need to implement a simplified shell with the following features:

1. Support the `exit` command to terminate the shell program
2. Support multi-level pipes

To simplify the requirements, you can assume that the command input and format are valid.

You can find several test cases in the “Sample test cases” section below

## Starting point: Compile and run myshell.c

`myshell_skeleton.c` is provided. Please rename it as `myshell.c` to start your project. You are not required to start from scratch as the base code already provides lots of features (e.g. command line parsing, the loop structure of the shell program)

In Lab 2 and 3, we will go through the base code, necessary programming concepts (e.g. C programming introduction) and briefly introduce how to extend the base code to complete the project requirements.

Please note that C programming language (instead of C++) must be used to complete this assignment. Here is the command to compile and run `myshell.c`

```
$> ls
myshell.c

$> gcc -o myshell myshell.c

$> ./myshell
```

## Feature 1 – `exit` command

The `exit` command MUST be implemented using `exit()` function call defined in `<stdlib.h>`. You should print out the process id of the shell program before its termination. Here is the sample input (highlighted in RED) and output. Please note that the process ID will be different every time you run the program. In this example, the process ID of the shell program is 7230.

```
The shell program (pid=7230) starts
$> exit
The shell program (pid=7230) terminates
```

## Feature 2 – Multi-level Pipes

In C programming in Linux, a process creates a pipe by:

```
int ps[2];
pipe(ps);
```

After the pipe function call, `ps[0]` will be assigned to a file pointer to the read end of the pipe, and `ps[1]` will be assigned to the write end of the pipe.

In a shell program, a pipe symbol (`|`) is used to connect the output of the first command to the input of the second command.

For example,

```
$> ls | sort
```

The `ls` command lists the contents of the current directory. As the output of `ls` is already connected to `sort`, it won't print out the content to the screen. After the output of `ls` has been sorted by `sort`, the sorted list of files appears on the screen.

In this project, you are required to support multiple-level pipes. We assume that there exists at most `16` pipe segments. Here is a pipe command with 4 segments:

```
$> ps aux | grep root | sort | less
```

## Sample Test Cases

*Note: Assume that input.txt exists in your current working directory*

| **Commands** | **Observation** |
|---|---|
| `$> ls` | `ls` displays the filenames of the current working directory |
| `$> cat input.txt \| less` | `less` is a text file viewer in Linux. This program will be used to view the content of `input.txt` |
| `$> ls \| sort` | `ls` displays the filenames of the current working directory, and the filenames are sorted in an ascending order |
| `$> ps aux \| grep root \| sort \| less` | root processes should be displayed in a sorted order using the `less` text file viewer |

## Marking Scheme

1. (30%) Explanation of `process_cmd` in the provided base code. You should use point form to clearly explain how you implement this function
2. (10%) Supports the exit command (e.g. `exit`)
3. (10%) Supports a basic command (e.g. `ls` )
4. (20%) Supports at least 2-level pipe (e.g. `ls | sort`)
5. (30%) Supports multi-level pipe (e.g. `ps aux | grep root | sort | less`)

*Plagiarism: Both parties (i.e. a student providing the codes and a student copying the codes) will receive 0 marks.*

## Submission

You only need to submit **myshell.c** via CASS on /before the due day mentioned above. CS Lab 2 is the development environment. Please use one of the following machines (`csl2wk`**`XX`**`.cse.ust.hk`), where **XX**=01…50.