Computer System Engineering
50.005
Dr. David Yau

Week 1: Lab 1 (25 marks)
Objective: Create a Shell Interface using Java or C program

Contact us
dima_rabadi@mymail.sutd.edu.sg
jie_yang@mymail.sutd.edu.sg
liza_ng@alumni.sutd.edu.sg

The Goal of this lab

- In this lab, we will learn how to write Java or C program to build a user shell interface. The interface accepts user commands and then executes each inserted command in an external process (shell).

What is shell interface?

- The shell is a program that takes your commands from the keyboard and gives them to the operating system to perform.
- It is a Command Line Interface (CLI).
- In the old days, it was the only user interface available on a computer.
- To open the Ubuntu terminal:
  - Press Ctrl +Alt + t
  - Go to Application then type terminal
  - You should see a shell prompt that contains your user name and the name of the machine followed by a dollar sign $

- Demo!

What to do!

- In this lab your code should handle three main requirements:
  1. **Creating an External Process**: modify the main() method of the given program so that an external process is created and executes the command specified by the user.
  2. **Changing Directories**: we encounter the concept of the current working directory (pwd), which is simply the directory you are currently in. The (cd) command allows a user to change current directories. Your shell interface must support this command.
  3. **Adding a History Feature:** that allows users to see the history of commands they have entered and to rerun a command from that history.

What is an external process?

- A process is an executing (i.e., running) instance of a program. Processes are also frequently referred to as tasks.
- To know the current running processes type ps in your terminal!
- How to create process in your code?
  - In Java: ProcessBuilder

    ```
    ProcessBuilder pb = new ProcessBuilder();
    pb.command(commandList);
    Process p = pb.start();
    ```
  - In C: System

    ```
    #include <stdlib.h>
    int system(const char *command);
    ```

ProcessBuilder

- The ProcessBuilder class is used to create operating system processes.
- ProcessBuilder returns after the command has been completed
- Example:

```
try {
        commandList = commandLine.split(" ");
        ProcessBuilder pb = new ProcessBuilder();
        pb.command(commandList);
        Process p = pb.start();
        BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream()));
        String cOutput;
        while((cOutput = br.readLine())!= null) {
                System.out.println(cOutput);
        }
        br.close();
} catch(Exception e) {
        // e.printStackTrace();     // anothor choice
        System.out.println(e.getMessage());
}
```

System

- The system() library function uses fork to create a child process that executes the shell command specified in command using exec
- system() returns after the command has been completed
- Example:

```
char command[100];//to store users command
//while loop to keep asking user for more inputs
while(1){

        printf("csh>");
        fgets(command,MAX_INPUT,stdin);//take input from user

        system(command);
        system("ls");
```

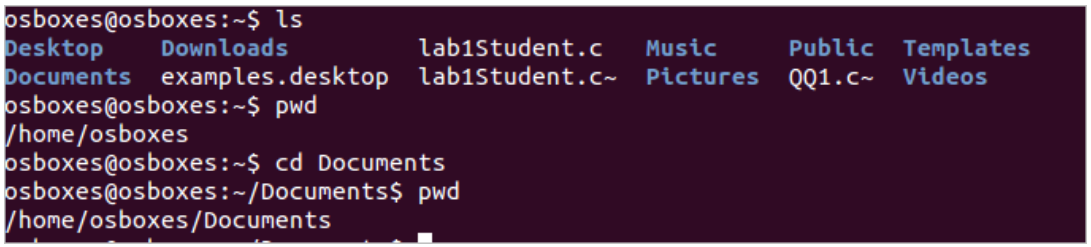
```
osboxes@osboxes:~/Desktop/TA/sampleFolder$ gcc lab1Student.c -o lab1Student
osboxes@osboxes:~/Desktop/TA/sampleFolder$ ./lab1Student
csh>pwd
/home/osboxes/Desktop/TA/sampleFolder
doc1  lab1Student  lab1Student.c  Things
csh>
```

What to do!

- In this lab your code should handle three main requirements:
  1. **Creating an External Process**: modify the main() method of the given program so that an external process is created and executes the command specified by the user.
  2. **Changing Directories**: we encounter the concept of the current working directory (pwd), which is simply the directory you are currently in. The (cd) command allows a user to change current directories. Your shell interface must support this command.
  3. **Adding a History Feature:** that allows users to see the history of commands they have entered and to rerun a command from that history.

Changing Directory

- To know your current working directory, type pwd in your terminal
- To change your working directory, type cd followed by the directory name -you want it to be your current working directoy- in your terminal
- Example:



- Problem: In Java and C, we cannot change the working directory of current process (i.e. the shell).
- Think about how to solve this!

Changing Directory

- To solve the previous problem, In Java:
- The ProcessBuilder class provides the following method for changing the working directory:

```
public ProcessBuilder directory(File directory)
```

- The current directory for the current user can be obtained by invoking the static getProperty() method in the System class as follows:

```
System.getProperty("user.dir");
```

- The home directory for the current user can be obtained by invoking the static getProperty() method in the System class as follows:

```
System.getProperty("user.home");
```

Changing Directory

- To solve the previous problem, In C:
- The C language provides the following function to change the current working directory:

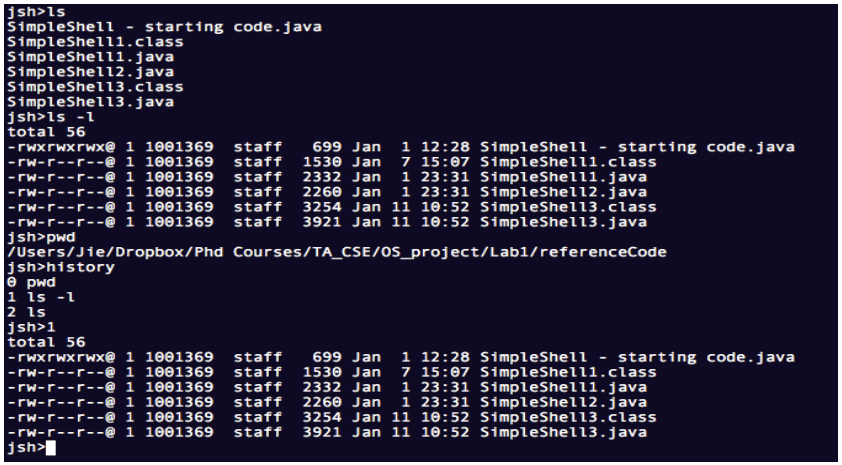
```
int chdir(const char *path);
```

What to do!

- In this lab your code should handle three main requirements:
  1. **Creating an External Process**: modify the main() method of the given program so that an external process is created and executes the command specified by the user.
  2. **Changing Directories**: we encounter the concept of the current working directory (pwd), which is simply the directory you are currently in. The (cd) command allows a user to change current directories. Your shell interface must support this command.
  3. **Adding a History Feature:** that allows users to see the history of commands they have entered and to rerun a command from that history.

History Feature

- Allow users to get the history of commands they have entered and to rerun a command from that history
  - When the user enters the command “history”, you will print out the contents of the history of commands that have been entered into the shell, along with the command numbers.
  - When the user enters “!!” , run the previous command in the history. If there is no previous command, output an appropriate error message.
  - When the user enters “<integer value *i*>”, run the *i*th command in the history.
  - For example, entering “4” would run the fourth command in the command history



Hints

- In this lab, you should think about each task before start writing your code, for example in each task, try to think about these details:
  1. How to take input from user
     - In java (readLine) in C (fget, scanf)
  2. How to parse user inputs
     - In java (split) in C (*strtok,strcmp*)
  3. How to execute command in Java/C
     - In java (.command) in C (system)
  4. How to change directory
  5. How to store all the entered lines from user to be able to use them later in your code
     - In java (<list>) in C (malloc or arrays)

From where to start!

- Open your eDimension and download the report for lab1
- Decide which language do you prefer based on your background
  - Java or C language
- Read the tasks one by one and use the help code provided in the report and the starting code in eDimention

- Don’t hesitate to ask for help from the teachers in the lab!
- Complete the shell with the required features and upload the Java or C file to eDimension before next lab