Computer System Engineering
50.005
Dr David Yau, Dr Jit Biswas

Week 1: Lab 1 (25 marks)

Objective: Create a Shell Interface using Java or C program

The Objective 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 $

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(commandTokens);
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 {
    String[] commandList = commandLine.split(" ");
    ProcessBuilder pb = new ProcessBuilder();
    pb.command(commandList);
    Process p = pb.start();
    BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream()));
    for (String line; (line = br.readLine()) != null;) {
        System.out.println(line);
    }
    br.close();
} catch (IOException e) {
    // e.printStackTrace();
    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[8192];
while (1) {
    printf("csh>");
    fgets(command, MAX_INPUT, stdin); // get user input
    system(command);
}
```

```
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>
```

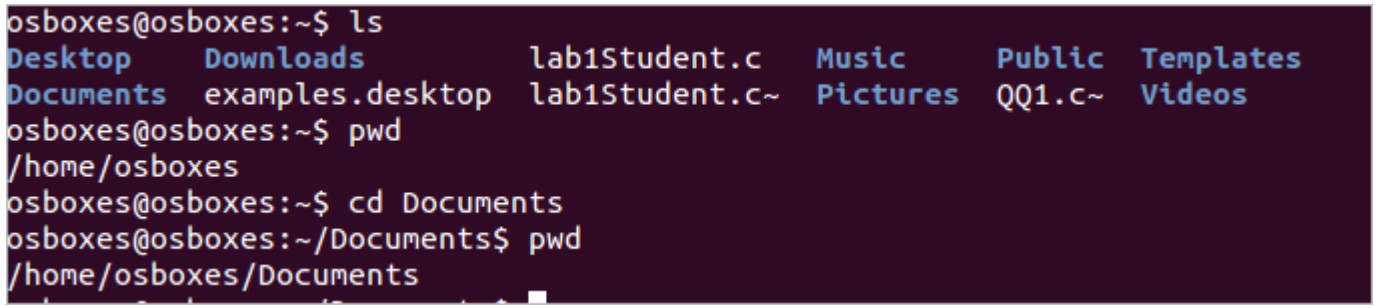
```
void exec(char **args) {
    pid_t pid;
    pid = fork();
    if (pid == 0) {
        // execute child process
        if (execvp(args[0], args) == -1) {
            perror("csh");
        }
        exit(EXIT_FAILURE);
    } else if (pid < 0) { // error forking
        perror("csh");
    }
}
```

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:



- How do you get and set the current working directory?

Changing Directory

In Java:

- To change the current directory of a ProcessBuilder instance, you can pass in a File object into its directory function:

```
ProcessBuilder pb = new ProcessBuilder();
pb.directory(new File("/Users/User/Documents"));
```

- The current directory for the ProcessBuilder instance can be obtained by:

```
File currentDir = pb.directory();
if (currentDir == null) currentDir = new File("");
```

- The home directory for the current user can be obtained by:

```
String homePath = System.getProperty("user.home");
File homeDir = new File(homePath);
```

- The parent directory can be obtained by:

```
File parentDir = new File(childDir.getAbsolutePath()).getParentFile();
```

Changing Directory

In C:

- The C language provides the following function to change the current working directory:

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

How to get the current directory in a C program:

```
#include <unistd.h>
char getcwd(char *buf, size_t size);
```

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 most-recent command in the history.
  - For example, entering “3” would run the third most-recent command

```
parallels@ubuntu:/media/psf/Ubuntu Portal/TA CSE Labs$ gcc lab1.c -o lab1_c && ./lab1_c
csh>ls
a  b  Lab1  lab1_c  lab1.c  Lab1_Shell Interface.pdf  sampleFolder  SimpleShell.class  SimpleShell.java  test
csh>mkdir c
csh>echo "HI"
HI
csh>history
1 echo "HI"
2 mkdir c
3 ls
csh>3
a  b  c  Lab1  lab1_c  lab1.c  Lab1_Shell Interface.pdf  sampleFolder  SimpleShell.class  SimpleShell.java  test
csh>
```

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 (fgets)
  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 store all the entered lines from user to be able to use them later in your code
     - In java (<list>) in C (arrays, or malloc for dynamic memory)

Requirements

- For this assignment, you can assume the following to simplify your code:
  1. The commands will not be more than 8000 characters.
  2. The commands will not have escaped whitespaces or whitespace within quotes. (e.g. mkdir “some dir” , mkdir some\ dir)
  3. The history list must be able to hold at least 10 most recent valid commands. You can hold more if you want.
- Of course, you are encouraged to make your program more versatile than the bare requirement.

Requirements

- Other requirements:
  1. You will only need to store valid commands in the history list.
  2. The commands can have consecutive tabs and spaces as whitespace, along with leading and trailing whitespace.
  3. The history list should be in *reverse chronological order, one-indexed*.
     (i.e. 1 will give the most recent valid command, 5 will give to 5^{th} most recent valid command)
  4. The '..', '.', and '~' shorthands for the parent, current and home directories respectively should be supported by the change directory feature.

Language Issues

- Java
  - Changing directory is more cumbersome for (“..”), due to the API for File.
  - Getting console output from process is not automatic.
  - No need memory management.
  - No need care about null termination of strings.

Language Issues

- C
  - May need to consider null termination in strings.
  - May need memory management.
    (depending on how conscientious you are)
  - Changing directories is easier.
    (`chdir` handles "..." automatically)
  - Getting process output is also automatic.
    (`system` does it automatically)

Where to start?

- Open your eDimension and download the handout 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 starting code provided on eDimension

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