Project 1: The OS Shell - Part A

Objectives

There are four objectives to this assignment:

- To familiarize yourself with the Unix/Linux programming environment.
- To develop your defensive programming skills in Java.
- To gain exposure to the necessary functionality in shells.
- To learn how processes are handled (i.e., starting and waiting for their termination).

Overview

The Shell or Command Line Interpreter is the fundamental User interface to an Operating System. Your first project is to write a simple shell - that has the following properties:

1. The shell must support the following internal commands:
   i. clr - clear the screen.
   ii. echo <comment> - display <comment> on the display followed by a new line (multiple spaces/tabs may be reduced to a single space)
   iii. help - display the user manual
   iv. pause - pause operation of the shell until 'Enter' is pressed
   v. exit - quit the shell
2. All other command line input is interpreted as program invocation which should be done by the shell **creating child processes**. The programs should be executed sequentially.
3. The shell must support background execution of programs. An ampersand (&) at the end of the command line indicates that the shell should return to the command line prompt immediately after launching that program.
4. The command line prompt must contain your venus username and ">" for example, in my case it would be `upvu1234>`

Project Requirements

1. Design a simple command line shell that satisfies the above criteria and implement it on venus.
2. Write a simple manual describing how to use the shell. The manual should contain enough detail for a beginner to UNIX to use it. For example, you should explain the concepts background program execution. The manual MUST be named readme and must be a simple text document capable of being read by a standard Text Editor. For an example of the sort of depth and type of description required, you should have a look at the on-line manuals for csh and tcsh (man csh, man tcsh). These shells obviously have much more functionality than yours and thus, your manuals don't have to be quite so large.
   You should NOT include building instructions, included file lists or source code - we can find

that out from the other files you submit. This should be an Operator's manual not a Developer's manual.

3. The source code **MUST** be extensively commented and appropriately structured to allow you to understand and easily maintain the code. Properly commented and laid out code is much easier to interpret and it is in your interests to ensure that the person marking your project is able to understand your coding without having to perform mental gymnastics!
4. The submission should contain only source code file(s) and the readme file (all lower case please). No compiled classes/executable program should be included. The marker will be automatically rebuilding your shell program from the source code provided. If the submitted code does not compile it can not be marked!
5. For example the files in the submitted directory could be:
   Project1a.java
   readme

Program Specifications

Your program must be invoked exactly as follows:
java Project1a

Defensive programming

is an important concept in operating systems: an OS can't simply fail when it encounters an error; it must check all parameters before it trusts them. In general, there should be no circumstances in which your java program will core dump, hang indefinately, or prematurely terminate. Therefore, your program must respond to all input in a reasonable manner; by "reasonable", we mean print an understandable error message and either continue processing or exit, depending upon the situation.

You should consider the following situations as errors; in each case, your shell should print a message (to System.err.println()) and exit gracefully:

- An incorrect number of command line arguments to your shell program.
- The file does not exist or cannot be opened.

For the following situation, you should print a message to the user (System.err.println()) and **continue** processing:

- A command does not exist or cannot be executed.

Your shell should also be able to handle the following scenarios, which are **not** errors (i.e., your shell should not print an error message):

- An empty command line.
- Extra white spaces within a command.

In no case, should any input or any command line format cause your shell program to crash or to exit prematurely.

Hints

Your shell is basically a loop: it repeatedly prints a prompt parses the input, executes the command specified on that line of input, and waits for the command to finish, if it is in the foreground. This is repeated until the user types "exit" or ends their input. You should structure your shell such that it creates a new process for each new command (except built in commands). There are two advantages of creating a new process. First, it protects the main shell process from any errors that occur in the new command. Second, it allows easy concurrency; that is, multiple commands can be started and allowed to execute simultaneously (i.e., in parallel style).

Miscellaneous Hints

Beat up your own code! You are the best (and in this case, the only) tester of this code. Throw lots of junk at it and make sure the shell behaves well. Good code comes through testing -- you must run all sorts of different tests to make sure things work as desired. Don't be gentle -- other users certainly won't be. Break it now so we don't have to break it later.

Keep versions of your code. More advanced programmers will use a source control system such as CVS. Minimally, when you get a piece of functionality working, make a copy of your .java file (perhaps a subdirectory with a version number, such as v1, v2, etc.). By keeping older, working versions around, you can comfortably work on adding new functionality, safe in the knowledge you can always go back to an older, working version if need be.

Submission : Due: 02/23/2017 at 11:59pm

For this project, you need to hand in two distinct items:

- **Your source code (no object files or executables, please!)**
- **A `readme` file as staged above**
- Under your **`$HOME`** directory, create a subdirectory: **`osproject1a` (case sensitive)**. Copy all of your .java source files into this directory. Do **not** submit any other files. **MAKE SURE TO GIVE ME PROPER PERIMISSIONS AS DISCUSSED IN CLASS**. On the deadline as listed above - system will copy all the files from your respective directory to mine for grading **Remember: No late projects will be accepted!**

Criteria (50)

- Submission of required files only, with name, student number etc on all submitted files (5)
- Warning free compilation and linking of executable with proper name (5)
- Performance of internal commands and aliases (10)
- External command functionality (10)
- Background execution (5)
- Readability, suitability & maintainability of source code (5)

- User manual (10)
    - Description of operation and commands
    - Description of background execution
    - Overall layout and display of understanding

Ensure that you address all the points specified in this project documentation. Finally, while you can develop your code on any system that you want, make sure that your code runs correctly on venus. Specifically, since libraries and environments sometimes vary in small and large ways across systems, you should verify your code executes on venus. This machine is where your projects will be tested.