Programming Assignment #1: Simple Shell

Due: 31 Jan, 2012 @ 11:59pm

Why this Assignment?

Shell also known as the command-line interpreter is one of the widely used human-computer interfaces. Understanding its operation in detail is one of first steps towards understanding the operation of a typical operating system. A shell provides interfaces to a variety of different functions provided by the underlying operation system. Some of these include process life-cycle management, file management, input/output management, and network management.

What is required as Part of this Assignment?

As part of this assignment, you are expected to design and implement a simple shell. You can **ONLY** use C to implement the shell. You need to handle user input, parse the user input and identify the command and the arguments given. Once the command is identified, you should classify the command as internal or external. The internal commands are completely handled by the shell. This means, you need to completely implement those internal commands in your simple shell. See below for the list of "built-in" commands that should be supported by your shell. The command not recognized as internal commands are external commands! For all external commands you should search the appropriate locations in the file system and execute the corresponding file.

Objectives in Detail

As part of the simple shell, you should implement the following internal commands:

- echo
- cd
- pwd
- history
- set
- env
- unset
- pushd
- popd
- exit

Type `man or which` in Linux to receive more information regarding these commands. One of the requirements is that you should implement the built-in commands without relying on any Linux provided commands. For example, Linux provides an echo command, but you should not use it to support echo in your shell. Because echo is a built-in command you should completely build the support within your shell. All other commands must be executed in a child process.

To implement the `cd` command, your shell should get the value of its current working directory (`cwd`) by calling `getcwd()` on start-up. When the user enters the `cd` command, you must change the current working directory by calling `chdir()`.

When executing a command, your shell should be able to support relative path.

For IO redirection your program have to support > and < syntax, i.e.:

```
my-shell % ls > ls.out # redirect ls's stdout to file ls.out

my-shell % wc < ls.out # redirect wc's stdin from file ls.out
```

Your shell also has to support pipeline, which chains a group of processes by their standard streams, for example:

```
ls -l | more
```

Please also make sure your shell correctly records the last exit code

```
% ls /

bin   coda  etc   lib         misc  nfs  proc  sbin  usr

boot  dev   home  lost+found  mnt   opt  root  tmp   var

% echo $?

0

% ls bogusfile

ls: bogusfile: No such file or directory

% echo $?

1
```

What to Hand in

Submit a zip file which contains:

1. A README file with:
    1.1. Your name
    1.2. If you have not fully implemented all shell functionality then list the parts that work so that you can be sure to receive credit for the parts you do have working.
2. All the source files needed to compile, run and test your code (Makefile, .c). Do not submit object or executable files.
3. Output from your testing of your shell program. Make sure to demonstrate:

3.1.simple Unix commands

3.2.built-in commands

3.3.I/O redirection

You can use the testing script, which will be provided 2 weeks before the due date, to capture the output like this:

```
python shell_testor.py [the executable file of your shell]
```

This testing script will run a group of commands in your shell and record output in `test_result.txt`.

We are going to use this script to test your program.

Suggestions

1. Make sure your source code could compile. If you have access to 2 computers, try to compile your code on both of them.
2. Read the content of `test_result.txt` to see if anything goes wrong.
3. You may find this book, Advanced Linux Programming, Chapter 1-3, useful for this project.

Proposed Approach

```
// Pseudo code for a VERY simple shell. It finds an executable in the PATH,
// then loads it and executes it (using execv).

main () {
   /* Initialize the shell parameters */
   parsePath(pathv);
   // read the environment variables and determine PATH
   // other initializations might go here such as setting prompt and
   // all other parameters
   loop = TRUE;
   while(loop) {
      printPrompt();
      /* Read the command line and parse it */
      readCommand(commandLine) ;
      parseCommand(commandLine, command);
      /* Get the full pathname for the file */
      command.name = lookupPath(command.argv, pathv);
      if(command.name == NULL) {
         /* Report error */
         continue;
      }
      /* Create child and execute the command */
      /* Wait for the child to terminate */
   }
   /* Shell termination */
}
```