Project 1

CS-UH 3010: Operating System

Due: Tuesday, February 19, by 11:59PM

The objective of this assignment is to become familiar with the use and programming in the Linux Operating System environment. In this project, you will create an application which is a simplified version of the *UNIX SHELL* using C/C++. **This is going to be an individual project, not a group project**.

Your Task

Your task is to create an application called *myshell* to parse command line input from the user and execute it if it is a built-in command. All command line input that user enters should be recorded in the history file. You also need to implement shell variables which need to be exported to your application through command line. You will implement the following built-in commands.

- pwd - print name of current/working directory.
- cd - change the current directory.
- export - names of exported shell variables.
- history - display the command history list with line numbers.
- exit - free all space allocated gracefully and exit the application.

Project Requirements

- Your task is to create an application to parse command line input from the user and to build the argv data structure in the form expected in any C/C++ main program. Input line is separated by space as given below. [**5pts**]

  identifier [identifier [identifier]]

- If it is a builtin command execute it, otherwise your program should search the directory system in the order specified by the *PATH variable* for a file with the same name as the first identifier and display the full path of the command and argument(s) passed. [**10 pts**]
- All path variables need to be exported to your application using *export* command. Variable name *PATH* should be exported by default in your application but the user should be able to modify it using *export* the command [**20 pts**]
- Application should record the command line inputs from the users and append it to the history file. On every invocation of your application, you should be able to see the content from history by using built-in command *history.* [**20 pts**]
- If the user types *exit* then free all space allocated gracefully, close all open file descriptors and exit the application. [**3 pts**]
- Your application can change the current directory using built-in *cd* command and you can verify it using *pwd* command. [**7 pts**]
- If the command is starting with the **!** symbol and followed by a number then your program needs invoke the line of command from the history file [**10 pts**]

Procedural Matters

- Your program is to be written in C++ or C and must run on the NYUAD Linux (CentOS) Server M-DCLAP-P302-CSD.abudhabi.nyu.edu(secure port: 4410)

  eg:- ssh -p 4410 nr83@M-DCLAP-P302-CSD.abudhabi.nyu.edu

- You will have to first submit your project electronically and subsequently, demonstrate your work.
- Nabil Rahiman (nr83@nyu.edu) will be responsible for answering questions as well as reviewing and marking the assignment.

What you Need to Submit

- A directory that contains all your work including source, header, Makefile, etc.
- The program needs to be properly commented.
- All the above should be submitted in the form of a tar or zip file bearing your name (for instance NabilRahimanProj1.tar).
- Submit the above tar/zip-ball using NYU classes

Grading Scheme

Aspect of Programming Assignment Marked	Total Grade 100 pts
Quality in Code Organization and Modularity	8pts
Addressing All Requirements	75pts
Use of Makefile and Separate Compilation	5pts
Properly Commented Code	5pts
Handling unexpected termination	7pts

Noteworthy Points

- You have to use separate compilation in the development of your program.
- Although it is understood that you may exchange ideas on how to make things work and seek advice from your fellow students, sharing of code is not allowed.
- If you use code that is not your own, you will have to provide appropriate citation (i.e., explicitly state where you found the code). Otherwise, plagiarism questions may ensue. Regardless, you have to fully understand what and how such pieces of code do.

Sample Program Output

```
$$./myshell
>> history
>>
>> ls
command not found
>>
>> export
PATH=
>>
>>
>> export PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
>> export
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
>>
>> ls -l -a
ls is an external command (/bin/ls)
command arguments:

>> cd h1
>>
>> pwd
/home/test1/h1
>> cd h2/h3
>>
>> pwd
/home/test1/h1/h2/h3
>>
>> history
    1  history
    2  ls
    3  export
    4  export PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
    5  export
    6  ls -l -a
    7  cd h1
    8  pwd
    9  cd h2/h3
   10  pwd
   11  history
>> export JAVA_PATH=/home/test/java/bin
>>
>> export
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
JAVA_PATH=/home/test/java/bin
```

```
>> history
    1  history
    2  ls
    3  export
    4  export PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
    5  export
    6  ls -l -a
    7  cd h1
    8  pwd
    9  cd h2/h3
   10  pwd
   11  history
   12  export JAVA_PATH=/home/test/java/bin
   13  export
   14 history
>>
>> !2
ls is an external command (/bin/ls)
command arguments:
-l
-a
>>exit
```

Your program terminates here. Invoke program again to verify you recorded the command history in history file.

```
$$./myshell
>> history
    1  history
    2  ls
    3  export
    4  export PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
    5  export
    6  ls -l -a
    7  cd h1
    8  pwd
    9  cd h2/h3
   10  pwd
   11  history
   12  export JAVA_PATH=/home/test/java/bin
   13  export
   14  history
   15  exit
   16  history
```

Additional Materials

Table 1: **Some Useful system calls**

`open(2)`	open and possibly create a file
`access(2)`	check real user's permissions
`read(2)`	read from a file descriptor
`write(2)`	write to a file descriptor, for details type: man 2 write
`getwd(3)`	get current working directory
`chdir(2)`	change working directory
`exit(2)`	terminate the program

Table 2: **Some Useful gdb commands**

`break foo`	Breakpoint at function `foo`
`break fees.c:39`	Breakpoint at line `39`
`run`	Run the program util breakpoint is hit
`cont`	Continue running (after hitting a breakpoint)
`bt`	Print a back trace (BT) of the function call stack
`print i`	Print value of local variable `i`
`print &i`	Print address of variable `i`
`info frame`	Print information about the current stack frame

Table 3: **Unix Development Tools**

`gcc/g++`	GNU C/C++ compiler
`gbd`	GNU debugging tool
`strace`	system call tracer
`vim`	Text command line editor
`make`	GNU make utility to maintain groups of programs