CS 460 Assignment #3 **Your Own Unix Shell**

Date Assigned: March 3, 2014
Date Due: March 17, 2014
Points: 75

Goal:
Learn about Unix processes, interprocess communication, fork(), exec(), pipe() dup2(), waitpid().

Description:
You will need to design a C program that implements a shell interface that accepts user commands and implements each command in a separate process. Your shell will give the user a prompt which is the process id followed by a >. The user then types in a shell command and your program will execute the command. Here is an example of how your completed program is to work:

```
zeus$ ./CS460_Shell
21364> ls
CS460_Shell.c  CS460_Shell.o CS460_Shell hint.txt
21364> cat hint.txt
you should look at man -s 2 pipe
recursion is fun!
21364> cat hint.txt | grep fun
recursion is fun!
21364> cat hint.txt > newFile.txt
21364> ls newFile.txt
newFile.txt
21364> firefox &
21364> ps | grep firefox
 2515 pts/1    00:00:01 firefox
21364> exit
zeus$
```

Your Program:

Your program will need to parse the command line input from the user and execute each command in a separate process. You need to handle the redirection symbol which will redirect stdout to a file (>). Users also need to be able to use the | to send the output of one shell program to the input of another. Finally, the & symbol at the end of a line will launch an application in the background and present the user with a shell prompt again before the background application terminates.

One technique for implementing your shell interface is to have the parent process read the user's input

from the command line and then create a child process that performs the command. The parent process will wait for the child to exit unless the command ends with a & in which case the child will continue running in the background.

The separate process is created using a fork() system call and the user's command will be executed using one of the system calls in the exec() family. The outline of a simple shell is:

```
#define MAX_LINE 256
#define MAX_ARGS 40

int main ()
{
  char *pTokens[MAX_ARGS];
  bool bIsRunning = true;

  while (bIsRunning)
  {
    // print prompt

    // Read user input
    // 1. if first token is "exit", break out of loop
    // 2. fork a child process
    // 3. child invokes proper exec () system call

  }

  return 0;
}
```

Part I

You must parse the user's input into separate tokens, storing the tokens in an array of character strings, pTokens, that can be processed accordingly. For example, suppose the user enters **`2043> ls –al`** from the command line. The values stored in pTokens would be as follows:

```
pTokens[0] = "ls"
pTokens[1] = "-al"
pTokens[2] = NULL
```

The array can then be passed into the execvp () function with the following prototype:

```
execvp (char *command, char *params[]);
```

Part II

Modify the shell interface so that it provides a history feature. This feature will list the most recent 25 commands. For example, suppose the user entered date, cat file, ls –al, ls, ps, the command history will output the following:

```
1 date
2 cat file
3 ls -al
4 ls
5 ps
```

Your shell must support two additional features for retrieving commands from the command history:

1. If the user enters !!, the most recent command in the history table is executed. Further, the executed command is echoed to the screen and added to the history. For the above example, if the user enters !!, then ps would be executed and ps would be added to the history table.
2. If the user enters ! followed by an integer N, the Nth command in the history table is executed. For the above example, if the user enters !1, then date would be echoed to the screen, executed and then added to the history table.

Notes:

1. If the user enters !! and there is no history, print the message **No History Commands**.
2. If the user enters !N and N is not in the range of 1 to the last command, print the message **No Such Command Exists**.
3. Each command line will be no more than 2048 characters.
4. Each symbol (> |) will be surrounded on either side by at least one space (**ls|more** is not allowed).
5. The & will be preceded by at least one space.
6. You do NOT need to support wild cards (**ls *.txt**).
7. Maintain the most recent 25 history commands.
8. A shell command will be well-formed.
9. The only *builtin* functions you need to create are **history** and **exit**.

Good online set of man pages: **http://linux.die.net/man/**

Part III

Unless otherwise specified, the parent process will wait for the child to complete before continuing. If the command ends with an &, the parent and child processes will run concurrently. Do not process &, >, or the | features until you get Parts I and II completely and correctly working.

Research the following functions:
fork(), exec??(), strtok_r(), dup2(), pipe(), waitpid(), STDOUT_FILENO, STDIN_FILENO

Subversion

You must store your source code in a Subversion repository on zeus. Name your project **CS460_Shell_PUNetID**.

Make Targets

CS460_Shell:	build the executable.
valgrind:	start the executable with valgrind

Executable Location

You must build the executable (CS460_Shell) at the root of your Eclipse Project.

Submitting

On the day your assignment is due, you are to submit a zipped up tar file on zeus called **`CS460_Shell_PUNetID.tar.gz`** and then using the submit script, type the following:

```
zeus$ submit cs460s14 CS460_Shell_PUNetID.tar.gz
```

Program Design

Don't just start writing code as your end result will be spagetti code. Design before coding writing well defined functions to do one major task. If your design is poor, you will lose significant points.