CIS*3110 Operating Systems (Winter 2022)

Assignment #1: Writing a Shell

Requirements and Specifications

Objective
In this assignment you are to implement a simple UNIX shell program. A shell is simply a program that conveniently allows you to run other programs. Read up on the *bash* shell to see what it does (some references are included at the end of this document and on CourseLink).

Specification
Your shell must support the functions described below. The functions are grouped into 3 sets. You may develop the functions in any order, but they are grouped in order of difficulty. It would be prudent to get the simpler functions working first. With each function there are some suggestions for POSIX system calls that may help with the construction of the function. These suggestions are not exhaustive, nor do you necessarily need to use them. They are there to stimulate your thinking.

Notes
You must check and correctly handle all return values. This means that you need to read the **man** pages for each function to figure out what the possible return values are, what errors they indicate, and what you must do when you get that error.

There are submissions instructions on the *CourseLink* site. With your fully commented code you must submit a **README.txt** file that explains what functions you have implemented, how they work (any assumptions or limitations) and how they were tested. You should also explain what you did not have time to implement. You must also submit a **Make** file.

This is an **individual** assignment - no code sharing is allowed. **Code will be checked for integrity.**

Set 1 Functions

1. The *internal* shell command "exit" which terminates the shell.
   - *Concepts*: exiting the shell
   - *System calls*: exit(), kill()

2. A command with **no** arguments. If the command does not start with **./** (i.e. it is an executable file in the current directory) then the executable exists in one of the appropriate system directories (/bin, /usr/bin)
   - *Example*: ls
   - *Details*: Your shell must block until the command completes and, if the return code is abnormal, print out a message to that effect.
   - *Concepts*: Forking a child process, waiting for it to complete; *synchronous* execution
   - *System calls*: fork(), execvp(), exit(), wait(), waitpid()

3. A command with arguments.
   - *Example*: ls -l
   - *Details*: Argument 0 is the name of the command
   - *Concepts*: Command-line parameters

4. A command, with or without arguments, executed in the **background** using &.
   - *Notes*: For simplicity, assume that if present the & is always the last thing on the line.
   - *Example*: ps aux &
   - *Details*: In this case, your shell must execute the command and return immediately, not blocking until the command finishes
   - *Concepts*: Background execution, signals, signal handlers, processes, *asynchronous* execution

Notes

You start your shell by typing:

```
$ ./myShell
> ls
myShell.c      makefile       testfile
> exit
myShell terminating…

[Process completed]
$
```

where the blue text represents your login shell and the black text is output from your myShell program and the red text represents commands that you have typed into your shell program.

When your shell exits it should explicitly kill any child processes that are still active (*i.e.,* those that are in the background).

Set 2 Functions

5. A command, with or without arguments, whose output is **redirected to** a file.
    - *Example*:
      ls -lt > foo
    - *Details*: This takes the output of the command and put it into the named file.
    - *Concepts*: File operations, output redirection
    - *System calls*: freopen()

6. A command, with or without arguments, whose input is **redirected from** a file.
    - *Example*:
      sort < testfile
    - *Details*: This takes the named file as input to the command.
    - *Concepts*: Input redirection, file operations
    - *System calls*: freopen()

7. A command, with or without arguments, whose output is piped to the input of another command.
    - *Example*:
      ls -lt | more
    - *Details*: This takes the output of the first command and makes it the input to the second command. You are required to implement **one** level of pipe.
    - *Concepts*: Pipes, synchronous operation
    - *Hint*: You might want to read the textbook to understand this function
    - *System calls*: pipe()

Set 3 Functions

Advanced shell functionality that must include the shell variables and built-in functions. Adding this functionality may require that you rewrite some of the code for Set 1 and 2 functions. This is because you are probably your code is relying on the shell program that you are executing your program in (probably *bash*) and it is searching for commands based on the information in its PATH environment variable. You must now replace the use of the environment variables from your account's shell and do everything (including searching for commands that contain a slash (/) in the command name.

8. Limited shell environment variables: myPATH, myHISTFILE, myHOME.
    - The shell allows for the definition and storage of variables and there are a number of default ones. You need to only provide 3 environment variables:
        - $myPATH: contains the list of directories to be searched when commands are executed. For this exercise the default will be **/bin** to facilitate testing. This is not the normal default. To find the default on a UNIX bash shell,

execute the following command: echo $PATH. *Hint*: execv() does not use the PATH variable from your account's shell.
    - $myHISTFILE: contains the name of the file that contains a list of all inputs to the shell. The default name of this file is ~/.bash_history. For the purposes of this assignment, the default will be the file ~/.CIS3110_history.
    - $myHOME: contains the home directory for the user.

9. Reading in the profile file on initialization of the shell and executing any commands inside.
    - By default in the bash shell, the profile file is call .bash_profile and is in the user's home directory. The profile file will be named .CIS3110_profile for the purposes of this assignment and will be in the user's home directory.
    - Commands in the profile file typically involve the setting of environment variables.
        - Using the $myPATH variable to set the path for commands and changing the myPATH using the **export** command.
    - Using your own $myPATH variable will mean that you cannot use the one designated by the shell from which you are executing your code (*i.e.* the bash shell). Therefore you must not use any commands in the exec() family that use the bash shell $PATH.

10. Implementing the builtin functions: export, history and cd.
    - The **export** builtin command
        - The POSIX standard defines export as follows:
          The shell shall give the **export** attribute to the variables corresponding to the specified names, which shall cause them to be in the environment of subsequently executed commands. If the name of a variable is followed by *= word*, then the value of that variable shall be set to *word*.
        - You may restrict the use of export to just the environment variables myPATH, myHOME and myHISTFILE.
        - The format of the export command is illustrated by the following examples:

          ```
          export
          myPATH=/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:$myHOME/bin
          export myPATH=/usr/local/sbin:$myPATH
          ```

    - The **history** builtin command

      history prints out all input to the shell. The format of its output is illustrated in the following example:

      ```
      1 echo $myPATH
      2 gcc assign1a.c -o assign1a
      3 ./assign1a
      4 ls /bin
      5 ls /usr/bin
      6 vi assign1a.c
      ```

```
7 gcc assign1a.c  -o  assign1a
8 ls -la ~ | more
```

- *(space) number (2 spaces) command line*
- The following two parameters must be recognized by your shell:
    - *history -c* which clears the history file
    - *history n* which outputs only the last *n* command lines

- The **cd** builtin command
    - The cd or "*change directory*" changes the notion of which directory the command is being issued from.
    - *Hint*: chdir()
    - If you implement the cd command (and thus your shell is aware of the current working directory's full path name) then replace the "> " prompt in your shell with "*cwd* > " where *cwd* is the full path name of the current working directory. For example:

```
$ ./myShell
/Users/dastacey/Teaching/CIS3110> cd ..
/Users/dastacey/Teaching> cd ..
/Users/dastacey> exit
$
```

References

- https://en.wikipedia.org/wiki/Unix_shell
- https://en.wikipedia.org/wiki/Bash_(Unix_shell)
- http://www.tldp.org/LDP/Bash-Beginners-Guide/html/Bash-Beginners-Guide.html#sect_01_01
- http://tldp.org/LDP/abs/html/internalvariables.html
- https://www.delftstack.com/howto/c/execvp-in-c/