

## Machine Problem 4: The UNIX Shell (Due: 03/10/17)

### Introduction:

Most useful interaction with a UNIX system occurs through the shell. Using a series of easy to remember and simple commands, one can navigate the UNIX file system and issue commands to perform a wide variety of tasks. Even though it may appear simple, the shell encapsulates many significant components of the operating system.

### Basic Shell Features:

#### **Environment**

The shell maintains many variables which allow the user to maintain settings and easily navigate the filesystem. Two of these that are particularly important are **the current working directory and the PATH**. As its name implies, the current working directory variable keeps track of the user's current directory. The PATH variable consists of string of colon separated pathnames. Whenever you type a name of a command, the kernel searches in the directories specified by the PATH variable starting with the leftmost directory first. If the executable is not found in any of the specified directories, then the shell returns with an error. One may modify the PATH at any time to add and remove directories to search for executables.

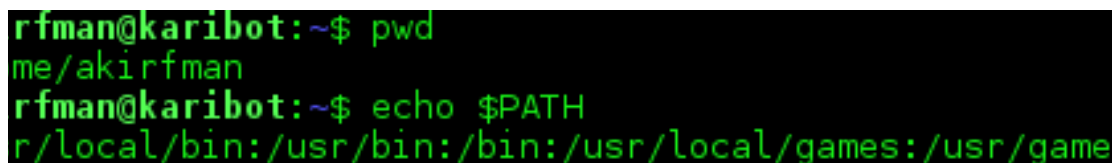
A terminal window with a black background and green text. The prompt is 'rfman@karibot:~\$'. The first command is 'pwd', which outputs 'me/akirfman'. The second command is 'echo \$PATH', which outputs 'r/local/bin:/usr/bin:/bin:/usr/local/games:/usr/game'.

Figure 1 -- Current Directory & PATH (*echo is a built-in command in the bash and C shells that writes its arguments to standard output.*)

#### **Pipelining**

UNIX provides a variety of useful programs for you to use (grep, ls, echo, to name a few). Like instructions in C++, these programs tend to be quite effective at doing one specific thing (Such as grep searching text, ls printing directories, and echo printing text to the console). However, programmers/OS users would like to accomplish large tasks consisting of many individual operations. Doing such requires using results from previous steps in order to complete a larger problem. The UNIX shell supports this through the pipe operation (represented by the character `$\vert$`). A pipe inbetween two commands **causes the standard output of one to be redirected into the standard input of another**. An example of this is provided below, using the pipe operation to search for all processes with the name "bash".

```
akirfman@karibot:~$ ps -elf | grep "bash" | grep -v grep | awk '{print$10}'  
5939  
5936
```

Figure 2 -- Piping Between Commands

## Input/Output Redirection

Many times, the output of a program is not intended for immediate human consumption (if at all). Even if someone isn't intending to look at the output of your program, it is still immensely helpful to have it print out status/logging messages during execution. If something goes wrong, those messages can be reviewed to help pinpoint bugs. Since it is impractical to have all messages from all system programs print out to a screen to be reviewed at a later date, sending that data to a file as it is printed is desired.

Other times, a program might require an extensive list of input commands. It would be an unnecessary waste of programmer time to have to sit and type them out individually. Instead, pre-written text in a file can be redirected to serve as the input of the program as if it were entered in the terminal window.

In short, the shell implements input redirection by redirecting the standard input of a program to an file opened for reading. Similarly, output redirection is implemented by changing the standard output (and sometimes also standard error) to point to a file opened for writing.

```
akirfman@karibot:~$ echo "This text will go to a file" > temp.txt  
akirfman@karibot:~$ cat temp.txt  
This text will go to a file  
akirfman@karibot:~$ cat < temp1.txt  
This text came from a file
```

Figure 3 -- Input/Output Redirection (<http://www.linfo.org/cat.html>)

### Assignment:

For this assignment, you are to design a simple shell which implements a subset of the functionality of the Bourne Again Shell (Bash). The requirements for your shell are as follows:

1. Continually prompt for textual user input on a command line.
2. Parse user input according to the provided grammar (see Table 1 below)
3. When a user enters a well formed command, execute it in the same way as a shell.
4. **You must use the commands fork and exec to accomplish this.** You may NOT use the C++ system() command.
5. Allow users to pipe the standard output from one command to the input of another an arbitrary number of times.
6. Support input redirection from a file and output redirection to a file.
7. Allow users to specify whether the process will run in the background or foreground using an '&'. (Commands to run in the foreground do not have an '&', and commands that run in the background do)
8. (Bonus) Allow users to specify a custom prompt which **supports printing the current directory, username, current date, and current time.**

```
valid string = unix_command PIPE unix_command || unix_command REDIRECTION  
filename || unix_command AMP || unix_command || special_command
```

```
unix_command = command_name PIPE command_name ARGS
```

```
special_command = cd DIRECTORY || exit
```

```
command_name = any valid executable/interpreted filename
```

```
AMP = &
```

```
ARG = string
```

```
ARGS = ARG ARGS || ARG
```

```
DIRECTORY = absolute path || relative path
```

```
PIPE = |
```

```
REDIRECTION = < || >
```

Table 1 -- Simple Shell Grammar

## Advanced Concepts:

### **File Pathnames**

The UNIX filesystem is organized as a giant tree. Leaf nodes are files, and non-leaf nodes are directories which either contain files or other directories. The highest node in the tree is root, denoted by a forward slash (/). All other files/directories in the system are child nodes of the root node.

Filenames in a UNIX system are specified by a pathname (such as /bin/bash stating that the file bash is inside the directory bin which is inside of root). Pathnames are provided either as absolute pathnames, which are given relative to the root of the filesystem (Ex: /usr/bin/env), or as relative, which are specified relative to the current working directory (../../andrew/homework/CSCE\\_313).

Side note: UNIX contains two special paths present in each directory. A single dot stands for the current directory, and two dots stands for the parent directory. To see these, type the command `ls -l` into a shell.

### **Process Address Space**

The address space of a process is usually divided up into 4 regions, stack segment, dynamic data, static data, and text segment (See Figure #4).

All code pertaining to the execution of the program is contained in the text segment. Program data is stored in two sections depending on the type. Static data (global variables and constants) is defined before the execution of the program. As a result, this segment is of a fixed size which can be allocated on loading. Dynamic data contains all data allocated to the process through calls to malloc/new. Since this changes continually throughout the process's life cycle, this section of the address space grows towards higher addresses and shrinks back towards lower addresses. Finally, the stack contains data pertaining to function calls. As with the dynamic data segment, the stack must grow and shrink with the execution of a process. Therefore, the stack grows downwards from high addresses towards lower ones and shrinks back to high addresses.

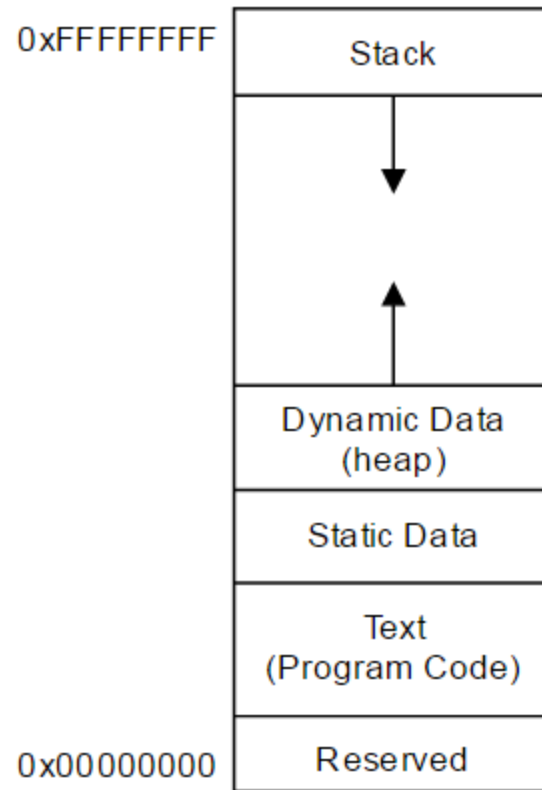


Figure 4 -- Simple Address Space

## UNIX man Pages:

One incredibly useful feature of UNIX operating systems that many new developers do not know about is the built in manual system. Using the command 'man', you can access information about most aspects of the operating system from general commands all the way to system call APIs.

The structure of man pages in UNIX are organized into sections by number follows (Wikimedia Foundation):

1. General Commands
2. System Calls
3. Library Functions (Specifically, the C standard library)
4. Special Files
5. File Formats
6. Games
7. Miscellanea
8. System Administration

You may find the following manual pages useful when creating this assignment. Each one of these lines can be executed as a valid shell command to open a particular manual page. Note, the number indicates the manual section that that function resides.

- man 3 exec
- man 2 fork
- man 2 chdir
- man 2 pipe
- man 2 dup

## Grading Rubric:

In MP4, we will test your shell with 9 different test

1. SimpleCommand1 [10 points]
  - a. `ls -t`
2. SimpleCommand2 [10 points]
  - a. `pwd -t`
3. SimpleArguments1 [10 points]
  - a. `ls -la -t`
4. SimpleArguments2 [10 points]
  - a. `df -h`
5. Redirection [15 points]
  - a. `Hello World > temp_shell`
  - b. `Cat < temp_shell`
6. multipleArgs [10 points]
  - a. `Hello World > temp_shell`
  - b. `tar -c -v -f temp.tar temp_shell`
  - c. `ls -la | grep temp.tar`
  - d. `ls -la | grep temp.tar`
7. simplePipe1 [10 points]
  - a. `ls | cat`
8. advancedPipe1 [10 points]
  - a. `ls | cat | cat | cat`
9. advancedPipe2 [15 points]
  - a. `ls -la | cat -A | cat -A | cat -A`

## Bibliography:

- [1] Bach, Maurice J. The Design of the UNIX Operating System. Englewood Cliffs, NJ: Prentice-Hall, 1986. Print.
- [2] Stevens, W. Richard. Advanced Programming in the UNIX Environment. Reading, MA: Addison-Wesley Pub., 1992. Print.
- [3] "Man Page." Wikipedia. Wikimedia Foundation, n.d. Web. 18 Feb 2017.