Operating Systems Project - 2017
Title: Unix Shell Project

Project Team:

Ujjwal Pasupulety (15IT150)
GVS Tushaar (15IT117)
Bharath A Kinnal (15IT114)
Rajeev Anirudh G (15IT230)

Unix Shell Project

Objective:

The **shell** is the part of the Operating System with which a computer usually interacts *(Unix - Command Line interpreter)*. The **shell** provides an interface for the User to run programs, execute commands and to manage files. The aim of the project will be build a simple yet powerful **shell** that will run on the Unix/ Linux System (*Programming will be in* ***C***).

Implementation:

The **shell** will be written in **C** language. The knowledge of both **C** language and **POSIX** System Calls is essential. Note that, usage of **scripting** in any form will **not** be done as the project does not make use of any existing shell.

Project Components:

- The **shell** will prompt the user, **read** a command string, execute the command and repeat the process.
- The **shell** will include an "**exit**" command, which will **terminate** the **shell**.
- The **shell** will also include the command "**type <file-name>**" which will print out the contents of the **<file-name>** onto the screen. For example, if the user types: **type Test.c** at the command-line, the contents of **Test.c** will be printed out.
- The **shell** will also include the command "**copy <file-name1> <file-name2>**" which **copies** the contents of **<file-name1>** to **<file-name2>**. The **shell** will create a **<file-name2>** (*If the <file-name2> does not exist*), open **<file-name1>** and copy the contents of **<file-name1>** byte-to-byte to **<file-name2>**.
- If the user types a command other than **exit**, **type**, **copy**, the **shell** will produce an appropriate error message and prompt again. If the user tries to "**type**" a file that does not exist or "**copy**" a file that does not exist, the **shell** will produce an appropriate error message and prompt again. In other words, the **shell** will be reasonably difficult to crash.
- At a higher level, if the user types a command that does **not** exist on the **shell** then the **shell** must assume that the command is the name of a **program** located in the current directory and should attempt to **run** the program. If the **program** does not exist or is not an executable, the **shell** will produce an appropriate error message and prompt again.
- The **shell** will include the command "**delete <file-name>**", which should **delete** *(Linux: rm -rf)* the **<file-name>** using the remove System Call.

To execute the programs the **shell** makes use of **execvp** System Call, which loads the program into the memory and executes it. Syntax: **execvp(<file-name> args)** **<file-name>** is a character array (string) consisting of the name of the program to be executed. Arguments **args** is an array of strings consisting of **Command-Line** arguments to be passed to the program. If **no** Command-Line arguments are to be passed then, we must define an empty array and pass that as **args**. This can be done:

```
char args[1][1] ;
args[0][0] = '\0' ;
```

The problem with using **execvp** will be that the when the **program** finishes its execution it will not go back to the **shell**, instead the **shell** will terminate. To overcome this situation, we must create a new process via **fork** System Call and use the **wait** System Call in the parent program and let the **shell** run in the background to take over when the program finishes. The **child** process (*PID = 0*) will call **execvp** to execute the program, while the **parent** process (*PID > 0*) will return to the prompt.

Extensions:

Executing a file is a matter of loading the program file into the memory. Executable files have a file header that tells the interpreter how to interpret the file. All executable files have **two** components that must be initialized:

- Machine Code (*called **.text***).
- Global Data Variables (*called **.data***).

The header specifies the location in memory as to where **.text** and **.data** must be placed. We must decode the **ELF** (*Executable and Linkable Format*) file format produced by **gcc**.

As an extension, the **shell** executes the **ELF** executable files without using **execvp**. In other terms, the shell will **not** use **execvp** System Call, but instead uses the following series of steps to execute a program:

- Read and decipher the **ELF** headers at the beginning of the file - Location of **section** headers.
- Read and decipher the **section** headers. There is a header for each **.text** and **.data** section. The header tells where the **section** is located in the file, and where to put it in the memory.
- Allocate the memory the the headers are asking for to a character array, using the **mmap** instruction - returns **0** if it fails to allocate.
- Copy the **sections** into appropriate places in the character array.
- The **entry point** to the program is given in the **ELF** header. We must jump to it.

The **struct** (*Structures in C*) for decoding the **ELF** headers and **session** headers are provided in “**elf.h**”.