Objectives

The goal of the project is to:

1) Implement a Shell

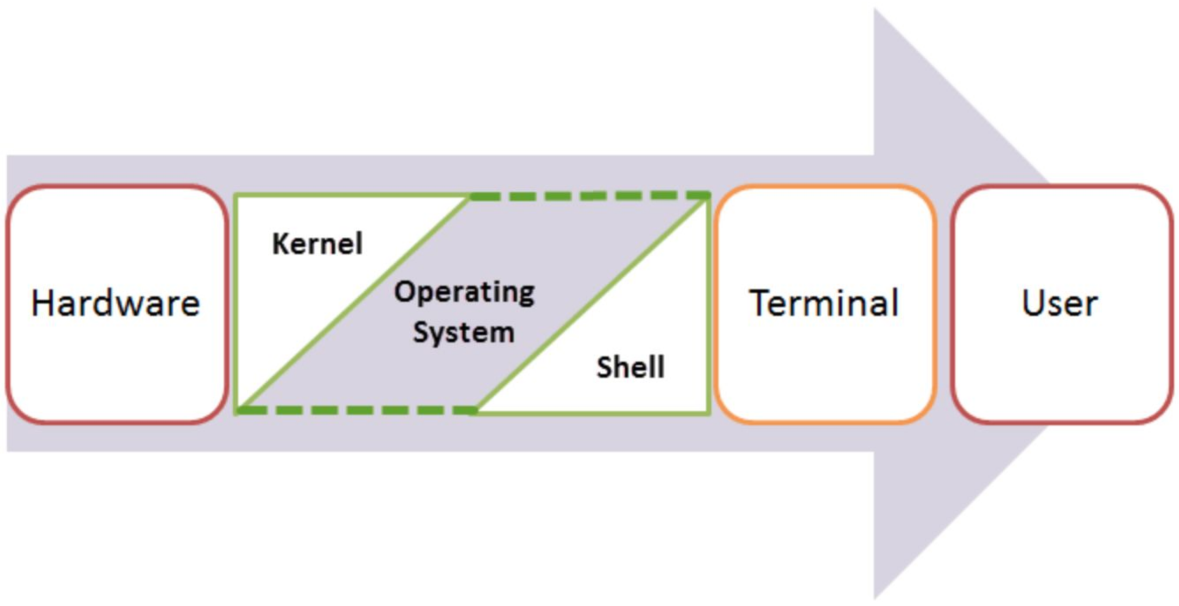
2) Implementation of features

- Piping and I/O redirection
- History feature
- Editor
- Aliasing

3) Implement custom functions

Abstract:

This project aims to create a virtual shell similar to unix shell.A shell is an interface that allows you to interact with the kernel of an operating system.The shell is the layer of programming that understands and executes the commands a user enters.



The shell is also called a command line interpreter. Users direct the operation of the computer by entering
commands as text for a command line interpreter to execute, or by
creating text scripts of one or more such commands. This project is
divided into three phases as explained below.

When you run the terminal, the Shell issues **a command prompt (usually $),** where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal.
The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name **Shell**.

After a command is entered, the following things are done:

1. Command is entered and if length is non-null, keep it in history.
2. Parsing : Parsing is the breaking up of commands into individual words and strings
3. Checking for special characters like pipes, etc is done
4. Checking if built-in commands are asked for.
5. If pipes are present, handling pipes.

6. Executing system commands and libraries by forking a child and calling execvp.
7. Printing current directory name and asking for next input.
8. For keeping history of commands, recovering history using arrow keys and handling autocomplete using the tab key, we will be using the readline library provided by GNU.

Implementation

Phase 1

A shell does three main things:

- Initialize: In this step, a typical shell would read and execute its configuration files. These change aspects of the shell's behavior.
- Interpret: Next, the shell reads commands from stdin (which could be interactive, or a file) and executes them.
- Terminate: After its commands are executed, the shell executes any shutdown commands, frees up any memory, and terminates

In our shell there won't be any configuration files, and there won't be any shutdown command. So, we'll just call the looping function and then terminate.

Out of the above 3 things, we are mostly concerned with scanning and would build our shell on that component. Now what does shell do as a part of scanning and executing is as follows:

1. Reading the input

2. Parsing the input to separate it into commands and arguments

3. Executing the commands from step 2.

The steps followed by it:

1) Taking input (command) from user

2) Forking a child process

3) Calling exec to replace the child process with the input taken in the first step.

4) Waiting for the child process to finish execution

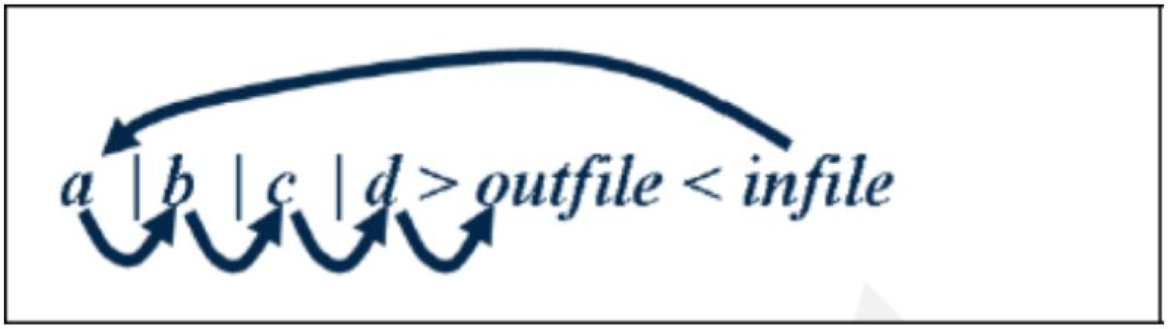
5) Exiting

Phase 2

In Phase 2, We add more functionality to the basic shell.
Here is a list of functions which our shell performs:

1. Piping and I/O redirection: Shell is to have
the parent process do all the piping and redirection before forking
the processes. In this way the children will inherit the redirection.

The parent needs to save input/output and restore it at the end.



2. History feature: We implement a history feature in
your shell which is able to give last 25 commands used with there
PID’s.

• List last 25 commands used with date and timestamp

• Search specific commands in history using grep

3. Editor: We Implement a customized editor in which we can write our
code and execute using the shell.This takes shell commands as input.

It should be able to take input on multiple lines and when we press \ enter. then combine the entire input as one job and execute it.

4. Aliasing: An alias is a name that the shell translates
into another name or command. Aliases allow you
to define new commands by substituting a string for the first token
of a simple command. They are typically placed in the ~/.bashrc

(bash) or ~/.tcshrc (tcsh) startup files so that they are available to
interactive subshells. Under bash the syntax of the alias builtin is
alias [name[=value]]

Phase 3:

Phase 3 involved the implication of custom functions. There are multiple approaches involved in carrying out this task.

We have implemented 4 custom commands using two different approaches.

The first approach was to write a C code to implement the custom functions. The two functions implemented were DirTrav and ownps. DirTrav works very similar to the built in ls command. Along with the path it gives details of what type of of files exist in the directory given, and also gives the nesting level.

Enhancing functionality of ps in ownps:

● Using the -z option, lists all the zombie processes

● Using the -cpu option, lists the top 10 cpu consuming processes.

● Using -memry option, lists the top 10 memory consuming processes.

● Using -ph option, gives the process hierarchy for the current process.

The second approach for the same could be by writing a shell script and creating a custom function by combining multiple existing Linux commands. We implemented two such commands: sysinfo and fcnt.

sysinfo command can be used to get different system information like the disk usage, free disk space etc.

fcnt command can be used to get the count of different types of files in the system. For example number of c files, shell scripts, java files, text files etc.