**Goal:** To implement a Linux shell with few added functionalities

**INTRODUCTION**

Shell is a UNIX term for the interactive user interface with an operating system. It is a 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.

**PHASE 1: Reading the user input, parsing and executing the command**

In this phase, the shell is initialized by the function **init\_shell()**. A child process is forked and replaced(with the input command) by an execvp system call.

On exec(), the basic Linux built-in commands like cd, ls, ps, cp, mv and exit are executed. Thus, the shell takes in these commands as input, performs the parsing and executes the required command.

**PHASE 2: Piping, History Feature, Editor and Aliasing**

**1. Piping and I/O Redirection**

Pipes are detected using strsep(“|”). The command is separated into two parts (before and after the pipe). After parsing each part, both parts are executed as two separated children, using execvp. The output of the first command is passed as input to the second one

File descriptors are stored in an integer array (0 to read and 1 to write). The pipe is opened through pipe(), and two children are created.

In child 1, fd1 is copied to stdout, and the first command is executed

In child 2, the input from pipe is taken, and fd0 copied to stdin. The second command is executed

**2. History Feature**

It is a command used to view the list of commands previously executed. On every command execution, the command is stored into the buffer through add\_history() of readline/history.h library. On execution of history() command the buffer is printed

The history buffer can be viewed as a Circular Buffer, where it wraps around when Upper bound is reached. Each new entry is added to the history buffer, which eliminates the oldest entry in the buffer if the buffer is full. The current history pointer is maintained for faster storage of new commands. The length of the command is not known in prior, and hence allocated dynamically. Any allocated command gets freed when it no longer exists in the buffer.

**3. Editor**

A text editor is implemented, which has the features to create or delete a file, append to a file, and display the file contents. File descriptors and C file library functions - fopen, fclose, feof (to test the EOF indicator for a given stream) are used to implement the Editor.

**4. Aliasing**

Alias command is an alternative name used for long strings that are frequently used. It is helpful for creating a simple name to replace a long command. Aliases on shell can be created using the command :

**alias name value**

where name is the alias name for the command and value is the actual name of the command.

The aliasing function takes an array of tokens and parses them as an executable program through fork() and execvp(). The input token is compared with the pre-defined internal commands and the appropriate action is performed depending on the alias command. For the given token , an alias name is created using createAlias() function.

**PHASE 3: Implementing Custom Functions**

In this phase, we implement two custom functions on the shell.

**1. lsz- Lists files of a directory that have a size of zero bytes**

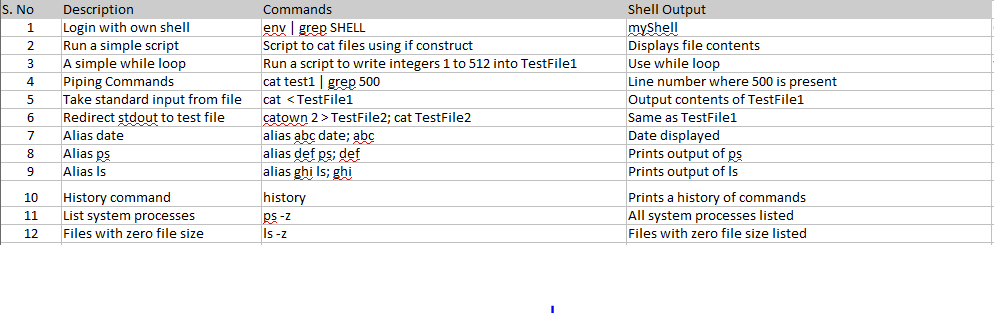
The directory, specified in the directory argument, is first opened using the function opendir(). The dirent structure is used to obtain file and directory properties. The directory is traversed, and contents are read through the function readdir(). If the type of the directory entry is a regular file (d\_type==DT\_REG), then the size of the file is read. The size is determined by the stat structure (st\_size attribute). If the size of the file is found to be zero, it is listed.

**2. myps - Listing out all the system processes**

Through this custom function, we list out all the system processes by reading the directory /proc. The directory /proc has a list of directories called “process directories” whose names are equal to process ids. Each pid directory consists a file named “comm” that has the command name associated with the process. We enter this path and display all the running processes with the following details:

* User\_id - Id of the user
* d\_name - directory name (pid) to which each command name belongs to
* Cpu\_usage - amount of cpu used for the process completion
* Memory\_usage - amount of memory used
* Vmsize - Virtual memory size used
* Memory\_rss - Memory occupied by a process held in main memory (RAM)
* State - The process status code
* Readbuf - Name of the process

**Test Cases**



**Final Outcomes**

We achieved building our shell through a sequence of 3 phases. Our shell can run the basic unix commands like ls, ps, cd, etc in addition to our own defined commands like help, exit, history. Piping command work in our shell. We have implemented a basic text editor where we can create files, add contents, append and also delete them. We also have the functionality of aliasing being incorporated. Lastly, we have defined 2 of our own custom functions to perform few tasks.