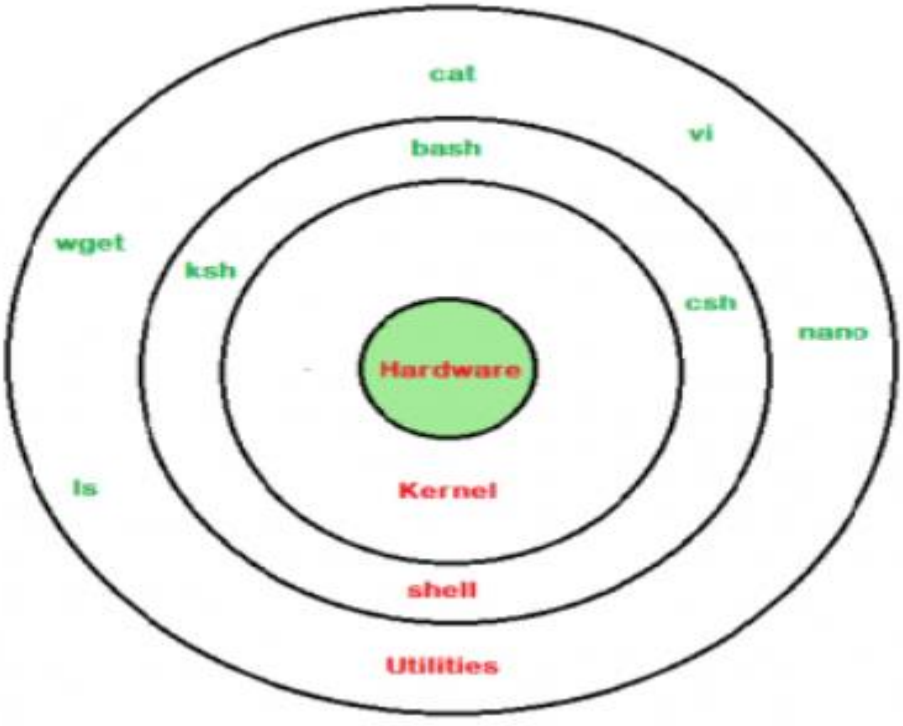
Description of shell build choices

I. Introduction :

if you are using any major operating system you are indirectly interacting to **shell**. If you are running Ubuntu, Linux Mint or any other Linux distribution, you are interacting to shell every time you use terminal.**so let's know what is a shell** ?

A shell is special user program which provide an interface to user to use operating system services. Shell accept human readable commands from user and convert them into something which kernel can understand. It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.



Shell is broadly classified into two categories :

- Command Line Shell
- Graphical shell

Description of shell build choices

II. Problem :

We all use the built in terminal window in Linux distributions like Ubuntu, Fedora, etc. But how do they actually work? In this article, We are going to handle some **under the hood** features and algorithms what actually work inside a shell. All Linux operating systems have a terminal window to write in commands. But how are they executed properly after they are entered?

Also, how are extra features like keeping the history of? All of this can be understood by creating your own shell.

III. solution :

In Our 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. Implementing output redirection
9. Can execute filename having suitable execute permissions in the current working directory using ./filename.
10. Can change the current working directory of the Shell using cd command.
11. Pipe | Operator for redirecting commands to another command.
12. Redirection > Operators for output redirection from / to a file respectively.

Description of shell build choices

IV. Choice of data structure :

My code uses several data structures and algorithms in order to implement a basic shell.

1. **Arrays**: The history array is used to store the last 10 commands entered by the user. The commands array is used to store the individual tokens of the command entered by the user.
2. **Strings:** The command variable is a string that stores the command entered by the user. The cwd variable is a string that stores the current working directory.
3. **readline library:** The readline library is used to take input from the user in a more user-friendly manner. It provides features such as line editing and history recall.
4. **Tokenizing:** The strtok function is used to tokenize the command entered by the user. It is used to split the command into its individual components, so that they can be processed individually.
5. **forking:** The fork() function is used to create a new process, and is used to execute the command entered by the user.
6. **execvp:** The execvp function is used to execute the command entered by the user.
7. **wait():** The wait() function is used to wait for the child process to complete execution.
8. **if-else statements:** The if-else statements are used to check for various conditions, such as whether the command entered is cd, history or quit.
9. **while loops:** The while loops are used to iterate through the commands array and the history array.
10. **modulus operator:** The modulus operator is used in the add_to_history function to ensure that the history array only stores the last 10 commands.

Overall, my code uses a combination of data structures and algorithms to provide a basic shell-like interface that allows users to navigate the file system, execute commands, and recall past commands.

Description of shell build choices

We need to use the appropriate system calls :

The **fork()** system call in a Unix-based operating system is used to create a new process by duplicating the calling process (the parent process). The new process, referred to as the child process, is an exact copy of the parent process with its own unique process ID.

The **wait()** system call is used by a parent process to wait for one of its child processes to terminate. When a child process terminates, the parent process receives a signal indicating that the child has exited and the parent can retrieve the child's exit status using the wait() system call.

The **execute()** system call is used to start a new program in a process. It replaces the current program in the process memory with a new program and starts the new program's execution. This is typically done by a child process after it has been created by the parent process using

V. choice of algorithm :

- **void init_shell()** : Greeting shell during startup
- **int takeInput(char* str)** : Function to take input
- **void printDir()** : Function to print Current Directory.
- **void executeCommand (char* command) :** function responsible for taking in a command, parsing it into an array of sub-commands, and then executing each sub-command. It also checks for the "cd" and «quit » commands and attempts to change the current working directory accordingly. It also handles history commands.
- It also appears that your function doesn't handle commands that include operators such as "&&", "||", ";", or "|".
- **runBatchFile(char* fileName) :** execute commands in batch mode
- **Main** : execute the previous commands.

Description of shell build choices

The Basics :

- Getting user name can be done by **getenv("USER")**
- Printing the directory can be done using **getcwd**.
- Parsing can be done by using **strsep("")**. It will separate words based on spaces. Always skip words with zero length to avoid storing of extra spaces.
- After parsing, check the list of built-in commands, and if present, execute it. If not, execute it as a system command. To check for built-in commands, store the commands in an array of character pointers, and compare all with **strcmp()**.
  Note: "cd" does not work natively using execvp, so it is a built-in command, executed with **chdir()**.
- For executing a system command, a new child will be created and then by using the execvp, execute the command, and wait until it is finished.
- Detecting pipes can also be done by using **strsep("|")**.To handle pipes, first separate the first part of the command from the second part. Then after parsing each part, call both parts in two separate new children, using execvp. Piping means passing the output of first command as the input of second command.
- For keeping history of commands, recovering historyusing **add_history()** and **strcpy(),** we will be using the readline library provided by GNU
- When the notation >filename is added to the end of a command, the output of the command is written to the specified file name. The > symbol is known as the output redirection operator.

- The output of a process can be redirected to a file by typing the command followed by the output redirection operator and file name. Using **dup2()** and **open().**
- Executing bash file by using **./myshell filename.** And to handling errors if bash file or Inadequate number of arguments.