***Abstract—This unix shell that is designed in C language will execute all external and some internal commands like “cd” “help” “jobs” and “exit” etc. It will also handel pipes and background and foreground jobs*.**

# INTRODUCTION

This document is about designing and implementing a simple, interactive shell program that prompts the user for a command, parses the command, and then executes it with a child process. In this solution we are using execvp() instead of execv(), which means that we have no need for the PATH environment, then search each directory in the PATH for the command file name that appears on the command line. This shell is also capable of doing background and foreground jobs. Here we are also handling pipes.

# BACKGROUND

Shell command line interpreter is an application program that uses the system call interface to implement the operation. Export the character-oriented human-computer interface and use the system call interface to call operating system functions. Each shell has its own language syntax and semantics. The executed command is the first word on the command line, and the remaining words are the parameters expected by the command. The number of parameters depends on the command being executed. Shell relies on an important convention to complete its work: the command is usually the name of the file that contains the executable program. For example, ls and cc are file names. In some cases, the command is not a file name, but a command implemented in the Shell. For example, cd is usually implemented in a shell rather than a file. Since most commands are implemented in files.

# RELATED WORK

These are the special steps that we observe to layout a shell.

1. ***Printing a prompt***

When the shell starts, it will look up the name of the computer and the name of the user running under it, and append it to the standard prompt characters. In addition, the shell program can also print the current directory at the command prompt. This means that every time the user uses cd, the query string will be redefined. After defining the query string, when it is ready to accept the command line, the shell prints it to standard output.

1. ***Getting the command line***

To get a command line, the shell plays a blocking off examine operation in order that the method/process that executes the shell could be blocked till the person types a command line in reaction to the prompt. When the command has been supplied by the person and that command terminated with a ‘\0’ character, the command line string is back to the shell and shell will start work on that input that supplied by the person

1. ***Command parsing***

As user input will consist of command as well as argument so at this stage we should need to parse these arguments from the original string or input that enter from the user. Because input string can also consist of pipe so in this step we should also need to check is user input consist of pipe or not.

1. ***Finding the file***

As we are not using execv() function instead of this function we are using execvp() function that requires just the command name and argument of that command so we should not need to find a path name. System calls like exec enable the process to run any program file, including executable binary files or shell scripts. Only we will discuss one system call from all of these that is execvp() system call. The execvp() system call takes two parameters; the first parameter is a string containing the name of the file to be executed. The second parameter is a pointer to an array of strings. More precisely, its type is char \*\*, which corresponds exactly to the argv array used. In the main program.

**e.g) int main (int argc, char \*\* argv)**

Please note that this parameter must be terminated with null.

1. ***Execute the command***

As when we try to use the exec family system command in our original process it will change our current process with the exec family process or command. So to handle this process we will use a fork() system call to create a child and parent process to execute the exec family function.

1. ***Execute internal commands***

For the external command like “cd” we are using chdir() system call that will change our directory. It consists of just one parameter that consists of a path name. If we pass “..” this as a string to this function it will move back one directory and vice versa. When the user “Exit” command enters it will simply terminate our infinite while loop by calling system call exit(0). If we enter “job” it will check our job list whose status is continued. It will simply print those job id, table number, status of job, and job name*.*

# THE PURPOSED WORK (THE GROUP WORK)

# 

# EXPERIMENTS

1. **Filename completion and command history**

We were thinking to store a history hashing table as it was the easiest way plus to store data due to accessibility,time complexity and as well as for the space complexity. I decide to control arrow key for this purpose for example when user will hit enter i should need to enter user input string in hash table in the same way when user hit upper arrow key i should need to bring last data from hash table that I had recently enter in hash table or if user hit “down key” it will bring the previous data in this way i can move to and fro in the history but I was not able to handle left, right, and TAB key.

we thought to use some built in function that c language provides us to take input from the user and provide ability to the user to edit commands on run time and for storing the history we are also using built in function for storing the history.

1. **Decision for exec family function**

There are 2 system functions that belong from exec family can done our job easily that are [execv() and execvp()] now the need was to decide which function we need to use for our job because execv() function will take the complete path to execute the external command and we now environmental path can be update at the runtime so if we try to use execv() function we should need to create a function that will get environmental path from the system and parse it in infinite while loop that will increase the time complexity. So that's why we decided to use execvp() function because it will take just command name or file name to execute with their parameters.

**Syntax of execv():**

**int execv(const char \*path, \*\*char const argument);**

**Syntax of execvp():**

**int execvp(const char \*fileName, const char \*\*argument);**

# CONCLUSION

This is a shell that provides the function of all other shell except some internal command like “bg”, “ll”, “fg” and many other only those internal command that supported by our shell are “exit”, “help”, ”jobs”, and ”cd”.

# REFERENCES

[1]<https://man7.org/linux/man-pages/man3/readline.3.html#:~:text=readline%20will%20read%20a%20line,text%20of%20the%20line%20remains>.

[2]<https://www.geeksforgeeks.org/exec-family-of-functions-in-c/>

[3]<https://www.geeksforgeeks.org/chdir-in-c-language-with-examples/#:~:text=The%20chdir%20command%20is%20a,the%20directory%20specified%20in%20path>.

[4]<https://www.educative.io/edpresso/what-are-linux-signals>

[5]<https://www.tecmint.com/run-linux-command-process-in-background-detach-process/#:~:text=How%20to%20Start%20a%20Linux,background%20jobs%20by%20typing%20jobs%20>.