**OPERATING SYSTEM**

# **Assignment - 2**

## ***SimpleShell : A Unix Shell in C from Scratch***

**Shell :**

A simple shell or command-line shell is a basic command-line interface program that allows user to interact with the operating system by executes commands.

Shell provides a way for user to control and manage their computers and laptop.

*Charactersitics of Shell* :

* Command Execution
* Pipelines
* Prompt
* Scripting
* History
* Redirection

Basically the shell i implemented carries a little of all above characteristics.

**Implementations :**

* Headers, constants & global variables**:  
  **
* **Main :**From main i call *void shell\_loop()* :
  + void shell\_loop():

Firstly it prints the promt and reads the stdin using char \*read\_user\_inputs(): and then checks for the enter command is valid or same as the previous or not it checks for the exit input i.e. terminating the shell, history i.e what commands we entered and also a bonus part 2 file input all in a do-while loop.  
All the commands provided by the user will run by calling launch that is also in a do-while loop in some of the above conditions. Here, we checked the needed erros and allocations.

* + char\* read\_user\_input():

It basically reads the user input. First we allocated the memory and size of the input using malloc, then we use fgets() to get the user input. It returns input.  
Here, also we checked for the errors and deallocation.

* + int launch(char \*command):  
    Basically all the commands provided by the user will run by calling launch.  
    In launch() we have our create\_process\_and\_run() which is responsible for running all the commands.
  + int create\_process\_and\_run(char \*command):

In this function we firstly checked for the pipe by using strtok and if pipe if present in the command we basically split that command and store it in a array.

We implemented this function only for 3 types of inputs :

→ inputs with 0 pipes i.e. simple commands like ls,echo,etc:

for implementation of this for and in the child we used execvp system call in the execvp we just simply passed commands and checked for the error and in parents we waited for the child to finish so we can add that to history and other things also

→ inputs with 1 pipes i.e. with two commands like cat t.txt | head :

for implementation of this we used pipe concepts as we know about the pipes

Pipes have two faces one is for write and other is for read so basically we

used this concept in fork. Here we gonna use 2 forks and 2 pipes i.e 2 childs and 2 parents process. In the first child we used the pipe concept i.e read closed, dup2() etc. we executed the first command using execlp() system call and stores the output in the pipe1 i.e write and in first parent we again waited for the child to finish and other things also then we created other child process in that parent process so we can used the pipe1 data now, in the second child we used the pipe2 write part to takes as an input from the pipe1 data i.e read part and in this child process we used that data and executed the second command using execlp system call() and stores the final data i.e output in the buffer and print it to the console using write() call. Here also we checked for the necessary error handling,deallocations etc.

→ inputs with 2 pipes i.e. with three commands like cat t.txt | head | sort :

for implementation of this we used the same above concept but with 3 forks and 3 pipes i.e with 3 childrens and 3 parents and used execlp for the execution.

* + void add\_to\_history():  
    we used this function to add the commands to an array named history, in this function we also done start,end time, pid etc this this function basically adds all the details of a process i.e command into the array and when we need it just we call that array.  
    We have done all the necessary error handling and allocation deallocations,etc.
  + void display\_history():

It just displays the all previous commands that we entered in the shell.

* + Here we done one thing also that when we enter exit it prints all the commands pid, execution,start,end timings. As we already stored it in the history array
  + We have done the bonus part2 for which we created a launch\_script() function in which we just takes the file as a input and read that file line by line and executing it commands along with it simply by calling launch.
* **Limitations** :

→ cd : The cd command cannot be executed directly within the shell. When you use cd it only affects the child process. When child process remains in the same directory. Or we can say that the cd command is not present int the execvp or execlp system call.

→ .jobs command : The jobs command is inbuild shell commands which means it is handled by the shell itself rather than being an external executable like other commands.

Group 17:

Prince - > main, shell\_loop(), launch\_script(), create\_process\_and\_run() i.e pipes and fork,get\_current\_time(), add executions details on exit,

Prajil -> add\_to\_history() function that adds history ,display\_history() that display history ,launch() all function are called by launch ,error handling, allocation dealloaction, headers helped in bonus part

Github link : <https://github.com/Prince22378/OS-Assignments-2023/tree/main/Assignment-2>