**Koc University**

**Spring2020**

**COMP304 – Operating Systems**

**Project 1: Shellgibi**

Group Members:

Arda Duman 60609

Elhan Oğuz 59898

**Project Aim:**

In this project, we developed an interactive Unix-style operating system shell. Which is divided into 3 main parts.

**Part I:**

In this part, our shell needs to run common commands with execv() system call. Our project does this as firstly find the path of the command by its name and after that execute the command with found path by execv() system call. This code part can be seen in the code between lines 625-634.

**A screen shot of a social media post

Description automatically generated**

Figure : Part 1 sample run

**Part II:**

In this part, our shell needs to support I/O redirection and program piping. If the redirection character is “>” then the shell creates an output file or if that output file exists, then the shell changes its information using the O\_TRUNC flag. If the redirection character is “>>” then the shell creates and output file or if that output file exists, append the given information to the one inside the file using the O\_APPEND flag. Lastly, if the redirection character is “<”, the shell redirects the given input to another command. Related code part can be seen between lines 594-617.

A close up of a person

Description automatically generated

Figure :Part2 I/O Direction sample run

For the piping part, all pipe actions are done by runPipe method which is a recursive method that uses pipes until it reaches the last command. This method firstly checks if we reached the last command or not for ending the recursive calls. If we are at the last command created file descriptor is used as an input and last command will be completed. If we are not at the last command, this function runs recursively when the child process finishes execution of runPipe(). Using pipes related codes are in the lines that 674,707-708 which are using dup2() calls. All method is between the lines 670-734.

A picture containing bird

Description automatically generated

Figure : Part2 Piping sample run

**Part III:**

For the last part of the project, we were given to implement a few new commands.

1. **Auto Complete:** This method works when the user pressed “Tab” button while typing a command. We searched through to all paths defined in environment with given word when tab is pressed. If we got only one proper match, this matched command will run immediately. If there are more than one possible commands in the list, it lists all the matching commands. Otherwise, if the command name is fully written the current directory is listed on the screen. Related codes are in lines between 508-591.**A screenshot of a social media post

   Description automatically generated**

Figure : Part 3 AutoComplete sample run

1. **Job Management:** This system contains 4 different commands. myjob command lets user to see their list of processes owned with their pid, program name and status (Code lines 385-393). pause command gets pid of a process and suspends it (Code lines 394-401). mybg command gets pid of a paused process and puts it to background at running state (Code lines 402-409). myfg command gets pid of a paused process and puts it to foreground at running state. (Code lines 410-418).**A close up of text on a white background

   Description automatically generated**

Figure 5 myjobs sample run

1. **Alarm:** This command takes a time input and a .wav file input and plays the file at the designated time. (Code lines 421-455)
2. **Custom Commands:** We have 3 custom commands for this project.
3. **History:** This command lists last used 10 commands of the user. (Code lines 457-467).
4. **Wait:** This command lets the user to wait for a specified time in seconds. (Code lines 468-477)
5. **Lshome:** This method lets the user to list everything in the home folder whereever the user is. (Code lines 479-485)

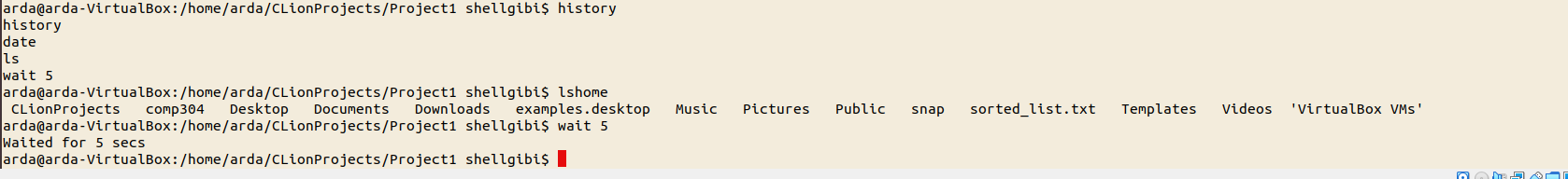
****

Figure 6 Custom Commands sample run