COMP 304 PROJECT 1

Bora Berke Şahin - bsahin17@ku.edu.tr - 64060
Büşra Işık - bisik18@ku.edu.tr - 69016

Link to Repository: https://github.com/boraberke/comp304-fall22-project1

Part 1:

In this part, we replaced execvp() with execv(). In order to be able to do this we need to find the paths of the built-in commands or add the path of the programs that are in the current directory. We used the environment variable for path and access() to see if the built-in command exists in any of those paths. We also checked if the user were trying to run a program within the directory and looked for the program there. On top of that, to process the background processes we just made the parent wait for that specific child process to terminate.

Part 2:

In this part, we have implemented I/O redirection for 4 different character commands. For '<', we opened the input file in readonly mode and using dup2, changed the stdin to that specific file. For '>' and '>>', we created/opened the output file and using dup2, changed the stdout to that specific file.

Following that, we created pipe command where we connect stdout of the previous command to stdin of the next command recursively. In the first command, we did not change the stdin, and only changed the stdout. For each command in between two commands, we have changed both stdin and stdout. And in the last command, we only changed the stdin, but kept stdout as it is.

Part 3:

A. uniq

This command returns every unique line from an input. In the implementation of this command, we have used a new struct uniq_t to keep the data as well as the next element. Instead of reallocating for each line, we have created a linked list. As the lines are in an alphabetical order, we only check the current line with the previous line. If they are the same, then we only increase the count of the previous line. However, if they are different, then we create a new line from the current line and continue to the process by assigning prev_line to current_line. If the user also asks the count, then we return the count as well.

B. chatroom

In the implementation of chatroom, we used named fifos under /tmp/ folder. If the given username or the roomname does not exist, then we create the named fifos and folders. If they are already created, then for each user in each room, whenever a user sends a message, we

send that message to each and every user's named fifo. Moreover, for each user, we read from their respective fifo to get the messages from other users.

C. wiseman

In this command, we parsed the argument given by the user to a text file in the correct format for crontab to execute fortune and espeak at every time interval specified. Then, we passed that file to the user crontab file. Thus, once the command is called, there will be a quote voiced over every given interval until the crontab file is cleared.

D. Bora's Custom Command: snake

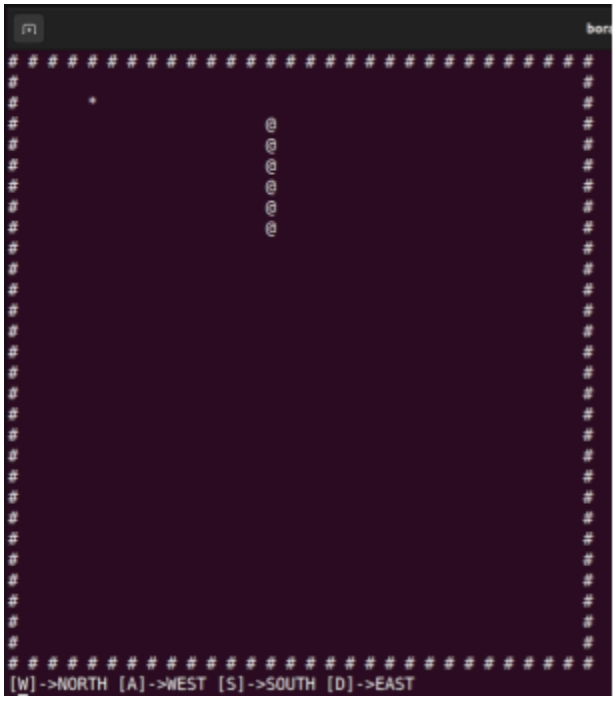
In this custom command, I have created a classic snake game where the player can specify the speed of the game and the frame size of the game. After specifying those two inputs, a game frame will be created and the user can play the game by collecting the food and try not to hit to the walls of the frame.

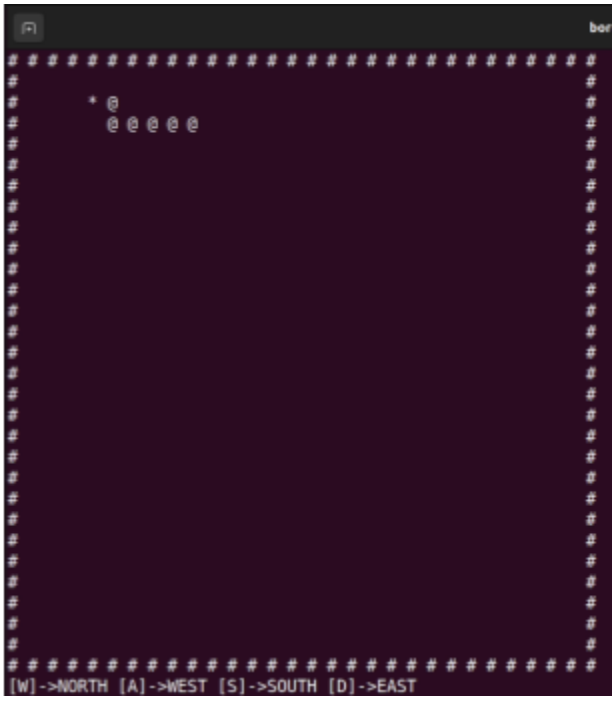
This custom command is written in python and the source code of it is in our github repository under snake.py file. Then using Pyinstaller, I have created a binary file which is runnable on linux.

```
usage: snake [-h] [--speed {1,2,3,4,5,6,7,8,9,10}]
             [--frame {small,medium,large}]

A simple and fun classic snake game!

optional arguments:
  -h, --help            show this help message and exit
  --speed {1,2,3,4,5,6,7,8,9,10}
                        the speed of the game. Between 1 and 10
  --frame {small,medium,large}
                        Size of the frame. small, medium, or big.
```





E. Büşra's Custom Command: dance

I created a command where you can have a cute small dance animation of your choice. There are three different choices of dances and you can also specify how many times you want the animation to be played with the second input.

Part 4:

The user gives a PID and we iterate over the children of the process and their children and so on on the kernel module we created. We printed the PIDs of the of all the processes and their start times on to the kernel log. We did these in a certain format so that we can later make use of graphviz tool to create the graph image for all the processes. On the main source code we read these printed out messages using dmesg command and pass them into a text file. Then we pass that text file into the graphviz tool and it outputs the graph image with the name that was given as input by the user.