COMP304 PROJECT 1-SEASHELL

Berke Can Rizai 69282 – Eren Barış Bostancı 68770

In the 1st part we changed the execvp() method for executing Linux programs to execv() system call which also needs the path of the binary that will be executed. We used /bin/ as a default path and added the name of the command end of the path to execute the binary.

```
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ ls
berke.txt    eren.txt      seashell.c  shortdir_memory.txt  t.txt
command.txt  ringtone.mp3  shell       tmp.txt
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$
```

In the 2nd part we have implemented shortdir command for storing the current directory for jump directly back in the future. We used a txt file which called shortdir_memory.txt for storing the set values. This file located in the same directory with the executable seashell binary. When user input the “shortdir set matlab”, the current directory saved with a matlab name, if shortdir_memory file is non existing, one will be created and if there is already a matlab command it will be overwritten. Whenever user want to change current directory to the previously set one, he/she can use shortdir jump matlab command. We used chdir() method for changing directory. Also, when user want to delete the shortdir, can use shortdir del matlab. We used additional file for removing specific shortdir which called tmp.txt. First, we copy every data to that file except the one that we remove. Then, clear the memory and put the copied data back into it. Additionally, there are shortdir list command which prints the currently saved shortdirs and shortdir clear for removing all previously saved shortdirs. For list command we just iterate the data in the memory and print them one by one. And for the clear command we just open shortdir_memory.txt with “w” flag, which removes everything stored, then closes the file, resulting in a clean file.

```
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ shortdir set test
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ shortdir list
test -> /media/sf_COMP304/PROJE

eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ cd ..
eren@eren-VirtualBox:/media/sf_COMP304 seashell$ shortdir set t2
eren@eren-VirtualBox:/media/sf_COMP304 seashell$ shortdir list
test -> /media/sf_COMP304/PROJE

t2 -> /media/sf_COMP304

eren@eren-VirtualBox:/media/sf_COMP304 seashell$ shortdir jump test
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ shortdir del test
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ shortdir list
t2 -> /media/sf_COMP304

eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ shortdir clear
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ shortdir list
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$
```

In the 3rd part we implemented highlight command which takes a word, one of the color characters (r, b, g) and file name as arguments. We iterate the file word by word and print them. If the word and the iterating token is a match, we print that word with the specified color. Firstly, we lowered case the given word as an argument and compare with each word which are also transformed to low case version if it matches, we print the original version of the iterating word. With this implementation we can eliminate the case sensitivity on the comparison. For example, if the user gives “book” as an argument than all version of the “book” will be colored like “Book”, “bOok”, “book”, “BOok”…

```
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ highlight book r t.txt
this is a test file book
book Book bOok BOokil
sfsdfas another testBook
testttttt book bookile _book bo_ok book_ bo ok
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$
```

In the 4th part If the user inputs “goodMorning” to shell, we open up a new txt file called command.txt, here we are going to put our command that will play a song in specific time (ringtone) according to the user’s input. First argument of the user’s input is divided into two parts with string tokenizer with token “.” Here we are able to divide the time into hour and minute since the crontab requires we put them divided with space. First we put minutes part into the file than a “ ” and hour. After that we put " * * * XDG_RUNTIME_DIR=/run/user/$(id -u) rhythmbox-client " to the file. In this string * * * means current day month and year, since we want to set the alarm for every day, we could specify the date if we wanted such as each Saturday etc. but it is not required. “XDG_RUNTIME_DIR=/run/user/$(id -u)” is necessary for ringtone to play, without it the command does not work. Lastly, we have rhythmbox-client which creates an instance of the rhytmbox program. Then we put the name of the mp3 file user specified in args and lastly the –play command that tells the rhtymbox to play that file. And we close the command file. We then, create another argument called argss and we allocate memory for it with malloc. As the first parameter we put “crontab”, second is our command files name “command.txt” and last one is null. With the execvp we call this argss argument with “crontab” specifier and program executes with the input accordingly to command.txt. When the time comes, rhtymbox app will open and song will play.

In the 5th part, we have implemented kdiff command which have two mods which one of them is comparing to two txt file line by line and the other one for the comparing any file byte by byte. For deciding the comparison mod, we used -a and -b arguments. For -a flag, we compare two txt file line by line and if the lines are different, we print both of their different lines and at the end we print the total count of different lines. If there is no -a or -b it will use -a by default.

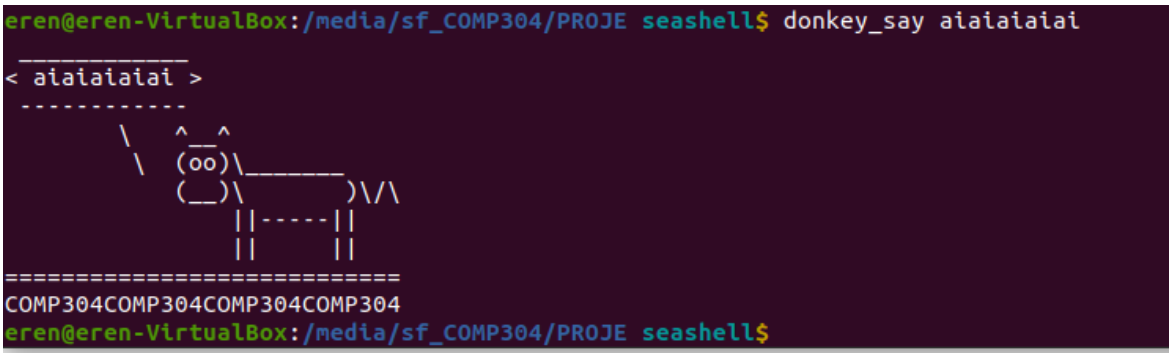
```
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ kdiff -a t.txt y.txt
t.txt:Line 2: book Book bOok BOokil

y.txt:Line 2: dgasfgasf

1 different lines found
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ kdiff -a t.txt same.txt
The two files are identical
```

For the -b flag, firstly we created two flags (i and j) that look if two input files are txt files for the a part. If command is -a we open both of the files in read mode and with line1, line2 strings. For the b part, we open both files without checking the file extension with rb which means read in bits mode. We call compare_bytes function. In this function we have unsigned long that holds current position. With getc, we get each byte one by one on both files until we hit EOF in first file or these bytes are different. If c1 is EOF and c2 is not then these two files are different, if they are both EOF they are same. If they are not same, we move on until we find EOF on both of them and since the difference of bytes is the size of biggest of these two files – position we make the calculation and print. We also print the position first difference is encountered.

```
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ kdiff -b t.txt y.txt
The two files are different in 36 bytes
file1 and file2 differ at position 26: 0x62 <> 0x64
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ kdiff -b t.txt same.txt
The two files are identical
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$ kdiff -b t.txt ringtone.mp3
The two files are different in 404720 bytes
file1 and file2 differ at position 0: 0x74 <> 0x49
eren@eren-VirtualBox:/media/sf_COMP304/PROJE seashell$
```

In the 6th part we have done two fun little projects to be used in our shell. First one, which is quite simple, takes a word as an argument and in the shell "paints" a donkey saying the word that was passed into the function as argument which is argv[0]. If donkey_say command is entered the

parameter is taken and printed text on the shell resembles a donkey and we can see it "saying" the input. This was implemented with printfs and is quite simple yet fun.



The second, one which took unnecessary amount of time for us to complete, is a hi-low guessing game and it is played as; Computer prompts a number randomly between 0-10 and you need to guess if the next random number is going to be higher or lower than the current one. If you manage to guess correctly you win 2 coins and if you can't guess correctly, you lose 2 coins. Your beggining balance is 50 and you progress. If you type quit, it leaves the game. Also, if you want to change the bet you need to type "change bet" then shell prompts a field, and you type your desired bet. Then all the plays you make are on the bet you entered, you win or lose that amount. For example, say we have 10 coins and current number is 7, now logically, we might think that since chances are that next number is going to be most likely lower than 7, we type low. If it is lower than 7, we gain 2 coins and now we have 12 coins. If it is same as the last drawn number since in gambling there is the rule "House always wins", player loses 2 coins.

In our implementation, we have char array (string) as input and while we are in the game loop with scanf("%s", input) we take the input from the shell. If change bet command is entered, we change the int bet which is our bet amount. Our random logic is with the srand(time(NULL)); which returns a random number, which is time, and with the rand() % 10 + 1; we make sure our number is between 0 and 10. In each loop our old int toCompare is made as compared and compared is assigned a new random number. In the next loop if new number (compared) is higher than old number(toCompare) and user input was "high" user wins and if it was lower and user again guessed right as "low" again user wins else user loses the bet amount. These bets are either added or deducted from the user's balance. If balance is lower than or equals to 0 than user lost the game and prompt "bankrupt" appears, game closes. If input is "exit" game is also closes whenever player desires.