Nursena Köprücü – 59843

Mehmet Enes Erciyes – 68906

April 4, 2021

COMP 304 - seashell - Your Custom Shell: Project 1 Report

Build Instructions

We used *CMake* to build our project. Follow these steps to build and run the code:

1. Extract the code.
2. Move into the folder.
3. Run the following commands to build:

   mkdir build

   cd build

   cmake ..

   cmake --build . --target p1

4. Run the executable:

   ./p1

IMPORTANT: In the CMakeLists.txt, version 3.10 is tagged as a minimum.

Part I

We implemented *execv* by first getting the PATH environment variable and looping over every directory separated by colons to find the command. We check whether the file is found, if so and if it is executable, then we call *execv*.

Part II

We implemented the *shortdir* command in this part. Here, we associate short names with the current directory. So then we can reach that directory by typing that short name without typing the whole path. We defined a new struct *shortdir* which contains alias and path variables to represent *shortdir* associations. And we keep a global FILE variable that points to the *shortdir* file. To implement all supportive options for the *shortdir*, we used hashmap which has an open-source code. [1] This global hashmap pointer holds shortdir associations in the memory.

IMPORTANT: Our program keeps shortdir associations and reads from /tmp/ directory. Our code is written and tested in Ubuntu 18.04. It may fail in other Linux distros or versions. If it does, please change this to an absolute path that does not require sudo privileges.

- *shortdir set name:* we defined *add_shortdir_alias* and used the *hashmap_set* function to associates the name with the current directory. We also defined the *save_shortdir_aliases* function which scans the *shortdir_map* (our hashmap) to save each association. So, each time we add a new *shortdir*, we save the association in the file.
- *shortdir jump name:* we implemented this one by first looking up the hashmap with the name as key and then using *chdir()* function to change the directory to the specified path.
- *shortdir del name:* we used the *hashmap_delete* function to remove the specified name-directory association.
- *shortdir clear:* we used the *hashmap_clear* function to delete all the name-directory associations.
- *shortdir list*: we called the *shortdir_list_iter* function (for each struct in the hashmap) inside the *hashmap_scan* function to list all the name-directory associations.

Part III

In this part, we implemented the highlight command. First, we set the color code according to the given argument. Then we read each line in the given text file. In the first pass we check whether the word exists, in the second pass we print out that line with the word as highlighted, if it exists.

Part IV

In this part, we implemented the goodMorning command which takes time and a music file as arguments. We created a crontab file because the crontab command allows us to run a scheduled task at the specified time. So, we write path information and the specified time into that crontab file, and we play the mp3 file by using *rhythmbox-client --play*.

Part V

In this part, we implemented *kdiff* utility to compare two files in two modes.

- *-a :* we defined the compare_txt_files function to read the given files and compare them line by line.
- *-b :* we defined the compare_bin_files function to read the given files and compare them byte by byte. We used *fseek*, and *ftell* functions to dynamically allocate just enough memory for the buffers we used.

Part VI

In this final part, we were supposed to implement a command of our choice in seashell. We came up with a new command *playCollatz* which is based on Collatz Conjecture. Here is the definition of Collatz Conjecture:

> *Let a_0 be an integer. Then one form of Collatz problem asks if iterating*
>
> $$a_n = \begin{cases} \frac{1}{2}\, a_{n-1} & \text{for } a_{n-1} \text{ even} \\ 3\, a_{n-1} + 1 & \text{for } a_{n-1} \text{ odd} \end{cases} \qquad [2]$$
>
> *always returns to 1 for positive a_0.*

Our custom command *playCollatz* takes two positive integers as arguments as player 1 and player 2 choices. Then, we print out the Collatz sequence for both numbers(players) where we have such conditions:

- The first term of the sequence is the given number by the user.
- To calculate the terms for the rest of the sequence:
  i) if the previous term is even, then take half of the previous term as the next term.
  ii) if the previous term is odd, then take 3*(previous term) + 1 as the next term.

Collatz Sequence guarantees that it always reaches 1 for given positive integers. In the end, the one who reaches the number 1 in fewer steps wins the game. Congratulations!

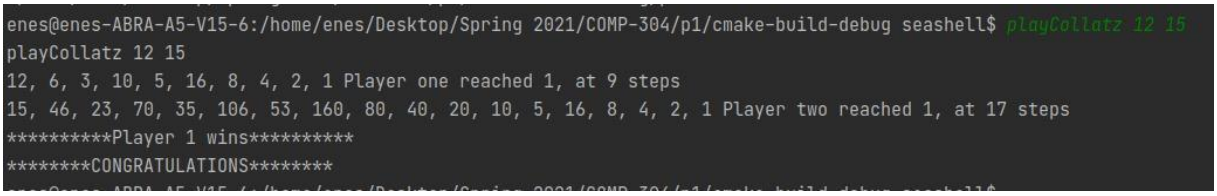

Figure 1: Sample usage of playCollatz Command

References

[1] https://github.com/tidwall/hashmap.c

[2] https://mathworld.wolfram.com/Sequence.html