Comp 304 — Project 1

Gul Sena Altintas[*]& İdil Defne Çekin [†]

April 4, 2021

Contents

Important Note: Our custom command for Part 6 uses `curl`, therefore it is required to compile our code as `gcc seashell.c -l curl -o seashell` this command is included in Makefile, so that the reader can run `make` in their terminal.

1. Part 1

1. Path variable of the machine is found and the locations in the variable is extracted to an array, `path_locations` in the beginning of the main function.
2. A function, `find_command_path(char *command_name, char *path)` is defined to iterate over the `path_locations` array, open each directory, search for the correct file name and copy the full path of the function to the `path` variable.
3. Then, `execv()` is called with the obtained absolute path and args.

2. Part 2

1. A new file named "shortdir_commands" is created if it does not already exists, to keep the shortcut names assigned to the directories.
2. The function, `handle_shortdir(struct command_t *command)` is defined to check the first argument of the command and execute the correct function.

[*]galtintas17@ku.edu.tr - 64284
[†]icekin17@ku.edu.tr - 64387

3. The function, `find_corresponding_location(char *command_name, char *location)` is defined to scan the "shortdir_commands" file and find the line index of the second argument `command->args[2]`, and set the variable `location` to its corresponding directory.

4. The function, `set_shortdir_command(char *command_name)` is defined to be executed when the first argument of the command is "set". This function opens the "shortdir_commands" file and adds the new shortcut to the end of the file. If the name entered is a duplicate, it deletes the previous entry using `delete_shortdir_command(char *command_name)` function, adds the shortcut to the file and prints a warning.

5. The function, `delete_shortdir_command(char *command_name)` is defined to be executed when the first argument of the command is "del". This function opens the "shortdir_commands" file, iterates over the lines and copies each line to another file that it creates during execution, except from the line that includes the name (args[2]) to be deleted. After the whole file is iterated over, deletes the old file and renames the new file as "shortdir_commands". Prints a warning if the name to be deleted does not exist as a shortcut.

6. The function, `find_corresponding_location(char *command_name, char *location)` is used to detect the location to be jumped when the first argument of the command is "jump". Then, the current directory is changed using `chdir` to where the shortdir directs to.

7. When the first argument of the command is "clear", "shortdir_commands" file is deleted and an empty file is opened with the same name.

8. The function, `list_shortdir_associations()` is defined to be executed when the first argument of the command is "list". This function iterates over the "shortdir_commands" file and prints the lines.

Outputs:

1. List:

```
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir list
Existing file associations:
lm /home/gsa/code/comp304/comp304/project1
l /home/gsa/code/comp304/comp304/project1
y /home/gsa/code/comp304/comp304
kkk /home/gsa/code/comp304/comp304/project1
```

2. Set:

```
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir set l
Setting
Replacing old command for shortdir l...
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir list
Existing file associations:
  lm /home/gsa/code/comp304/comp304/project1
  y /home/gsa/code/comp304/comp304
  kkk /home/gsa/code/comp304/comp304/project1
  l /home/gsa/code/comp304/comp304/project1
```

3. Jump:

```
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir jump y
/home/gsa/code/comp304/comp304
gsa@gsa:/home/gsa/code/comp304/comp304 seashell$
```

4. Clear:

```
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir list
Existing file associations:
  lm /home/gsa/code/comp304/comp304/project1
  y /home/gsa/code/comp304/comp304
  kkk /home/gsa/code/comp304/comp304/project1
  l /home/gsa/code/comp304/comp304/project1
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir clear
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir list
Existing file associations:
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$
```

5. Delete:

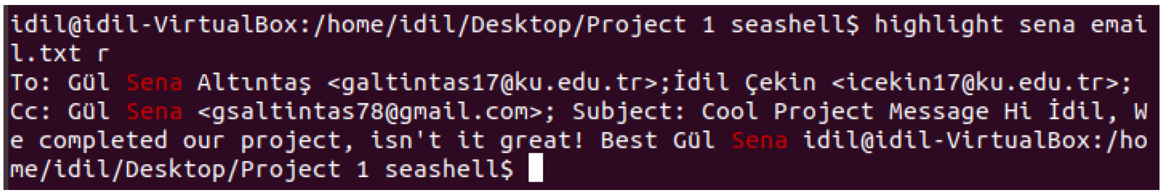
```
(base) gsa@gsa:~/code/comp304/comp304/project1$ ./seashell
gsa@\gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir list
Existing file associations:
  h /home/gsa/code/comp304/comp304/project1
  y /home/gsa/code/comp304/comp304/project1
  yy /home/gsa/code/comp304/comp304/project1
  o /home/gsa/code/comp304/comp304/project1
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir del y
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir list
Existing file associations:
  h /home/gsa/code/comp304/comp304/project1
  yy /home/gsa/code/comp304/comp304/project1
  o /home/gsa/code/comp304/comp304/project1

gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir list
Existing file associations:
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ shortdir del g
Shortdir command doesn't exit: g
```

3. Part 3

We learned how to add color to the output from here.

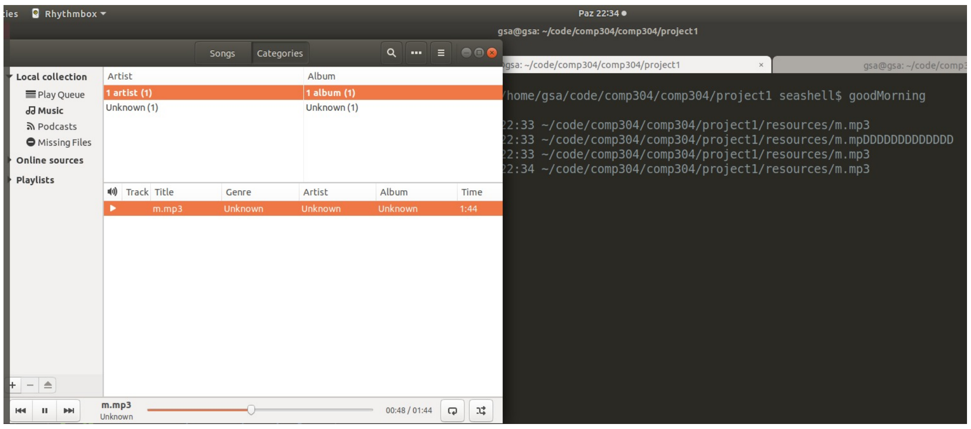
1. When the highlight command is called, first, the number of arguments in the command is verified and a warning is raised if it does not match, in process_command function.
2. Then, the third argument is checked for whether it is a valid color, and a warning is raised otherwise.
3. When everything is validated, `highlight(char *word, char *file_location, char *color)` is called to handle the process.
4. In the function, the file is opened, read with fscanf function and if the scanned word fits the first argument of the command (not case-sensitive), the word is printed with the specified color, else, it is printed regularly.
5. Finally when there are no words left to scan, file is closed and allocated memory is freed.



```
idil@idil-VirtualBox:/home/idil/Desktop/Project 1 seashell$ highlight sena emai
l.txt r
To: Gül Sena Altıntaş <galtintas17@ku.edu.tr>;İdil Çekin <icekin17@ku.edu.tr>;
Cc: Gül Sena <gsaltintas78@gmail.com>; Subject: Cool Project Message Hi İdil, W
e completed our project, isn't it great! Best Gül Sena idil@idil-VirtualBox:/ho
me/idil/Desktop/Project 1 seashell$
```

4. Part 4

1. When the goodMorning command is called, first the number of arguments is checked if it is valid, else a warning is raised.
2. Then, the hour and minute information are extracted from the first argument by tokenizing.
3. Then, `schedule_alarm(char *hour, char *minute, char *song)` is called to handle the process.
4. In the function, arguments are concatenated and written into a text file, as minute hour * * * XDG_RUNTIME_DIR=run/user/$(id -u) DISPLAY=:0.0 / usr/bin/rythmbox-client –play song
5. crontab is executed by `execvp("crontab", argv)`, where argv includes "crontab", name of the text file, NULL.



```
/home/gsa/code/comp304/comp304/project1 seashell$ goodMorning

22:33 ~/code/comp304/comp304/project1/resources/m.mp3
22:33 ~/code/comp304/comp304/project1/resources/m.mpDDDDDDDDDDDDD
22:33 ~/code/comp304/comp304/project1/resources/m.mp3
22:34 ~/code/comp304/comp304/project1/resources/m.mp3
```

5. Part 5

1. When the kdiff command is called, first the first argument is checked, if it is "-b" `kdiff_binary(char *f1, char *f2)` fuction is called; else if it is "-a" or not specified, `kdiff_all_lines(char *f1, char *f2)` is called to handle the process.
2. In binary comparison, files are opened in binary mode and read byte by byte by fgetc function. Both files are read simultaneously until one of them reaches EOF. Each byte is compared and different bytes are counted. The result is printed.
3. In line by line comparison, files are opened in read mode, and both files are read by getline function until one of the files end. In each iteration lines are compared and different lines are printed along with the number of lines that are different.
4. Files are closed at the end of both functions.

Kdiff for all lines:

```
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ cd resources
gsa@gsa:/home/gsa/code/comp304/comp304/project1/resources seashell$ kdiff -a xlang.txt
    lang_diff.txt
lang.txt:Line 0: The programming language used for this code is C.
lang_diff.txt:Line 0: The programming language used for this code should be C.
lang.txt:Line 1: The first three letters of English language are A B and C.
lang_diff.txt:Line 1: Python is one of the most commonly used programming languages.
2 different lines found.
gsa@gsa:/home/gsa/code/comp304/comp304/project1/resources seashell$ kdiff lang.txt
    lang_same.txt
The two files are identical.
```

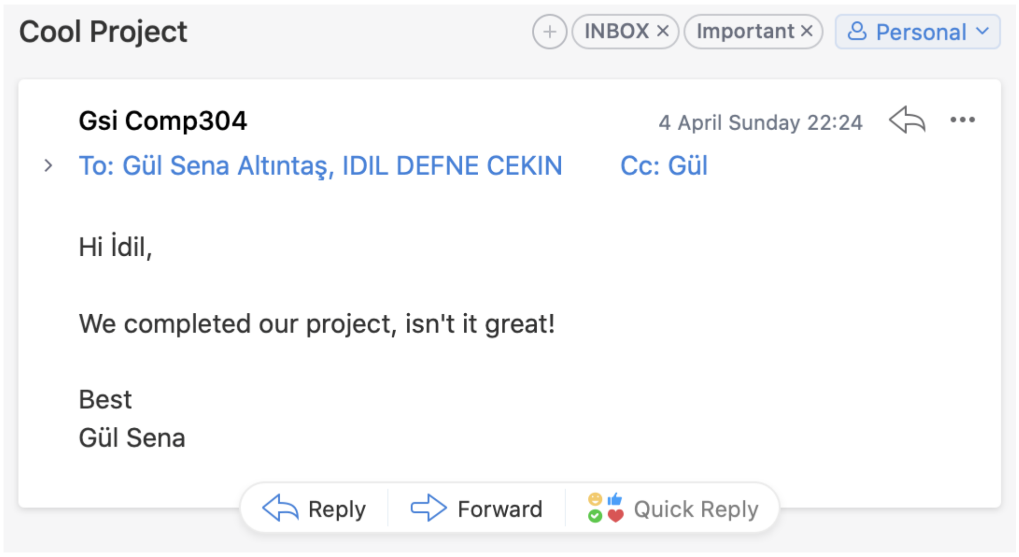
Kdiff byte by byte:

```
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ kdiff -b seashell s
956 bytes are different.
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ kdiff -b seashell seashell.c
4669 bytes are different.
gsa@gsa:/home/gsa/code/comp304/comp304/project1 seashell$ kdiff -b seashell seashell
Two files are identical.
```

6. Part 6

Our command: email

1. This command uses curl library to send an email, therefore seashell should be compiled as `gcc seashell.c -o seashell -l curl`. We learned how to use curl library from libcurl API.
2. There should exist two text files: user.txt including user name (the name from which the email will be sent from); email; password. email.txt including the "To:", "Cc:", "Subject:" and "Body:" in separate lines.
3. When email command is called, `send_email()` function is used to handle the process.
4. In the function, user information is kept in a user_email struct, that has 3 attributes name, email and password. Each line of the user.txt file is read and saved into a user_email struct and this struct is returned by `get_email_details()` function.
5. If the user information is not read successfully, the `get_email_details()` function raises a warning and returns null. If the read is succesfull, required curl structures are initialized and email.txt file is read in `get_payload(user, recipients` function. User is the user_email struct that includes the user information, and recipients is a curl_slist struct that will be used to keep the email addresses of the recipients.
6. In the `get_payload(user, recipients` function, email.txt file is read line by line, and parsed to create the payload_text in the required format, and fill the recipient list accordingly.
7. Payload_text is a global char array, that will keep the content of the mail. It should include the information in both txt files in the desired format to be used in the built in functions in the curl library.
8. After all information is read, curl is initialized, setup and the mail is sent.
9. **Discussion:** We are sending the email over a TLS connection. However, under an attack to the machine the credentials of the user may be compromised, as the password is printed as base64 encoded to the terminal. We use an external `user.txt` file to enclose user's credentials so that they are not included in the bash history. Advised users should use the command carefully.


Cool Project

Gsi Comp304 4 April Sunday 22:24

To: Gül Sena Altıntaş, IDIL DEFNE CEKIN Cc: Gül

Hi İdil,

We completed our project, isn't it great!

Best
Gül Sena

7. References

[1]s. C, A. Cainikovs and P. S, ”stdlib and colored output in C”, Stack Overflow, 2021. [Online]. Available: https://stackoverflow.com/a/3219471/11557945. [Accessed: 01- Apr- 2021]

[2]”libcurl - API”, Curl.se, 2021. [Online]. Available: https://curl.se/libcurl/c. [Accessed: 01- Apr- 2021]