**COMP 304 PROJECT 1**

**Our Github Repository:** https://github.com/edemirbas17/COMP304\_Project\_1.git

**PART 1 – BASIC COMMANDS:**

In question 1 we are asked to execute basic UNIX commands such as ls, mkdir, cp, etc. To execute these commands, we used fork and exec. We used fork to not exit shellfire when exec is called and make the child process call exec. In exec we added bin/ in front of these commands since these basic UNIX commands are located at bin directory. If the user adds a “&” sign at the end of their command user wants the command to run in background therefore after the fork the parent process does not wait for child process’s exec call to end.

**PART 2 – CUSTOM COMMANDS:**

**Filesearch**

In this command, we searched filenames with matching a keyword in a directory. If nothing specified, our file search command just return the file names with matching keyword.

![metin içeren bir resim

Açıklama otomatik olarak oluşturuldu]()

Example output of filesearch without any option argument:

**![metin içeren bir resim

Açıklama otomatik olarak oluşturuldu]()**

If -r option is specified, our command is researching all matching files under subdirectories. In order to do that, we created recursive\_file\_search method. After taking arguments in process\_command, we are calling this recursive\_file\_search method. This method does not return anything, but it just prints out the matching file names under subdirectories as well.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Example output of filesearch with -r option:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

If -o option is specified, our filesearch command opens matching files with default applications used in your computer under current directory and prints out matched file names to the terminal. In order to open files, with default applications in your computer, we have used “xdg-open”. This command is installed in most of the Ubuntu machines. If you are using Windows, you need to install “xdg-open” package to your terminal for running this option of filesearch command.

![metin içeren bir resim

Açıklama otomatik olarak oluşturuldu]()

If both -o -r options are specified, our filesearch command will print matching files under subdirectories (including current directory) and open them with default applications. For this purpose, we have used our recursive\_file\_search command. If it takes -o parameter with -r parameter, it will activate opening option inside of it. If it does not take -o parameter, it just prints out files recursively. The implementation of recursive\_file\_search command is put above. While writing options, the user can write options with any order (-o -r or -r -o). Our command is implemented in a way that works for both orders.

**Cdh:**

In cdh the first thing I have done is create history txt file in the current directory we are in where all the directories we have visited are added. The created file name should be like current\_directory/filename.txt. If the current\_directory part is not added when trying to open the file, we would create a new file in the new directory we are in and can’t access the old file. The filename is made to be a global variable so that we can access from it any method, anytime we want. The current directory is added to the file whenever the current directory is different from the last directory. This is checked at the beginning of process\_command.

When cdh is called if there are more than 10 lines in directory history file it is first shorten down to the last ten directories. Then the directories are printed on screen with letters and numbers next to them. After that using scanf user input of which directory they want to go to is taken. But using scanf in parent process results in segmentation fault, therefore fork is done and the user input is communicated through pipes to the parent. Then using chdir the directory is changed.

The file is not deleted to save the history between sessions.

**Take:**

We take inputs of this method in process\_command method. Then, if the command->name is matched with take, we call a command called take. This command takes input as path given by user and returns nothing. Inside of the method, we tokenize path given by user with “/”. Then, for each directory, we checked status of directory (whether it exists or not). If the directory does not exist, first we create directory and then changed our current directory into it. If the directory exists, we just changed our current directory into it. For this operation we used “for loop” over our tokenized path string.

![metin içeren bir resim

Açıklama otomatik olarak oluşturuldu]()

Example output is given below:

![metin içeren bir resim

Açıklama otomatik olarak oluşturuldu]()

**Joker:**

In a similar way with other commands, we have called a method named as joker inside of the process\_command method. To implement this method first we tried to use the command shown in the ps (crontab -l | {cat; echo "51 13 \* \* \* env DISPLAY=:0 /usr/bin/gnome-calculator";} | crontab -). However, we could not manage to achieve getting random jokes in this way. We mentioned this situation to ISMAYIL (Eray e-mailed him). Since we could not solve this problem with ISMAYIL, we used another approach. We created a file called as “cron.txt”. Then, we have written our crontab command into it. Then, we execute crontab with this txt file. We used execvp in this method. To use execvp, we have utilized from forked command. The below image indicates our joker method implementation.

![metin içeren bir resim

Açıklama otomatik olarak oluşturuldu]()

Example output:



**Our Awesome Commands:**

**Shoplist**

Shoplist is a command for the user to keep track of the shopping items he/she will buy at the grocery store. Just like cdh this list is saved through sessions by entering the items into a file. When user enters “shoplist -a the\_item\_he\_wants number\_of\_that\_item”, they are added to the list. For example: “shoplist -a eggs 5”. When the user wants to see his/hers shopping list he should enter shoplist -l and the list is printed into the terminal. Finally if the user has bought all the items from the shop and wants to clear its shopping list he/she should write “shoplist -d”. A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

**Carry**

Carry is a command that takes a file from one directory and move it to another directory. After moving the file and its contents, it deletes the existing file in the source directory. For this command, we inspired from mv command of Linux machines. However, the implementation of the carry method is completely belonging to us. As we have done in our other methods, after taking inputs in process\_command, we called a separate method named as carry. This method takes source and destination from user. In the source path, first we extract given file name and stored in separate character pointer. Then, we append this file name at the end of user given destination to open file with this name in desired destination. In order to make these string operations, we have utilized from strcpy, strcat, strtok, strrchr function. Then, we read every character from source file and write to the desired destination. If the given source or directory paths are not existed, our code will raise an error. Below image is our implementation of carry command.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

This is the example output of carry command:

**![metin içeren bir resim

Açıklama otomatik olarak oluşturuldu]()**

**WARNING: YOU SHOULD NOT PUT “/” AT THE END OF THE DESTINATION.**

**PART 3 – KERNEL MODULES:**

**Pstraverse:**

The implementation of this command consists of different parts. First we wrote a c file outside of shellfire in which we load a module called my\_module.c into the kernel. my\_module.c has methods for initializing:

Text, letter

Description automatically generated

**and exiting:**

A screenshot of a computer

Description automatically generated with low confidence**.**

**There are also private methods for dfs and bfs:** Text

Description automatically generated**.**

**This module can also take input from user space using module\_param methods:** A picture containing text, orange

Description automatically generated**.**

**The inputs are the root PID for the search and in which traversal mode (dfs or bfs) user wants to traverse (0 for dfs and 1 for bfs). In our initializing method we take these parameters do the traversal search starting from the root PID and using the traversal method user wants.**

**There is also a Makefile where my\_module.c is initialized therefore before using the pstraverse command Makefile needs to be run by writing make to shell beforehand.**

**In shellfyre.c in the pstarverse command method it first checks whether my\_module is loaded in the kernel. If it is not loaded it loads it by calling sudo insmod using execv method. To send the inputs which are the root PID and traverse method (dfs, bfs) “my\_pid=PID\_USER\_CHOSE” and “my\_mode=MODE\_USER\_CHOSE” are inputted as arguments of sudo insmod. This loads my\_module into the kernel by calling my\_module’s init method which also makes the dfs and bfs traversals using the root pid and traversal method user wants. Finally when exit command is called in the shellfire it removes my\_module from the kernel.**

**For Pstraverse we used some online sources for help. Here is a list of all the references we used for completing this command:**

* [**https://www.youtube.com/watch?v=4tgluSJDA\_E&ab\_channel=Johannes4GNU\_Linux**](https://www.youtube.com/watch?v=4tgluSJDA_E&ab_channel=Johannes4GNU_Linux)
* [**https://www.youtube.com/watch?v=j331b\_xS4NY&ab\_channel=Johannes4GNU\_Linux**](https://www.youtube.com/watch?v=j331b_xS4NY&ab_channel=Johannes4GNU_Linux)
* **https://github.com/BuptBoyang/Project-2-Linux-Kernel-Module-for-Listing-Tasks**