COMP 304
PROJECT 1 REPORT

Bilgehan Şahlan

80084

Mahmut Zahid Göksu

79520

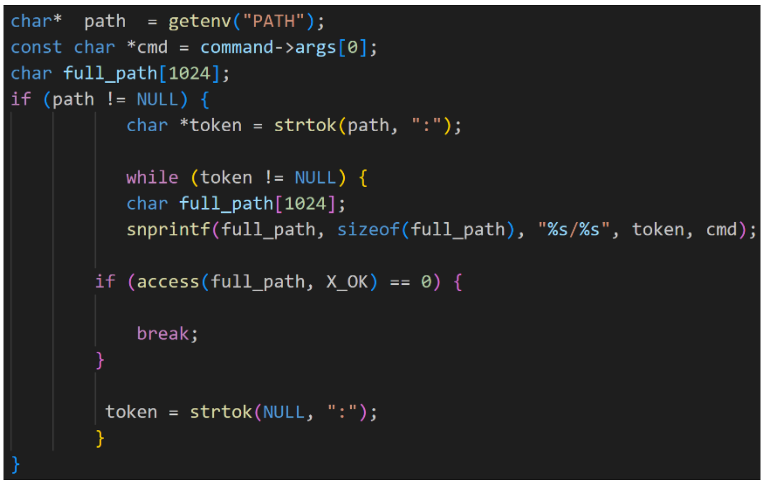
1.Set Up

In order to implement some features and our custom commands we used several external libraries, and we used python for our custom command and draw_graph scripts. Therefore, there are some libraries that need to be installed before execution and "make" command must be called in some directories. Following bullet points are explaning how to set up the project.

- Call "make" command in both main project directory (in our case it is project-1-shell-vim-masters) and in the module directory
- Pip install beautifulsoup4, request, graphviz, ktrain, tensorflow libraries
- To be able to use the costum command offensive_detect you will need to create a conda environment. you will first need to install conda and the create a conda venv using the command: `conda create --name myenv python=3.11`
- After creating the conda env run command: `source Path_OF_ENV/bin/activate` to activate conda venv
- Include a file called "aliases.txt" (otherwise fgets will give a segmentation fault).

2.Part I

In order to reach the command from the bin directories we used "getenv("PATH")" function. "getenv("PATH")" gives the directories containing executable Linux command, by splitting them with a ":". As it can be seen from the figure 1, we created a new string for each directory and checked that if the executable is in that directory by using "access" function's "X_OK" mode, which checks the "execute" permission of the given file. Besides, if the command is not a background command, we used "wait(0)" to wait child process to finish. As a disclaimer we got some help from Chat GPT for using the "access" function.



```
char*  path  = getenv("PATH");
const char *cmd = command->args[0];
char full_path[1024];
if (path != NULL) {
           char *token = strtok(path, ":");

           while (token != NULL) {
           char full_path[1024];
           snprintf(full_path, sizeof(full_path), "%s/%s", token, cmd);

        if (access(full_path, X_OK) == 0) {

            break;
        }

         token = strtok(NULL, ":");
        }
}
```

Figure 1

3. Part II

Redirects:

We have done this part by implementing the same behavior that our shell uses. we have used the parser that has been provided to us. while giving an argument to append or truncate/create the format is : "program arg1 arg2 >file1 >>file2"

This part works by creating file descriptors using dup2 and creating file descriptors and depending on the redirect appends or replaces the content inside the file. for the case "<" the code opens the file provided and redirects the content to a given command.

it uses the STDOUT_FILENO and STDIN_FILENO file descriptor to write the content that has been printed on the terminal and redirect it to the file that has been provided.

Our code works perfectly for the cases of single ">" , ">>" and "<" but has some issues when used with multiple redirects.

4. Part III

4.1 Hexdump

For the convenience we implemented 2 different function one for "-g" flag option called "hexdump_g" and one for normal hexdump called "hexdump" (since those functions are quite long, we could not provide a screenshot). "hexdump" function is called, if there is no -g flag after the "xdd" command and enters state that every input's hex value is printed. Different than the classic "xxd" command, we added an exit utility for convenience, so when "quit" is typed the xdd mode is terminated.

On the other hand, if there is a -g flag the program first checks the accessibility of the file by "access(*file_name*, F_OK)". If there is an accessible file, the "xdd" command will output the hex values of the content of the files. However, if there is no accessible file, the result will be the hex value of the argument after the -g flag.

In order to be as close as the original “xxd” command we printed the hex value of the group size with a range of 8 by using printf’s “%08” utility.

4.2 Alias

As in the linux, our alias function takes the first argument as the alias and the following arguments as the command to be stored. After that, arguments are appended to the “aliases.txt” file as “alias_name=command”. This procedure takes places in a function called “addAlias”.

While parsing an argument we first check whether the name (first word) of the argument is an alias by calling the “isAlias” function. “isAlias” function takes command name as input and checks the aliases.txt file for the command name. If the command name present in the file, it returns the command which is after the “=” sign. After that, the remaining arguments from the terminal checked, if any they are appended to the agrs dynamic array of a temporary command_t struct called alias_command.

After, creating and populating the alias_command, we assigned alias_command->args to command->args and alias_command->arg_count to command->arg_count for parse_command function to finalize our new command.

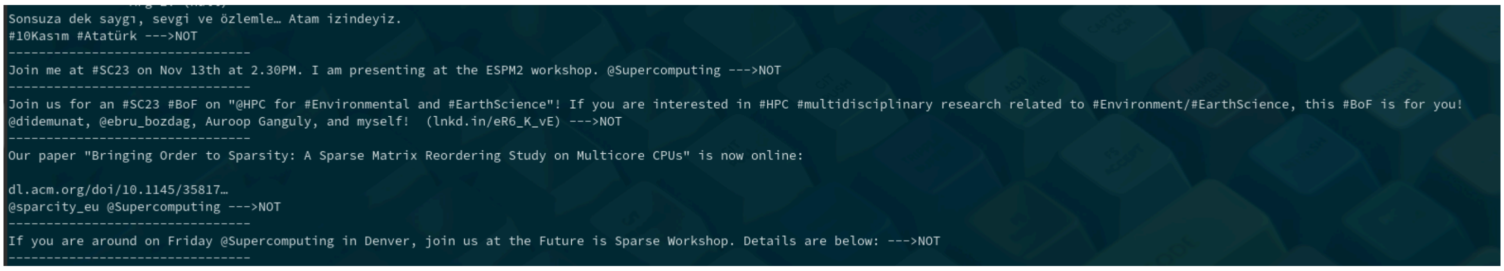
4.3 good_morning

The good_morning command works by getting the number of minus and the file location of the mp3 file. After these are provided it first calculates the time that the cronjob will be executed and then creates a command in the format that crontab uses. we have used the mpg123 command to play the audio. After building the command that will be given to crontab it executes the command “crontab -l” to get previous cronjobs and appends all the cronjobs to a tempfile and executes the command “crontab tempfile” that will write the file to crontab.

4.4.1 what_to_watch (Bilgehan’s Custom Function)

What to watch function is a special custom function recommends a random anime from the top 50 anime list of the “myanimelist.net”. The command is implemented in python and makes a system call for executing the “rand_anime.py“ file in the main directory.

rand_anime.py files imports beautifulsoup4 (bs4) and request libraries. bs4 library is used for web scrapping and request library is for connecting the web server. The script



```
Sonsuza dek saygı, sevgi ve özlemle… Atam izindeyiz.
#10Kasım #Atatürk --->NOT
--------------------------------
Join me at #SC23 on Nov 13th at 2.30PM. I am presenting at the ESPM2 workshop. @Supercomputing --->NOT
--------------------------------
Join us for an #SC23 #BoF on "@HPC for #Environmental and #EarthScience"! If you are interested in #HPC #multidisciplinary research related to #Environment/#EarthScience, this #BoF is for you!
@didemunat, @ebru_bozdag, Auroop Ganguly, and myself!  (lnkd.in/eR6_K_vE) --->NOT
--------------------------------
Our paper "Bringing Order to Sparsity: A Sparse Matrix Reordering Study on Multicore CPUs" is now online:

dl.acm.org/doi/10.1145/35817…
@sparcity_eu @Supercomputing --->NOT
--------------------------------
If you are around on Friday @Supercomputing in Denver, join us at the Future is Sparse Workshop. Details are below: --->NOT
--------------------------------
```

Figure 2

basicly sends a request to the website and if the request is a success, it takes the information (name, rank, rating, season, etc.) of the top 50 anime and stores them in a list. The strings from the website contains lots of "\n" and whitespaces. In order to, handle them a "split("\n", " ")" operation followed by "strip" operation used for parsing the strings.

After a random anime selected from the list, a string including anime name, season, platform, ranking and rating is built and printed to the terminal as a recommendation. If the internet connetion is weak, the command can take some time to execute.

4.4.2 offensive_detect (Mahmut's Custom Command)

This command works in two different ways :

- "offensive_detect r X_userName" where X_userName is the tag of the X user.
- "offensive_detect X_userName"

The first command with the r flag will give you the last 5 tweets that a user has written.

The second command will detect if print out the last 5 tweets and detect if the tweets are offensive or not.

Example: "offensive_detect didemunat"

This command works by first sending a request to Nitter which is a free and open source alternative viewer for Twitter/X. After sending the appropriate request it gets the first 5 tweets and then puts each text to the offensive language detection model. It first loads the Model and stores it in cache so that every time the command is called the model is not loaded. I does inference using the tweets and gives the Labels "OFF" if offensive or "NOT" if the tweet is not offensive. I have used the ktrain library which is a lightweight wrapper of the Tensorflows Keras.

Note: the number of tweets can be adjusted using the python code.

5. Part 4

"psvis" command takes two arguments a pid of the root process and name of the output file. After, taking the inputs the command makes consecutive system calls for loading a kernel module called "mymodule.ko". Firstly, for saving the log entries only belonging to the mymodule.ko "system('sudo dmesg --clear')" called for deleting the past log entries. After that a system call is done with the string "sudo insmod ./module/mymodule.ko root_pid=*pid* file_name=*file_name*".

In the mymodule.c, a dfs is performed for extracting child process's information. We created a recursive function called traverse_processes and printed pid, name, parent pid and creation time of the child processes to the kernel log. After printing information of the child processes, a system call "dmesg > log.txt" is made for giving them to the draw_tree.py script.

draw_tree.py script is plotting a tree diagram of the processes as in figure 3. In order to draw the tree, we used graphviz library. Firstly, we initialized the root_node with the name "Root_node" and the id of the node as the pid of the root_node. After that, we started to extract the parent, child, pid, and name informations of the processes. The connected nodes' pid's are stored as a tupple and they were given to the dot.edges() function as a list of tupples for creating edges between nodes.

In order to control whether the kernel module we got some help from chat gpt and implemented a "is_module_loaded() " function, which looks into the proc directory for the given module name (in this case mymodule), and returns 1 if the module is loaded, return 0 otherwise.

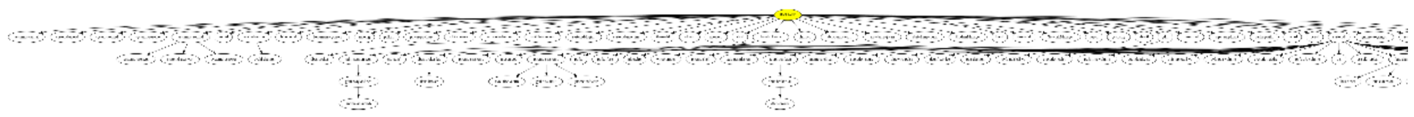


Figure 3