COMP304 PROJECT 1 REPORT

R.Sude Güngör - Lara Gül Küçükdevlet

Part1:

First we checked the executable file in current directory. In order to do this we put ‘if’ statement to see if command starts with “./”. We put execv command to execute the file if it exists. If the file does not exist, we get PATH environment variables and stored the list of directories where Shell search for executable files. For tokenization created duplicate of path variables. Path variable is a string of directories separated by colon. So, we need to separate colons to get each directory. Strtok function is used to separate strings using a delimiter. Iterated through these directories with acess function and if we find an executable directory it executes it with execv function. If the command can not found, it prints an error message. We executed Part1 under the void exec_command(struct command_t *command). We have implemented an extra method called file_exists that checks whether a specified file exists in the file system.

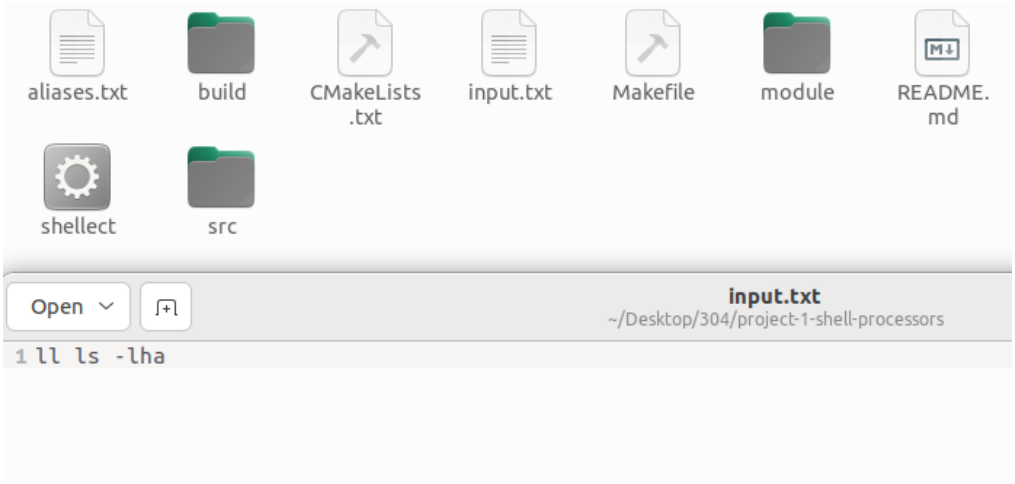
```
    if(command->args[0][0] == '.' && command->args[0][1] == '/') {
        if (file_exists(command->name)) {
            //get current working directory
            execv(command->name, command->args);
        }
    }
    else
    {
        char *path = getenv("PATH");
        // stores list of directories where shell search for executable files
        char *dupPath = strdup(path);

        char *dir = strtok(dupPath, ":");
        char cmdPath[1000]; // upper limit for the length of an exec path is 1000
        while (dir) {
            sprintf(cmdPath, "%s/%s", dir, command->name);
            if (access(cmdPath, X_OK) == 0) {
                break;
            }
            dir = strtok(NULL, ":");
        }
        execv(cmdPath, command->args);


    }
    printf("-%s: %s: command not found\n", sysname, command->name);
    // end of part 1
}
```

```
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ ls -lha
total 96K
drwxrwxr-x 8 sude sude 4,0K Kas 12 15:22 .
drwxrwxr-x 7 sude sude 4,0K Kas 12 12:13 ..
-rw-rw-r-- 1 sude sude    6 Kas 12 15:13 aliases.txt
-rw-r--r-- 1 sude sude   12 Kas 12 15:13 append.txt
drwxrwxr-x 5 sude sude 4,0K Kas 12 15:22 build
-rw-rw-r-- 1 sude sude 3,3K Eki 28 11:21 .clang-format
-rw-rw-r-- 1 sude sude  486 Eki 28 11:21 CMakeLists.txt
drwxrwxr-x 8 sude sude 4,0K Kas  9 21:22 .git
drwxrwxr-x 4 sude sude 4,0K Eki 28 11:21 .github
-rw-rw-r-- 1 sude sude  568 Eki 28 11:21 .gitignore
-rw-r--r-- 1 sude sude   12 Kas 12 13:45 input.txt
-rw-rw-r-- 1 sude sude 1,4K Kas 12 12:38 Makefile
drwxrwxr-x 2 sude sude 4,0K Kas 12 12:48 module
-rw-r--r-- 1 sude sude    0 Kas 12 13:42 output.txt
-rw-rw-r-- 1 sude sude  529 Eki 28 11:21 README.md
-rwxrwxr-x 1 sude sude  31K Kas 12 15:22 shellect
drwxrwxr-x 2 sude sude 4,0K Kas 12 10:55 src
drwxrwxr-x 2 sude sude 4,0K Kas 11 23:46 .vscode
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ ls
aliases.txt  append.txt  build  CMakeLists.txt  input.txt  Makefile  module  output.txt  README.md  shellect  src
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$
```

Part2:

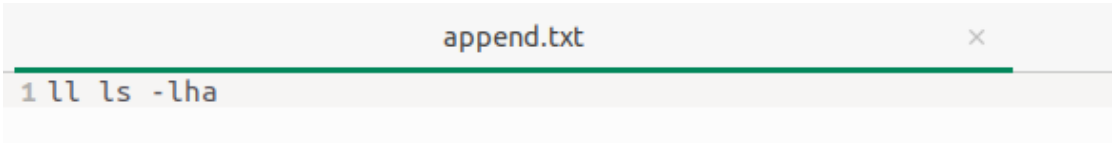
In this part, we implemented I/O redirection for 4 commands. These are '>', '<' , '>>' . First, we checked if input redirection is not NULL. Opened the file as'read-only'. If there is an error opening the file, printed an error message. Then checked if the output redirection is not NULL. Opened the file as 'write-only'. If file does not exist, it creates; if it exists, it truncates. We used O_CREAT | O_WRONLY | O_APPEND flags. If there is an error about opening the file, printed an error message, if not the file descriptor (fd1) is duplicated, made fd1 new output file descriptor and closed it. Lastly, checked if append output redirection is NULL. Then checked for errors and duplicated fd2 and made it new standard output file. Then closed it. We executed this part under exec_command(struct command_t *command).

```
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ cat aliases.txt >input.txt
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$
```



We wrote "aliases.txt" to input.txt and run command below.

```
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ cat aliases.txt <input.txt >>append.txt
heresude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$
```



Part3:

1-Hexdump:

In this part, we wrote a command to show a hex dump of a given file - that is showing the file contents as hexadecimal numbers. We created a **void hexdump(struct command_t *command)** function. First, we checked the arg_counts and handled errors if the file is not opening. Then, If the arg_count is greater than one, parsed the second argument and assigned it to group_size. 'group_size' equals to sixteen divided by the given number. Sixteen is the number of bytes represented in each line. Declared a buffer to store bytes from file. We decaled a bytes_read variable

to return the number of bytes read from the file descriptor fd. Declared a loop until read returns 0, means it reached the end of the file. For each line, printed the offset of the first byte. Offset is the position of the first byte in each line. 08x is the format specifier for printing hexadecimal numbers. Printed | and moved to next line by printf("|\n"). Closed file descriptor. Used strcmp function under the process_command() function to checks if the command name is hexdump.

```
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ hexdump -g 16 input.txt
00000000 61 |
00000001 6c |
00000002 69 |
00000003 61 |
00000004 73 |
00000005 65 |
00000006 73 |
00000007 2e |
00000008 74 |
00000009 78 |
0000000a 74 |
0000000b 0a |
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ hexdump -g 2 input.txt
00000000 61 6c 69 61 73 65 73 2e |
00000008 74 78 74 0a |
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$
```

2-Aliases:

In this part, we handled the creation of aliases command. We created a method called **void alias(struct command_t *command)**. Created a buffer named alias to store the concatenated string. Copied the second argument of command to buffer. Started to iterate from third argument and added a space separator. Then concatenated the argument with alias using strncat. Opened file 'aliases.txt' as read only, tokenized each line with /n and iterated through it to check if alias already exits. If it already exists printed an error message. Else, opened the alias.txt to add new alias with fprintf. Closed it. Under process_command(struct command_t *command) function we called alias functiom to execute.

```
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ alias kk cat aliases.txt
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ kk
mm ls -lha
ll ls -a
cc cat aliases.txt
kk cat aliases.txt
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$
```

3-Good Morning:

In this part, we scheduled an alarm that will play a given audio file after a specified number of minutes. First we created a function named **void good_morning(struct command_t *command).** Checked if the number of arguments equals to 4. If it is not, we printed an error message. Parsed second argument and stored it as int minute. Assigned path of the audio file to char *audio variable. Created a buffer named command1 to store the constructed command for playing the audio file using mpv. Constructed cron command to schedule the audio play and used system to execute it.

```
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ good_morning 2 /home/sude/Music/izmirMarsi
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ crontab -l
```

```
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
*/2 * * * * mpv /home/sude/Music/izmirMarsi
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$
```

4-Custom Commands:

a)Sude: To run this command you need to type **game <number[0-4]>**. You will start with 3 lives and must remember and input the number shown on the screen within a limited time.There are different themes such as Cat, Dog, Penguin, Frog, and Owl, which players can select at the start by giving a number among 0 to 4. Correct inputs increases the player's score; incorrect inputs or failing to respond in time costs a life. The goal is to successfully complete 10 rounds. The time allowed to remember the number decreases every 5 correct answers, increasing the challenge.

```
Lives: 3                Score: 0

   @..@
  (----)
 ( >__< )
 ^^ ~~ ^^

 Number: 9830954

 You will be asked to enter the number in: 5 seconds
```

```
Lives: 2                Score: 0

You have 5 seconds left to enter the number. Press enter to continue
```

b)Lara:

In this part, I created a multiplication game. It has 10 rounds and it gives 5 seconds for answering the questions correctly with sleep(5). First you have to press enter to start the game. Then you have to answer the questions correctly with entering integers. At the end, it calculates your score and prints prizes.

```
sude@sude:/home/sude/Desktop/304/project-1-shell-processors Shellect$ lara
Hello sude, welcome to the multiplication game!
You will be given 10 questions to answer.
You have 5 seconds to answer each question.
If you answer correctly, you will get a star.
If you answer incorrectly, you will not get a star.
If you answer all 10 questions correctly, you will get a prize!
Good luck!
Press enter to start the game.

Question 1: 4 x 7 =
```

Part 4:

In this part, first we provided macros. Then declared a module parameter called curr_pid which gives current pid. To initialiaze the module we used simple_init(void) function. Found the parent process with given pid. If parent process can not be found returned -1 which means that the module could not be loaded. Else called the psvis_traverse(parent) function and returned 0 which means the module loaded successfully. Then created a void psvis_traverse(struct task_struct *parent) to traverse the process tree. In this function we created functions named find_oldest_child(list_children, &parent->children) and show_process_tree(list_children , &parent->children) which find the oldest child based on the starting time to color it in a different color. Printed the process tree. Finally, to exit the module we used simple_exit(void) function. In shell-skeleton.c file we opened void psvis(struct command_t *command) to construct the command to visualize the process tree. In child process opened file output.txt and wrote the first line of the file to output.txt. Wrote the process tree to output.txt. Converted output.txt to png. Exited from child process. In parent process used wait to wait for the child process to finish.