HOMEWORK 2 REPORT

Mustafa Mercan

1901042676

1) SYSTEM REQUIREMENTS

We were asked to develop a terminal emulator in this project. The terminal emulator should be capable of running 20 different commands in a single line. Additionally, the use of the system() function is prohibited in this project. Instead, the "fork()", "execl()", "wait()", and "exit()" functions found in the standard C library should be used. Furthermore, each command entered into this terminal is expected to be executed in a different child process. Preventing memory leaks is also expected in our program. Multiple command pipes (|) have been linked together in this project, and redirections ("<", ">") have been used to direct file descriptors. To exit the program, the user must type the ":q" command into the terminal and press the enter key.

Additionally, when this program is first run, it creates a .txt file in the "logs" folder in the current directory, which carries the timestamp of that moment. This txt file records which pid number the commands and redirections executed in that terminal. Also, the "srcs" file in the directory contains the ".c" extension files for the functions we used in the terminal. Our program starts running in the "main.c" part in our main directory.

The "terminal.h" file is the header file where the functions are defined. Finally, our directory also contains the makefile.

2) PROBLEM SOLUTIONS APPROACH

1→ Firstly, I designed the working environment that I would use to implement my project. According to this design, there would be a main file named "main.c" with a C extension that would run the program. Then, I would have a folder named "srcs" containing different functions that I would use in the program. Within this "srcs" folder, three C extension files would be created by categorizing the functions I used within themselves as "log_utils.c", "utils.c", and finally "terminal_utils.c". In the main directory, there would be a header named "terminal.h" where the functions are defined, a folder named "logs" where the program logs are written, and each time the program runs, the pid numbers and the commands entered are stored in this folder. Finally, there would be a makefile file that can compile all of these components.

2→ I prepared a function named "terminal_prompt()" that can display the current directory in the terminal. This function shows the path of the current directory before entering any command in the terminal and prints it on the screen

3→After this, I used the "fgets()" function to get input from the user. Then, I made some adjustments on this input. Afterwards, I performed different processes based on the assigned pid number by using the fork() function.

4→ After getting input from the user using the 'fgets()' function, I performed some modifications on the input. Then, I used the 'fork()' function to perform different operations based on the assigned pid number. If the pid number is 0, I first determined whether the entered input consists of a single command or multiple commands. If there is a single command, I split it into several pieces based on the '<' and '>' symbols. After making the necessary modifications, if the '<' symbol is present, I used the 'smaller_than_handling()' function, and if the '>' symbol is present, I used the 'greater_than_handling()' function. These functions are responsible for performing the appropriate redirection based on the symbol found in the input value, using the 'dup2()' function. Each time these functions are called, they write the performed operation to the 'timestamp.txt' file in the 'logs' directory. After completing these operations, the 'execute_command()' function is called, which executes the command passed as a parameter. Like the previous functions, this function also writes the performed operation to the 'timestamp.txt' file in the 'logs' directory as a log.

5→ If the input value taken from the user contains multiple commands (command1 | command2 | command3), then the pipe_handling() function is called. The purpose of this function is to ensure that the value to be written on the screen when command1 is executed is successfully transferred to command2.

6→The "pipe_handling()" function includes a variable called "char ** all_commands". This variable holds the string structures that are separated according to the pipe symbols (|) in the user input using the "multiple_command_handling()" function. Then, pipes and pids were created according to the number of inputs entered by the user. A while loop was created based on the number of commands. In this loop, each command was split into smaller parts using the "parse_single_line_input()" function based on the "<" and ">" signs. Then, necessary redirects were made using pipes. Depending on the presence of "<" and ">" signs within the command, the "greater_than_handling()" and "smaller_than_handling()" functions were used to perform the necessary redirects. After all of these processes were completed, the "execute_command()" function was used to run the command

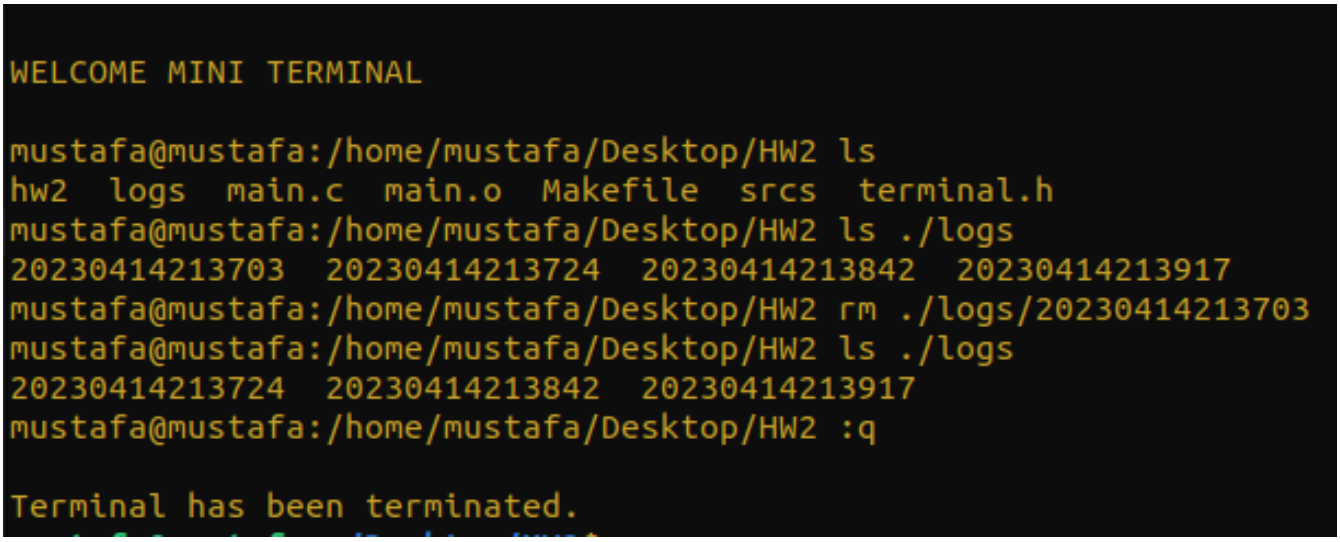
7→ I made sure that all parent processes wait for the child process to terminate, so that no zombie processes are created.

8→ Then, I made the necessary changes for the program to exit when the user enters ":q" input. Finally, I checked for memory leaks using valgrind and terminated my project.

3) Challenging parts of the project

The most challenging part of this project for me was dealing with the memory leaks that occurred while parsing the user input. Despite all my efforts, I couldn't solve the memory leaks issue in my self-created 'strtrim' function. As a result, I had to find a different solution, which caused significant changes in my planned design and made it challenging to adjust

4)Test Outputs



```
WELCOME MINI TERMINAL

mustafa@mustafa:/home/mustafa/Desktop/HW2 ls -al | grep "ma" | sort | uniq | wc -l
2
mustafa@mustafa:/home/mustafa/Desktop/HW2 ls -al | grep "ma" | sort | uniq | wc -l > output.txt
mustafa@mustafa:/home/mustafa/Desktop/HW2 :q

Terminal has been terminated.
```

output.txt

```
1  2
```

```
WELCOME MINI TERMINAL

mustafa@mustafa:/home/mustafa/Desktop/HW2 ls
hw2  input.txt  logs  main.c  main.o  Makefile  srcs  terminal.h
mustafa@mustafa:/home/mustafa/Desktop/HW2 cat < input.txt
test1
aaaaa
test33
bbbbb
test2
test55
asfasf
asfsafamustafa@mustafa:/home/mustafa/Desktop/HW2 cat < input.txt | grep "test" | sort > output.txt
mustafa@mustafa:/home/mustafa/Desktop/HW2 ls
hw2  input.txt  logs  main.c  main.o  Makefile  output.txt  srcs  terminal.h
mustafa@mustafa:/home/mustafa/Desktop/HW2 cat < output.txt
test1
test2
test33
test55
mustafa@mustafa:/home/mustafa/Desktop/HW2
```

```
WELCOME MINI TERMINAL

mustafa@mustafa:/home/mustafa/Desktop/HW2 ^C
SIGINT signal received!
^Zmustafa@mustafa:/home/mustafa/Desktop/HW2
SIGTSTP signal received!
^Zmustafa@mustafa:/home/mustafa/Desktop/HW2
SIGTSTP signal received!
^Cmustafa@mustafa:/home/mustafa/Desktop/HW2
SIGINT signal received!

mustafa@mustafa:/home/mustafa/Desktop/HW2 mustafa@mustafa:/home/mustafa/Desktop/HW2
```

```
Terminal has been terminated.
==14753==
==14753== HEAP SUMMARY:
==14753==     in use at exit: 0 bytes in 0 blocks
==14753==   total heap usage: 36 allocs, 36 frees, 30,714 bytes allocated
==14753==
==14753== All heap blocks were freed -- no leaks are possible
==14753==
==14753== For lists of detected and suppressed errors, rerun with: -s
==14753== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```