

CSE 344 – Spring 2023

HW2 Report

GONCA EZGİ ÇAKIR – 151044054

1. Solution Approach

Terminal emulator program implementation steps are listed below.

- Program works in a `while(1)` loop in order to keep getting commands from user till the exit command (`:q`) received.
- Command taken from user after the program started with “myShell>”. **fgets** takes the command as a line and checks validity. The command shouldn't be containing “&&, ||, &, ;” separators also there can't be more than 20 commands in a single line. If these cases occurs program simply prints the error with proper usage information and asks for new command. If the command is valid, the line splitted according to `|` token by using **strtok**. Then these commands are stored in a string array, also the number of commands are calculated.
- Children processes are created in a for loop that iterates as number of commands. Child is created by using **fork** system call. Fork's return value checked if it is equal to 0, child process called. If it is greater than 0, parent process runs. Otherwise there is an error occurred while fork system call, program prints the error and exits.
- For the basic of the program children processes have to communicate between each other. Because each child process executes one command and gives the result to other child process in order to get the proper result at the end.

So, pipe is created at the beginning of the for loop before child process created by fork, and stored in an array of pipes. Each child process closes the read end of the pipe (`pipe[0]`) and duplicates the `STDOUT_FILENO` to pipes write end (`pipe[1]`). This provide an access to other child process to continue from the `STDOUT_FILENO`. At the end child process closes the pipes write end (`pipe[1]`) and executes **execl** system call to run command.

Parent firstly closes the write end of the pipe (`pipe[1]`) and duplicates the `STDIN_FILENO` for pipe to read end (`pipe[0]`). At the end parent process closes the pipes read end (`pipe[0]`).

- If the for loop reaches the *last child* (by simply checking loop iterator to be equal number of command amount minus 1) shell command's result which is taken from pipes read end(`pipe[0]`) is printed to terminal. If there is nothing to print (might be written to file), program simply continues.
- Signals are handled in a signal handler method which is set with `sigaction` struct and implemented at the beginning of the program. When a program received `SIGINT` or `SIGTERM` signal handler print the signal name and program requests another command prompt from the user.

For SIGKILL it is not possible to catch or handle this signal because it is handle by the kernel. Also valgrind gave a warning, thats why it not added in the code.

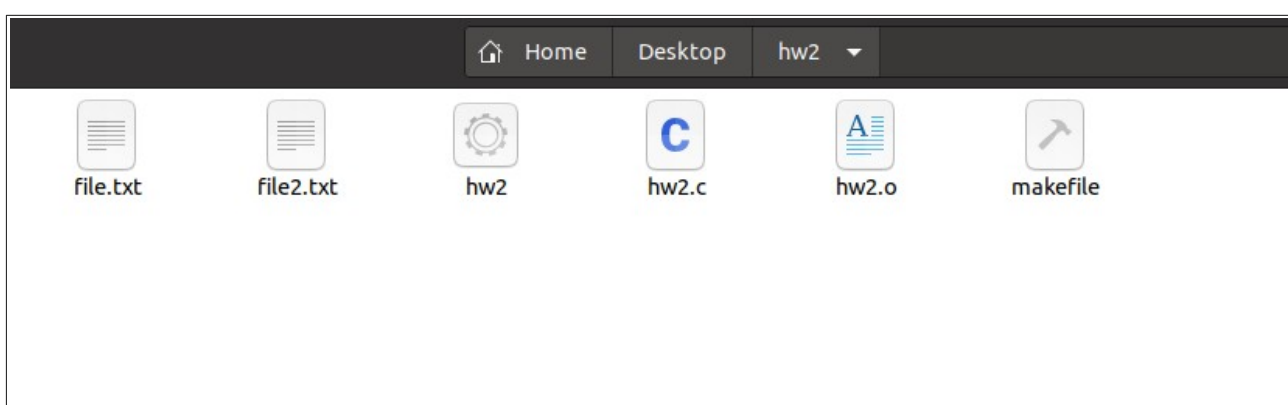
```
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/hw2$ make
gcc -c hw2.c
gcc hw2.o -o hw2 -Wall -std=c99 -pedantic
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/hw2$ valgrind -s ./hw2
==14488== Memcheck, a memory error detector
==14488== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==14488== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==14488== Command: ./hw2
==14488==
==14488== Warning: ignored attempt to set SIGKILL handler in sigaction();
==14488==          the SIGKILL signal is uncatchable
myShell>
34         }else if(signal == SIGTERM){
35             perror("\n-----SIGTERM signal is caught-----\n ");
36         }else if(signal == SIGKILL){
```

- At the end of executing the command all child process pids and their executed command information are written into log file with a filename as log_timestamp.txt .
- When exit command(:q) received all children process are ended by **wait** system call.

2. Test Results

Multiple test case senarios applied and screenshots are added.

Below you will see some program input result. This is the hw2 folder content at the beginning:



Content of file.txt:



A screenshot of a text editor window titled "file.txt" with a subtitle "~/Desktop/hw2". The window has a dark header bar with "Open", a dropdown arrow, and a "J+l" icon. The text content is as follows:

```
1 test case
2 zeynep
3 ezgi
4 ali
5 aliye
6 gursel
7 idil
8 gonca
9 gonca
10 idil|
11
```

Content of file2.txt:

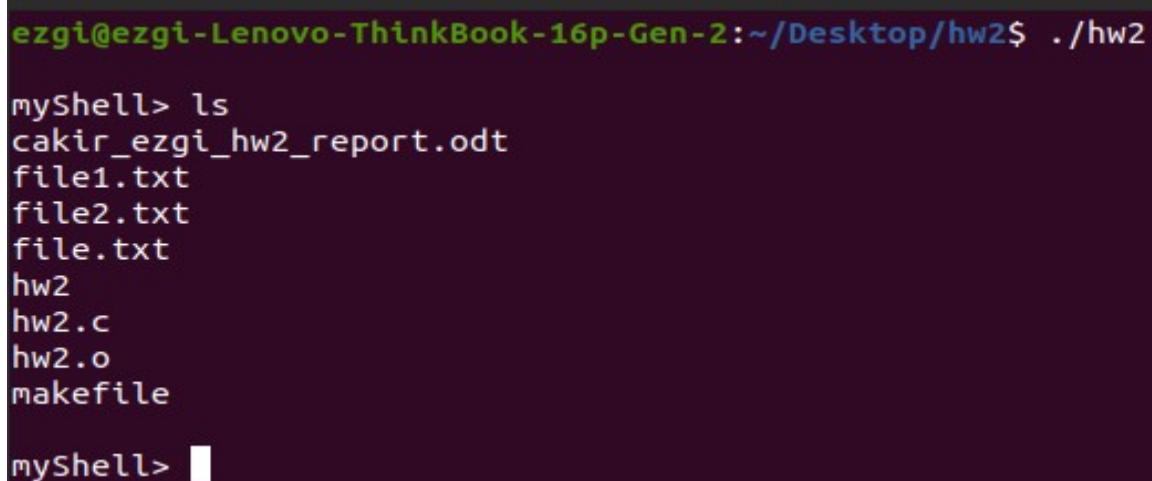


A screenshot of a text editor window titled "file2.txt" with a close button "X" in the top right corner. The text content is as follows:

```
1 |
```

2.1. ls

Program output for command:

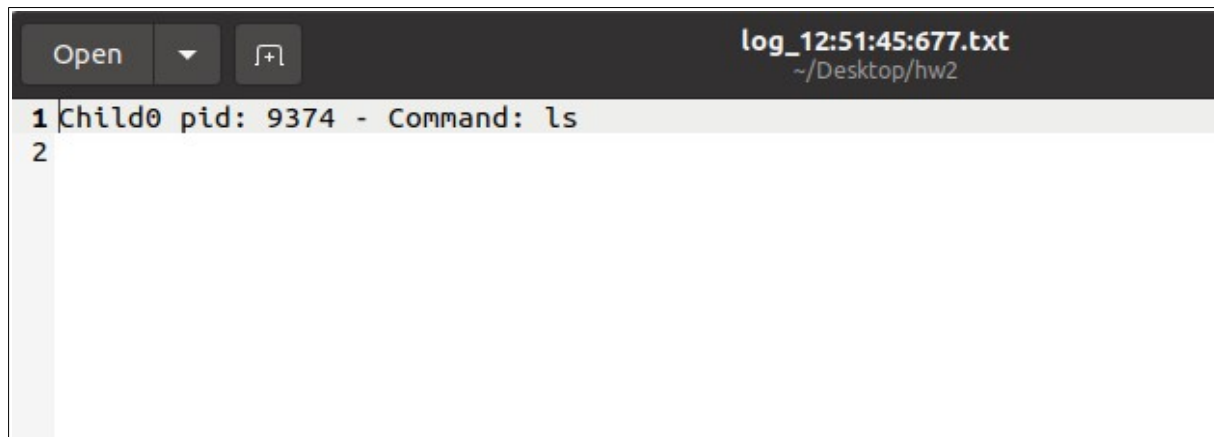


A screenshot of a terminal window showing the output of the 'ls' command. The prompt is "ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/hw2\$./hw2". The output is:

```
myShell> ls
cakir_ezgi_hw2_report.odt
file1.txt
file2.txt
file.txt
hw2
hw2.c
hw2.o
makefile

myShell> 
```

Log file of the command:

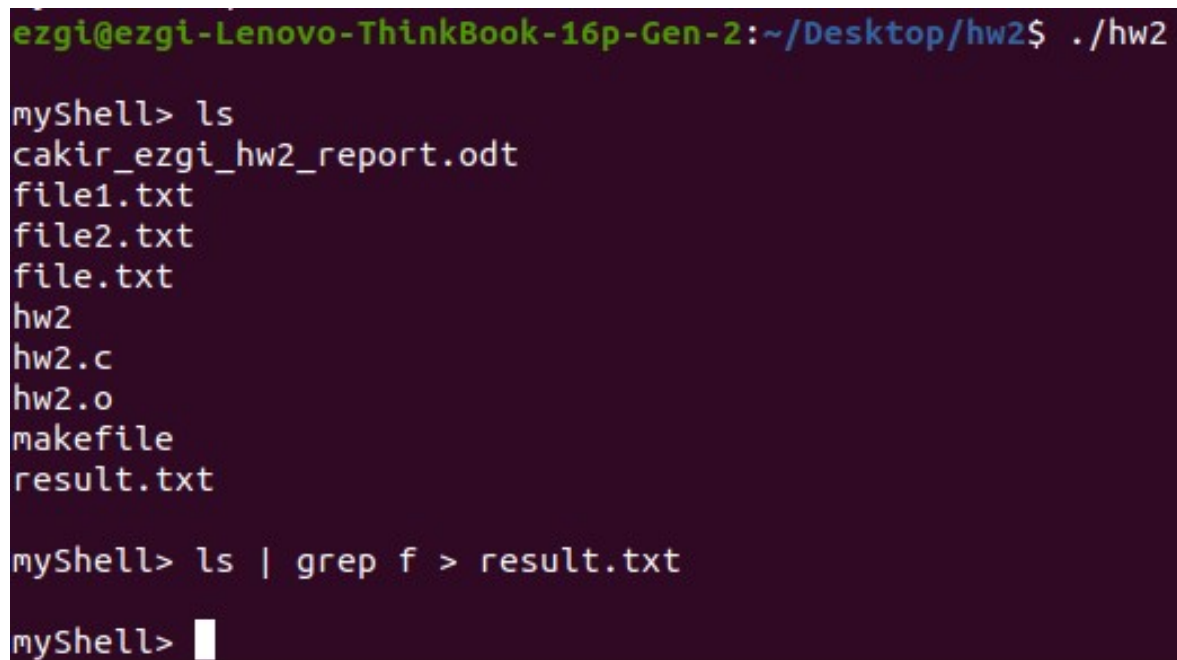


The screenshot shows a log file viewer window titled "log_12:51:45:677.txt" with the path "~/Desktop/hw2". The window contains a list of log entries. The first entry is highlighted and shows "1 | Child0 pid: 9374 - Command: ls". The second entry is "2".

```
Open [v] [icon] log_12:51:45:677.txt
~/Desktop/hw2
1 | Child0 pid: 9374 - Command: ls
2
```

2.2. ls | grep f > result.txt

Program output for command:



The screenshot shows a terminal window with the prompt "ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/hw2\$./hw2". The terminal output shows the execution of "myShell> ls" which lists the files: "cakir_ezgi_hw2_report.odt", "file1.txt", "file2.txt", "file.txt", "hw2", "hw2.c", "hw2.o", "makefile", and "result.txt". The second command "myShell> ls | grep f > result.txt" is also shown, followed by a prompt "myShell>".

```
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/hw2$ ./hw2
myShell> ls
cakir_ezgi_hw2_report.odt
file1.txt
file2.txt
file.txt
hw2
hw2.c
hw2.o
makefile
result.txt

myShell> ls | grep f > result.txt

myShell>
```

Log file of the command:

```
log_12:52:06:318.txt
~/Desktop/hw2

1 Child0 pid: 9402 - Command: ls
2 Child1 pid: 9403 - Command: grep f > result.txt
3
```

result.txt after execution:

```
result.txt
~/Desktop/hw2

1 file2.txt
2 file.txt
3 makefile
```

2.3. cat file.txt | sort | uniq

Program output for command:

```
myShell> cat file.txt | sort | uniq

ali
aliye
ezgi
gonca
gursel
idil
test case
zeynep

myShell> 
```

Log file of the command:

```
Open  ▼  [🔍]  log_12:52:23:457.txt
~/Desktop/hw2
1 Child0 pid: 9444 - Command: cat file.txt
2 Child1 pid: 9445 - Command: sort
3 Child2 pid: 9446 - Command: uniq
4
```

2.4. sort < file.txt

Program output for command:

```
myShell> sort < file.txt

ali
aliye
ezgi
gonca
gonca
gursel
idil
idil
test case
zeynep

myShell> █
```

Log file of the command:

```
Open  ▼  [🔍]  log_12:52:32:664.txt
~/Desktop/hw2
1 Child0 pid: 9472 - Command: sort < file.txt
2
```

2.5. cat > file2.txt file.txt

Program output for command:

```
myShell> cat > file2.txt file.txt  
myShell> 
```

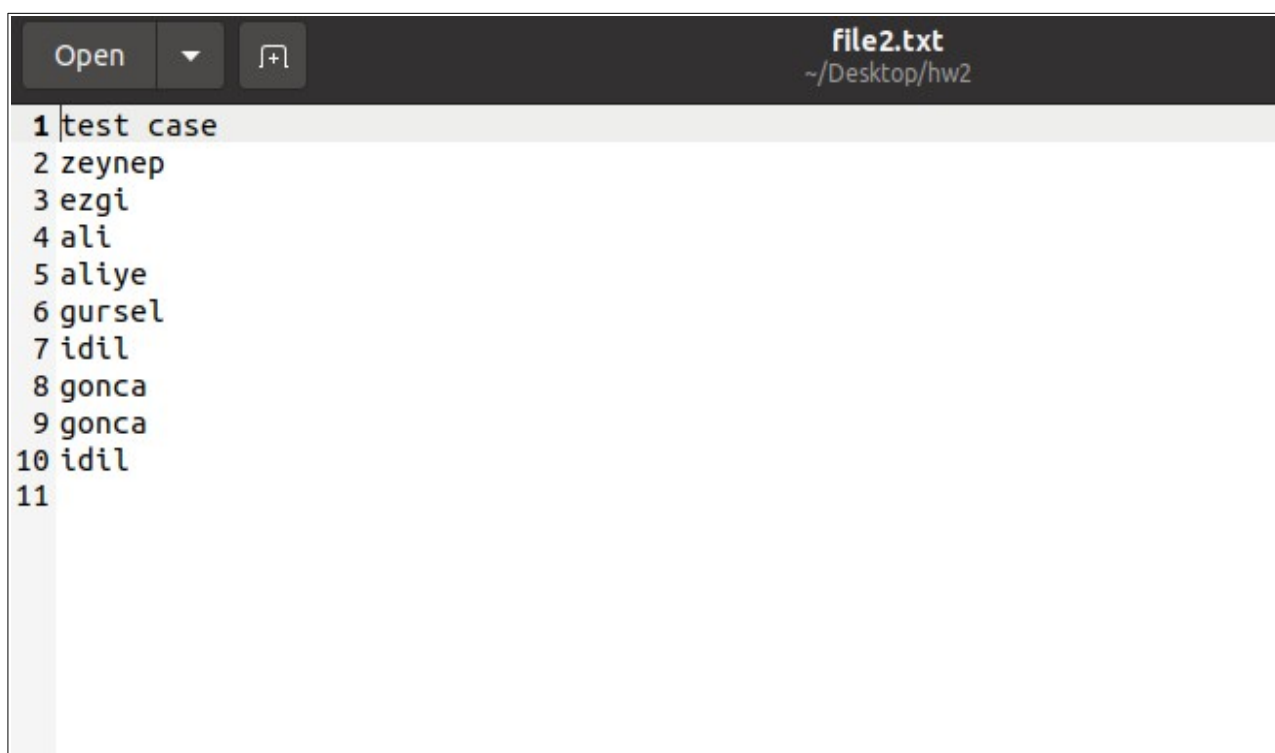
Log file of the command:



The screenshot shows a log file viewer window titled "log_12:52:55:634.txt" with a subtitle "~/Desktop/hw2". The window contains a list of log entries. The first entry is highlighted and shows the command "Child0 pid: 9484 - Command: cat > file2.txt file.txt". The second entry is "2".

```
1 Child0 pid: 9484 - Command: cat > file2.txt file.txt  
2
```

file2.txt after execution:



The screenshot shows a text file viewer window titled "file2.txt" with a subtitle "~/Desktop/hw2". The window contains a list of lines from the file. The first line is "test case", followed by names: "zeynep", "ezgi", "ali", "aliye", "gursel", "idil", "gonca", "gonca", "idil". The list ends with "11".

```
1 test case  
2 zeynep  
3 ezgi  
4 ali  
5 aliye  
6 gursel  
7 idil  
8 gonca  
9 gonca  
10 idil  
11
```

2.6. cat && file.txt, cat || file.txt, ls &, ls ; (unsupported command, Invalid)

Program output for command:

```
myShell> cat && file.txt
---
Invalid command '&&'.
Usage: Enter a shell commands with <, > or |.
---

myShell> cat || file.txt
---
Invalid command '||'.
Usage: Enter a shell commands with <, > or |.
---

myShell> ls &
---
Invalid command '&'.
Usage: Enter a shell commands with <, > or |.
---

myShell> ls ;
---
Invalid command ';'.
Usage: Enter a shell commands with <, > or |.
---

myShell> 
```

2.7. a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a (25 commands, Invalid)

Program output for command:

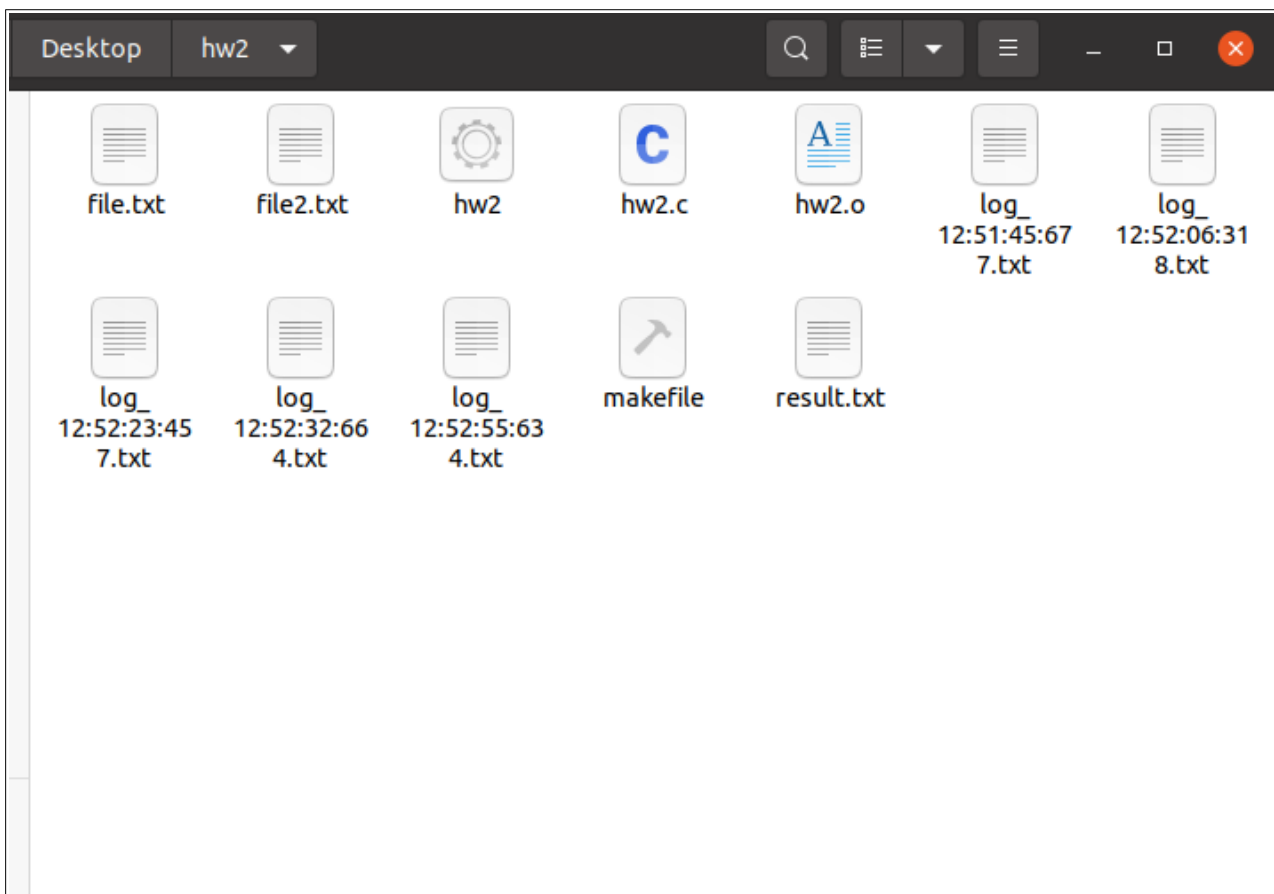
[illegible]

2.8. :q (quit command) and valgring result

Program output for command:

```
myShell> :q
==9372==
==9372== HEAP SUMMARY:
==9372==    in use at exit: 0 bytes in 0 blocks
==9372==   total heap usage: 20 allocs, 20 frees, 10,978 bytes allocated
==9372==
==9372== All heap blocks were freed -- no leaks are possible
==9372==
==9372== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/hw2$
```

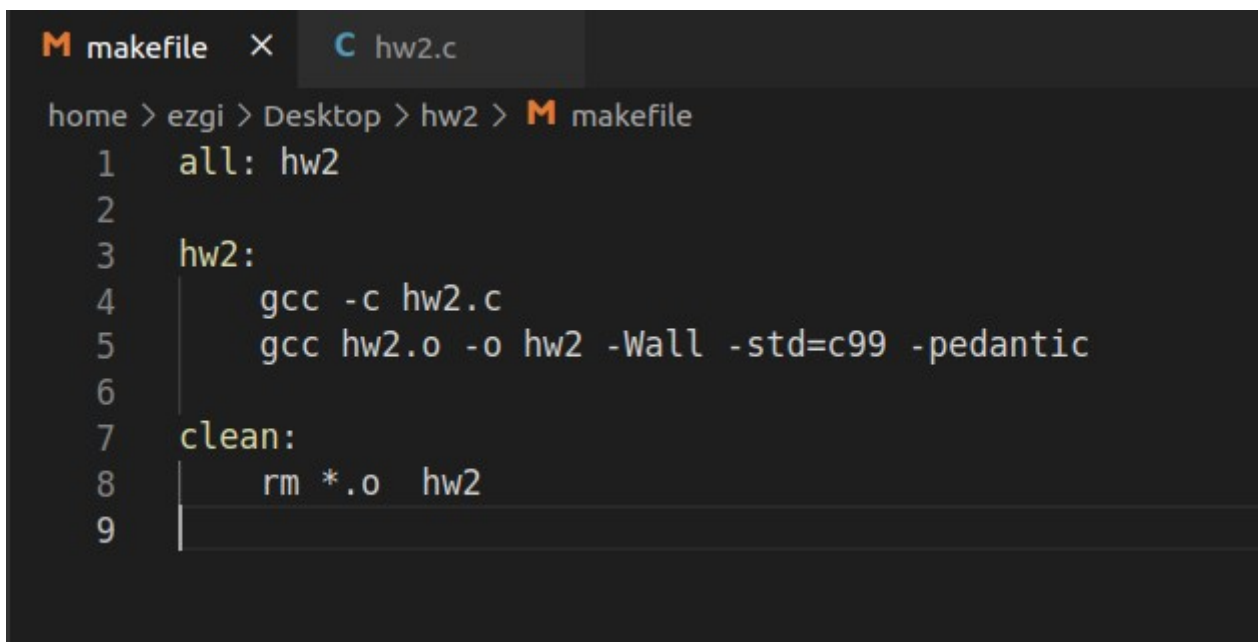
After the execution of the program, folder content:



3. Makefile

You can see makefile content down below. It only compiles the codes with warning flags by “make” command and cleans .o files with “make clean” command.

Note: I didn’t add the valgrind because it doesn’t work without the executable file. You can test it as “valgrind -s ./hw2” (Also the result of valgrind is in the part 2.8)



```
M makefile  X  C hw2.c
home > ezgi > Desktop > hw2 > M makefile
1  all: hw2
2
3  hw2:
4      gcc -c hw2.c
5      gcc hw2.o -o hw2 -Wall -std=c99 -pedantic
6
7  clean:
8      rm *.o  hw2
9
```



```
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/hw2$ make clean
rm *.o  hw2
ezgi@ezgi-Lenovo-ThinkBook-16p-Gen-2:~/Desktop/hw2$ make
gcc -c hw2.c
gcc hw2.o -o hw2 -Wall -std=c99 -pedantic
```