CSE344 – System Programming

HW2

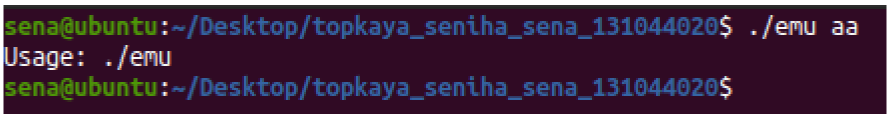
Report

- Seniha Sena Topkaya
- 131044020

Emu:

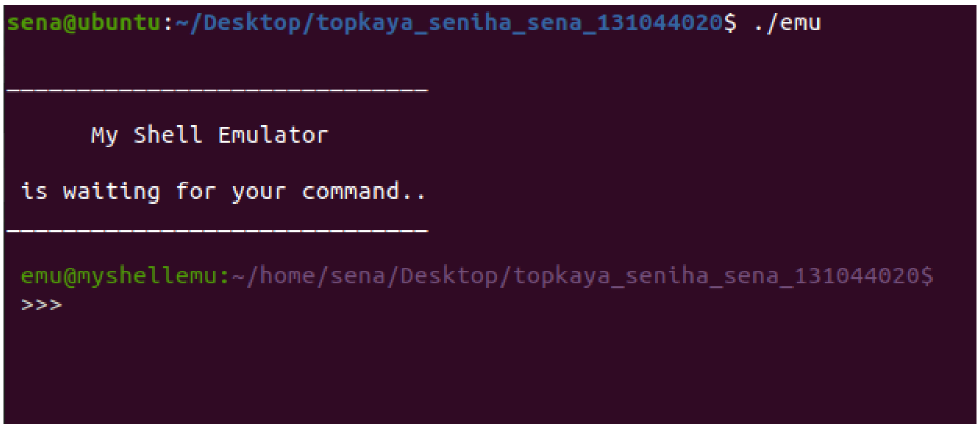
If command line arguments are invalid, print usage information and exit:

I check all the arguments and if they are invalid then I call printUsage function to print valid input format.



```
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$ ./emu aa
Usage: ./emu
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$
```

When execute emu.c file the terminal looks like:



```
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$ ./emu

______________________________

      My Shell Emulator

 is waiting for your command..
______________________________

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>
```

Algorithm Design and Explanation

I developed this project on vmware - ubuntu 20.04.

main():

First of all, I checked for the correct execution of the program and printed the usage in the wrong state. To be able to handle the signals, I first created the signal mask and added the signals to the mask. After printing my input screen, I started the endless loop. (colors are used when printing here. I hope it won't give an error :D)

In an infinite loop:

- I got the commands. Since I'm using fgets, I replaced the last character with a Null character so that it's a valid input. I checked whether the entered input is ":q" or sigkill, so I terminated the program.
- I found the current time and created a log file and closed the file.
- I checked if there is a special operator (<,>,|) character in the input and kept this query in a flag.

- If there is no special operator and only one command is entered; In this case, I called the **commandExe()** function, which can run a single command.
- If there is a private operatör; In this case, since there are more than one command, I split the commands according to the pipe and kept the commands separately in the allCommands array. If the number of commands exceeds 20, the program terminates.
  - If there is one command; In this case I called the **commandDetect()** function which can detect the command.
  - If the number of commands is more than one and an even number; In this case, I called the **myPipe()** function, giving the commands in pairs as parameters.
  - If the number of commands is more than one and odd; In this case, I called the **myPipe()** function, giving the commands in pairs as parameters. I called the **commandDetect()** function to detect the only last command left.

- I did a few signal checks using the sigaction function.

commandDetect():

In this function, I first checked whether there is an input or output redirection in the command and kept the control result in the flag.

- If the flag is 0, it means there is only one command. Therefore, I called the **commandExe()** function, which runs a single command.
- If flag is 1, it means there is output redirection. That's why I split the command according to the '>' character. Then I split the second command according to the space character so that the file name does not start with a space. Finally, I called the **outRedirect()** function by giving these two commands as parameters.
- If the flag is 2, it means there is an input redirection. That's why I split the command according to the '<' character. Then I split the second command according to the space character so that the file name does not start with a space. Finally, I called the **inpRedirect()** function by giving these two commands as parameters.

commandExec():

A single shell command runs in this function.

I created a child process by forking, **fork()**

- I added the process id to the globally maintained id array. I wrote the process information to the log file by calling the **logFile()** function. I ran the command using the **execl()** function inside the child process and ended it with exit.
- With the parent process **waitpid()**, I waited for the child process to terminate and I added the id information to the global id array.

outRedirect():

In this function, two shell commands with the '>' character between them work.

I created a child process by using **fork()**.

- In child process, I added the process id to the globally maintained id array. I created the file where the output will be written and duplicated the file descriptor using **dup2()**. I wrote the process information to the log file by calling the **logFile()** function. I ran the command using the **execl()** function inside the child process and ended it with exit.
- With the parent process waitpid(), I waited for the child process to terminate and I added the id information to the global id array.

inpRedirect():

In this function, two shell commands with a '<' character between them are executed.

I created a child process using **fork()**.

- In the child process, I added the Process id to the globally held id array. I opened the file to read the input from and duplicated the file descriptor using **dup2()**. I wrote the process information to the log file by calling the **logFile()** function. I ran the command using the execl() function inside the child process and ended it with exit.
- With the parent process **waitpid()**, I waited for the child process to terminate and I added the id information to the global id array.

myPipe():

This function has 2 commands as arguments. First I created a pipe with **pipe()** to establish a connection between the two commands.

- For the first command, I created a child process using **fork()**. In this child process, I added the Process id to the globally held id array. I wrote the process information to the log file by calling the **logFile()** function. Then I duplicated the pipe's write end using **dup2()**. Then I called the **commandDetect()** function for the first command. Finally, I closed both ends of the pipe and ended with exit.

- For the second command, I created a child process using **fork()**. In this child process, I added the Process id to the globally held id array. I wrote the process information to the log file by calling the **logFile()** function. Then I duplicated the pipe's read end using **dup2()**. Then I called the **commandDetect()** function for the second command. Finally, I closed both ends of the pipe and ended with exit.

- With the parent process **waitpid()**, I waited for the child processes to terminate and I added the id information to the global id array for both forks.

killHandler():

This function is triggered only if SIGKILL signal comes and kills all globally held process ids one by one using **kill()**. It closes the open log file and ends.

signalHandler():

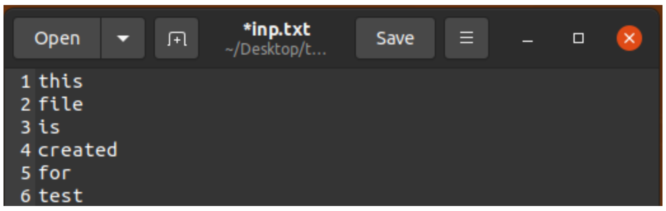
In this function, 'signal captured' information is printed for the captured signal. The signals it contains are Sigkill, Sigint, Sigterm and Sigtstp. **killHandler()** is called only when sigkill is caught.

logFile():

In this function, the log file is opened and the process id and command information is written and the file is closed.

Test cases:

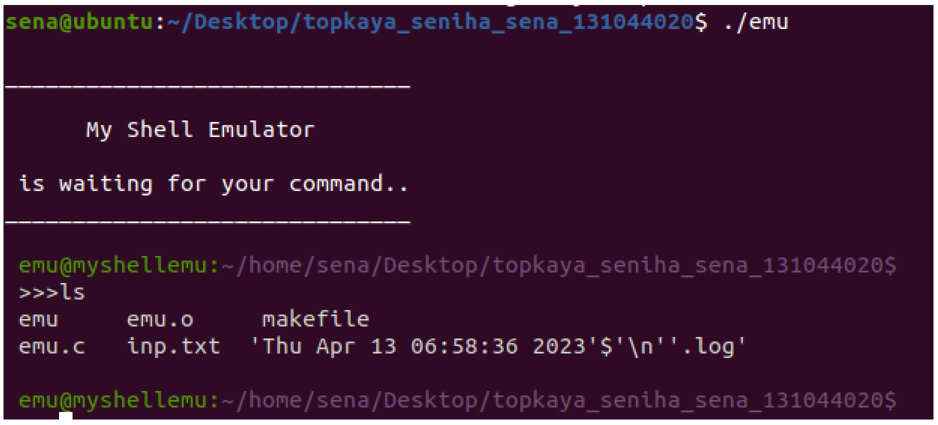
İnput file that I use for test:



```
this
file
is
created
for
test
```

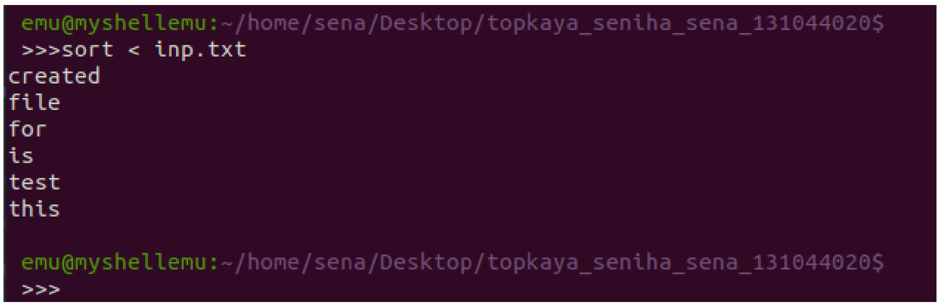
I tested my emullator with different inputs:

- Command: **ls**



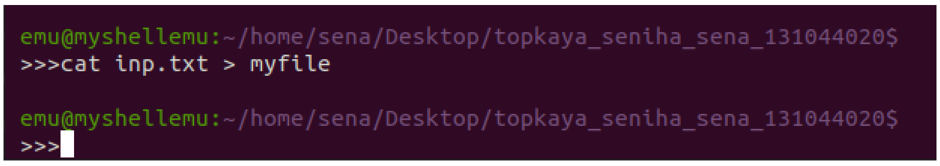
```
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$ ./emu

------------------------------

      My Shell Emulator

 is waiting for your command..
------------------------------

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>ls
 emu      emu.o      makefile
 emu.c    inp.txt   'Thu Apr 13 06:58:36 2023'$'\n''.log'

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
```

- Command: **sort < inp.txt**



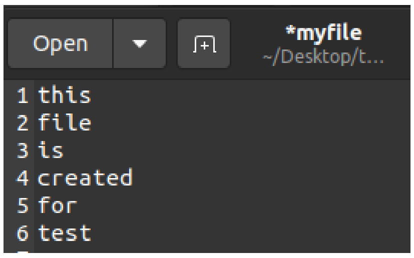
```
 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>sort < inp.txt
created
file
for
is
test
this

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>
```

- Command: **cat inp.txt > myfile**



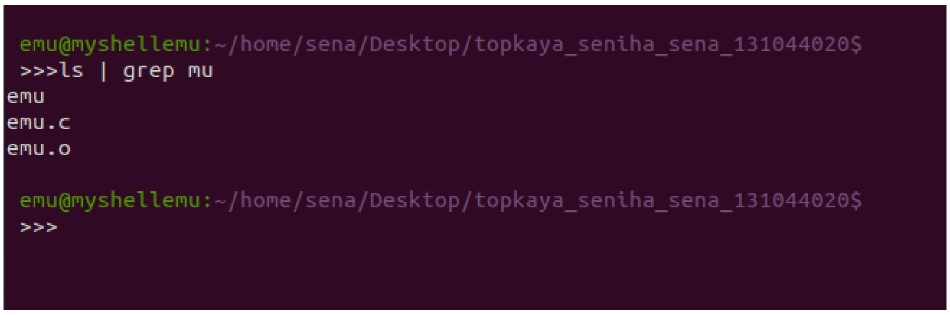
```
 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>cat inp.txt > myfile

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>
```

Myfile ->



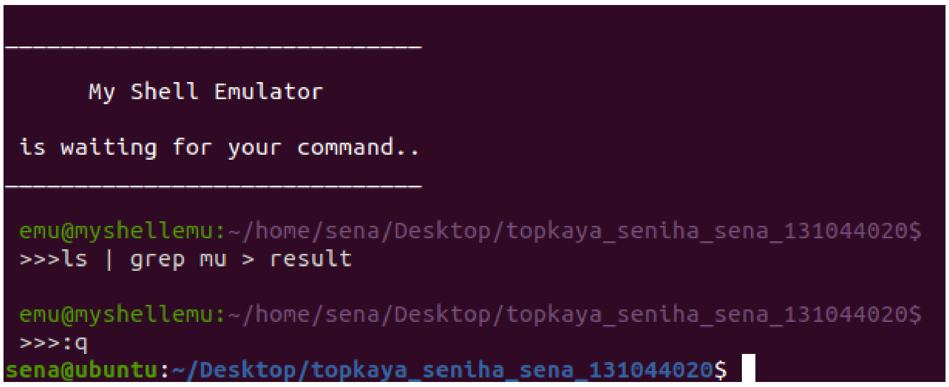
```
this
file
is
created
for
test
```

- Command: **ls | grep mu**



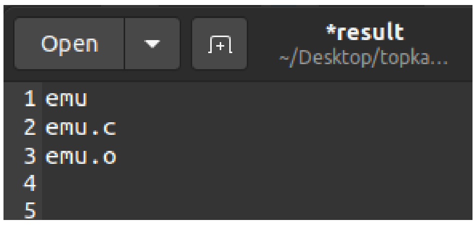
```
 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>ls | grep mu
emu
emu.c
emu.o

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>
```

- Command: **ls | grep mu > result**



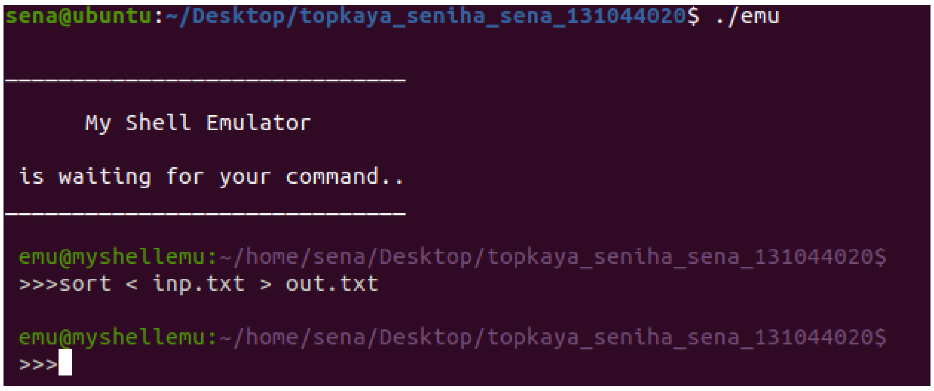
```
________________________________

      My Shell Emulator

 is waiting for your command..
________________________________

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>ls | grep mu > result

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>:q
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$
```

Result->



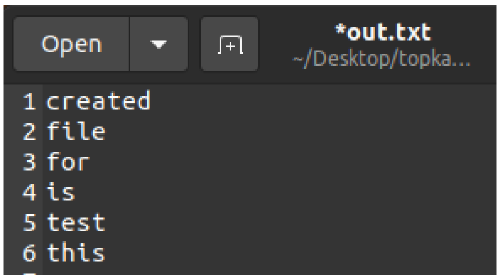
```
emu
emu.c
emu.o
```

- Command: **sort < inp.txt > out.txt**



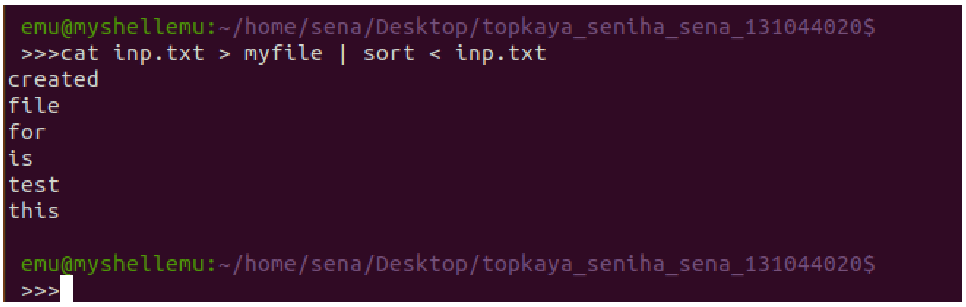
```
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$ ./emu

________________________________

      My Shell Emulator

 is waiting for your command..
________________________________

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>sort < inp.txt > out.txt

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>
```

Out.txt->

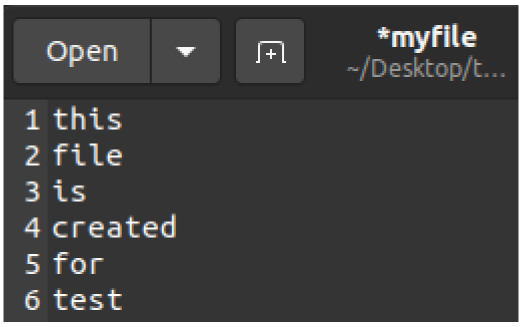


```
created
file
for
is
test
this
```

- Command: **cat inp.txt > myfile | sort < inp.txt**

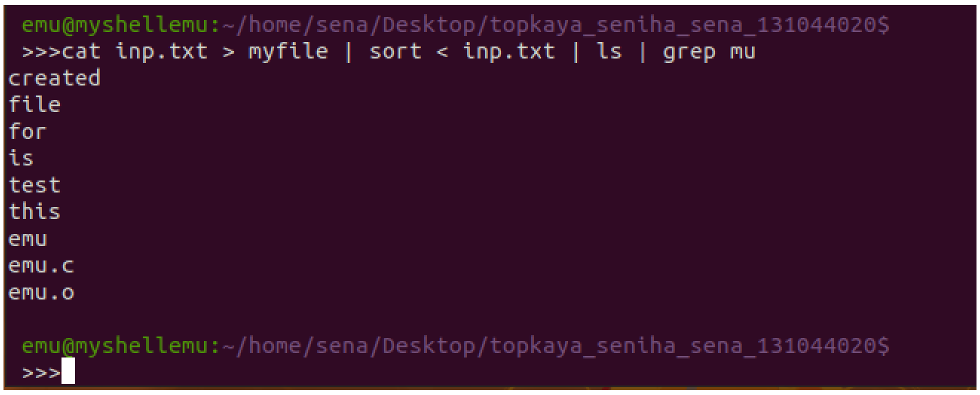


```
 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>cat inp.txt > myfile | sort < inp.txt
created
file
for
is
test
this

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>
```

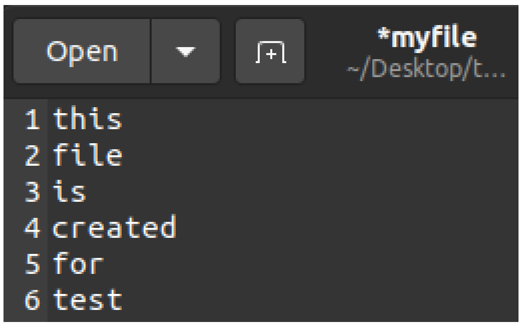
Myfile-> 

```
this
file
is
created
for
test
```

- Command: **cat inp.txt > myfile | sort < inp.txt | ls | grep mu**

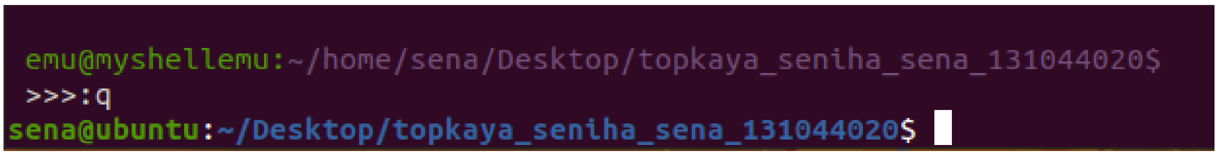


```
 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>cat inp.txt > myfile | sort < inp.txt | ls | grep mu
created
file
for
is
test
this
emu
emu.c
emu.o

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>
```

Myfile-> 

```
this
file
is
created
for
test
```

- Command: **:q**

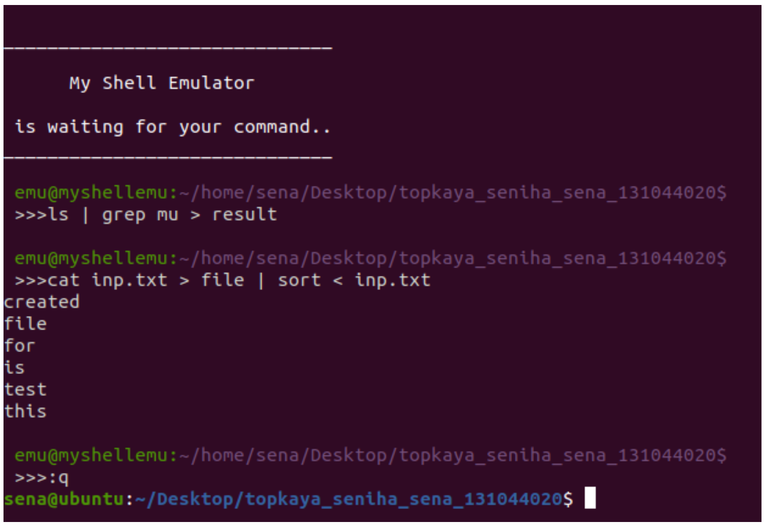


```
 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>:q
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$
```

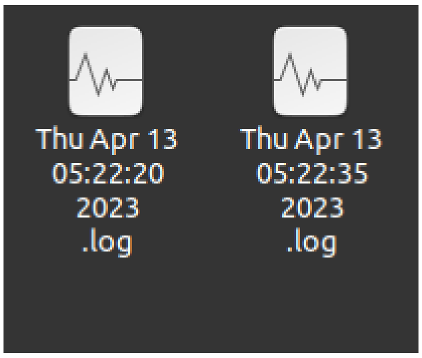
Log File Handling

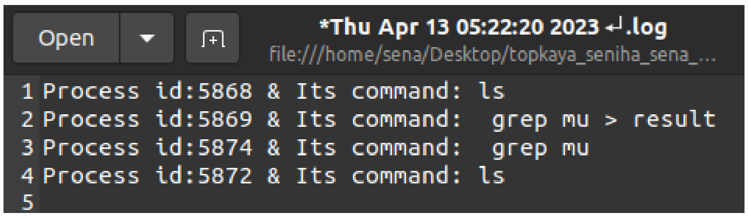
Each execution should create a new log file with a name corresponding to the current timestamp and all pids of child processes with their corresponding commands should be logged in a separate file.

- Command: **ls | grep mu > result**
- Commands: **cat inp.txt > file | sort < inp.txt**

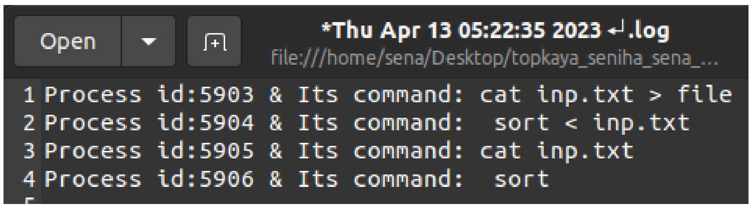


```
------------------------------
      My Shell Emulator

 is waiting for your command..
------------------------------
 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>ls | grep mu > result

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>cat inp.txt > file | sort < inp.txt
created
file
for
is
test
this

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>:q
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$
```




Open
*Thu Apr 13 05:22:20 2023 ↵.log
file:///home/sena/Desktop/topkaya_seniha_sena_...

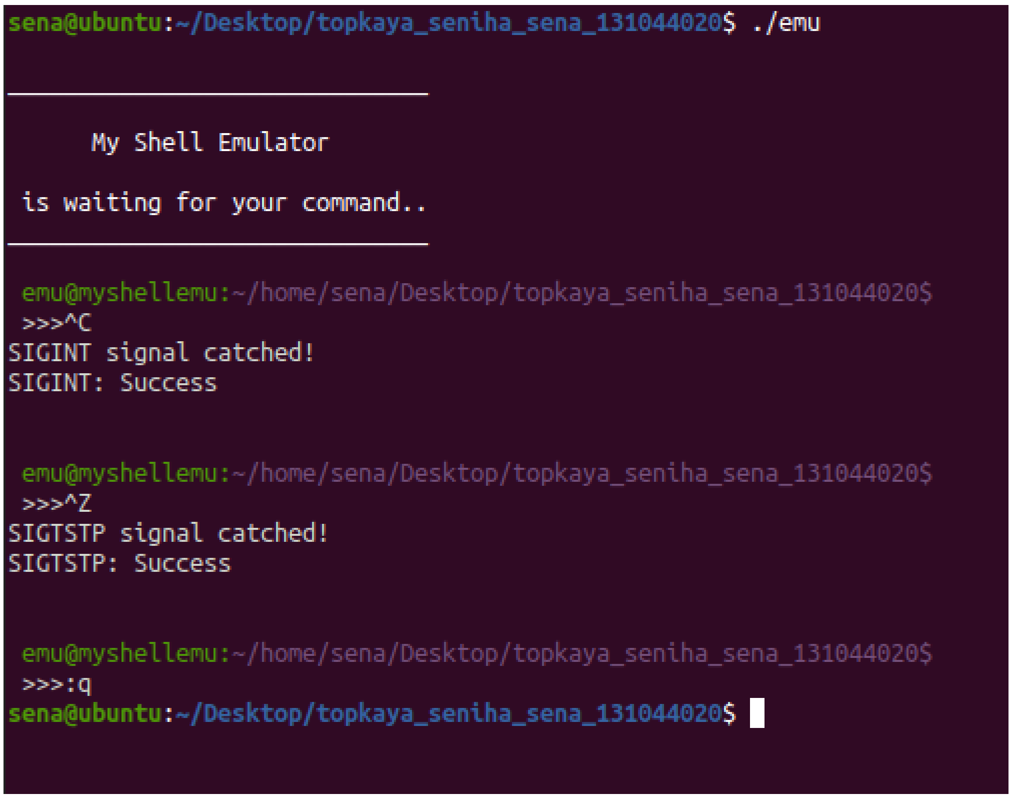
```
Process id:5868 & Its command: ls
Process id:5869 & Its command:  grep mu > result
Process id:5874 & Its command:  grep mu
Process id:5872 & Its command: ls
```


Open
*Thu Apr 13 05:22:35 2023 ↵.log
file:///home/sena/Desktop/topkaya_seniha_sena_...

```
Process id:5903 & Its command: cat inp.txt > file
Process id:5904 & Its command:  sort < inp.txt
Process id:5905 & Its command: cat inp.txt
Process id:5906 & Its command:  sort
```

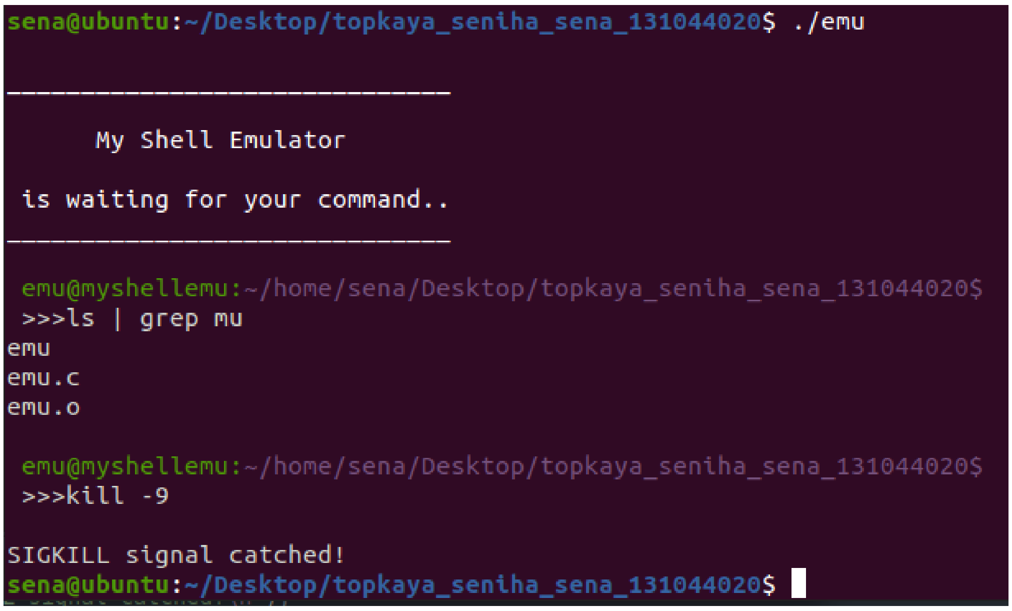
Signal Handling

Aside from a SIGKILL (which also should be handled properly) the program must wait for ":q" to finalize its execution. You should **press Enter** after send a signal. (**YOU NEED TO PRESS ENTER AFTER** Ctrl-C or Ctrl-Z etc. **TO GET BACK TO PROMPT.** Because the algorithm assume the signal as a command)



```
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$ ./emu

______________________________

      My Shell Emulator

 is waiting for your command..
______________________________

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>^C
SIGINT signal catched!
SIGINT: Success


 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>^Z
SIGTSTP signal catched!
SIGTSTP: Success


 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>:q
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$
```

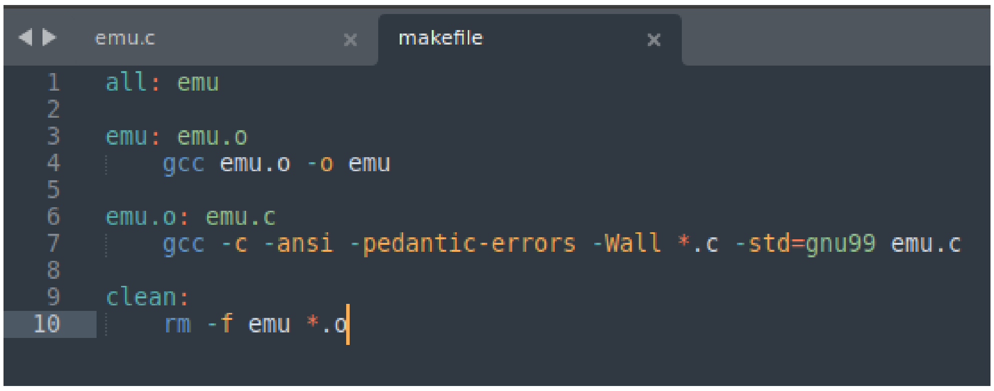
- **SIGKILL signal handled**



```
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$ ./emu

________________________________

      My Shell Emulator

 is waiting for your command..
________________________________

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>ls | grep mu
emu
emu.c
emu.o

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>kill -9

SIGKILL signal catched!
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$
```

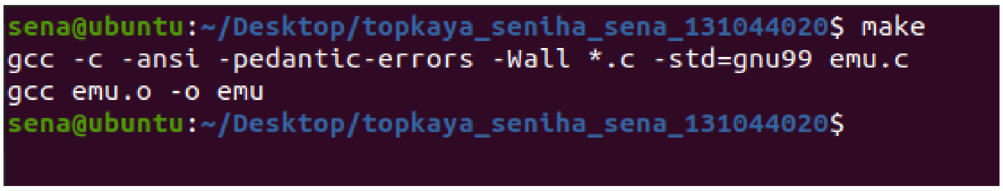
General rules:

- Makefile with make clean that just compile the program not run for all source files.



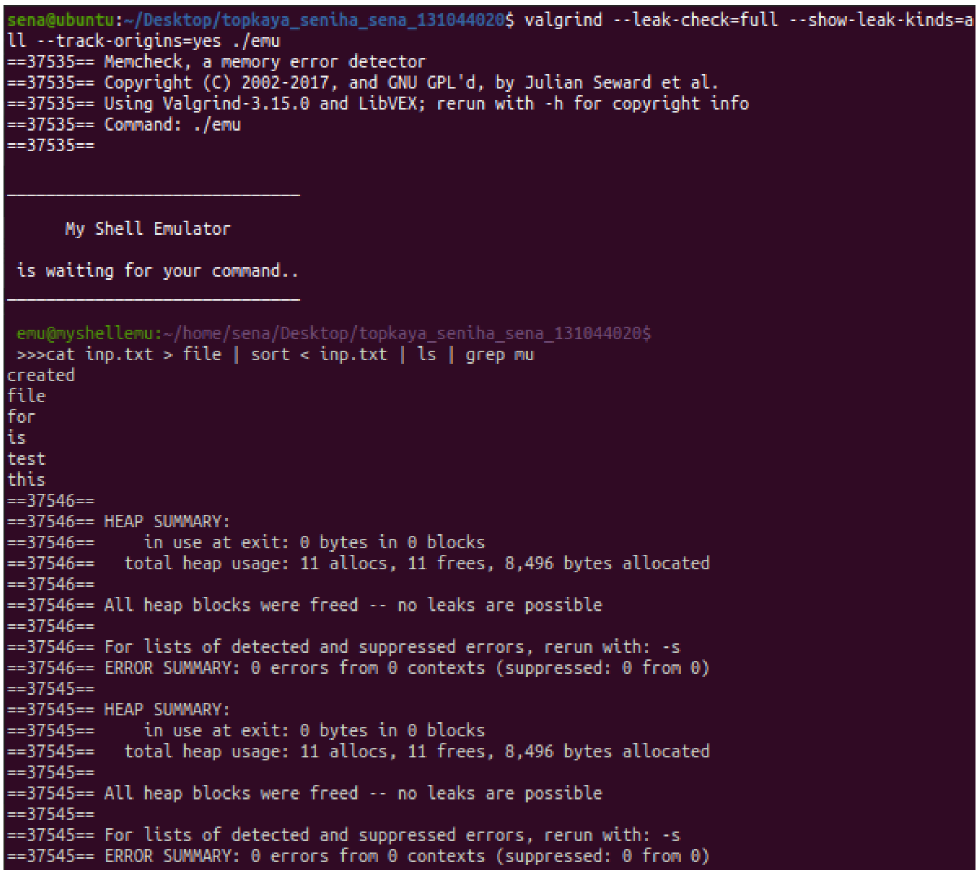
```
all: emu

emu: emu.o
	gcc emu.o -o emu

emu.o: emu.c
	gcc -c -ansi -pedantic-errors -Wall *.c -std=gnu99 emu.c

clean:
	rm -f emu *.o
```

- There is no compilation error



```
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$ make
gcc -c -ansi -pedantic-errors -Wall *.c -std=gnu99 emu.c
gcc emu.o -o emu
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$
```

- There is no memory leak: valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes ./emu



```
sena@ubuntu:~/Desktop/topkaya_seniha_sena_131044020$ valgrind --leak-check=full --show-leak-kinds=a
ll --track-origins=yes ./emu
==37535== Memcheck, a memory error detector
==37535== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==37535== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==37535== Command: ./emu
==37535==

______________________________

      My Shell Emulator

 is waiting for your command..
______________________________

 emu@myshellemu:~/home/sena/Desktop/topkaya_seniha_sena_131044020$
 >>>cat inp.txt > file | sort < inp.txt | ls | grep mu
created
file
for
is
test
this
==37546==
==37546== HEAP SUMMARY:
==37546==     in use at exit: 0 bytes in 0 blocks
==37546==   total heap usage: 11 allocs, 11 frees, 8,496 bytes allocated
==37546==
==37546== All heap blocks were freed -- no leaks are possible
==37546==
==37546== For lists of detected and suppressed errors, rerun with: -s
==37546== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==37545==
==37545== HEAP SUMMARY:
==37545==     in use at exit: 0 bytes in 0 blocks
==37545==   total heap usage: 11 allocs, 11 frees, 8,496 bytes allocated
==37545==
==37545== All heap blocks were freed -- no leaks are possible
==37545==
==37545== For lists of detected and suppressed errors, rerun with: -s
==37545== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

End of the report