# GTU Department of Computer Engineering
# CSE 344 – Spring 2023
# Homework #2 Report

**Hasan Mutlu**
**1801042673**

## Implementation and Program Flow of Main Function

- ➔ Firstly, the signal handler is prepared for SIGINT signals. When the SIGINT signal is received, the program prints the signal number and terminates.
- ➔ Secondly, log.txt file is created and prepared.
- ➔ Thirdly, the piped commands are parsed and  sent to the execute_commands() function to be handled.

## execute_commands() Function

**This is the most important part of the program, where the problem is solved.** It takes an array of commands and quantity of commands as arguments.

It firstly creates an array of pids to record the pids of the children processes it will create, and an array of pipe file descriptor pairs to establish communication between each command's child process. Sizes of these arrays are equal to number of commands.

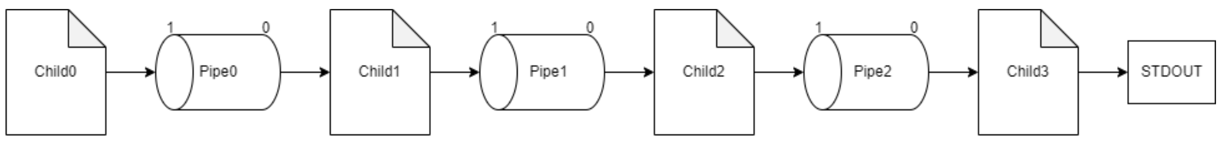
All pipes are created before creating the child processes.

After that, in a loop as much as number of commands, child processes are created. The parent process closes the unused write end of the pipe.

Child processes duplicates STDOUT_FILENO and STDIN_FILENO depending on their position on the command chain, if there are more than one command of course.

- ➔ If the child process executes the first command (i ==0), it only duplicates the STDOUT_FILENO into the next (0th) pipe's write end (1).
- ➔ If the child process executes the last command (i == numCommands – 1), it only duplicates the  STDIN_FILENO into the previous (i-1th) pipe's read end (0).
- ➔ If the child executes one of the commands in the middle, it duplicates STDOUT_FILENO in the next (ith) pipe's write end and also duplicates the STDIN_FILENO into the previous (i-1th) pipe's read end (0).
- ➔ After all duplications, the duplicated file descriptor is closed.

After these steps, a communication structure as follows is formed between the child processes:



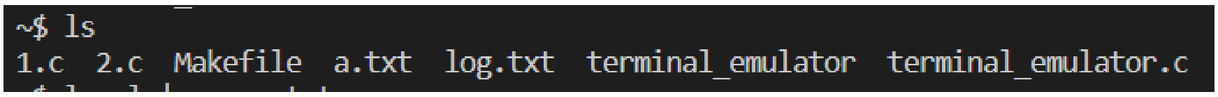
After creating the required connection, the child processes execute their commands using the “execl("/bin/sh", "sh", "-c", commands[i],  (char *) NULL)” function and then the child process terminates.

After creating child processes, the parent process waits for its children to terminate using waitpid() function, using the saved pid numbers in the pids[] array. After each children is terminated, the pid and the command of that chil process executed is printed to the log file.

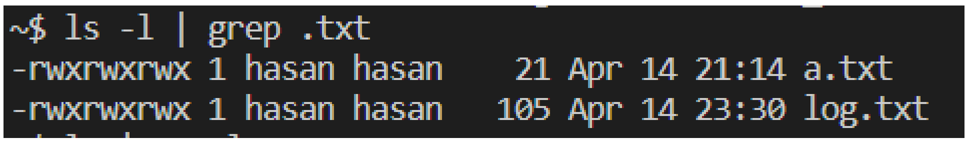
## Requirements Met

- ➔ Mutliple piped commands in shell are handled. However, redirections couldn’t be exclusively handled.
- ➔ Usage information is printed properly.
- ➔ Error messages and SIGINT signals are printed.
- ➔ Pids of child processes and corresponding commands are logged to a file.,

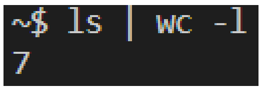
## Example Running Results



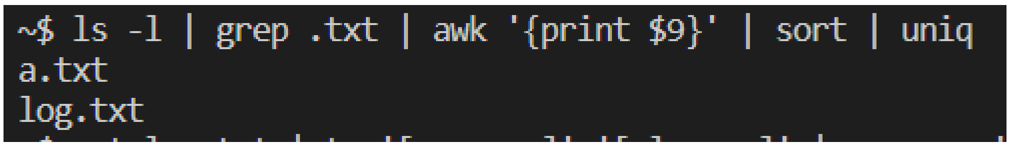
```
~$ ls
1.c  2.c  Makefile  a.txt  log.txt  terminal_emulator  terminal_emulator.c
```



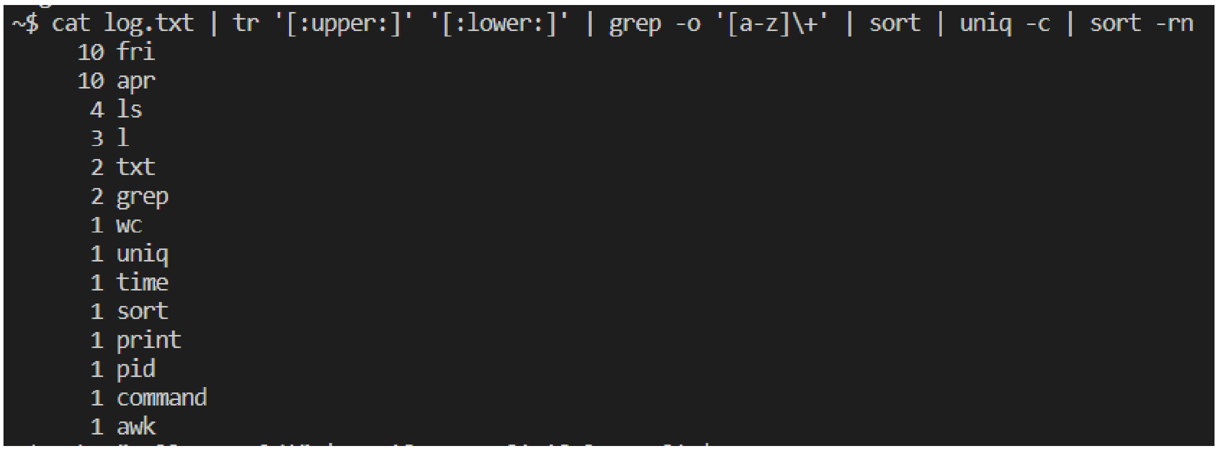
```
~$ ls -l | grep .txt
-rwxrwxrwx 1 hasan hasan    21 Apr 14 21:14 a.txt
-rwxrwxrwx 1 hasan hasan   105 Apr 14 23:30 log.txt
```



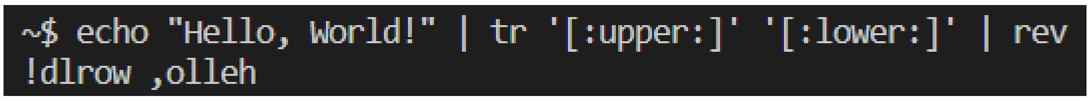
```
~$ ls | wc -l
7
```



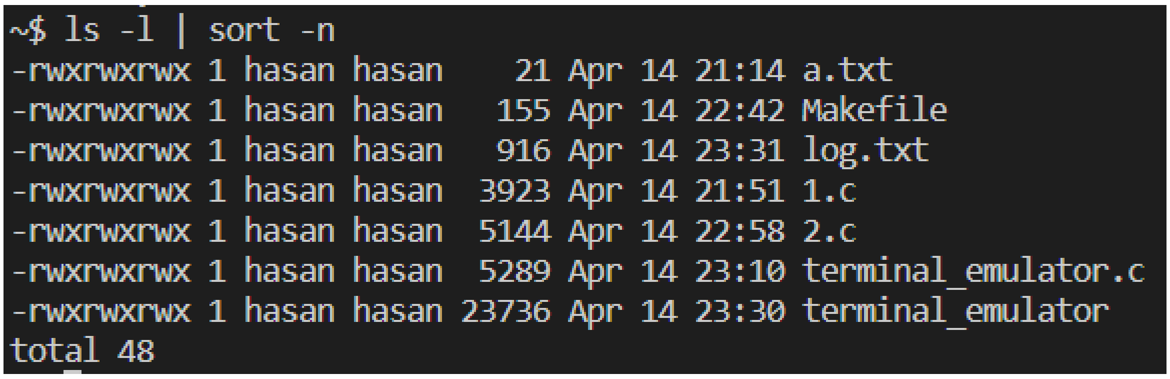
```
~$ ls -l | grep .txt | awk '{print $9}' | sort | uniq
a.txt
log.txt
```



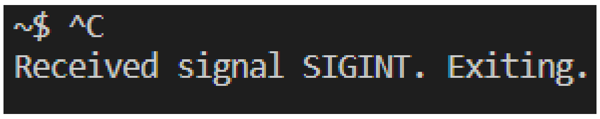
```
~$ cat log.txt | tr '[:upper:]' '[:lower:]' | grep -o '[a-z]\+' | sort | uniq -c | sort -rn
     10 fri
     10 apr
      4 ls
      3 l
      2 txt
      2 grep
      1 wc
      1 uniq
      1 time
      1 sort
      1 print
      1 pid
      1 command
      1 awk
```



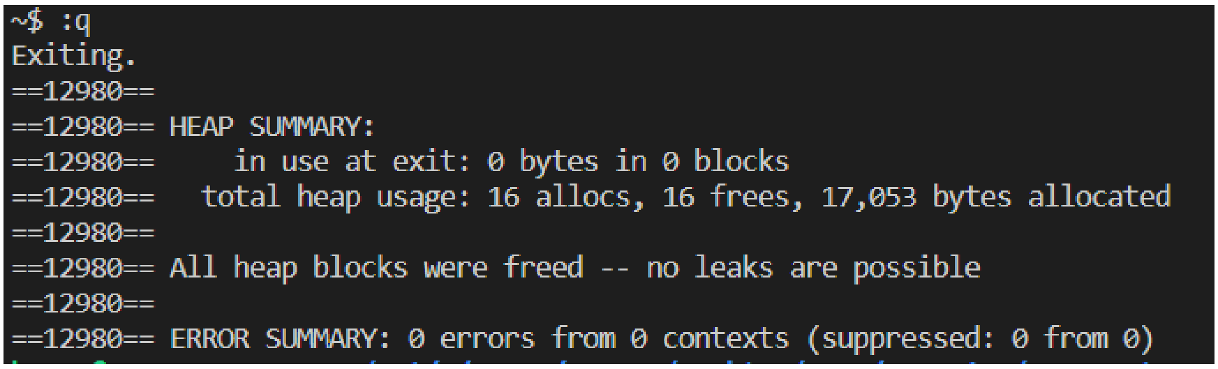
```
~$ echo "Hello, World!" | tr '[:upper:]' '[:lower:]' | rev
!dlrow ,olleh
```



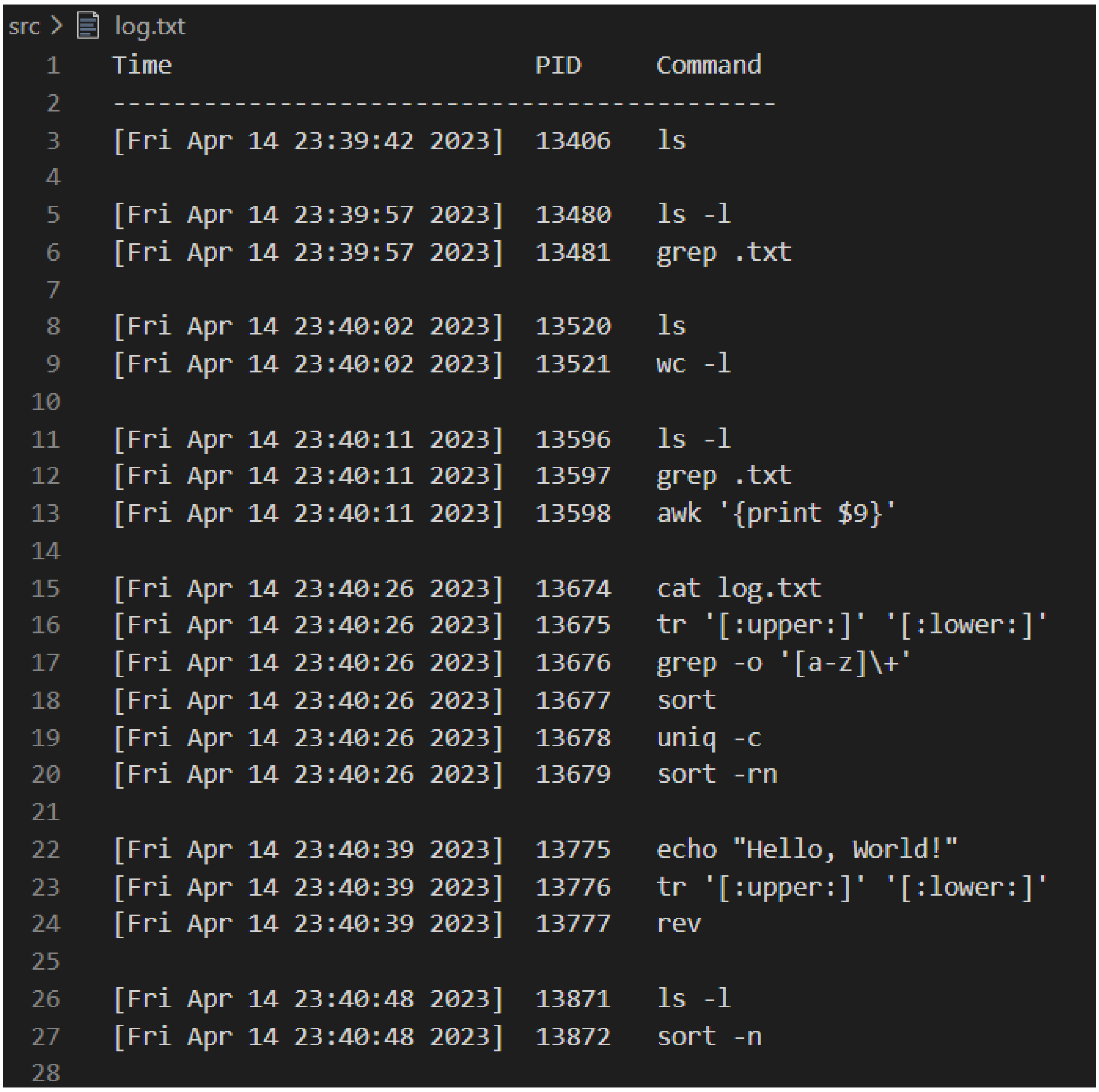
```
~$ ls -l | sort -n
-rwxrwxrwx 1 hasan hasan    21 Apr 14 21:14 a.txt
-rwxrwxrwx 1 hasan hasan   155 Apr 14 22:42 Makefile
-rwxrwxrwx 1 hasan hasan   916 Apr 14 23:31 log.txt
-rwxrwxrwx 1 hasan hasan  3923 Apr 14 21:51 1.c
-rwxrwxrwx 1 hasan hasan  5144 Apr 14 22:58 2.c
-rwxrwxrwx 1 hasan hasan  5289 Apr 14 23:10 terminal_emulator.c
-rwxrwxrwx 1 hasan hasan 23736 Apr 14 23:30 terminal_emulator
total 48
```



```
~$ ^C
Received signal SIGINT. Exiting.
```



```
~$ :q
Exiting.
==12980==
==12980== HEAP SUMMARY:
==12980==     in use at exit: 0 bytes in 0 blocks
==12980==   total heap usage: 16 allocs, 16 frees, 17,053 bytes allocated
==12980==
==12980== All heap blocks were freed -- no leaks are possible
==12980==
==12980== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```



```
Time                        PID     Command
--------------------------------------------
[Fri Apr 14 23:39:42 2023]  13406   ls

[Fri Apr 14 23:39:57 2023]  13480   ls -l
[Fri Apr 14 23:39:57 2023]  13481   grep .txt

[Fri Apr 14 23:40:02 2023]  13520   ls
[Fri Apr 14 23:40:02 2023]  13521   wc -l

[Fri Apr 14 23:40:11 2023]  13596   ls -l
[Fri Apr 14 23:40:11 2023]  13597   grep .txt
[Fri Apr 14 23:40:11 2023]  13598   awk '{print $9}'

[Fri Apr 14 23:40:26 2023]  13674   cat log.txt
[Fri Apr 14 23:40:26 2023]  13675   tr '[:upper:]' '[:lower:]'
[Fri Apr 14 23:40:26 2023]  13676   grep -o '[a-z]\+'
[Fri Apr 14 23:40:26 2023]  13677   sort
[Fri Apr 14 23:40:26 2023]  13678   uniq -c
[Fri Apr 14 23:40:26 2023]  13679   sort -rn

[Fri Apr 14 23:40:39 2023]  13775   echo "Hello, World!"
[Fri Apr 14 23:40:39 2023]  13776   tr '[:upper:]' '[:lower:]'
[Fri Apr 14 23:40:39 2023]  13777   rev

[Fri Apr 14 23:40:48 2023]  13871   ls -l
[Fri Apr 14 23:40:48 2023]  13872   sort -n

```