CSE 344 System Programming

Homework #2

14.04.2023

Yusuf Arslan

200104004112

Problem Definition:

In this homework, you are expected to develop a terminal emulator capable of handling up to 20 shell commands in a single line, without using the "system()" function from the standard C library. Instead, you should utilize the “fork()”, “execl()”, “wait()”, and “exit()” functions.

Your terminal emulator should include the following features:
• Each shell command should be executed via a newly created child process, meaning that multiple commands will result in multiple child processes.
• Proper handling of pipes ("|") and redirections ("<", ">") by redirecting the appropriate file descriptors.
• Usage information should be printed if the program is not called properly.
• Error messages and signals that occur during execution should be printed, and the program should return to the prompt to receive new commands.
• Aside from a SIGKILL (which also should be handled properly) the program must wait for “:q” to finalize its execution.
• Upon completion, all pids of child processes with their corresponding commands should be logged in a separate file. Each execution should create a new log file with a name corresponding to the current timestamp.

Make sure to test your program with multiple shell commands in /bin/sh to ensure its functionality.

Problem Solution Approach:

First, I started with dividing the problem into smaller pieces.

I wrote a shell simulator program that works with only 1 command. I have completed the required signal handling and log printing topics in the homework before proceeding to step 2.

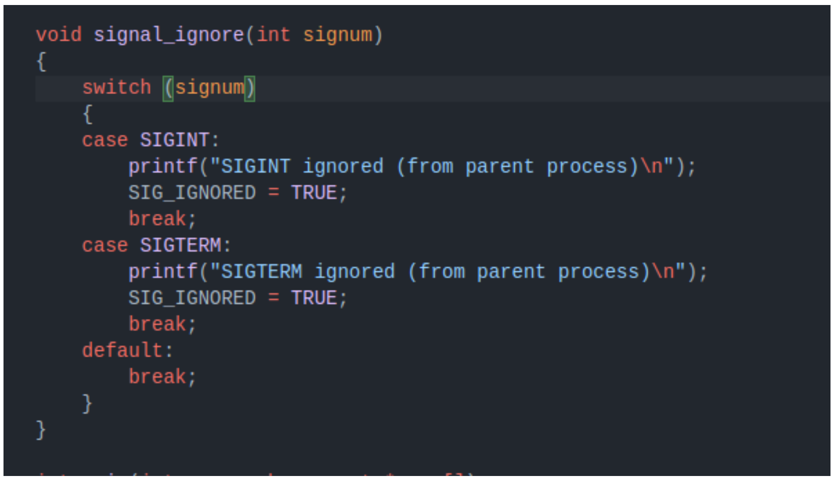
Afterwards, I completed entire program by adding pipes and redirections.

Signal Handling:

Using the sigaction function, I ignore the SIGINT and SIGTERM signals before each new command.

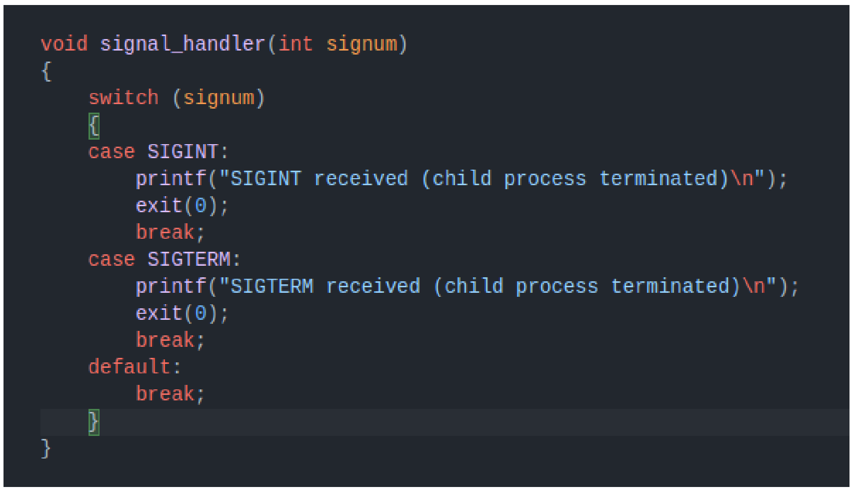
Then, I change the sa_handler function in the children I created with fork() so that these two signals can exit the child proccesses.

Signal handler for parents:



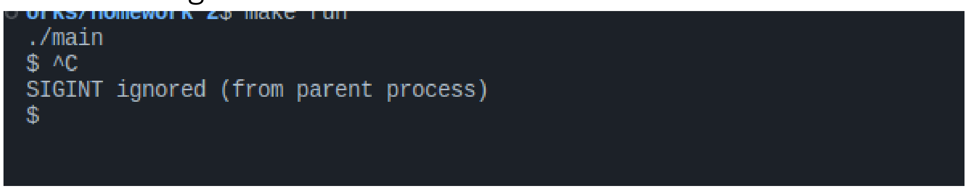
```
void signal_ignore(int signum)
{
    switch (signum)
    {
    case SIGINT:
        printf("SIGINT ignored (from parent process)\n");
        SIG_IGNORED = TRUE;
        break;
    case SIGTERM:
        printf("SIGTERM ignored (from parent process)\n");
        SIG_IGNORED = TRUE;
        break;
    default:
        break;
    }
}
```

Signal handler for childs:



```
void signal_handler(int signum)
{
    switch (signum)
    {
    case SIGINT:
        printf("SIGINT received (child process terminated)\n");
        exit(0);
        break;
    case SIGTERM:
        printf("SIGTERM received (child process terminated)\n");
        exit(0);
        break;
    default:
        break;
    }
}
```

Console Image:



```
orks/homework 2$ make run
./main
$ ^C
SIGINT ignored (from parent process)
$
```

Command Execution:

I handled this in two ways:

1- Single Command: e.g. “ls -l > out”

In this case, I create a child process with fork() and execute the command with the execvp() system call.

But if there are IO redirections it need to be handled. I separate each word into tokens and check if there is a "<" or ">" in it. I redirect inputs by changing STDOUT or STDIN if any.

2- Multiple Command: e.g. “ls -l | sort”

In this case, I needed the pipe system call. I created pipes as much as one less than the number of commands. I was able to create a long pipe by copying the file descriptions of the first command's input and the last command's output, then turning off unnecessary file descriptors.

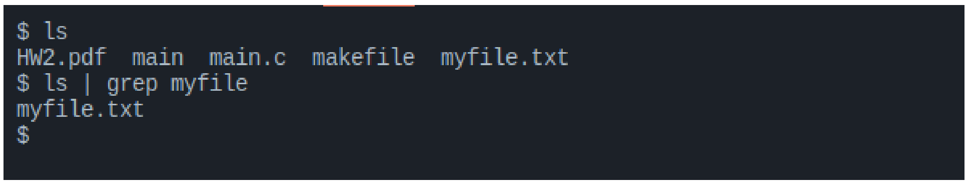
I also created a redirection operation similar to the one for pipe.

Log files

When a child operation finishes, the parent waits for itself with wait(). After the wait operation done, the log file is being created with the current timestamp file name and what the pid and command are added to it.

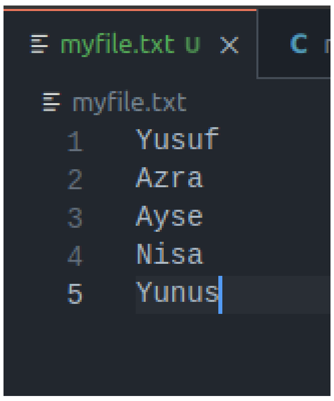
Tests:

`ls` and `ls | grep myfile`



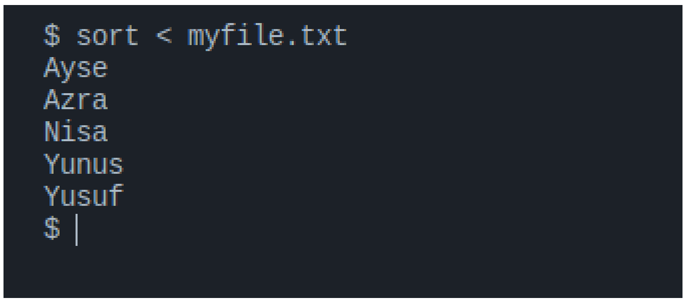
```
$ ls
HW2.pdf  main  main.c  makefile  myfile.txt
$ ls | grep myfile
myfile.txt
$
```

` sort < myfile.txt `



```
Yusuf
Azra
Ayse
Nisa
Yunus
```

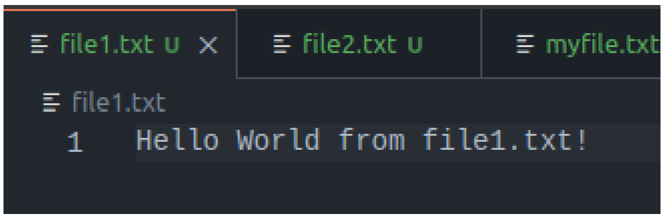
myfile.txt



```
$ sort < myfile.txt
Ayse
Azra
Nisa
Yunus
Yusuf
$ |
```

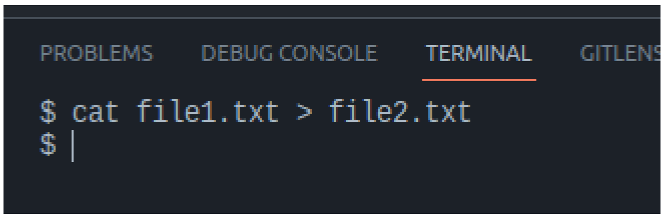
console

` cat file1.txt > file2.txt `



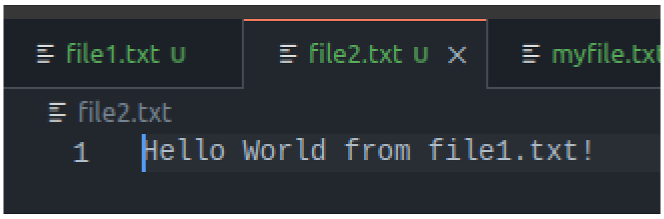
```
Hello World from file1.txt!
```

file1.txt



```
$ cat file1.txt > file2.txt
$ |
```

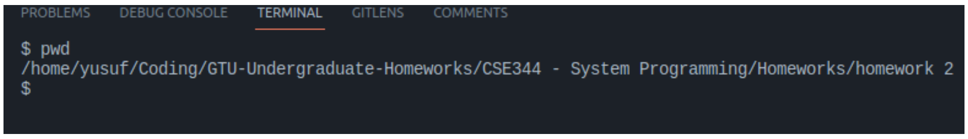
console



```
Hello World from file1.txt!
```

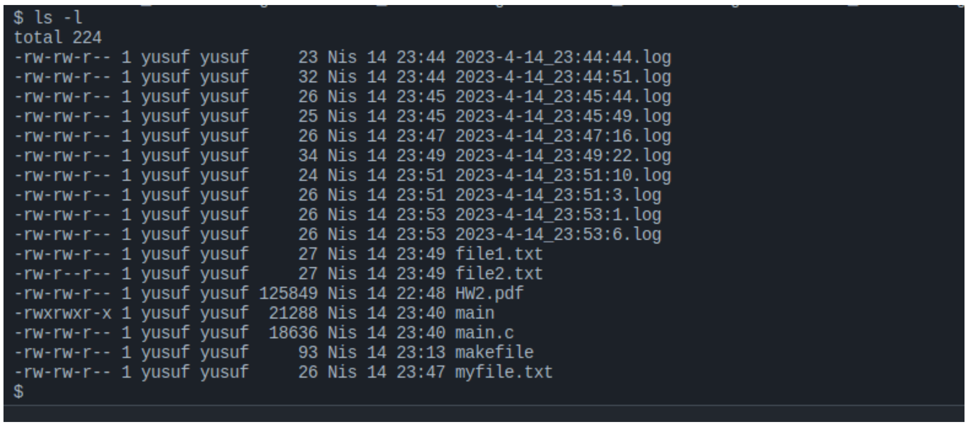
file2.txt

` pwd `



```
$ pwd
/home/yusuf/Coding/GTU-Undergraduate-Homeworks/CSE344 - System Programming/Homeworks/homework 2
$
```

` ls -l ` after operations



```
$ ls -l
total 224
-rw-rw-r-- 1 yusuf yusuf     23 Nis 14 23:44 2023-4-14_23:44:44.log
-rw-rw-r-- 1 yusuf yusuf     32 Nis 14 23:44 2023-4-14_23:44:51.log
-rw-rw-r-- 1 yusuf yusuf     26 Nis 14 23:45 2023-4-14_23:45:44.log
-rw-rw-r-- 1 yusuf yusuf     25 Nis 14 23:45 2023-4-14_23:45:49.log
-rw-rw-r-- 1 yusuf yusuf     26 Nis 14 23:47 2023-4-14_23:47:16.log
-rw-rw-r-- 1 yusuf yusuf     34 Nis 14 23:49 2023-4-14_23:49:22.log
-rw-rw-r-- 1 yusuf yusuf     24 Nis 14 23:51 2023-4-14_23:51:10.log
-rw-rw-r-- 1 yusuf yusuf     26 Nis 14 23:51 2023-4-14_23:51:3.log
-rw-rw-r-- 1 yusuf yusuf     26 Nis 14 23:53 2023-4-14_23:53:1.log
-rw-rw-r-- 1 yusuf yusuf     26 Nis 14 23:53 2023-4-14_23:53:6.log
-rw-rw-r-- 1 yusuf yusuf     27 Nis 14 23:49 file1.txt
-rw-r--r-- 1 yusuf yusuf     27 Nis 14 23:49 file2.txt
-rw-rw-r-- 1 yusuf yusuf 125849 Nis 14 22:48 HW2.pdf
-rwxrwxr-x 1 yusuf yusuf  21288 Nis 14 23:40 main
-rw-rw-r-- 1 yusuf yusuf  18636 Nis 14 23:40 main.c
-rw-rw-r-- 1 yusuf yusuf     93 Nis 14 23:13 makefile
-rw-rw-r-- 1 yusuf yusuf     26 Nis 14 23:47 myfile.txt
$
```