CSE 344
SYSTEM PROGRAMMING

HOMEWORK 2
REPORT

OGUZ MUTLU
1801042624

1. PROBLEM DEFINITION

In this homework, we 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.

- Each shell command executed with newly created child process
- Handling of pipes (“ | ”) and redirections(“<”,“>”)
- Usage information should be printed
- Error message and signal handling.
- :q command will terminate our shell
- All child processes with their corresponding commands should be logged in a separate file.

2. PROBLEM SOLUTION APPROACH

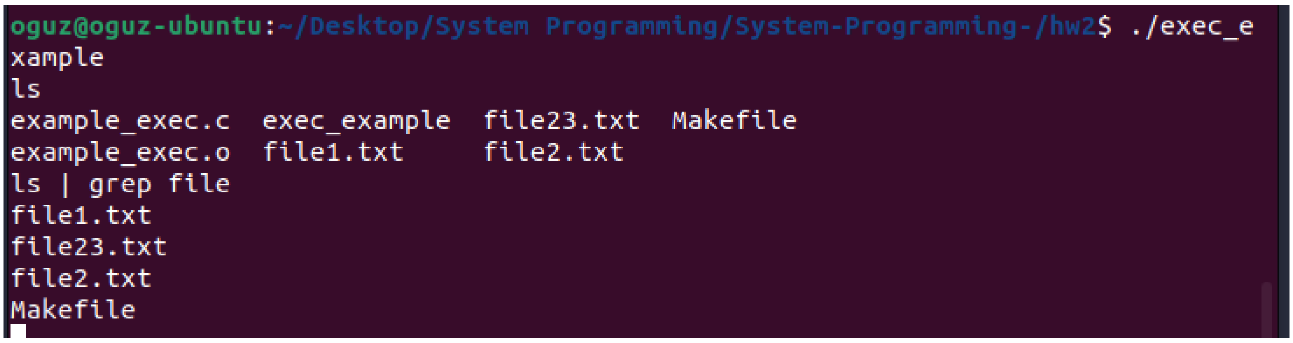
1.1 First think that we had to think about was about determining the pipe state that will occur between the process and child

1.2 Searched and read documentation about pipeline dup2 exec fork wait.

1.3 System calls and variable forms to be used in the code were searched. Afterwards, examples of the use of these changes were examined.

1.4 The algorithm to be used in the code content has been designed will be explained in implementation part.

3.IMPLEMENTATION



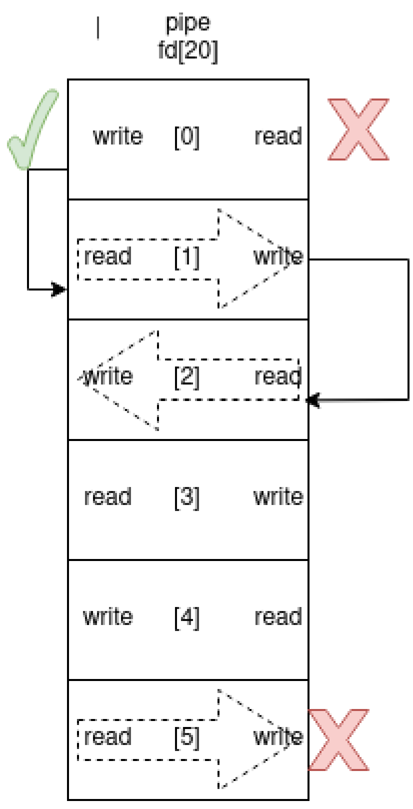
```
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw2$ ./exec_e
xample
ls
example_exec.c  exec_example  file23.txt  Makefile
example_exec.o  file1.txt     file2.txt
ls | grep file
file1.txt
file23.txt
file2.txt
Makefile
```

Initially, it was thought that the program should be run with an infinite loop, so it was designed in such a way that the terminal emulator takes inputs endlessly within the while loop using the read() system call.
Afterwards, the program will be terminated with a specific instruction instead of an infinite loop by performing signal handling in the while.

Since the value sample received from the user is like this, the entered value must be parsed first. Parsed with strtok.

First, I divided all the pieces into blank space values and saved them dynamically in a 2-dimensional char array, then I defined them as static values due to the segment errors that occurred.

After obtaining information about how the pipe will work,The general process-child relationship was examined and the construction of the structure to be formed was provided according to this general situation.



The program works like this: first with strtok | values are parsed, then these parsed values are stored in an array. Each of these stored values is thought to represent a command.

As these commands are called, they are stored in the array. Commands are delivered to their required places by keeping a counter.

After the necessary pipe is created, if the queue is processing the first process, first of all, the child process is created with the help of fork()

The read descriptor of the first pipe value is turned off, the write descriptor is duplicated.

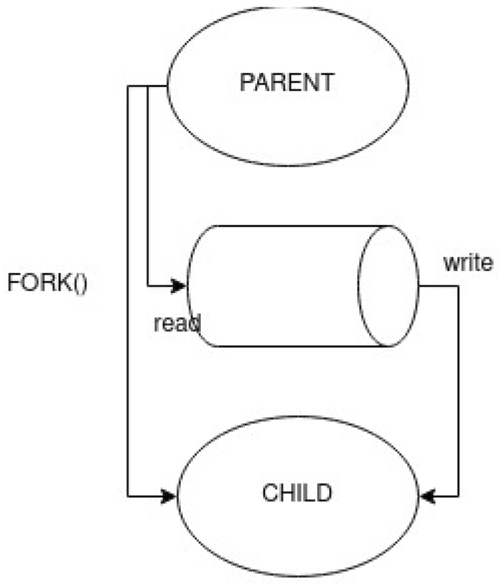
This write value written to the STD_OUT value must be read by the next process, so after the necessary file descriptor shutdown operations are done, the child process ends and a new child process is created.

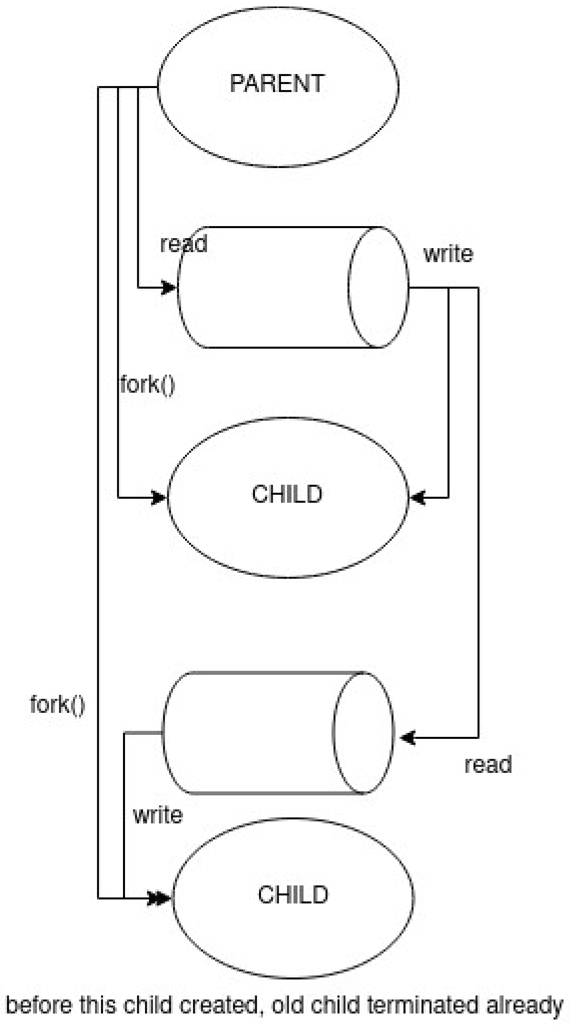
By showing the pfd[i-1] of the new process created, it accesses the output of the previous file descriptor and reads this value by keeping the read descriptor open.

In the last operation, the only thing to do is to take the process of the previous process from the read pipe.

If else constructs are made according to the above-mentioned.
The fork operation is controlled by the switch case structure without using any auxiliary functions. I define fixed size of command array which is [50][100] in total.

The normal parent child relationship is as shown left, the parent child relationship I use for my code is like the one on the top page, its more detailed explanation is given in the screenshot at the right bottom of this page. The bottom process has been created and visualized for two child processes.



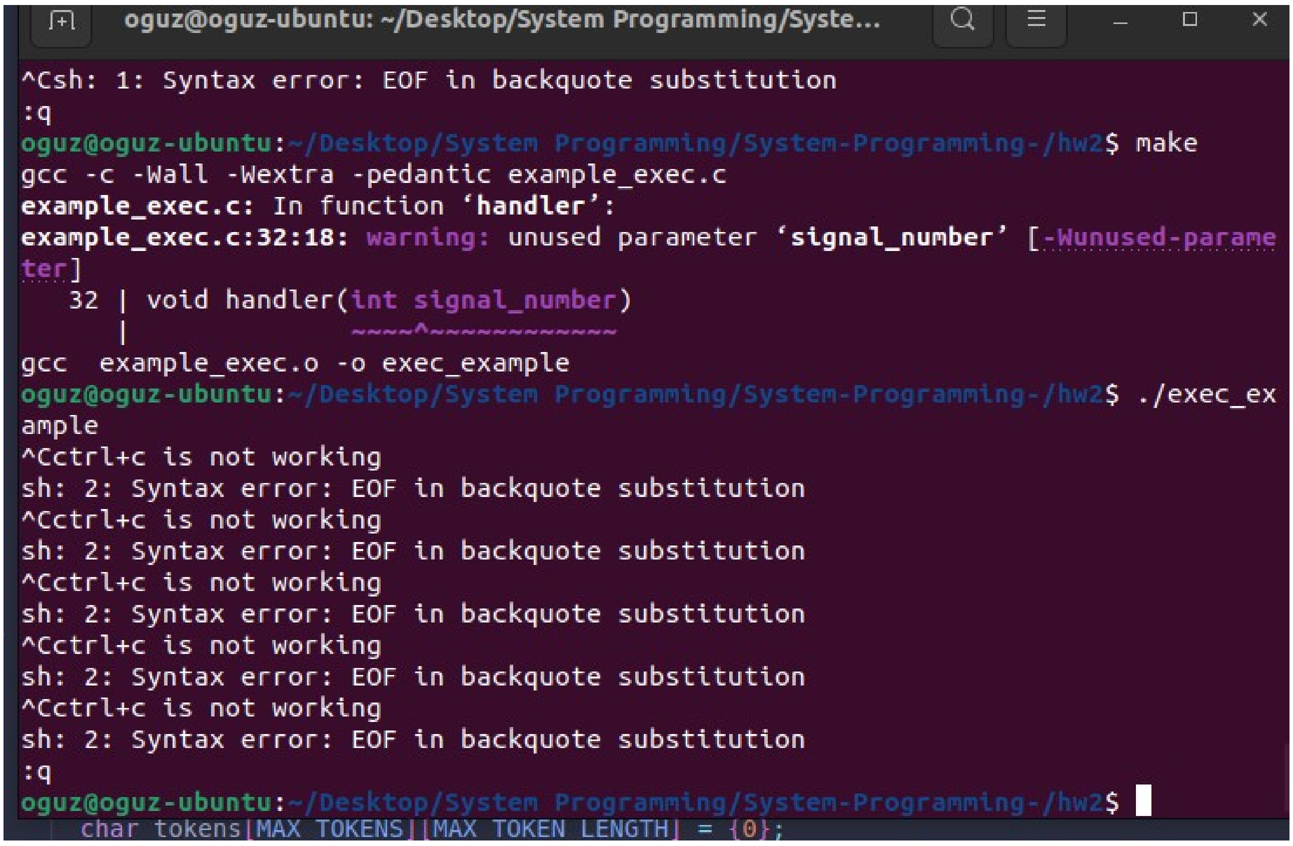


In addition, when :q is taken, the program must be terminated. For this, the SIGINT signal is handled by using the signal handler and this process is terminated.

4.TEST CASES AND RESULTS

1. Signal handling

Our system will only quit with :q command ctrl+c (SIGINT) is not exit the shell

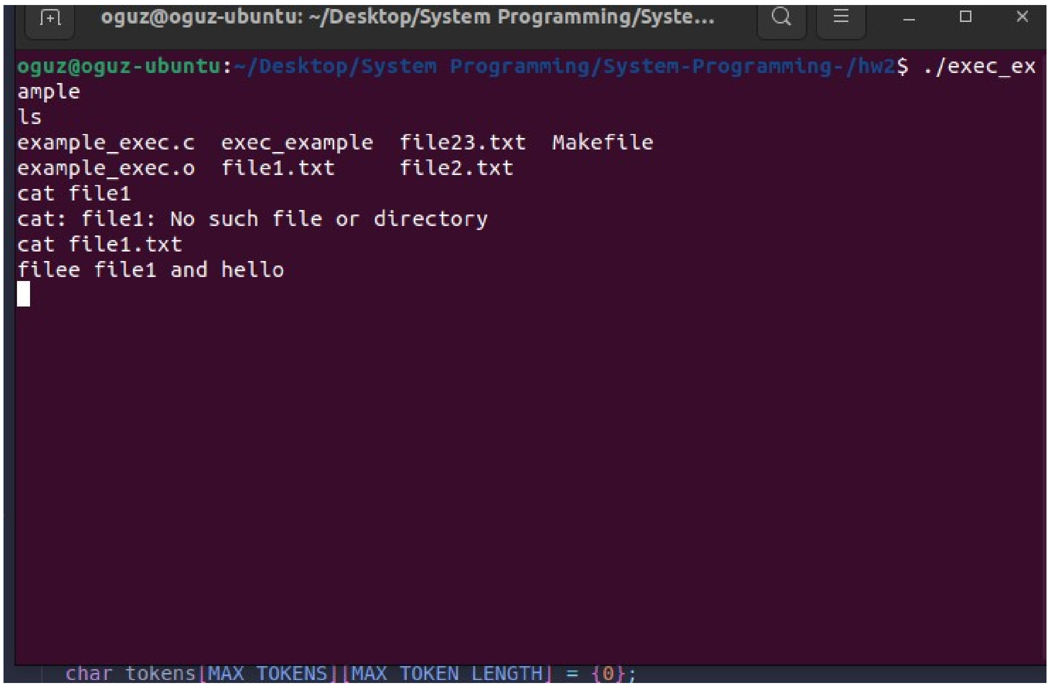


oguz@oguz-ubuntu: ~/Desktop/System Programming/Syste...

```
^Csh: 1: Syntax error: EOF in backquote substitution
:q
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw2$ make
gcc -c -Wall -Wextra -pedantic example_exec.c
example_exec.c: In function ‘handler’:
example_exec.c:32:18: warning: unused parameter ‘signal_number’ [-Wunused-parame
ter]
   32 | void handler(int signal_number)
      |              ~~~~^~~~~~~~~~~~~
gcc  example_exec.o -o exec_example
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw2$ ./exec_ex
ample
^Cctrl+c is not working
sh: 2: Syntax error: EOF in backquote substitution
^Cctrl+c is not working
sh: 2: Syntax error: EOF in backquote substitution
^Cctrl+c is not working
sh: 2: Syntax error: EOF in backquote substitution
^Cctrl+c is not working
sh: 2: Syntax error: EOF in backquote substitution
^Cctrl+c is not working
sh: 2: Syntax error: EOF in backquote substitution
:q
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw2$
    char tokens[MAX_TOKENS][MAX_TOKEN_LENGTH] = {0};
```

2. 1 command without pipe

You can see cat and ls command without any pipe.

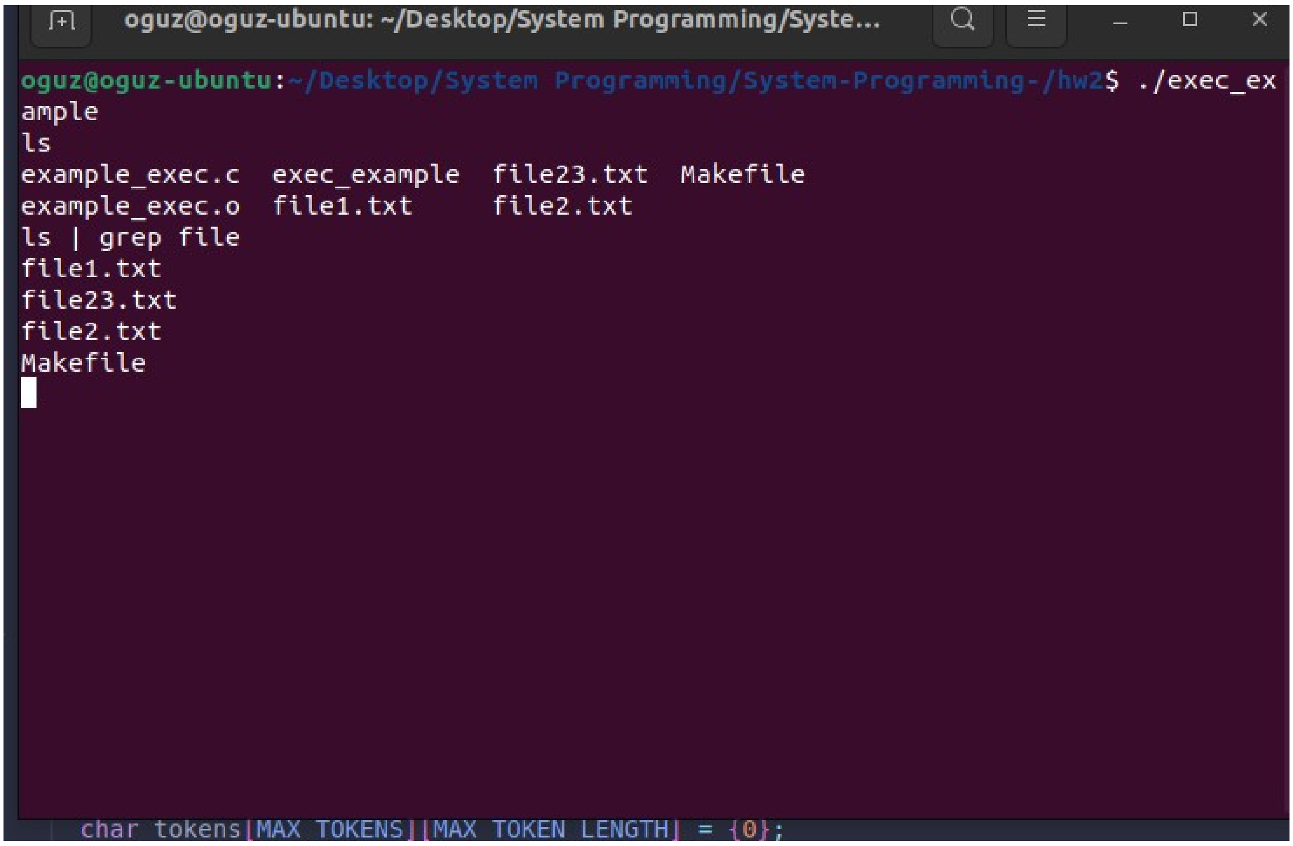


oguz@oguz-ubuntu: ~/Desktop/System Programming/Syste...

```
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw2$ ./exec_ex
ample
ls
example_exec.c  exec_example  file23.txt  Makefile
example_exec.o  file1.txt     file2.txt
cat file1
cat: file1: No such file or directory
cat file1.txt
filee file1 and hello

    char tokens[MAX_TOKENS][MAX_TOKEN_LENGTH] = {0};
```

3. 2 command with one pipe
You can see ls and ls | grep file result with one pipe operator



```
oguz@oguz-ubuntu: ~/Desktop/System Programming/Syste...
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw2$ ./exec_ex
ample
ls
example_exec.c  exec_example  file23.txt  Makefile
example_exec.o  file1.txt     file2.txt
ls | grep file
file1.txt
file23.txt
file2.txt
Makefile

    char tokens[MAX TOKENS][MAX TOKEN LENGTH] = {0};
```

5. CONCLUSION AND NOTES

I did not achieve to reach end of the homework my results work on only pipe tokenization and only 2 commands included, with 3 pipe command my homework gives bad file descriptor but at the end of this error gives true result I could not recognize which file descriptor that I did not close. But signal handling part and making fork and pipe workes properly.