

HW2 REPORT

Ozan Çolak

1901042645

The implementation is done with two source code files. The first and main file is '*terminal.c*' and the second file for executing the shell commands is '*my_system.c*'.

Terminal.c:

The '*terminal.c*' source code starts by checking the usage of program calling, and if it is wrong, it returns an error message and prints out the proper usage. If everything is done correctly, the program will wait for the users input to execute. It will take the stdin and send the input to the mySystem in a loop, until the user enters 'q', which terminates the whole program. Then it will check the status to handle any occurring error and if not, it will be ready for the new input.

```
while(1){
    printf("$ ");
    fflush(stdout);
    if(fgets(str, MAX_CMD_LEN, stdin)==NULL)
        break;
    else if(strcmp(str, ":q\n") == 0)
        break;

    status = mySystem(str);

    if(status == -1){
        printf("Process has encountered an error!\n");
        exit(-1);
    }
}
```

Signal Handling of Terminal.c:

The terminal handles and changes the behavior of some signals (such as SIGINT, SIGTSTP, SIGTERM and SIGQUIT) to make them unable to finalize the execution. This is done by *sigaction* function, when one of the signals that is handled is received, the program will print out the signal and the indication that the program can not be stopped with the given signal.

```
struct sigaction sa;
memset(&sa, 0, sizeof(sa));
sa.sa_handler = &handleSig;
sa.sa_flags = SA_RESTART;
if (sigemptyset(&sa.sa_mask) == -1){
    printf("Signal handlers failed.\n");
}
else{
    sigaction(SIGINT, &sa, NULL);
    sigaction(SIGTSTP, &sa, NULL);
    sigaction(SIGTERM, &sa, NULL);
    sigaction(SIGQUIT, &sa, NULL);
}

54 void handleSig(int sig){
55     if(sig==2)
56         printf("\nYou can not terminate with interrupt
(SIGINT) signal.\n$ ");
57     else if(sig==3)
58         printf("\nYou can not terminate with quit (SIGQUIT)
signal.\n$ ");
59     else if(sig == 15)
60         printf("\nYou can not terminate with terminate
(SIGTERM) signal.\n$ ");
61     else if(sig == 15)
62         printf("\nYou can not terminate with terminate
(SIGTERM) signal.\n$ ");
63 }
```

My_system.c:

Firstly, the program will tokenize the given command by separating with pipe '|' symbol. It will create separate pipes for each command, this is needed to communicate between child processes.

```
//Creating pipes for all commands
int fd[comCount][2];
for(i = 0; i < comCount; i++){
    if(pipe(fd[i]) == -1){
        printf("Pipe error!\n");
        return -1;
    }
}
```

Then, it will fork the parent process for each separate command to handle them individually in a for loop. In the loop, for every child it will run the code below. The parent process will skip it and wait for the childs.

The program uses the index of the current child for reading fd and the index of the next child for writing. For example, if there are 3 commands, the first child (i=0) will write its output to fd[1][1], the second child will take its input from fd[1][0] and write its output to fd[2][1], and the last child will read fd[2][0] and give its output to regular stdout.

```
int j;
for(j = 0; j < comCount; j++){
    if(i != j)
        close(fd[j][0]);
    if(i + 1 != j)
        close(fd[j][1]);
}
//If there is only one command
if(comCount == 1){
    close(fd[i][0]);
}

//If it is the first command
else if(i==0){
    close(fd[i][0]);
    dup2(fd[i+1][1], STDOUT_FILENO);
    close(fd[i+1][1]);
}

//If it is the last command
else if(i==comCount-1){
    dup2(fd[i][0], STDIN_FILENO);
    close(fd[i][0]);

//If it is one of the middle commands
else{
    dup2(fd[i][0], STDIN_FILENO);
    dup2(fd[i+1][1], STDOUT_FILENO);
    close(fd[i][0]);
    close(fd[i+1][1]);
}

logChild(getpid(), commands[i], i+1);
execl("/bin/sh", "sh", "-c", commands[i], (char *) NULL);
```

When the loop starts for if(pid == 0), it will close all the fd's that the current child will not use. These fd's are every one of them except read part of the current fd and write part of the next fd. After that, it will determine which part of the command we are at.

If command count is one, then it will close the only fd that is open and continue without any dup2.

If command count is not one, it will be different. For the first command, it will close the read part of fd[0] and dup2 the output of the execution to fd[1][1]. And then, it will close the fd[1][1]. This is okay because the parent still holds the information of it.

If we are at the last command, it will read from the read part of its fd, which is fd[i][0], make it its input and close it.

If we are at some middle command, it will both read and write. Then it will close both of the fd's.

After completion, we will log the current child in a file and execute the current command using execl().

While these executions of child processes happening, the parent process closed all of its fd's and started to wait for the children processes to end. After all of the child processes are executed, the parent process will finally skip the wait function and return 1 for success.

```
for(i = 0; i < comCount; i++){
    close(fd[i][0]);
    close(fd[i][1]);
}

for (i = 0; i < comCount; i++) {
    wait(NULL);
}

return 1;
```

Signal Handling of my_system.c:

The terminal handles and changes the behavior of some signals (such as SIGINT, SIGTSTP, SIGTERM and SIGQUIT) to make them unable to finalize the execution. This is done by *sigaction* function, when one of the signals that is handled is received, the program will print out the signal and the indication that the program cannot be stopped with the given signal.

```
if(pids[i] == 0){
    //Some signal preperation for childs.
    if (sigemptyset(&sa.sa_mask) == -1){
        printf("Signal handlers failed.\n");
    }
    else{
        sigaction(SIGINT, &sa, NULL);
        sigaction(SIGTSTP, &sa, NULL);
        sigaction(SIGTERM, &sa, NULL);
        sigaction(SIGQUIT, &sa, NULL);
    }
}

void handleSigChild(int sig){
    if(sig==2)
        printf("\nSIGINT received. All child processes has been terminated.
\n");

    else if(sig==3)
        printf("\nSIGQUIT received. All child processes has been terminated.
\n");

    else if(sig == 15)
        printf("\nSIGTERM received. All child processes has been terminated.
\n");

    else if(sig == 20)
        printf("\nSIGTSTP received. All child processes has been terminated.
\n");

    else
        printf("Unknown signal caught.\n$ ");

    fflush(stdout);
    kill(getpid(), SIGKILL); //Killing the child process.
}
```

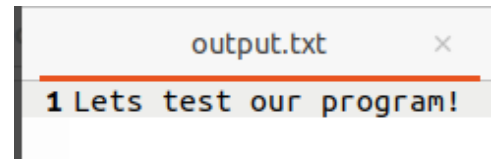
It gives the four signals to sigaction function with sa. The third parameter is null because we don't need to set a new signal handler for it.

When we receive the signal, we print the signal and indicate that all running child processes has been terminated. After that, we flush the stdout and kill the current child with SIGKILL. This will be done for every child for that parent. Then, it will go back to prompt to receive new commands.

Tests:

- Let's test with a basic instruction that uses pipe with 2 commands.

```
==126598== Memcheck, a memory error detector
==126598== Copyright (C) 2002-2017, and GNU GPL'd, by Ju
==126598== Using Valgrind-3.18.1 and LibVEX; rerun with
==126598== Command: ./terminal
==126598==
$ echo "Lets test our program!" | tee output.txt
Lets test our program!
```



```
output.txt
1 Lets test our program!
```

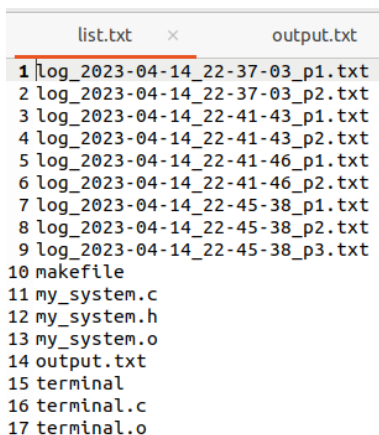
Echo will print "Let's test our program!" and give its output to tee, then tee will both print the message and give its output to "output.txt".

- ls | sort will list all of the files that exists in the current directory, and sort will sort them.

```
$ ls | sort
log_2023-04-14_22-37-03_p1.txt
log_2023-04-14_22-37-03_p2.txt
log_2023-04-14_22-41-43_p1.txt
log_2023-04-14_22-41-43_p2.txt
makefile
my_system.c
my_system.h
my_system.o
output.txt
terminal
terminal.c
terminal.o
```

- Now let's try to copy this information to a file.

```
$ ls | sort | cat > list.txt
```



```
list.txt
1 log_2023-04-14_22-37-03_p1.txt
2 log_2023-04-14_22-37-03_p2.txt
3 log_2023-04-14_22-41-43_p1.txt
4 log_2023-04-14_22-41-43_p2.txt
5 log_2023-04-14_22-41-46_p1.txt
6 log_2023-04-14_22-41-46_p2.txt
7 log_2023-04-14_22-45-38_p1.txt
8 log_2023-04-14_22-45-38_p2.txt
9 log_2023-04-14_22-45-38_p3.txt
10 makefile
11 my_system.c
12 my_system.h
13 my_system.o
14 output.txt
15 terminal
16 terminal.c
17 terminal.o
```

- Taking the list file and filtering to only words with 'my', then printing its word count with wc -w.

```
$ cat list.txt | grep my
my_system.c
my_system.h
my_system.o
$ cat list.txt | grep my | wc -w
3
```

- Reversely sorting the list.txt and putting its output to listReverse.txt

```
$ sort -r < list.txt > listReverse.txt
```

list.txt ×	listReverse.txt ×	list.txt ×	listReverse.txt ×	outp
1 log_2023-04-14_22-37-03_p1.txt	1 terminal.o	1 terminal.o		
2 log_2023-04-14_22-37-03_p2.txt	2 terminal.c	2 terminal.c		
3 log_2023-04-14_22-41-43_p1.txt	3 terminal	3 terminal		
4 log_2023-04-14_22-41-43_p2.txt	4 output.txt	4 output.txt		
5 log_2023-04-14_22-41-46_p1.txt	5 my_system.o	5 my_system.o		
6 log_2023-04-14_22-41-46_p2.txt	6 my_system.h	6 my_system.h		
7 log_2023-04-14_22-45-38_p1.txt	7 my_system.c	7 my_system.c		
8 log_2023-04-14_22-45-38_p2.txt	8 makefile	8 makefile		
9 log_2023-04-14_22-45-38_p3.txt	9 log_2023-04-14_22-45-38_p3.txt	9 log_2023-04-14_22-45-38_p3.txt		
10 makefile	10 log_2023-04-14_22-45-38_p2.txt	10 log_2023-04-14_22-45-38_p2.txt		
11 my_system.c	11 log_2023-04-14_22-45-38_p1.txt	11 log_2023-04-14_22-45-38_p1.txt		
12 my_system.h	12 log_2023-04-14_22-41-46_p2.txt	12 log_2023-04-14_22-41-46_p2.txt		
13 my_system.o	13 log_2023-04-14_22-41-46_p1.txt	13 log_2023-04-14_22-41-46_p1.txt		
14 output.txt	14 log_2023-04-14_22-41-43_p2.txt	14 log_2023-04-14_22-41-43_p2.txt		
15 terminal	15 log_2023-04-14_22-41-43_p1.txt	15 log_2023-04-14_22-41-43_p1.txt		
16 terminal.c	16 log_2023-04-14_22-37-03_p2.txt	16 log_2023-04-14_22-37-03_p2.txt		
17 terminal.o	17 log_2023-04-14_22-37-03_p1.txt	17 log_2023-04-14_22-37-03_p1.txt		

- Trying signals to terminate the program.

```
$ ^C
You can not terminate with interrupt (SIGINT) signal.
$ ^Z
You can not terminate with stop (SIGTSTP) signal.
```

- Trying to write wrong commands.

```
$ runOzan
sh: 1: runOzan: not found
```

- Terminating the program properly with :q

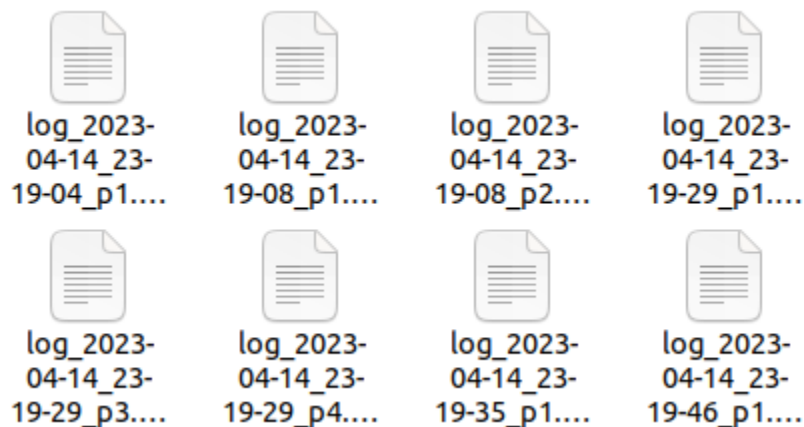
```
$ :q
==126598==
==126598== HEAP SUMMARY:
==126598==      in use at exit: 0 bytes in 0 blocks
==126598==    total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==126598==
==126598== All heap blocks were freed -- no leaks are possible
==126598==
==126598== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

As it can be seen, there is not any memory leak.

```
ozan@ozan-VirtualBox:~/Desktop/system/hw2$ ps -g
  PID TTY          STAT TIME COMMAND
  1905 tty2      Ssl+  0:00 /usr/libexec/gdm-wayl
  1911 tty2      Sl+   0:00 /usr/libexec/gnome-se
123941 pts/0      Ss    0:00 bash
127080 pts/0      R+    0:00 ps -g
```

Also, no zombies have been created.

- Log files are created and waiting to be examined.



End of the tests and the report.