

GEBZE
TEKNİK ÜNİVERSİTESİ



SPRING 2023 – CSE 344 SYSTEM PROGRAMMING

HOMEWORK 2

MERT GÜRŞİMŞİR

1901042646

How to run?

First you should use make command.

Then you are going to have 1 executable files which is "hw2".

You can use it like follows:

```
./hw2
```

After that, you can use "make clean" to delete .o file and executable.

TERMINAL EMULATOR

For this homework, first I carefully examined my notes about creating a terminal that I took at lectures of Erkan Zergeroğlu. I searched about processes on the internet and watch lots of videos about this subject. Commands that I have mostly examined are (with reasons how I have used them):

- fork()
 - I have made research about it to learn how it creates a child process, what is the id of it, how can I use that process, what if we terminate but child process didn't terminate which is the situation for zombies.
 - It creates a child process that is copy of parent process.
 - fork() returns child pid for parent process, 0 for child process.
 - We can use that created child process with using execl command. As we said at the lectures, it is preferable to use explicit fork() and execl() calls to execute the desired program.
- execl() – to execute the command we want
 - As it is stated at the homework, I have used execl function to implement a command.
 - When we are trying to implement system() at lectures we have used execl() function there too so I have used it rather than other exec functions.
 - I gave pathname "/bin/sh" as it is stated at the homework pdf and also this is what we did at lectures.
- wait() – wait for execution of that command to finish
 - This is the function that we MUST do when we are using fork().
 - We are blocked until child returns.
 - I have waited for each child I have created with fork() to return to avoid zombies.
- exit()
 - exec command overwrite to child process so to say.
 - So I have used exit(127) right after the execl function to indicate that we could not exec the command.

system() vs. my code

If I have used `system()`, then at least 2 processes creation will be required. One of these 2 processes is for the shell and one or more for the commands.

In my implementation I have used the same `execl` command that we have used at lectures. I forked the process, give command to `execl`, exit if it fails, wait for child to return, and handle the signals.

Explanation of Solution

As we mentioned at lectures, If we give command to `execl` as a whole, it executes it but this is not the thing that is wanted from us. So I have implement these steps one by one that are designed by me:

1. Tokenize the input from pipes to have array of commands.
2. Create number of commands minus one pieces of pipes to connect commands. One process' stdout will be the stdin of the other process that is at the other end of the pipe.
3. For each command at the input, create a child process with **fork**.
4. For the child process:
 - a. Connect one former process' STDOUT to next process' STDIN using pipes.
 - b. Close all pipe ends to avoid read/writes at the end.
 - c. **Execute** the current command.
 - d. **Exit** when exec fails.
5. At the end of these forkings, upon completion, all pids of child processes, which are created by fork, will be logged in a separate file. Commands are already in the array so iterate through all the commands and write their process ids as well as command name. File name will be the current timestamp. This step will be done only in the parent process. Child processes will only be used for command executions and redirections.
6. Next step is closing all the pipe descriptors in the parent process.
7. The most important step is this which is **waiting** for children. There are command number of children so wait for each of them.
8. Take inputs at the main function and send them to the terminal emulator. If input is `":q"`, then finalize the execution.

At step 5, I faced with a problem that is having the same timestamp for each of the commands at the input. So I have solved it by using a global variable and add it to end of the file name by incrementing by one for each process. At the newly given input, it will be zeroized.

I have go through each steps and implement them one by one and also I have handled the signals. The hardest part for me is definitely the handling the pipes after tokenized the commands. I have tried handling pipes with many `dup2` files but at each time I have done some small mistakes. Although, I have achieved to handle pipes with correct redirections at the end by going through the commands one by one.

Tests

Finalizing

```
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./hw2
Welcome dear user to my terminal emulator. Enjoy!


Command $> :q

Thanks for using! Good-bye!
```

One command and log file

```
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ./hw2
Welcome dear user to my terminal emulator. Enjoy!

Command $> ls
' test.txt '          'General-20230414_142130-Meeting Recording.mp4'          'Meeting in _General_-20230411_104715-Meeting Recording.mp4'
20230414_22-28-52_0.txt 'General-20230414_150653-Meeting Recording.mp4'          Report.docx
'3. Sinif'            Hw2.pdf                                                    Verify.md5
90s                  Makefile                                                    Verify.sfv
Computer             'Meeting in _General_-20230411_103620-Meeting Recording.mp4' 'Virtual Machines'
```

 20230414_22-28-52_0 - Notepad


File Edit Format View Help

CHILD: Command: ls, Process ID: 1408

Commands with pipe


```
Command $> ls -l | wc
37      339      2477
```

```
cabanel@MSI:/mnt/c/Users/Mert/Desktop$ ls -l | wc
37      339      2477
```

 20230414_22-31-07_0 - Notepad

File Edit Format View Help

CHILD: Command: ls -l , Process ID: 1412

 20230414_22-31-07_1 - Notepad

File Edit Format View Help

CHILD: Command: wc, Process ID: 1413


Commands with pipes

```
Command $> ls -l | sort -n | grep t | wc -l  
25
```

 20230414_22-33-22_0 - Notepad


File Edit Format View Help

CHILD: Command: `ls -l` , Process ID: 1427

 20230414_22-33-22_1 - Notepad


File Edit Format View Help

CHILD: Command: `sort -n` , Process ID: 1428

 20230414_22-33-22_2 - Notepad

File Edit Format View Help

CHILD: Command: `grep t` , Process ID: 1429

 20230414_22-33-22_3 - Notepad

File Edit Format View Help

CHILD: Command: `wc -l`, Process ID: 1431

Command with redirection

```
Command $> whoami | wc -c > mert.txt
```

 20230414_22-41-01_0 - Notepad


File Edit Format View Help

CHILD: Command: `whoami` , Process ID: 1445

 20230414_22-41-01_1 - Notepad

File Edit Format View Help

CHILD: Command: `wc -c > mert.txt`, Process ID: 1446

 mert - Notepad


File Edit Format View Help

|8

Handling signals

```
Command $> ^C
Ctrl+C received. Press Enter...
```

```
Command $> whoami
cabanel
```


 20230414_22-43-10_0 - Notepad

File Edit Format View Help

CHILD: Command: whoami, Process ID: 1458

```
Command $> sleep 10
^C
Ctrl+C received. Press Enter...
```

```
Command $> whoami
cabanel
```

 20230414_22-51-47_0 - Notepad

File Edit Format View Help

CHILD: Command: whoami, Process ID: 1488