

CSE 3033 Project 2 Report

Ahmet Turgut : 150117046
Koray Emre Şenel : 150117037
Mehmet Etkü Uzun: 150118504

Briefing

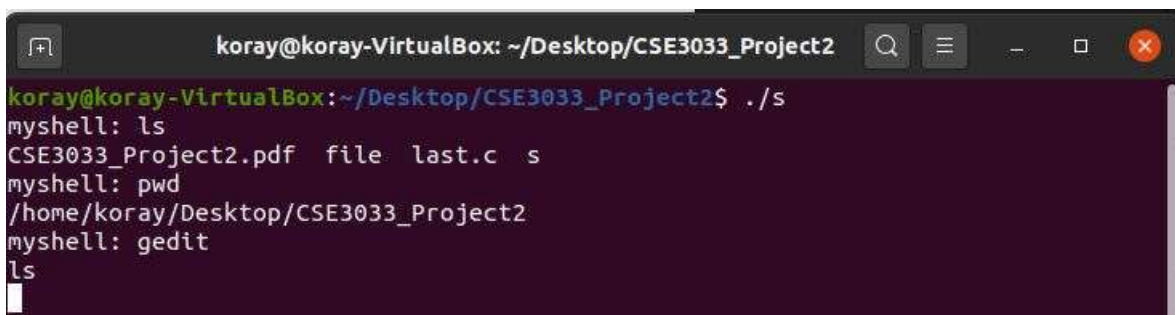
In this project , our goal is to make a shell terminal. Also we are working with fork and wait functions so we can learn multi process concept. In the shell that we make we add another commands like `ps_all`, `search`, `bookmark` etc.

This project require three main part.

1. If the input is a shell command this command have to run. If it has '`&`' at the end it will work on background.
2. There are some commands that required to create ;
 - a. **`ps_all`** – Shows the status of background processes.
 - b. **`^Z`** – Stops the currently running foreground process.
 - c. **`search`** – Searches the given keyword or phrase in the files.
 - d. **`bookmark`** – It bookmarks frequently used commands.
 - e. **`exit`** – If there are no active background process it terminates shell.
3. It should execute the given I/O redirections.

Part A

In this part we check the command if it is background or foreground. Then we use fork to create child process and execute the commands using `execv`. If the command is foreground the shell waits until it ends with the help of `wait` function. If it is a background process the shell can read another command and execute it.

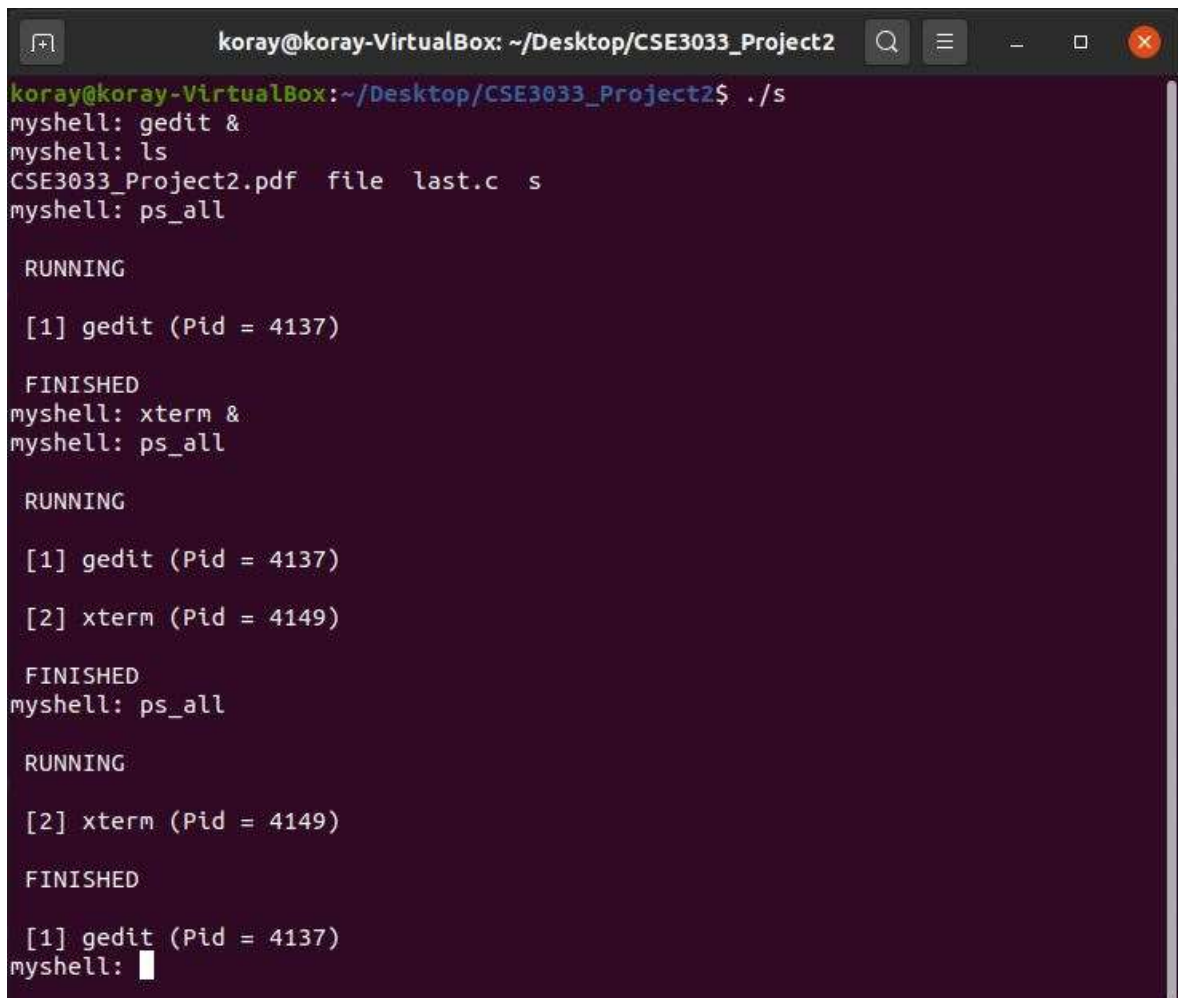


```
koray@koray-VirtualBox: ~/Desktop/CSE3033_Project2
koray@koray-VirtualBox:~/Desktop/CSE3033_Project2$ ./s
myshell: ls
CSE3033_Project2.pdf  file  last.c  s
myshell: pwd
/home/koray/Desktop/CSE3033_Project2
myshell: gedit
ls
```

Part B

- **PS_ALL**

This command returns all finished and running background processes. It does this with the help of linked list. When the background process starts it inserts this process to linked list. After that it checks the return of waitpid. If the waitpid returns the same value with child pid it means that process is finished so it will be deleted from the current linked list and send to another linked list. This method returns the all the things in these linked lists as output.



```
koray@koray-VirtualBox: ~/Desktop/CSE3033_Project2
koray@koray-VirtualBox:~/Desktop/CSE3033_Project2$ ./s
myshell: gedit &
myshell: ls
CSE3033_Project2.pdf  file  last.c  s
myshell: ps_all

RUNNING

[1] gedit (Pid = 4137)

FINISHED
myshell: xterm &
myshell: ps_all

RUNNING

[1] gedit (Pid = 4137)
[2] xterm (Pid = 4149)

FINISHED
myshell: ps_all

RUNNING

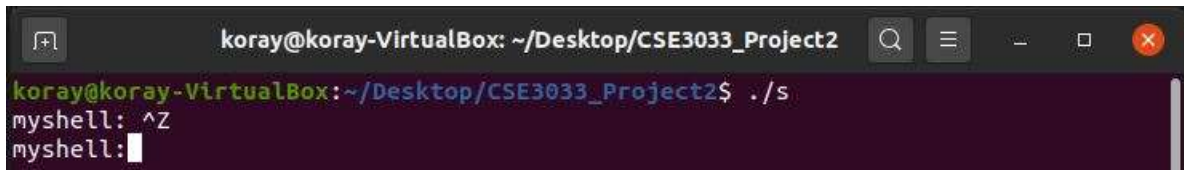
[2] xterm (Pid = 4149)

FINISHED

[1] gedit (Pid = 4137)
myshell: 
```

- ^Z

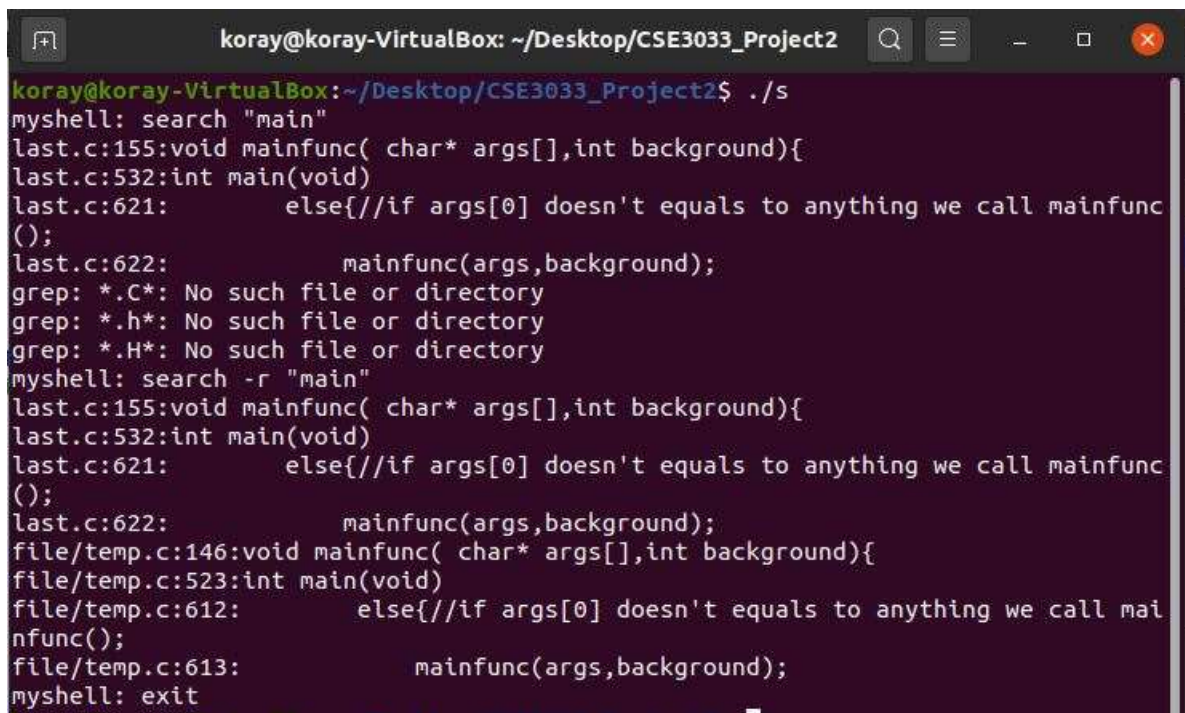
This method catch the ^Z signal. It creates signal struct and initializes the signal handler. After it checks if there any errors. If there are errors prints the error message. It checks if there any foreground process . If there are it kills the process with all the childs.



```
koray@koray-VirtualBox: ~/Desktop/CSE3033_Project2
koray@koray-VirtualBox:~/Desktop/CSE3033_Project2$ ./s
myshell: ^Z
myshell:
```

- SEARCH

This command search the given input string in the same directory files or recursively in all subdirectory files. We do this by calling the system method. system() method is executes the general commands in the c code. The executed general command is 'grep' which command that does the searching. For the recursion it should be called as "search -r". 'grep -r' is used for recursively searching.



```
koray@koray-VirtualBox: ~/Desktop/CSE3033_Project2
koray@koray-VirtualBox:~/Desktop/CSE3033_Project2$ ./s
myshell: search "main"
last.c:155:void mainfunc( char* args[],int background){
last.c:532:int main(void)
last.c:621:         else{//if args[0] doesn't equals to anything we call mainfunc
();
last.c:622:         mainfunc(args,background);
grep: *.C*: No such file or directory
grep: *.h*: No such file or directory
grep: *.H*: No such file or directory
myshell: search -r "main"
last.c:155:void mainfunc( char* args[],int background){
last.c:532:int main(void)
last.c:621:         else{//if args[0] doesn't equals to anything we call mainfunc
();
last.c:622:         mainfunc(args,background);
file/temp.c:146:void mainfunc( char* args[],int background){
file/temp.c:523:int main(void)
file/temp.c:612:         else{//if args[0] doesn't equals to anything we call mai
nfunc();
file/temp.c:613:         mainfunc(args,background);
myshell: exit
```

- **BOOKMARK**

This method makes a list of give input strings. It stores these things in a linked list. There are 4 different types of bookmark tag. "bookmark "command"" is adds the given command to the bookmarks to use it easily. "bookmark -l " is returns the lists of current bookmarks. "bookmark -i n" is gets the command that stored in the n'th index and executes it."bookmark -d n" is deletes the n'th bookmark.

```
koray@koray-VirtualBox: ~/Desktop/CSE3033_Project2
koray@koray-VirtualBox:~/Desktop/CSE3033_Project2$ ./s
myshell: bookmark "ls"
myshell: bookmark "ls -l | wc -l"
myshell: bookmark -l

[0] "ls"

[1] "ls -l | wc -l"
myshell: bookmark -i 1
7
myshell: bookmark -i 0
CSE3033_Project2.pdf file file.in file.out last.c s
myshell: bookmark -d 0
myshell: bookmark -l

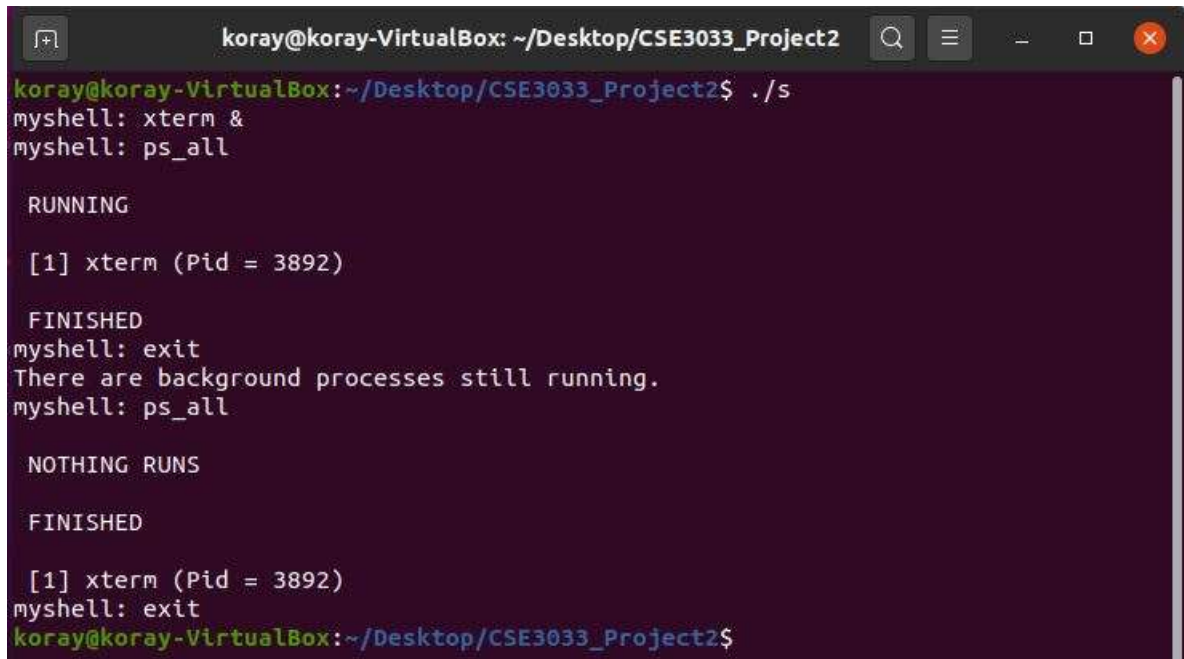
[0] "ls -l | wc -l"
myshell: bookmark -d 0
myshell: bookmark -l

NOTHING
myshell: bookmark "pwd"
myshell: bookmark -l

[0] "pwd"
myshell: exit
```

- EXIT

This command terminates the shell process if there are no any currently running process. We check the linked list of running's header , if it is null that means there are no currently running background process so it terminates the shell. If there are a currently running background process it returns a message to user.



```
koray@koray-VirtualBox: ~/Desktop/CSE3033_Project2
koray@koray-VirtualBox:~/Desktop/CSE3033_Project2$ ./s
myshell: xterm &
myshell: ps_all

RUNNING

[1] xterm (Pid = 3892)

FINISHED
myshell: exit
There are background processes still running.
myshell: ps_all

NOTHING RUNS

FINISHED

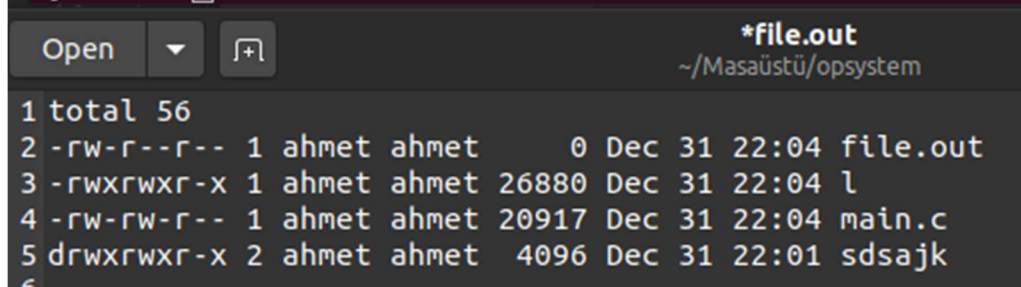
[1] xterm (Pid = 3892)
myshell: exit
koray@koray-VirtualBox:~/Desktop/CSE3033_Project2$
```

Part C

- `myprog[args] > file.out`

In this part we take command as input and write the output on file.out file. If file.out doesn't exist in current directory we create that file.

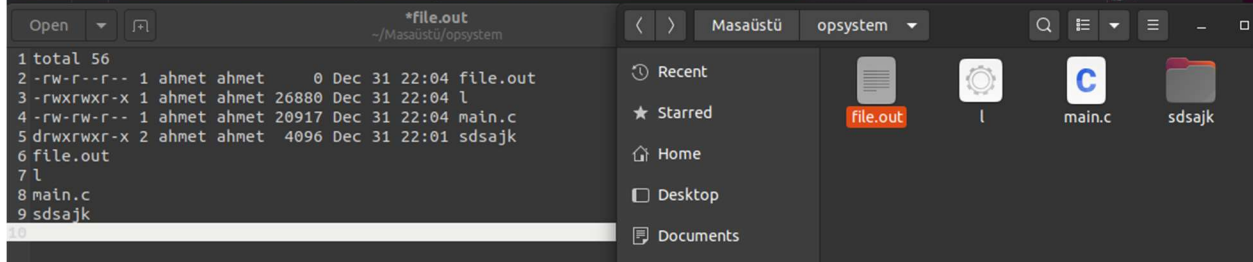
```
ahmet@ahmet:~/Masaüstü/opsystem$ ./l
myshell: myprog ls -l > file.out
myshell: 
```



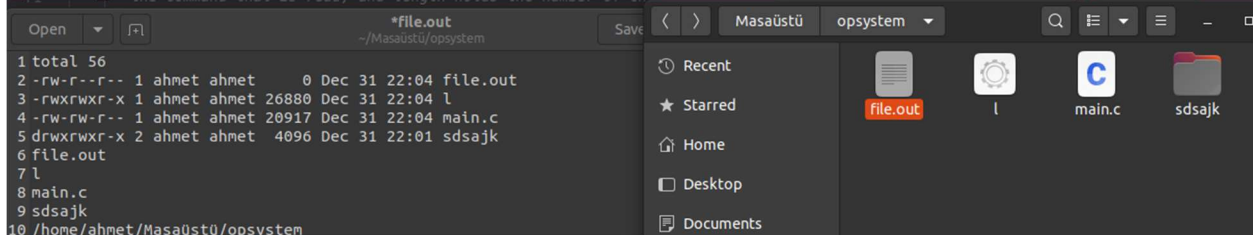
- `myprog[args] >> file.out`

In this part we take command as input and write the output on file.out file.. If file.out exist in current directory, we append output on file.out. If file.out doesn't exist in current directory we create that file.

```
ahmet@ahmet:~/Masaüstü/opsystem$ ./l
myshell: myprog ls -l > file.out
myshell: myprog ls >> file.out
myshell: 
```

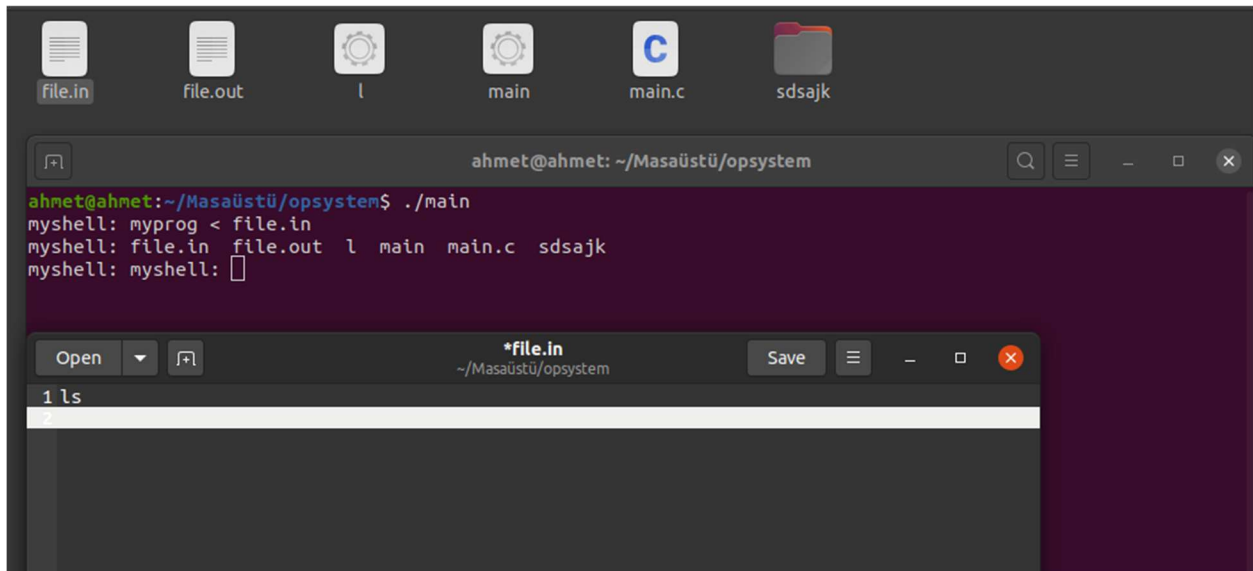


```
ahmet@ahmet:~/Masaüstü/opsystem$ ./l
myshell: myprog ls -l > file.out
myshell: myprog ls >> file.out
myshell: myprog pwd >> file.out
myshell: 
```



- `myprog[args] < file.in`

In this part we read file.in file and we execute command on file.in file. If file.in doesn't exist in current directory we print error on screen.

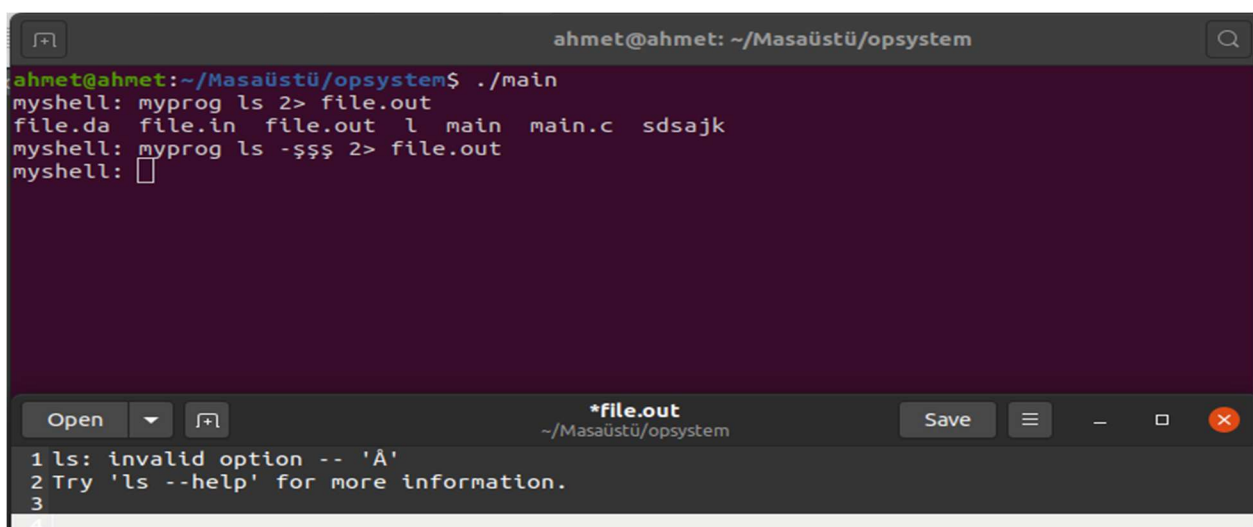


The screenshot shows a terminal window titled "ahmet@ahmet: ~/Masaüstü/opsystem". The user has executed the command `./main`. The output shows the program reading from `file.in` and executing the command `ls`. The output of `ls` is displayed in a separate window titled `*file.in`.

```
ahmet@ahmet:~/Masaüstü/opsystem$ ./main
myshell: myprog < file.in
myshell: file.in file.out l main main.c sdsajk
myshell: myshell: 
```

- `myprog[args] 2> file.out`

In this part we execute the given args. If there are no errors in the args it prints to terminal but if there are any errors it will write it to file.out file.

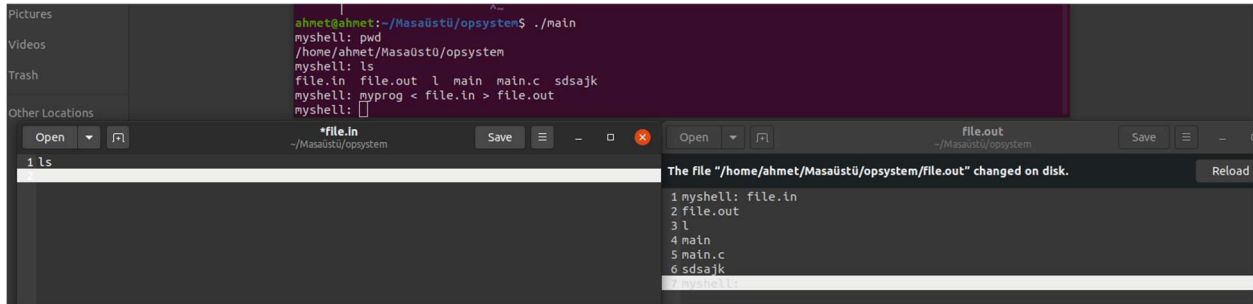


The screenshot shows a terminal window titled "ahmet@ahmet: ~/Masaüstü/opsystem". The user has executed the command `./main`. The output shows the program writing errors to `file.out`. The output of `ls` is displayed in a separate window titled `*file.out`.

```
ahmet@ahmet:~/Masaüstü/opsystem$ ./main
myshell: myprog ls 2> file.out
file.da file.in file.out l main main.c sdsajk
myshell: myprog ls -$$$ 2> file.out
myshell: 
```

- `myprog[args] < file.in > file.out`

In this part we combine one and three in Part C. So, we read `file.in` file and we execute command on `file.in` file and write this output to `output.file`. If `file.in` doesn't exist in current directory we print error on screen and If `file.out` doesn't exist in current directory we create that file.



The screenshot shows a terminal window and two file editors. The terminal window, titled "ahmet@ahmet:~/Masaüstü/opsystem", shows the following commands and output:

```
ahmet@ahmet:~/Masaüstü/opsystem$ ./main
myshell: pwd
/home/ahmet/Masaüstü/opsystem
myshell: ls
file.in file.out l main main.c sdsajk
myshell: myprog < file.in > file.out
myshell:
```

Below the terminal, there are two file editors. The left editor, titled "*file.in", shows the output of the `ls` command:

```
1 ls
```

The right editor, titled "file.out", shows the output of the `myprog` command:

```
1 myshell: file.in
2 file.out
3 l
4 main
5 main.c
6 sdsajk
```

A notification bar at the bottom of the right editor states: "The file "/home/ahmet/Masaüstü/opsystem/file.out" changed on disk. Reload".