

CSE 3033

OPERATING SYSTEMS

Programming Assignment #2

Instructor: Ali Haydar ÖZER

T.A.: Zuhra ALTUNTAŞ

Ertuğrul Sağdıç – 150116061

Eray Ayaz – 150116053

## Introduction

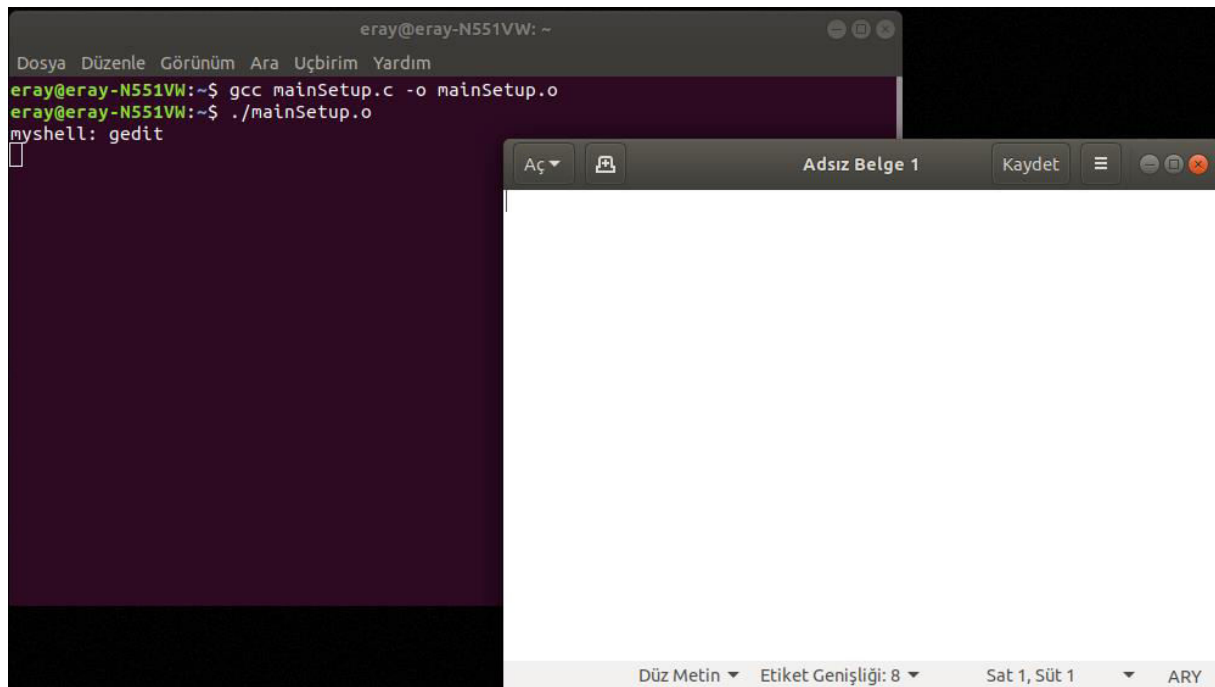
In this project, we are implementing our shell program in c programming language, which works on terminal, with the given functionalities.

The main() function of our program presents the command line prompt “myshell: ” and then invokes setup() function which waits for the user to enter a command. This program is terminated when the user enters ^D (<CONTROL><D>); and setup function then invokes exit. The contents of the command entered by the user is loaded into the args array.

## PART A

First, we need to take all commands as input and execute that in a new process. To do that, we need the to use fork() system call and fork() the processors as in parent and child processes. In child process, we are taking the path environmental variable into substrings. To do that, we need to take that environmental variable while calling system call which is called getenv() function. Then, call this function in example “getenv(“PATH”); “. Then, we are dividing the path with using strtok(string) function. While we are dividing, we are adding executable file name that appears on the command line to the substrings. After that, we are calling execv() function on every cycle of the loop with using substrings.

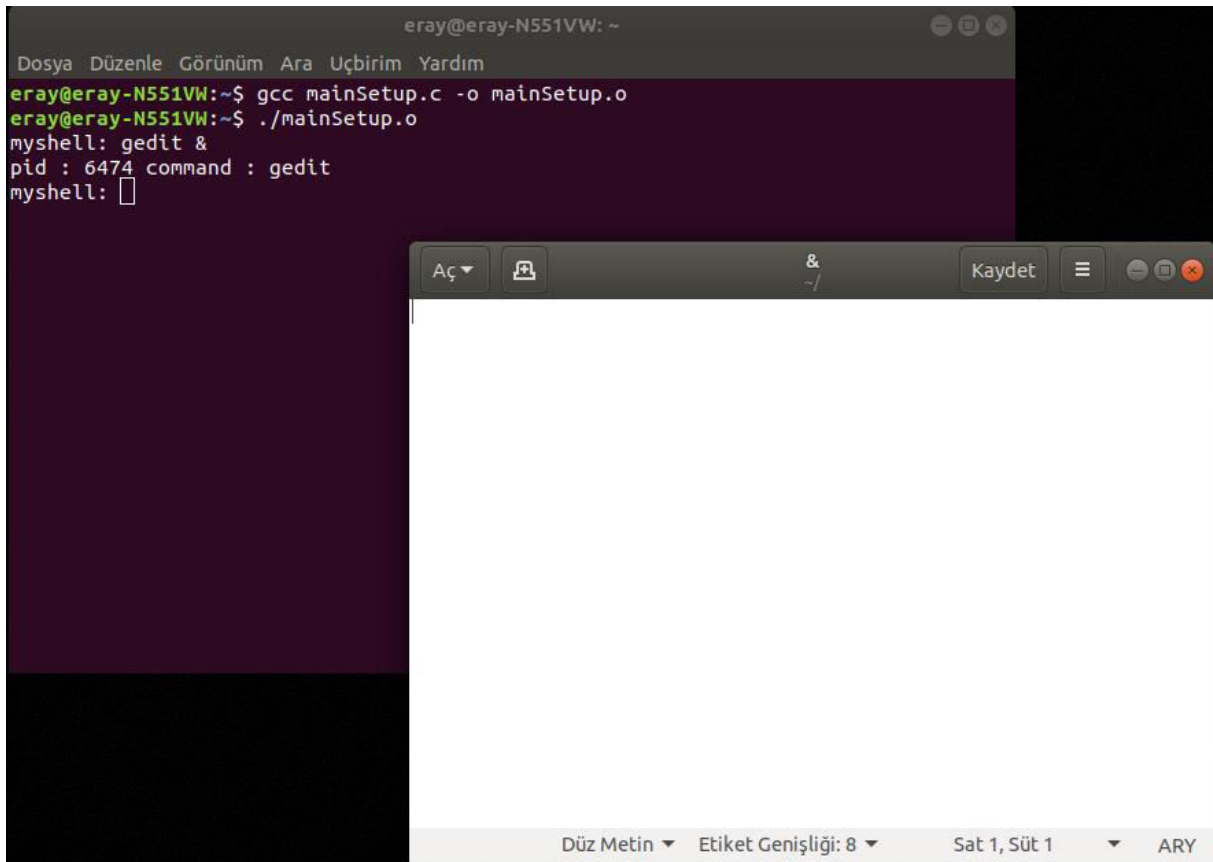
We are also handling with background and foreground processes. . If we give to command without & sign, the process runs itself on foreground. If a process run in foreground, the shell is waiting for that task to complete then prompts the user another command.



```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: gedit  
█
```

As we can see from this example, the shell takes `myshell: gedit` command and it waits for the foreground process until it finishes its task.

If we give to command with & sign, the process runs itself on background. If a process run in background, the shell is not waiting for that task to complete, but prompts the user another command.



```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: gedit &  
pid : 6474 command : gedit  
myshell: 
```

The screenshot shows a terminal window with a dark purple background. The prompt is 'eray@eray-N551VW: ~'. The user has entered 'gcc mainSetup.c -o mainSetup.o' and './mainSetup.o'. The output shows 'myshell: gedit &' and 'pid : 6474 command : gedit'. The prompt is now 'myshell: '. A second window, 'Aç', is open in the foreground, showing a blank white area. The window title bar includes '& ~/ Kaydet' and the status bar at the bottom shows 'Düz Metin Etiket Genişliği: 8 Sat 1, Süt 1 ARY'.

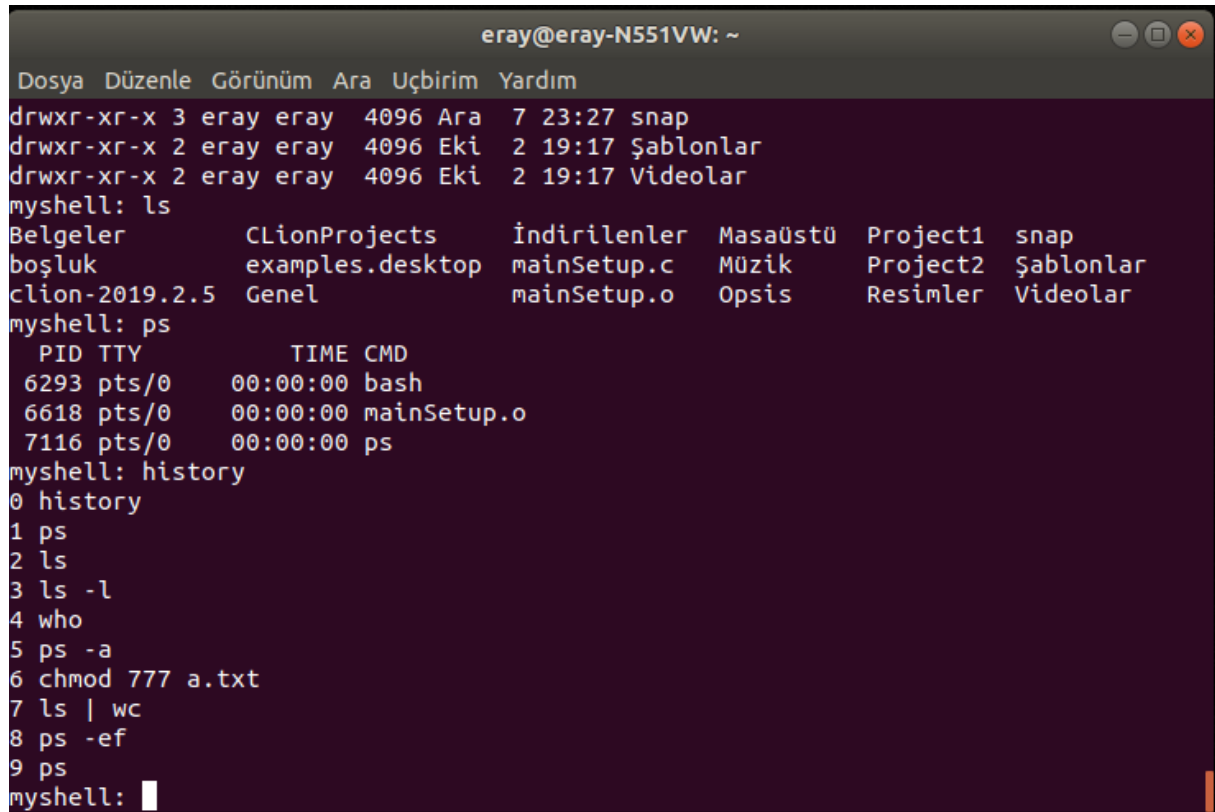
As shown, the shell prompts the user command line immediately. It does not wait for process to end its task.

## PART B

The shell supports some commands. We need to provide the conditions for the commands. The shell supports the following internal commands:

- history: prints list of the last 10 element on screen

We entered some commands to shell before. They have stored in the History.



```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
drwxr-xr-x 3 eray eray 4096 Ara 7 23:27 snap  
drwxr-xr-x 2 eray eray 4096 Eki 2 19:17 Şablonlar  
drwxr-xr-x 2 eray eray 4096 Eki 2 19:17 Videolar  
myshell: ls  
Belgeler          CLionProjects      İndirilenler      Masaüstü          Project1          snap  
boşluk            examples.desktop   mainSetup.c        Müzik             Project2          Şablonlar  
clion-2019.2.5    Genel              mainSetup.o        Opsis             Resimler          Videolar  
myshell: ps  
  PID TTY          TIME CMD  
 6293 pts/0        00:00:00 bash  
 6618 pts/0        00:00:00 mainSetup.o  
 7116 pts/0        00:00:00 ps  
myshell: history  
0 history  
1 ps  
2 ls  
3 ls -l  
4 who  
5 ps -a  
6 chmod 777 a.txt  
7 ls | wc  
8 ps -ef  
9 ps  
myshell: 
```

As shown in the screenshot, if give history command to shell, it prints the last 10 command entered.

- history -i num: try to execute the command which is entered with given index called num

First, we need to know which index that we want to execute. We can check it from history. After choosing the index from history list, we call `myshell: history -i 3` command and it will run the third index of the history.

```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
myshell: history  
0 history  
1 ls /bin  
2 history  
3 ps  
4 ls  
5 ls -l  
6 who  
7 ps -a  
8 chmod 777 a.txt  
9 ls | wc  
myshell: history -i 3  
  PID TTY          TIME CMD  
 6293 pts/0    00:00:00 bash  
 6618 pts/0    00:00:00 mainSetup.o  
 7142 pts/0    00:00:00 ps  
myshell: history  
0 history  
1 ps  
2 history -i 3  
3 history  
4 ls /bin  
5 history  
6 ps  
7 ls  
8 ls -l  
9 who  
myshell: █
```

As shown, Third command was **3 ps** and “ps” command has executed. Also, the executed part has been added to history list again.

- ^Z: stops the currently running foreground process

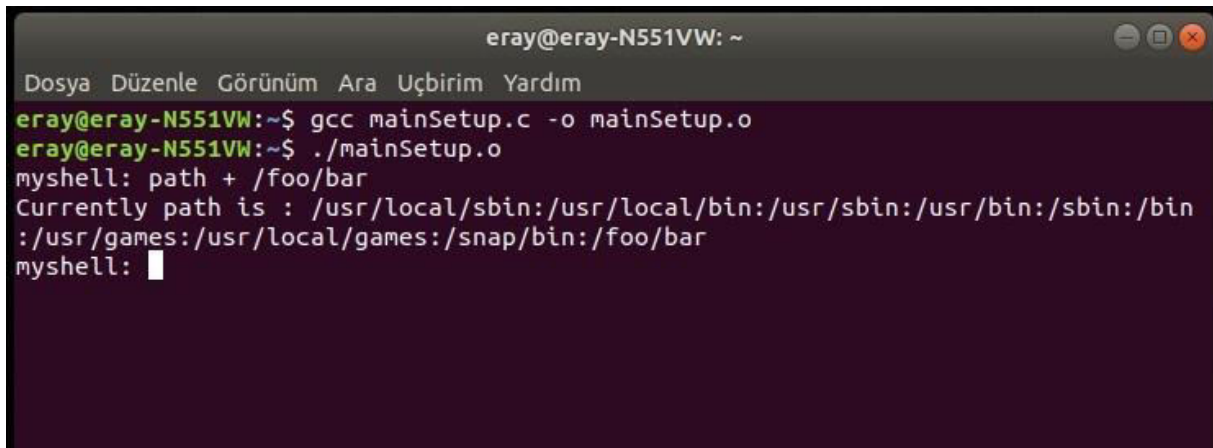
We create a method for this action. We kept a counter to find out if foreground are still executing or finished. If they are finished before the our action this signal has no effect. Otherwise this signal stop the the running foreground process.

```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: gedit  
^Z  
myshell: █
```

As we can see, ^Z terminates the process. We will prove this later on with other commands.

- path: shows current path name

First, we need to create global pathname variable and make it equal to path environmental variable using getenv("PATH") function. Then, we just print it to the screen.

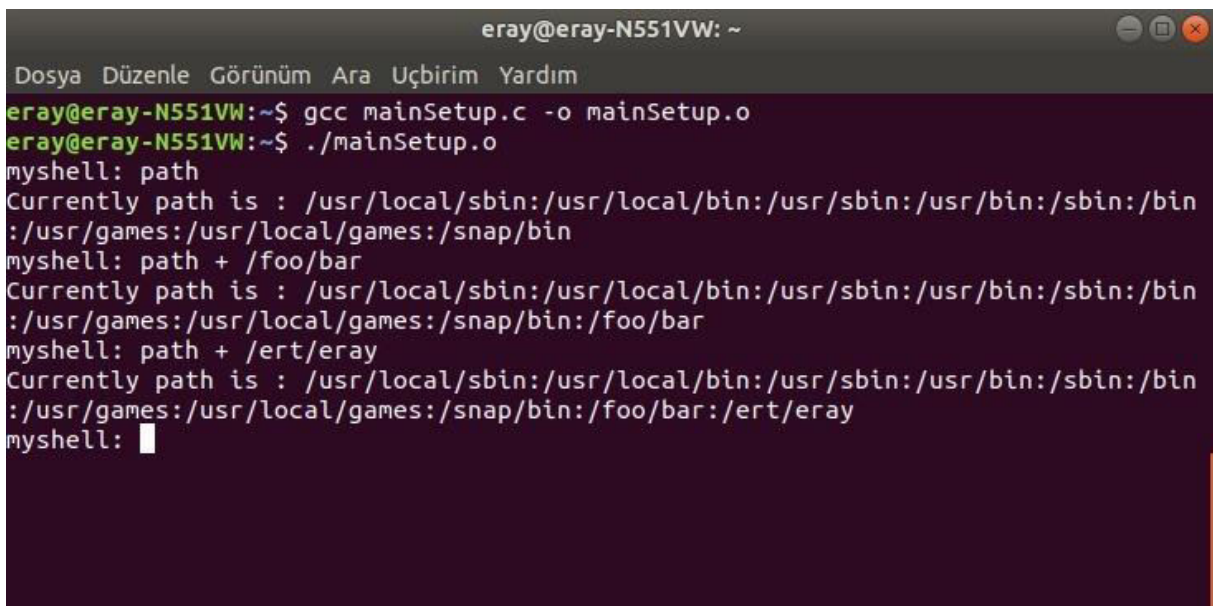


```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: path + /foo/bar  
Currently path is : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
:/usr/games:/usr/local/games:/snap/bin:/foo/bar  
myshell: █
```

As shown in the screenshot, it prints the path.

- path + /foo/bar: appends the pathname to the path variable

If we would like to add a path, we just simply call `myshell: path + /foo/bar` command. User can modify the /foo/bar part. It will add the path which entered by user.



```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: path  
Currently path is : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
:/usr/games:/usr/local/games:/snap/bin  
myshell: path + /foo/bar  
Currently path is : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
:/usr/games:/usr/local/games:/snap/bin:/foo/bar  
myshell: path + /ert/eray  
Currently path is : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
:/usr/games:/usr/local/games:/snap/bin:/foo/bar:/ert/eray  
myshell: █
```

As in the screenshot, it has added /foo/bar and /ert/eray paths to path, and it printed out the modified path variable to the screen.

- path - /foo/bar: removes the pathname from the path variable

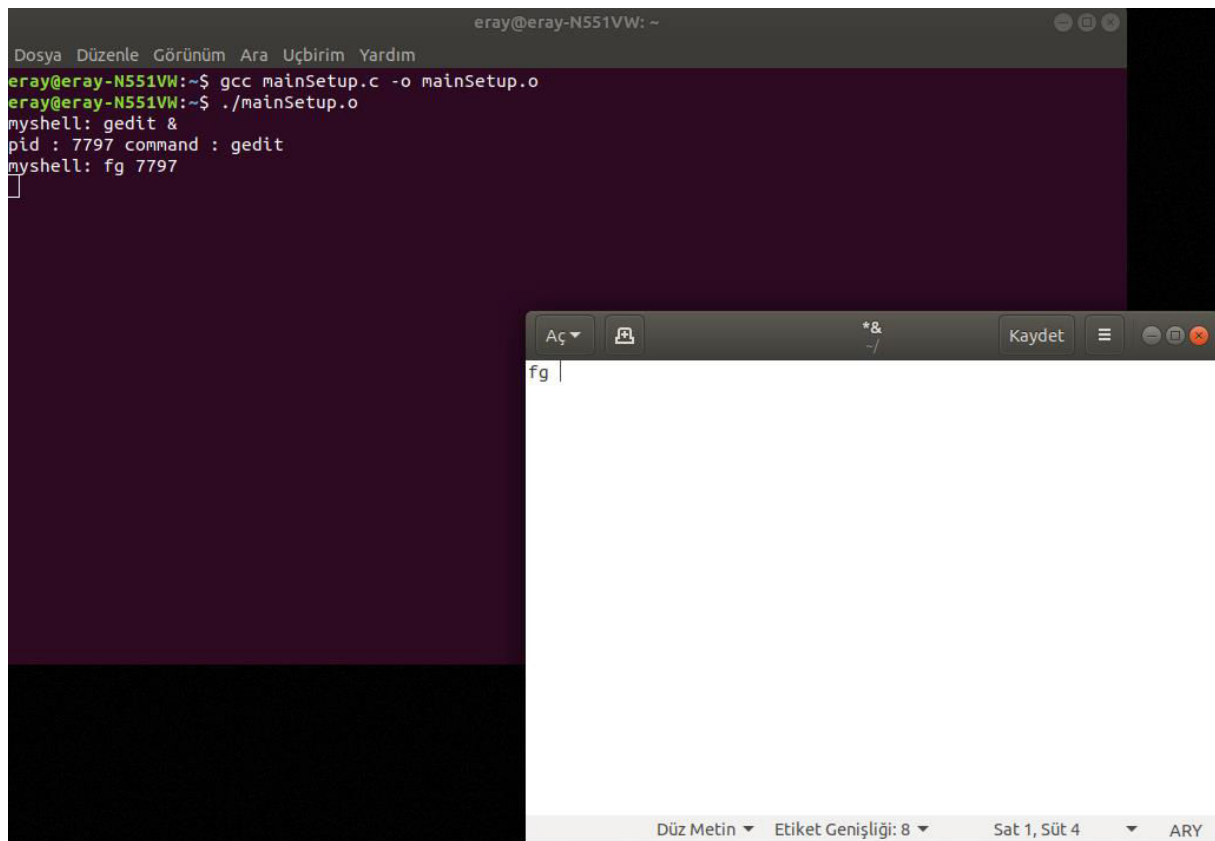
If we would like to remove a path from the path environmental variable, we just call `myshell: path - /foo/bar` command. User can modify the /foo/bar part. It will remove the path which entered by user.

```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: path  
Currently path is : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
:/usr/games:/usr/local/games:/snap/bin  
myshell: path + /foo/bar  
Currently path is : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
:/usr/games:/usr/local/games:/snap/bin:/foo/bar  
myshell: path + /ert/eray  
Currently path is : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
:/usr/games:/usr/local/games:/snap/bin:/foo/bar:/ert/eray  
myshell: path - /foo/bar  
Currently path is : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
:/usr/games:/usr/local/games:/snap/bin:/ert/eray  
myshell: █
```

As in the screenshot, it have deleted /foo/bar path from path, and it printed out the modified path variable to the screen.

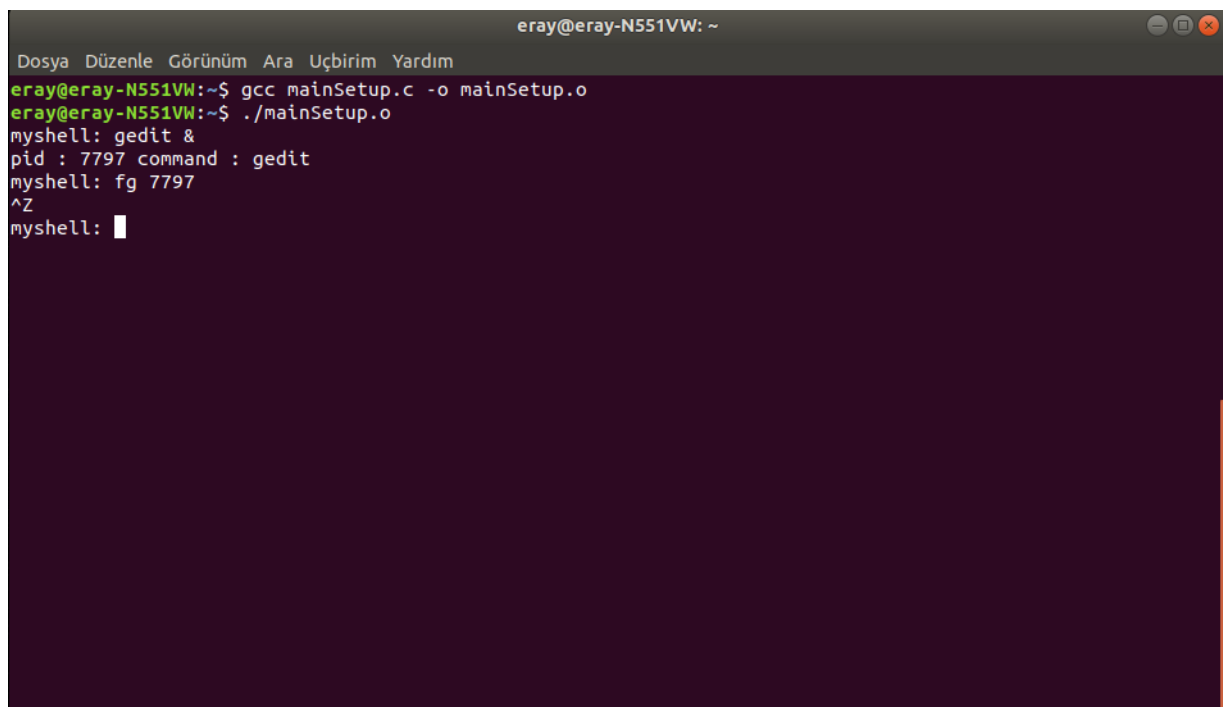
- `fg %num`: moves the background process to the background

When the user want to execute the command in background, our program will print the command id and command name. Then program will added to our linked list structure, which has created before the program. Later when the user enter 'fg' command with id of the command. Program takes the executable command from background and put it into a foreground.



```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: gedit &  
pid : 7797 command : gedit  
myshell: fg 7797  
fg |
```

As shown, background process has moved to foreground. To prove it has moved to foreground, we can use ^Z command.

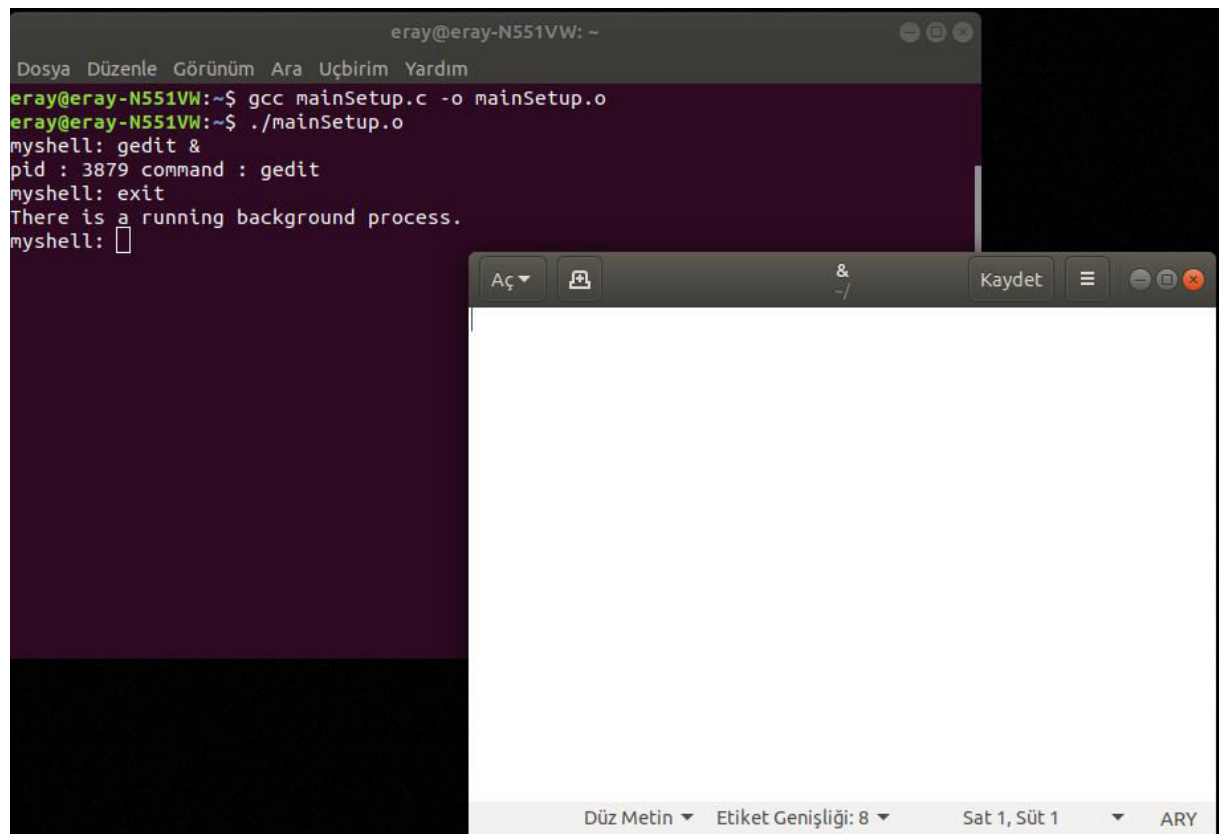


```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: gedit &  
pid : 7797 command : gedit  
myshell: fg 7797  
^Z  
myshell: █
```

It is obvious that the process has been terminated.

- exit: terminates shell process. If there are background processes still running then does not terminates the shell unless all background processes terminated by user.





```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: gedit &  
pid : 3879 command : gedit  
myshell: exit  
There is a running background process.  
myshell: 
```

If the user want to exit from the system can press the 'exit' command. But program after the 'exit' command will search the background processes. If program find out any background process in the system then it will print the error message and will wait until the command finished.

As we can see, we created a background process by using gedit & command. After entering exit, it gave the error.

```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: gedit &  
pid : 3879 command : gedit  
myshell: exit  
There is a running background process.  
myshell: exit  
eray@eray-N551VW:~$
```

After terminating background process, exit command works totally fine. In addition, we are looking to our linked list structure that we created for holding all background processes. After the 'exit' command entered if the linked list empty user can exit from the program easily.

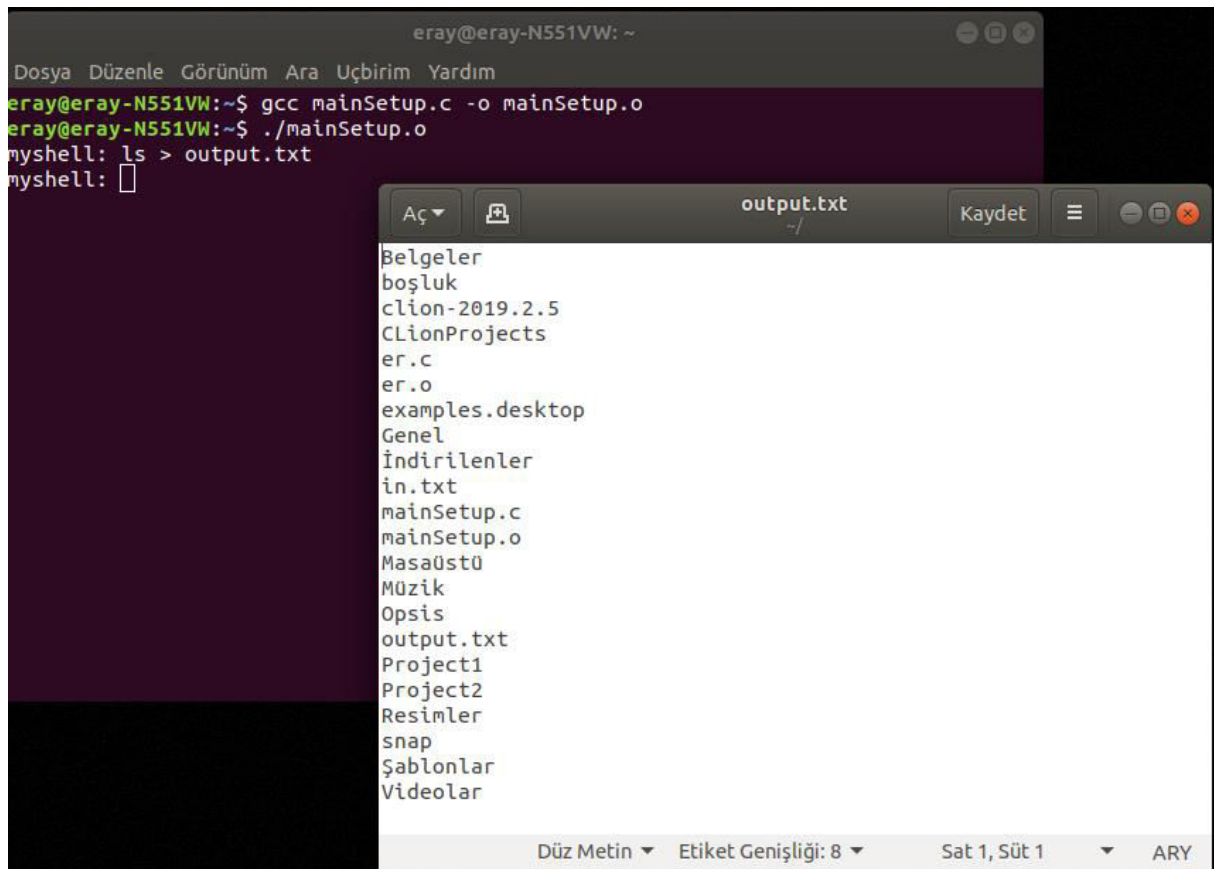
## PART C

The shell program supports I/O redirection on stdin, stdout and, stderr.

- It can either truncates or appends the output to the output file which is created with the given file name.
- It can use the contents of the input file as the standard input to program.
- It can write the standard error of the program to the error file which is created with the given file name.

If we give the command **myshell: myprog [args] > file.out** to shell, it redirects the standard output as output file which entered by user. The file created if file does not exists and truncated if it does exist.

For example if we give **myshell: ls > output.txt** command, it will redirects the outputs to output.txt file.



The screenshot shows a terminal window titled 'eray@eray-N551VW: ~' with a menu bar containing 'Dosya', 'Düzenle', 'Görünüm', 'Ara', 'Uçbirim', and 'Yardım'. The terminal shows the following commands and output:

```
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o
eray@eray-N551VW:~$ ./mainSetup.o
myshell: ls > output.txt
myshell: █
```

Overlaid on the terminal is a file explorer window titled 'output.txt' with a menu bar containing 'Aç', a search icon, and 'Kaydet'. The file list shows the following files and folders:

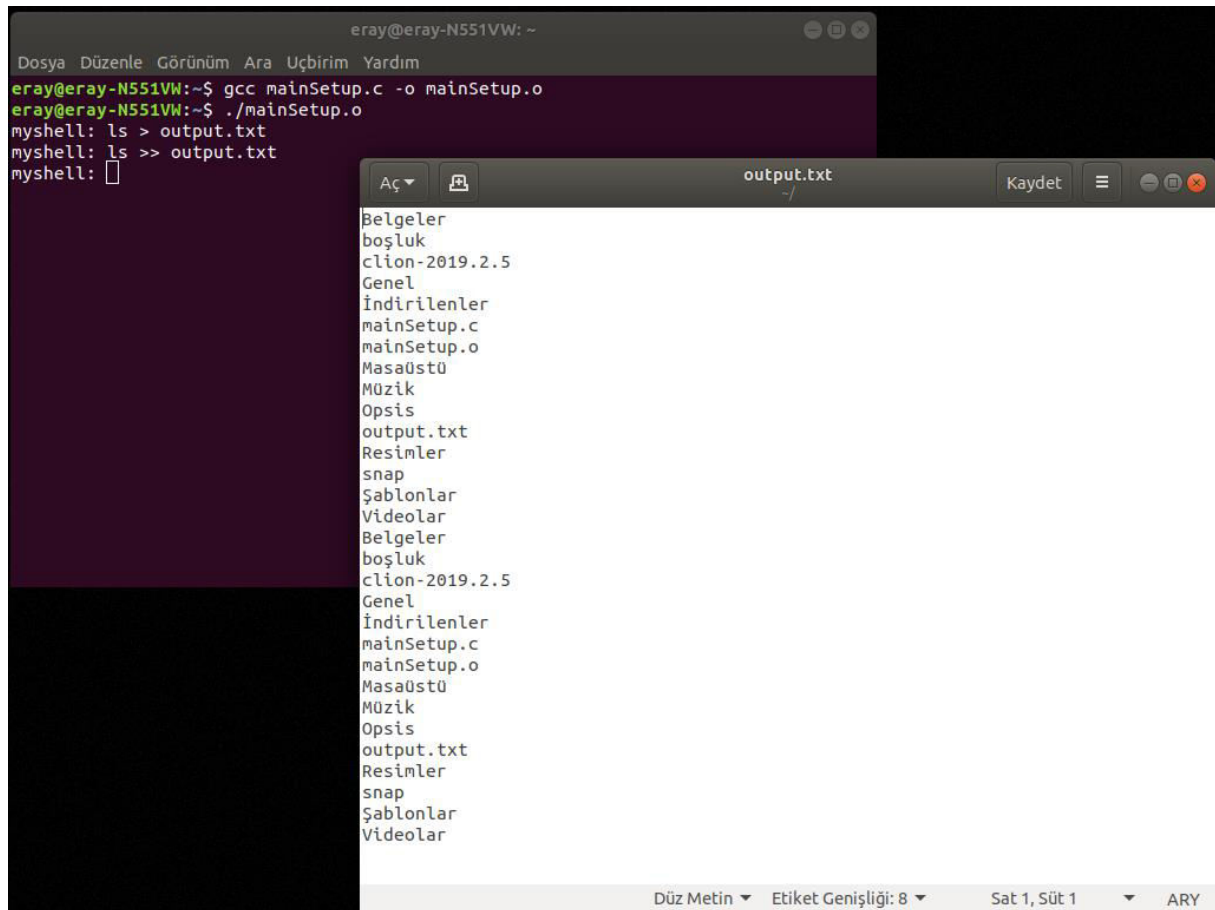
- Belgeler
- boşluk
- clion-2019.2.5
- CLionProjects
- er.c
- er.o
- examples.desktop
- Genel
- İndirilenler
- in.txt
- mainSetup.c
- mainSetup.o
- Masaüstü
- Müzik
- Opsis
- output.txt
- Project1
- Project2
- Resimler
- snap
- Şablonlar
- Videolar

The status bar at the bottom of the file explorer shows 'Düz Metin', 'Etiket Genişliği: 8', 'Sat 1, Süt 1', and 'ARY'.

As you can see from the screenshot, the output written to output.txt file.

If we give the command ***myprog [args] >> file.out*** to shell, it redirects the standard output as output file which entered by user. The file created if file does not exists and appended if it does exist.

For example if we give **myshell: `ls >> output.txt`** command, it will redirects the outputs to output.txt file.



The screenshot shows a terminal window on the left and a file editor on the right. The terminal window has a title bar 'eray@eray-N551VW: ~' and a menu bar 'Dosya Düzenle Görünüm Ara Uçbirim Yardım'. The terminal content shows the following commands and output:

```
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o
eray@eray-N551VW:~$ ./mainSetup.o
myshell: ls > output.txt
myshell: ls >> output.txt
myshell: 
```

The file editor on the right has a title bar 'output.txt' and a menu bar 'Aç Kaydet'. The editor content shows a list of files and directories:

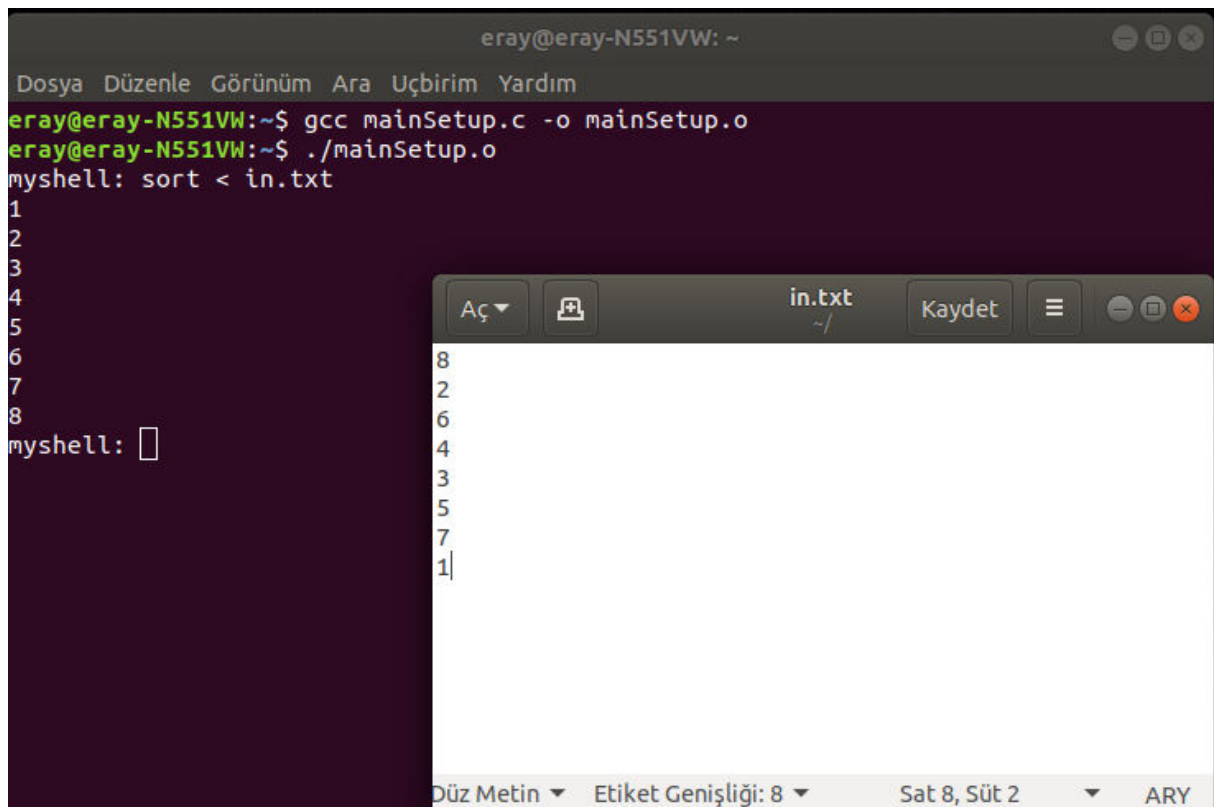
```
Belgeler
boşluk
clion-2019.2.5
Genel
İndirilenler
mainSetup.c
mainSetup.o
Masaüstü
Müzik
Opsis
output.txt
Resimler
snap
Şablonlar
Videolar
Belgeler
boşluk
clion-2019.2.5
Genel
İndirilenler
mainSetup.c
mainSetup.o
Masaüstü
Müzik
Opsis
output.txt
Resimler
snap
Şablonlar
Videolar
```

The status bar at the bottom of the file editor shows 'Düz Metin', 'Etiket Genişliği: 8', 'Sat 1, Süt 1', and 'ARY'.

As you can see from the screenshot, the output written to output.txt file. Also, we can see that the file has appended from this example. It has written the outputs with the old outputs.

If we give the command ***myprog [args] < file.in*** to shell, it redirects the standard input as input file which entered by user.

For example if we give **myshell: sort < in.txt** command, it will redirects the input file. We are expecting from shell, to take inputs from input file and sorts them and prints them to its standard output.



The screenshot shows a terminal window with the following commands and output:

```
eray@eray-N551VW: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o  
eray@eray-N551VW:~$ ./mainSetup.o  
myshell: sort < in.txt  
1  
2  
3  
4  
5  
6  
7  
8  
myshell: █
```

Overlaid on the terminal is a text editor window titled "in.txt" showing the contents of the file:

```
8  
2  
6  
4  
3  
5  
7  
1|
```

The text editor has a status bar at the bottom showing "Düz Metin", "Etiket Genişliği: 8", "Sat 8, Süt 2", and "ARY".

As you can see, it took inputs from input file and printed them to screen.

If we give the command ***myprog [args] 2> file.out*** to shell, it redirects the standard error to file which entered by user.

For example if we give **myshell: ls 2> out.txt** command, it will redirect the error file. We are expecting from shell, if there is an error message, then write that error to file.

The screenshot shows a terminal window titled 'eray@eray-N551VW: ~' with a menu bar containing 'Dosya', 'Düzenle', 'Görünüm', 'Ara', 'Uçbirim', and 'Yardım'. The terminal output is as follows:

```
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o
eray@eray-N551VW:~$ ./mainSetup.o
mysHELL: ls 2> out.txt
Belgeler          er.o          mainSetup.c    mysHELLson.c   out.txt        Videolar
boşluk            Genel         mainSetup.o    mysHELLson.o   Resimler
clion-2019.2.5    İndirilenler Masaüstü       Opsis          snap
er.c              in.txt        Müzik          output.txt     Şablonlar
mysHELL: 
```

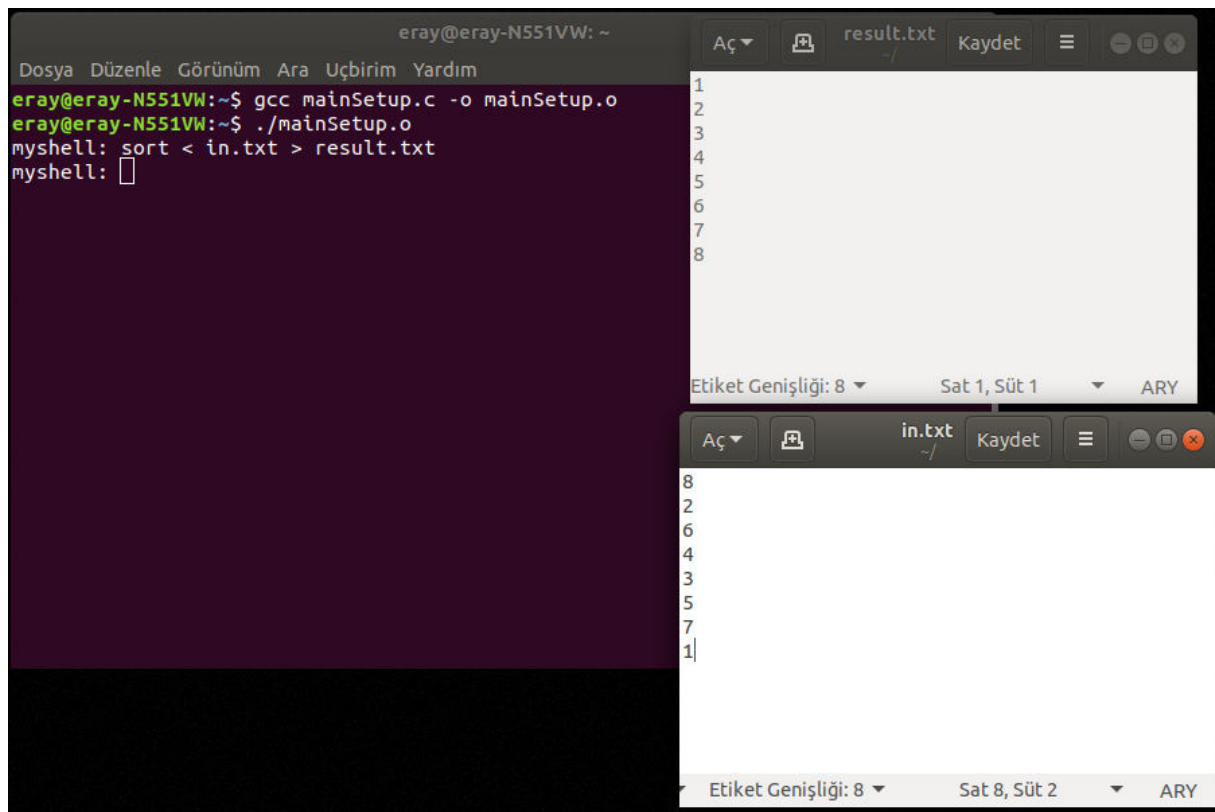
Below the terminal, a file editor window titled 'out.txt' is open, showing an empty file. The editor has a menu bar with 'Aç', a file icon, 'Kaydet', and a hamburger menu. The status bar at the bottom indicates 'Düz Metin', 'Etiket Genişliği: 8', 'Sat 1, Süt 1', and 'ARY'.

As you can see, there is no error. So, the file is empty. Let us, try it with

Obviously, there is no executable called "aa". So, it has written the error to file.

If we give the command ***myprog [args] < file.in > file.out*** to shell, it redirects the standard input as input file and it redirects the standard output as output file which are entered by user.

For example if we give **mysHELL: sort < in.txt > result.txt** command, it will redirect input. We are expecting from shell, take all inputs from the input file and write them to output file.



The screenshot shows a terminal window on the left and two text editors on the right. The terminal window has a title bar 'eray@eray-N551VW: ~' and a menu bar 'Dosya Düzenle Görünüm Ara Uçbirim Yardım'. The terminal output is as follows:

```
eray@eray-N551VW:~$ gcc mainSetup.c -o mainSetup.o
eray@eray-N551VW:~$ ./mainSetup.o
myshell: sort < in.txt > result.txt
myshell: 
```

The top text editor window has a title bar 'result.txt' and a menu bar 'Aç Kaydet'. It contains a list of numbers 1 through 8, with line 1 selected. The bottom text editor window has a title bar 'in.txt' and a menu bar 'Aç Kaydet'. It contains a list of numbers 1 through 8, with line 1 selected.

As we can see, the inputs has taken from input file and they written to output file.

## Conclusion

We have learned that handling signals, dealing with background and foreground processes, I/O redirections, how to deal with pipelines, different kind of shell commands, and so on.