Islamic University of Gaza
Faculty of Engineering
Department of Computer Engineering
ECOM 4010: Operating Systems Lab

Lab # 3

Fork() & execv()

Eng. Haneen El-Masry

October, 2013

Fork ( )

The process can perform more than one function at a same time, such it can create an entirely separate processes.

Most operating systems identify processes according to a unique process identifier (pid), which is typically an integer number.

In Unix the system call fork is used to create a new process, which becomes the child process of the caller. Both processes will execute concurrently from the next instruction following the fork() system call. We have to distinguish the parent from the child. This can be done by testing the returned value of fork() such it returns pid with value:

- **-1** if an error happened and fork() failed.
- **0** to the child process.
- **Positive Process id of child** to the parent.

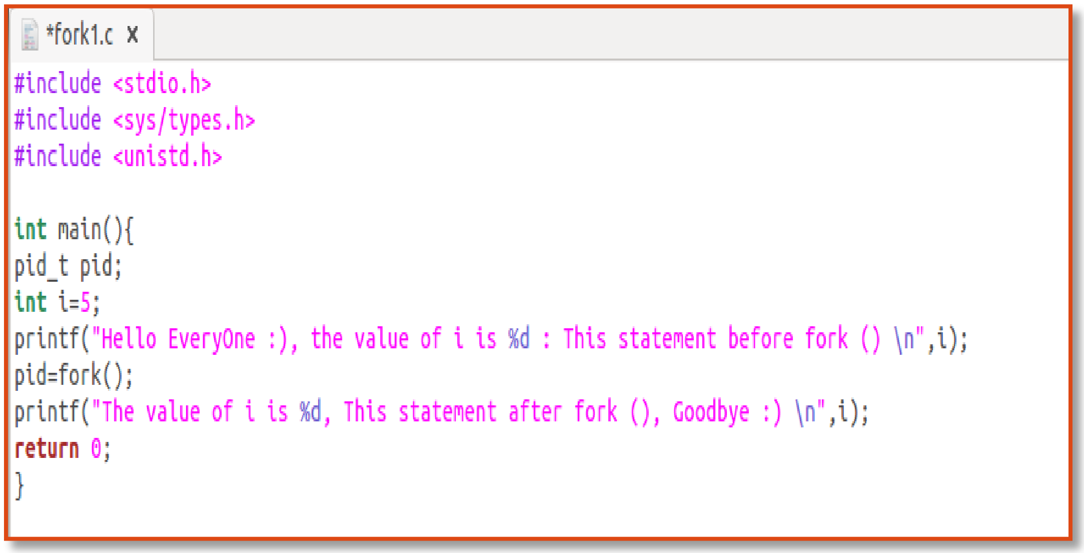
When a process creates a new process, two possibilities exist in terms of execution:

- The parent continues to execute concurrently with its children.
- The parent waits until some or all of its children have terminated.

There are also two possibilities in terms of the address space of the new process:

- The child process is a duplicate of the parent process (it has the same program and data as the parent).
- The child process has a new program loaded into it.

Example1



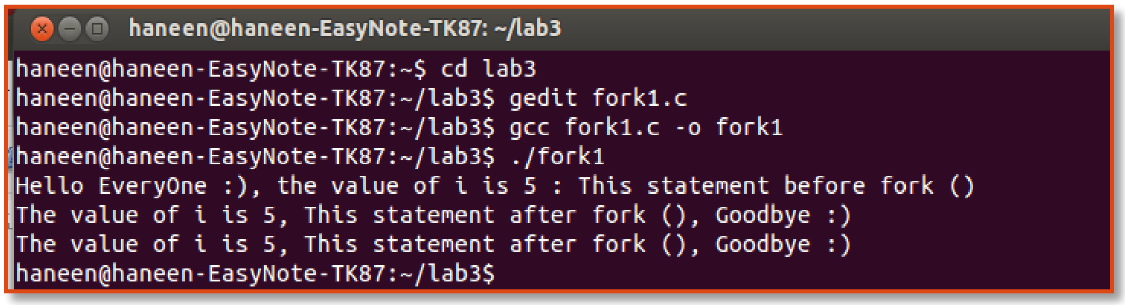
```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
pid_t pid;
int i=5;
printf("Hello EveryOne :), the value of i is %d : This statement before fork () \n",i);
pid=fork();
printf("The value of i is %d, This statement after fork (), Goodbye :) \n",i);
return 0;
}
```

Notes

<sys/types.h> : Defines pid_t definition.

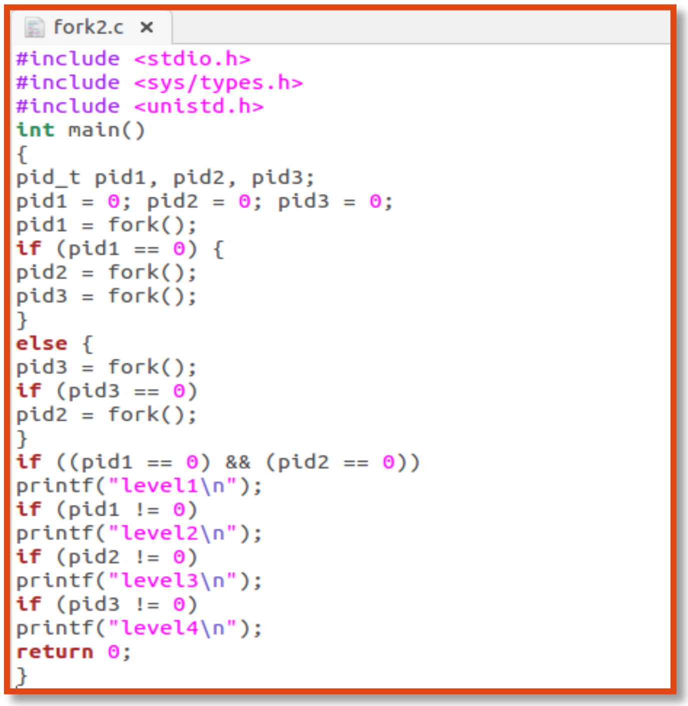
<unistd.h>: Defines System Calls including fork() .

Practical



```
haneen@haneen-EasyNote-TK87:~$ cd lab3
haneen@haneen-EasyNote-TK87:~/lab3$ gedit fork1.c
haneen@haneen-EasyNote-TK87:~/lab3$ gcc fork1.c -o fork1
haneen@haneen-EasyNote-TK87:~/lab3$ ./fork1
Hello EveryOne :), the value of i is 5 : This statement before fork ()
The value of i is 5, This statement after fork (), Goodbye :)
The value of i is 5, This statement after fork (), Goodbye :)
haneen@haneen-EasyNote-TK87:~/lab3$
```

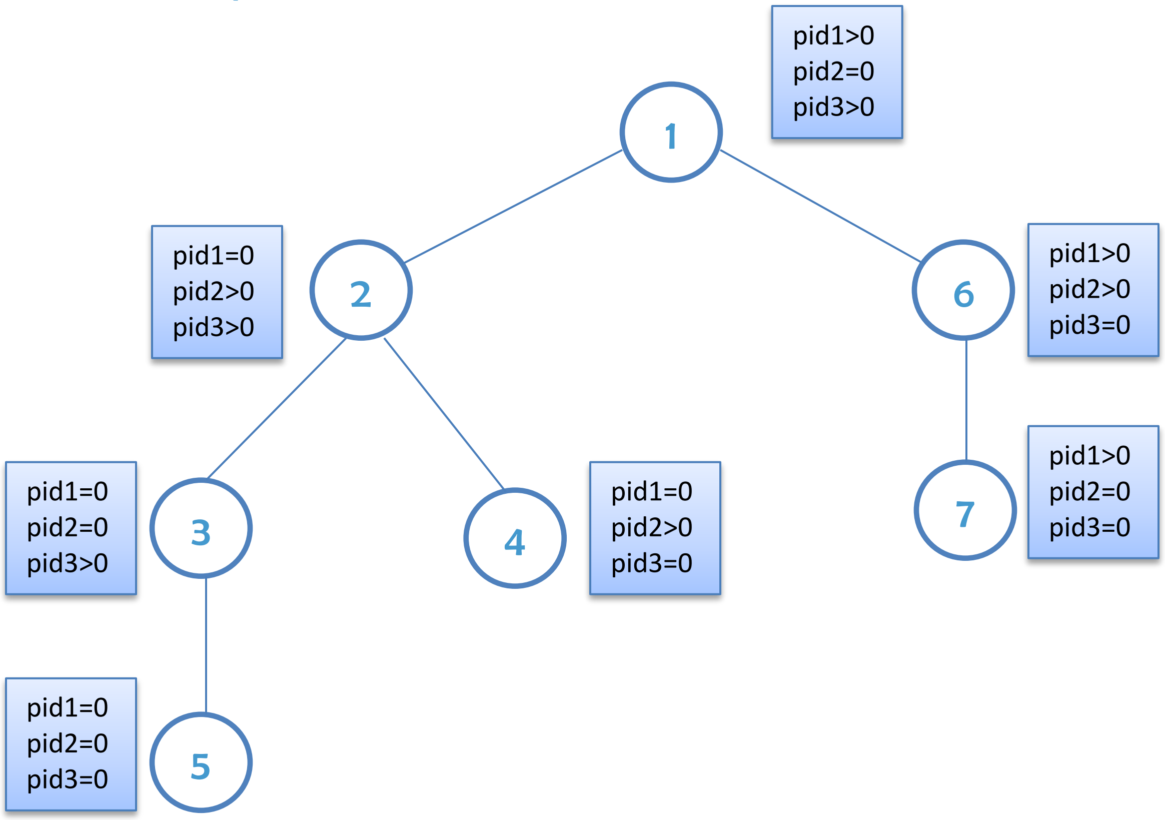
Example2



fork2.c

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
pid_t pid1, pid2, pid3;
pid1 = 0; pid2 = 0; pid3 = 0;
pid1 = fork();
if (pid1 == 0) {
pid2 = fork();
pid3 = fork();
}
else {
pid3 = fork();
if (pid3 == 0)
pid2 = fork();
}
if ((pid1 == 0) && (pid2 == 0))
printf("level1\n");
if (pid1 != 0)
printf("level2\n");
if (pid2 != 0)
printf("level3\n");
if (pid3 != 0)
printf("level4\n");
return 0;
}
```

The output:



Then:

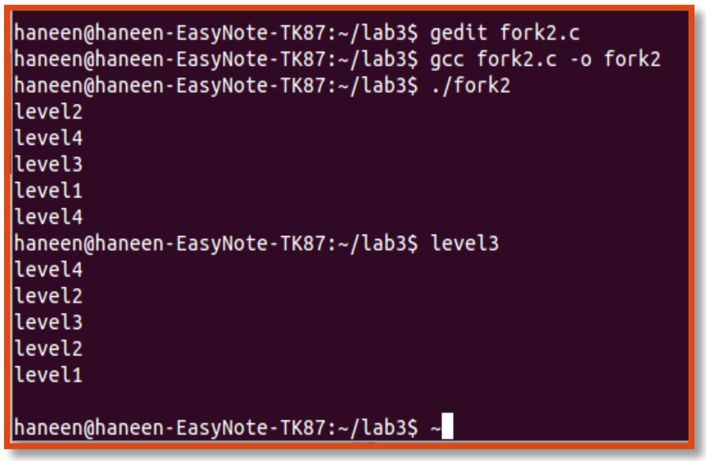
Level 1: printed 2 times.

Level 2: printed 3 times.

Level 3: printed 3 times.

Level 4: printed 3 times.

Practical



```
haneen@haneen-EasyNote-TK87:~/lab3$ gedit fork2.c
haneen@haneen-EasyNote-TK87:~/lab3$ gcc fork2.c -o fork2
haneen@haneen-EasyNote-TK87:~/lab3$ ./fork2
level2
level4
level3
level1
level4
haneen@haneen-EasyNote-TK87:~/lab3$ level3
level4
level2
level3
level2
level1

haneen@haneen-EasyNote-TK87:~/lab3$ ~
```

In Example 1 &2:

- The parent continues to execute concurrently with its children.
- The child process is a duplicate of the parent process.

Wait( )

We can arrange for the parent process to wait until the child finishes before continuing by calling wait.

The syntax for wait would be:

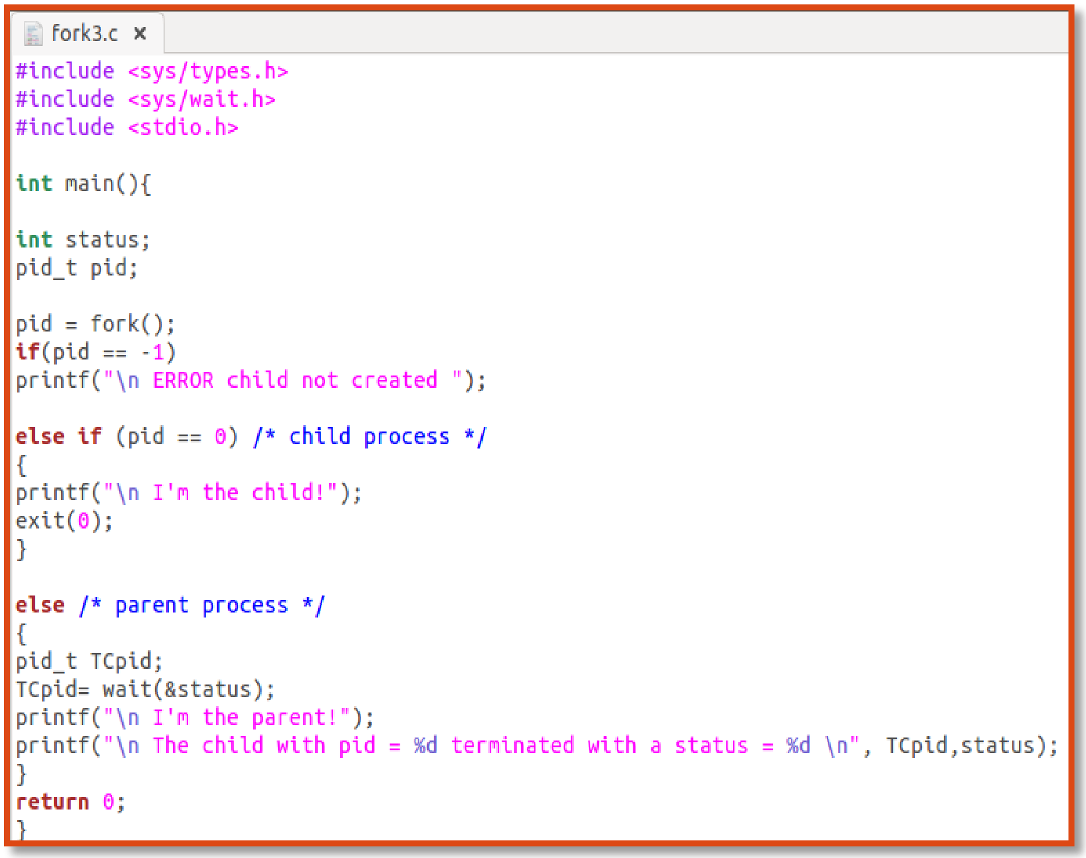
pid_t wait(int *stat_val);

The return value is PID of the child process that has terminated.

The argument is the status information that allows the parent process to determine the exit status of the child process, that is, the value returned from main or passed to exit.

If stat_val is not a null pointer, the status information will be written to the location to which it points.

Example 3:



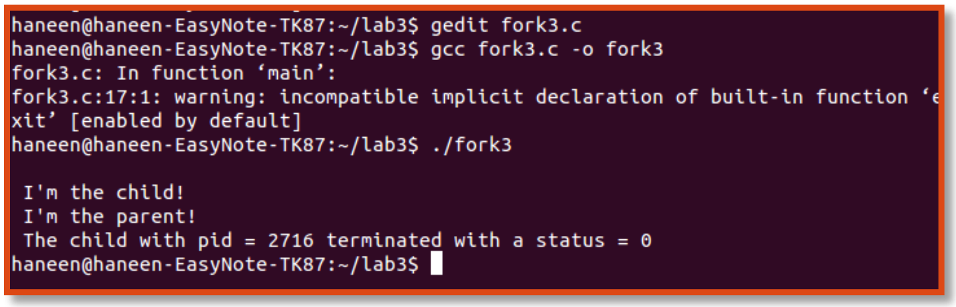
fork3.c ×

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>

int main(){

int status;
pid_t pid;

pid = fork();
if(pid == -1)
printf("\n ERROR child not created ");

else if (pid == 0) /* child process */
{
printf("\n I'm the child!");
exit(0);
}

else /* parent process */
{
pid_t TCpid;
TCpid= wait(&status);
printf("\n I'm the parent!");
printf("\n The child with pid = %d terminated with a status = %d \n", TCpid,status);
}
return 0;
}
```

Notes:

<sys/wait.h>: Defines wait System Call.

Practical:



```
haneen@haneen-EasyNote-TK87:~/lab3$ gedit fork3.c
haneen@haneen-EasyNote-TK87:~/lab3$ gcc fork3.c -o fork3
fork3.c: In function ‘main’:
fork3.c:17:1: warning: incompatible implicit declaration of built-in function ‘e
xit’ [enabled by default]
haneen@haneen-EasyNote-TK87:~/lab3$ ./fork3

 I'm the child!
 I'm the parent!
 The child with pid = 2716 terminated with a status = 0
haneen@haneen-EasyNote-TK87:~/lab3$ 
```

execv ( )

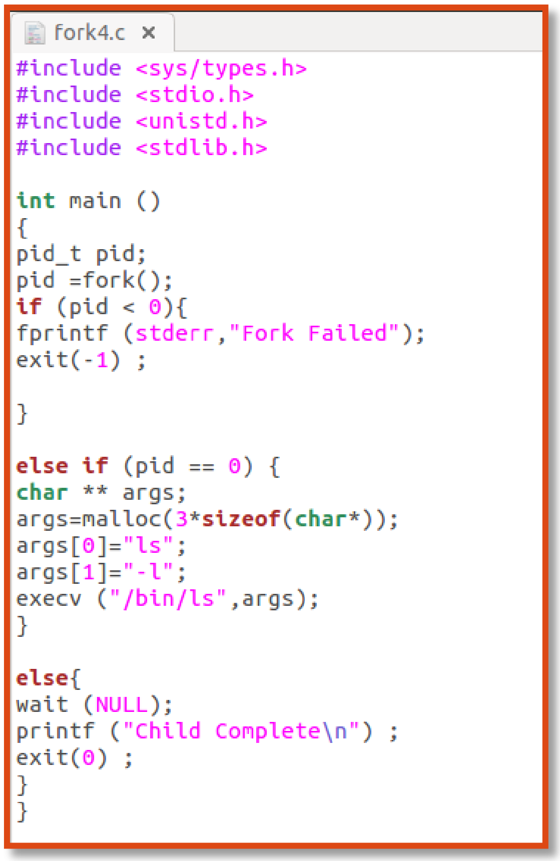
execv System call replaces the currently executing program with a newly loaded program image.

The syntax for execv would be:

execv(const char *program, char **args);

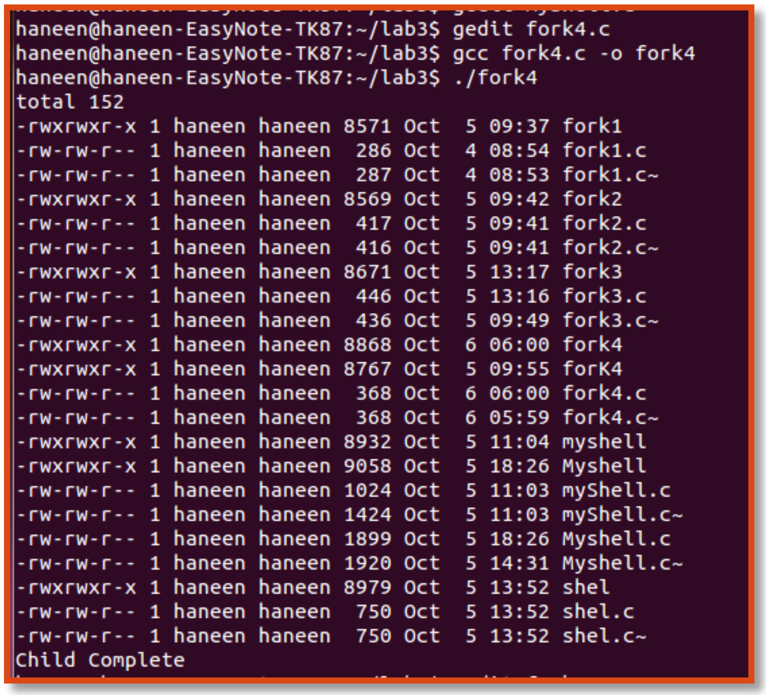
- The first argument to execv() will be a path to the program.
- The second argument to execv() will be an array of pointers to null-terminated strings that represent the argument list available to the new program.

Example 4


fork4.c

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main ()
{
pid_t pid;
pid =fork();
if (pid < 0){
fprintf (stderr,"Fork Failed");
exit(-1) ;

}

else if (pid == 0) {
char ** args;
args=malloc(3*sizeof(char*));
args[0]="ls";
args[1]="-l";
execv ("/bin/ls",args);
}

else{
wait (NULL);
printf ("Child Complete\n") ;
exit(0) ;
}
}
```

Practical:



```
haneen@haneen-EasyNote-TK87:~/lab3$ gedit fork4.c
haneen@haneen-EasyNote-TK87:~/lab3$ gcc fork4.c -o fork4
haneen@haneen-EasyNote-TK87:~/lab3$ ./fork4
total 152
-rwxrwxr-x 1 haneen haneen 8571 Oct  5 09:37 fork1
-rw-rw-r-- 1 haneen haneen  286 Oct  4 08:54 fork1.c
-rw-rw-r-- 1 haneen haneen  287 Oct  4 08:53 fork1.c~
-rwxrwxr-x 1 haneen haneen 8569 Oct  5 09:42 fork2
-rw-rw-r-- 1 haneen haneen  417 Oct  5 09:41 fork2.c
-rw-rw-r-- 1 haneen haneen  416 Oct  5 09:41 fork2.c~
-rwxrwxr-x 1 haneen haneen 8671 Oct  5 13:17 fork3
-rw-rw-r-- 1 haneen haneen  446 Oct  5 13:16 fork3.c
-rw-rw-r-- 1 haneen haneen  436 Oct  5 09:49 fork3.c~
-rwxrwxr-x 1 haneen haneen 8868 Oct  6 06:00 fork4
-rwxrwxr-x 1 haneen haneen 8767 Oct  5 09:55 forK4
-rw-rw-r-- 1 haneen haneen  368 Oct  6 06:00 fork4.c
-rw-rw-r-- 1 haneen haneen  368 Oct  6 05:59 fork4.c~
-rwxrwxr-x 1 haneen haneen 8932 Oct  5 11:04 myshell
-rwxrwxr-x 1 haneen haneen 9058 Oct  5 18:26 Myshell
-rw-rw-r-- 1 haneen haneen 1024 Oct  5 11:03 myShell.c
-rw-rw-r-- 1 haneen haneen 1424 Oct  5 11:03 myShell.c~
-rw-rw-r-- 1 haneen haneen 1899 Oct  5 18:26 Myshell.c
-rw-rw-r-- 1 haneen haneen 1920 Oct  5 14:31 Myshell.c~
-rwxrwxr-x 1 haneen haneen 8979 Oct  5 13:52 shel
-rw-rw-r-- 1 haneen haneen  750 Oct  5 13:52 shel.c
-rw-rw-r-- 1 haneen haneen  750 Oct  5 13:52 shel.c~
Child Complete
```

Shell using fork ( ) and execv ( )

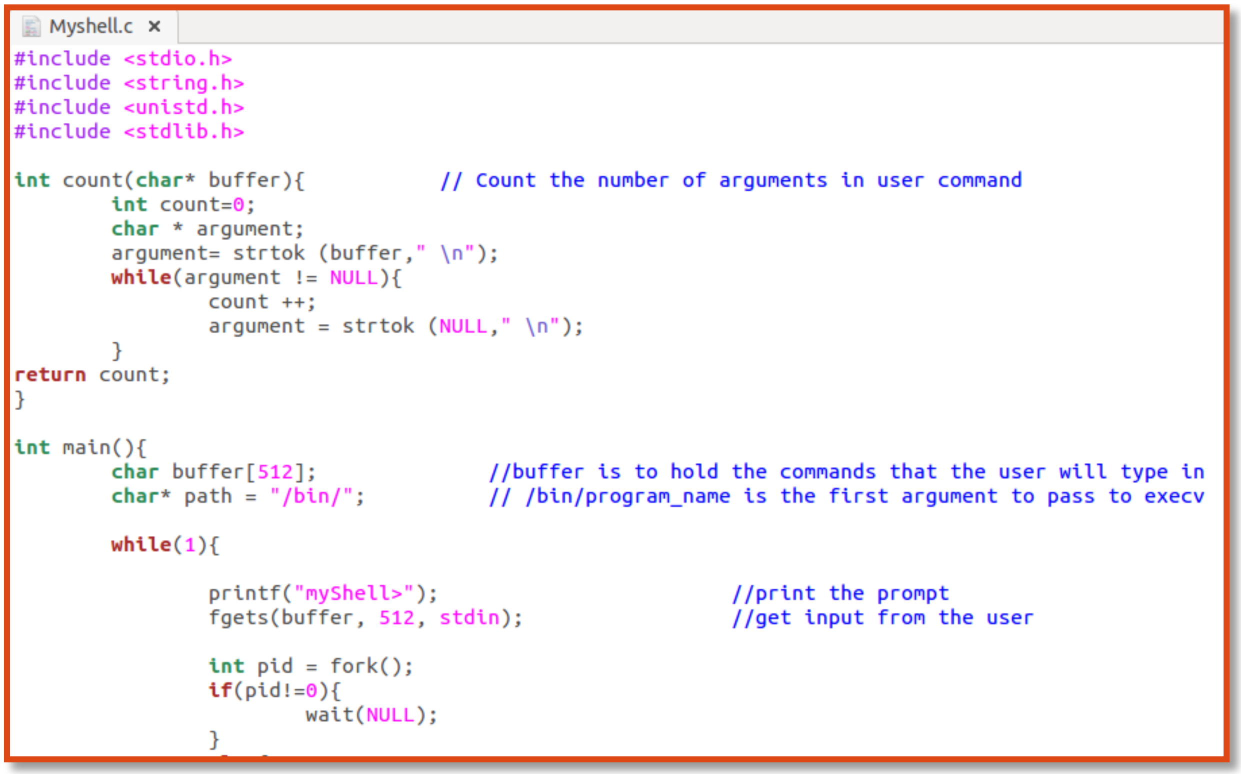
Shell is a program that takes commands from the keyboard and gives them to the operating system to perform.

Terminal is a program that opens a window and lets you interact with the shell.

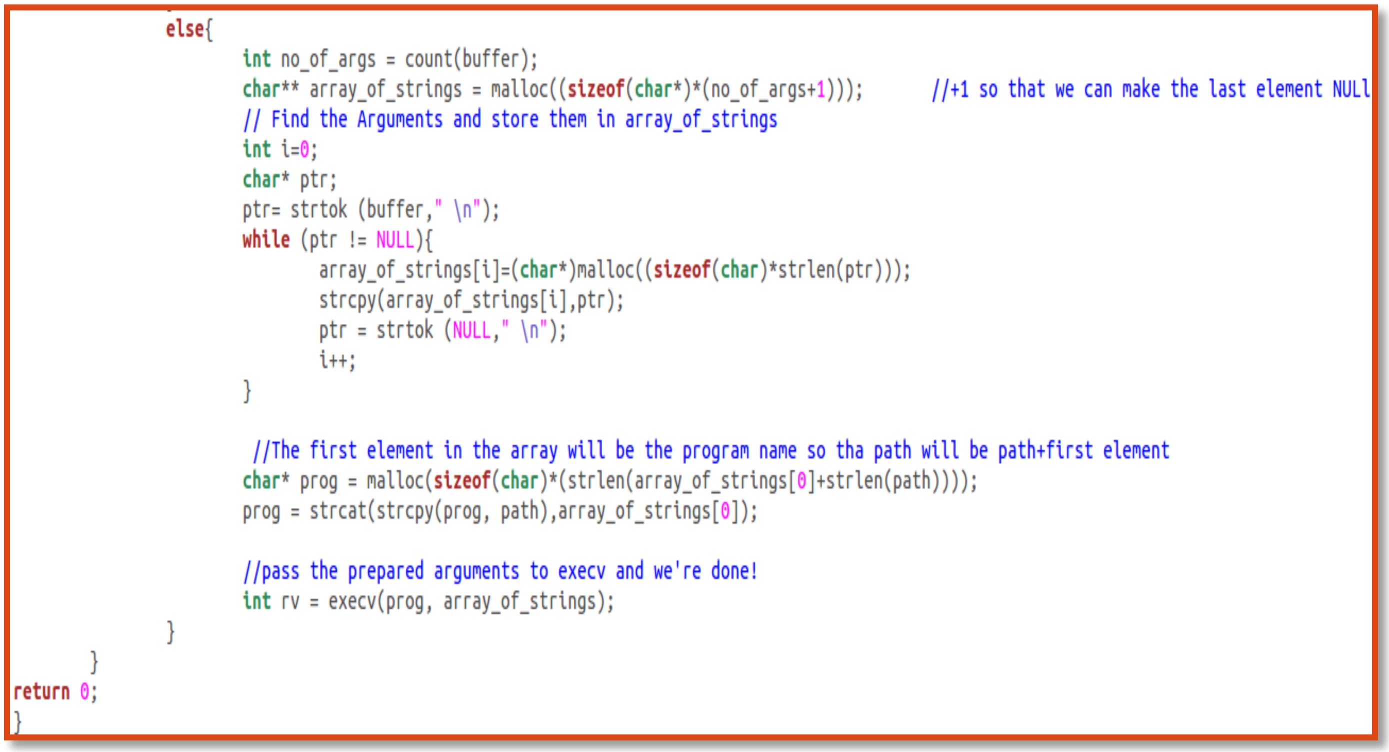
We want to create a shell interpreter that can execute commands, so:

1- Firstly, we would require fork() to spawn a new process. The new process will be an exact copy of the calling process.
2- Secondly we would require execv() to replace the currently executing program with a newly loaded program image.

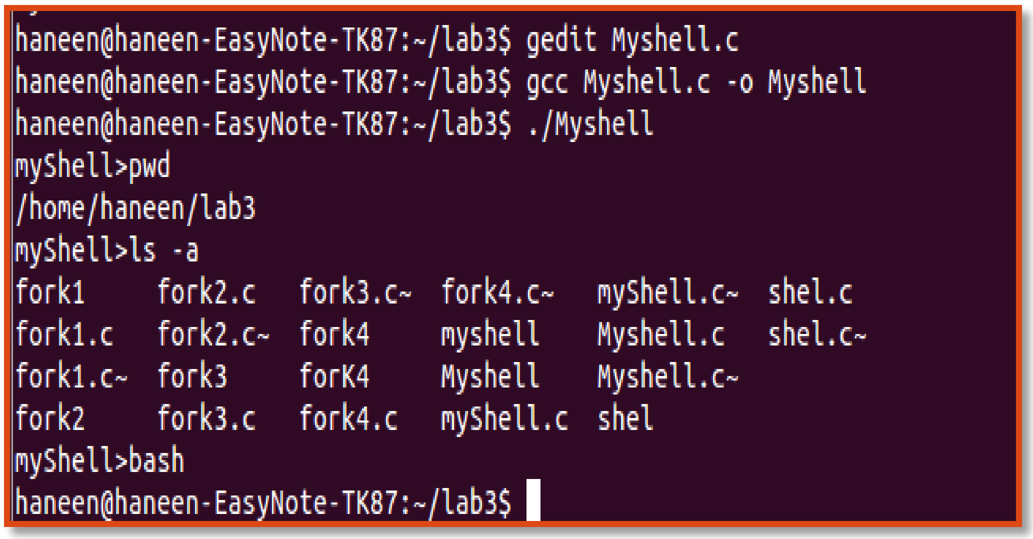
Example 5


Myshell.c ×

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

int count(char* buffer){            // Count the number of arguments in user command
        int count=0;
        char * argument;
        argument= strtok (buffer," \n");
        while(argument != NULL){
                count ++;
                argument = strtok (NULL," \n");
        }
return count;
}

int main(){
        char buffer[512];              //buffer is to hold the commands that the user will type in
        char* path = "/bin/";          // /bin/program_name is the first argument to pass to execv

        while(1){

                printf("myShell>");                    //print the prompt
                fgets(buffer, 512, stdin);             //get input from the user

                int pid = fork();
                if(pid!=0){
                        wait(NULL);
                }
```



```c
                else{
                        int no_of_args = count(buffer);
                        char** array_of_strings = malloc((sizeof(char*)*(no_of_args+1)));      //+1 so that we can make the last element NULL
                        // Find the Arguments and store them in array_of_strings
                        int i=0;
                        char* ptr;
                        ptr= strtok (buffer," \n");
                        while (ptr != NULL){
                                array_of_strings[i]=(char*)malloc((sizeof(char)*strlen(ptr)));
                                strcpy(array_of_strings[i],ptr);
                                ptr = strtok (NULL," \n");
                                i++;
                        }

                         //The first element in the array will be the program name so tha path will be path+first element
                        char* prog = malloc(sizeof(char)*(strlen(array_of_strings[0]+strlen(path))));
                        prog = strcat(strcpy(prog, path),array_of_strings[0]);

                        //pass the prepared arguments to execv and we're done!
                        int rv = execv(prog, array_of_strings);
                }
        }
return 0;
}
```



```
haneen@haneen-EasyNote-TK87:~/lab3$ gedit Myshell.c
haneen@haneen-EasyNote-TK87:~/lab3$ gcc Myshell.c -o Myshell
haneen@haneen-EasyNote-TK87:~/lab3$ ./Myshell
myShell>pwd
/home/haneen/lab3
myShell>ls -a
fork1     fork2.c   fork3.c~  fork4.c~   myShell.c~  shel.c
fork1.c   fork2.c~  fork4     myshell    Myshell.c   shel.c~
fork1.c~  fork3     forK4     Myshell    Myshell.c~
fork2     fork3.c   fork4.c   myShell.c  shel
myShell>bash
haneen@haneen-EasyNote-TK87:~/lab3$ 
```

☺ *Best Wishes* ☺