#include<sys/types.h>

#include<stdio.h>

#include<unistd.h>

int main()

{

pid\_t pid, pid1;

pid = fork();

if(pid<0)

{

fprintf(stderr,"Fork Failed");

return 1;

}

else if(pid == 0) /\* child process \*/

{

pid1 = getpid();

printf("child: pid = %d\n",pid);

printf("child: pid1 = %d\n",pid1);

}

else /\* parent process \*/

{

pid1 = getpid();rintf("parent: pid = %d\n",pid);

printf("parent: pid1 = %d\n",pid1);

}

return 0;

}

and its o/p:

parent: pid = 1836

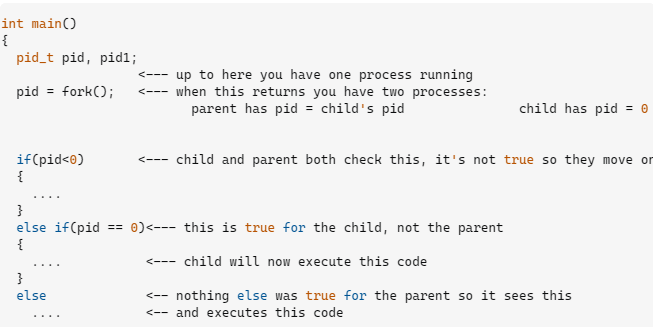
parent: pid1 = 1835

child: pid = 0

child: pid1 = 1836

Explaination

You should read up on [fork()](http://linux.die.net/man/2/fork). Once you hit a fork() statement a second process is started, it has a copy of everything the parent process has but it can run a separate execution, and the return it sees from the fork is different than what the parent sees.



So yes, you are correct, once you hit the if, or the else if or the else you're not going to get into another branch of the code, *in a single process’ execution*. You’re seeing the else if and the else because you have two processes running.

note how the pid1's are different, because getpid() is returning which process is running that code, and you can see you have two different processes, one picks the else if the other picks the else

fork actually returns zero for the child process id and non-zero for the parent means the actual process id of child is returned to parent

after else if(pid ==0 ) {...} is child process

and else {...} is parent process.

Basic meaning to fork is to create new process .

If main calls one fork() then only child is created with it's own address space. just below the fork call all the statements are same for the parent and child process.

creating a process using fork means these two parent and child are independent process and don't share any data between them by default

But in your case after fork call these two are independent so the order of execution of these two processes are unspecified

There are two different processes which are running your program. After the call of fork, there are two executions of the following instructions, and they have different values of pid. So both else if and else parts are executed, each by one of the two processes.