fork()

- Fork system call is used for creating a new process, which is called ***child process***, which runs concurrently with the process that makes the fork() call (parent process).
- After a new child process is created, both processes will execute the next instruction following the fork() system call.
- A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

It takes no parameters and returns an integer value. Below are different values returned by fork().

Negative Value: creation of a child process was unsuccessful.

Zero: Returned to the newly created child process.

Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

To install C on Centos

 # yum groups install Development Tools

How to create programme

nano fork.c

To compile

gcc fork.c

To run

#./a.out

fork() in C

There are no arguments in fork() and the return type of fork() is integer. You have to include the following header files when fork() is used:

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

When working with fork(), <sys/types.h> can be used for type ***pid_t*** for processes ID's as pid_t is defined in <sys/types.h>.

The header file <unistd.h> is where fork() is defined so you have to include it to your program to use fork().

The return type is defined in <sys/types.h> and fork() call is defined in <unistd.h>. Therefore, you need to include both in your program to use fork() system call.

Example 1: Calling fork()

Consider the following example in which we have used the fork() system call to create a new child process:

CODE:

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

int main()
{
    fork();
    printf("Using fork() system call\n");
    return 0;
}
```

OUTPUT:

```
Using fork() system call
Using fork() system call
```

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

int main()
{
    pid_t p;
    p = fork();
    if(p==-1)
    {
        printf("There is an error while calling fork()");
    }
    if(p==0)
    {
        printf("We are in the child process");
    }
    else
    {
        printf("We are in the parent process");
    }
    return 0;
}
```

OUTPUT:

We are in the parent process
We are in the child process

To create Zombie process

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

int main()
{
int pid=fork(); //create new process

if(pid>0)
{
        sleep(60); //parent sleeps for 60 sec

}
else
{       exit(0);  // child exits before the parent & child becomes zombie

}
 return 0;
}
```

OUTPUT

 # ./a.out &

[1] 9006

 # pstree -p 9006

a.out(9006) ------ âââa.out(9007) // 9007 is child PID

 # ps -aux | grep 9007

user1 9007 0.0 0.0 0 0 pts/0 Z 12:19 0:00 [a.outct>

To create Orphan process

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

int main()
{
int pid=fork(); //create new process

if(pid>0)
{
        exit(0);  //parent exit before child

}
else if(pid==0)
{

sleep(60);  // child sleeps for 60 second
}
 return 0;
}
```

```
[user1@localhost shell]$ ./a.out &
[1] 10506
[1]+  Done                    ./a.out

[user1@localhost shell]$ ps -aux | grep a.out

user1     10510  0.0  0.0   4212    88 pts/0    S    12:32   0:00 ./a.out
user1     10515  0.0  0.0 112812   984 pts/0    R+   12:33   0:00 grep --color=auto a.out

[user1@localhost shell]$ ps -o ppid 10510
 PPID
    1

[user1@localhost shell]$ pstree -p 1

systemd(1)─┬─ModemManager(6586)─┬─{ModemManager}(6639)
           │                    └─{ModemManager}(6642)
           ├─NetworkManager(6726)─┬─dhclient(7036)
           │                      ├─{NetworkManager}(6736)
           │                      └─{NetworkManager}(6738)
           ├─VGAuthService(6584)
           ├─a.out(10510)
           ├─abrt-watch-log(6588)
           ├─abrt-watch-log(6590)
```