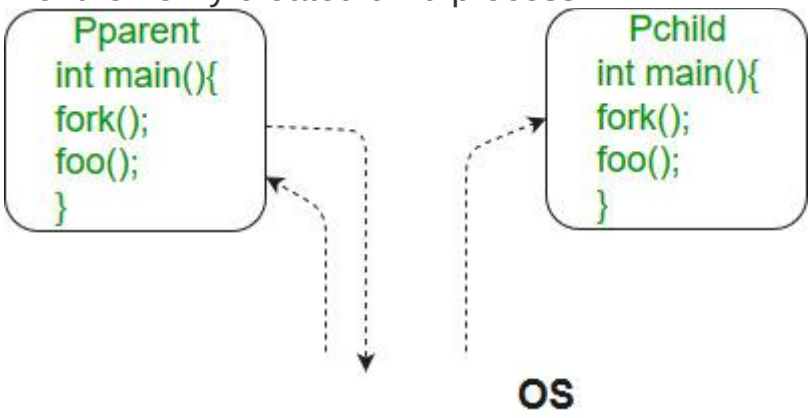
fork() in C

The Fork system call is used for creating a new process in Linux, and Unix systems, which is called the ***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.
The child process uses the same pc(program counter), same CPU registers, and 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**: The creation of a child process was unsuccessful, then it returns -1.
- **Zero**: Returned to the newly created child process.
- **Positive value**: Returned to parent or caller. The value contains the process ID of the newly created child process.



Note: *fork() is threading based function, to get the correct output run the program on a local system.*

Also, if you're interested in understanding process control and memory allocation in C, the **C Programming Course Online with Data Structures** covers these topics in depth.
Please note that the below programs don't compile in a Windows environment.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{

    // make two process which run same
    // program after this instruction
    pid_t p = fork();
```

```
    if(p<0){
      perror("fork fail");
      exit(1);
    }
    printf("Hello world!, process_id(pid) = %d \n",getpid());
    return 0;
}
```

Output

```
Hello world!, process_id(pid) = 31

Hello world!, process_id(pid) = 32
```

Example 2: Calculate the number of times hello is printed.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork();
    fork();
    fork();
    printf("hello\n");
    return 0;
}
```

Output

```
hello

hello

hello

hello

hello

hello

hello

hello
```

Explanation

The number of times 'hello' is printed is equal to the number of processes created. Total Number of Processes = 2^n, where n is the number of fork system calls. So here n = 3, 2^3 = 8 Let us put some label names for the three lines:

```
fork ();    // Line 1
fork ();    // Line 2
fork ();    // Line 3
        L1        // There will be 1 child process
    /      \      // created by line 1.
```

```
  L2      L2    // There will be 2 child processes
 /  \    /  \   //  created by line 2
L3  L3  L3  L3  // There will be 4 child processes
                // created by line 3
```

So there is a total of eight processes (new child processes and one original process). If we want to represent the relationship between the processes as a tree hierarchy it would be the following:

- Main process: **P0**
- Processes created by the **1st fork: P1**
- Processes created by the **2nd fork: P2, P3**
- Processes created by the **3rd fork: P4, P5, P6, P7**

Example 3: Predict the Output of the following program.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
   pid_t p;
   p = fork();
   if(p<0)
   {
     perror("fork fail");
     exit(1);
   }
   // child process because return value zero
   else if ( p == 0)
      printf("Hello from Child!\n");

   // parent process because return value non-zero.
   else
      printf("Hello from Parent!\n");
}
int main()
{
   forkexample();
   return 0;
}
```

Output

```
Hello from Parent!

Hello from Child!
```