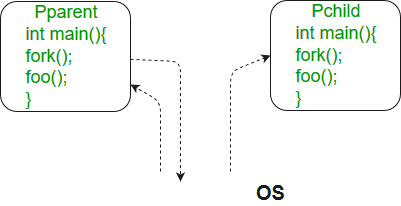
**fork() system call**

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.
* ***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.*

***Please note that the above programs don’t compile in a Windows environment.***

**Example of fork() in C**

* C

|  |
| --- |
| #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.**

* C

|  |
| --- |
| #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 = 2n, where n is the number of fork system calls. So here n = 3, 23 = 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: The 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

P0

/ | \

P1 P4 P2

/ \ \

P3 P6 P5

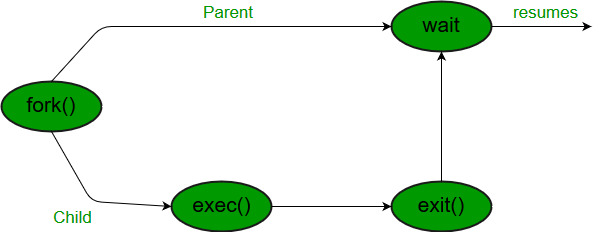
/

P7

**Wait System Call**

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent ***continues*** its execution after wait system call instruction.   
Child process may terminate due to any of these:

* It calls exit();
* It returns (an int) from main
* It receives a signal (from the OS or another process) whose default action is to terminate.



**execv system call**

The execv system call is a function in the C programming language that replaces the current process image with a new process image. The new process image is loaded into the current process space and begins execution from the entry point of the new program. The execv system call is part of the exec family of functions, which are used to run a new program in the current process space. [The exec family of functions can be used to run any program files, including binary executables or shell scripts 1](https://www.geeksforgeeks.org/exec-family-of-functions-in-c/)