2. Gyak

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.

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// make two process which run same

// program after this instruction

fork();

printf("Hello world!\n");

return 0;

}

Output:

Hello world!

Hello world!

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void forkexample()

{

// child process because return value zero

if (fork() == 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:

1.

Hello from Child!

Hello from Parent!

(or)

2.

Hello from Parent!

Hello from Child!

In the above code, a child process is created. fork() returns 0 in the child process and positive integer in the parent process.  
Here, two outputs are possible because the parent process and child process are running concurrently. So we don’t know whether the OS will first give control to the parent process or the child process.

**Important:** Parent process and child process are running the same program, but it does not mean they are identical. OS allocate different data and states for these two processes, and the control flow of these processes can be different.

#include <stdio.h>

#include <sys/types.h>

int main()

{

fork();

fork();

fork();

printf("hello\n");

return 0;

}

Output: (8db hello)

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

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