UNIX Fork

In UNIX, The ‘fork’ system call is the way to create a new process from an existing process. When a process calls ‘fork’, it creates a copy, called the child, of the original process, called the parent. After the ‘fork’ is called, both the parent and child process will execute the instructions immediately after the system call. Changes to the variables in one process will not affect the variables in the other.

The parent and the child process will have different address spaces. The code will be shared, but the data can and probably will be subject to change. The operating system will use the copy-on-write strategy for efficient memory management and to handle the changing data. Instead of the child process having an exact copy of the parent process memory, the parent and the child will share the memory, which is marked read-only. When one process needs to change something, that portion is copied, instead of everything being copied.

When calling ‘fork()’, it returns a value, of type “pid_t” (a signed integer), to both the child and the parent process. The child process is returned 0, and the parent process is returned the identifier of its child process, called the PID. It is a number greater than 0. If ‘fork()’ returns something less than 0, then the system call had failed and a new process was not created. This happens when the system lacks the resources to create a new process, such as too many processes running.

The parent process is in the running state when it initially calls the fork. The child process is in the new state as it is created, and then goes into the ready state as it waits for the OS to put it in the running state.

As a demonstration of the fork process, here is some C code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    fork();
    fork();
    fork();
    printf("My Process ID is %d\n", (int) getpid());
    return 0;
}
```

The first line is calling the 'fork()' function, creating a child process. The child process will start executing at the second line, also calling another 'fork()', and then a third 'fork()' is called. Because the child process starts executing the next line of instructions, some of these child processes will have children of their own. The program prints some text, including its PID using the 'getpid()' function. The output will look something like this:

```
My Process ID is 4716
My Process ID is 4719
My Process ID is 4717
My Process ID is 4718
My Process ID is 4720
My Process ID is 4721
My Process ID is 4722
My Process ID is 4723
```

Links

https://man7.org/linux/man-pages/man2/fork.2.html
https://stackoverflow.com/questions/7455161/what-happens-when-i-call-fork-in-unix
https://www.geeksforgeeks.org/fork-system-call/