Tiffany Tang
tt2405

The UNIX “fork” command is a method of process creation where a ‘child’ process will be created and split from its ‘parent’ process. This system call is a fundamental feature in UNIX operating systems and is a cornerstone of network programming.

In UNIX there is only one system call to create a new process - fork(). The fork command takes no parameters and will return a process ID upon creation. This call creates a ‘child’ process, the calling process (‘parent’ process) in which, the two processes will have the exact same memory image, environment string and open files. If fork is successfully executed, UNIX will make two separate identical address spaces (process ID), one for the parent and the other for the child, and both processes will start their execution at the next statement following the fork() command. Usually the child process can then execute execve or a similar system call to change its memory image and run a new program.

Before the fork command is executed, the parent process must already be in the *running* state and actively executing its instruction. After the fork command, both the parent and the child process are in the *running* state. As the parent calls on the fork command and the operating system begins to create the child, the chile process is in the *new start*. At this point the child process is an exact replica of the parent but has its own PID. Once all the resources are gathered for the child process, it is said to be in the *ready* state and is waiting for the operating system to schedule when to run on the CPU. A *blocked* state occurs when a process makes a request that is going to take a significant amount of time. This state indicates that the process is no longer ready to run, but it is not quite done yet. Both the parent and the child can enter the blocked state if they make a request that the operating system deems too costly. When a process finally finishes, it moves to the *exit* state. Here the parent process will wait for the child process to terminate with the system call waitpid() and will give the return value back and release the resources associated with the child, once the child is finished. There can be three values returned: a negative value, which signifies that the creation of the child process was unsuccessful, a zero, which will be returned to the newly created child process, and a positive value, which is the child's PID that will be returned back to the parent process.

The UNIX "Fork" command has been a critical aspect of network programming over the years. I would like you to write a (roughly) one page summary of what the function does and what activities you'd expect to see in the operating system after calling this function from a program. Please pay careful attention to the various states that a process goes through and explain which states you'd expect each process to be in and why.

Notes and rough outline:

A new process is created by having an existing process execute a process creation system call
System tells OS to create a new process and indicates which program to run it
The function creates a new copy called the *child* out of the *parent* process.When the parent process closes or crashes, the child process is also killed.

In UNIX, there is only one system call to create a new process: fork

- Call creates an exact clone of the calling process
- After the fork, the two processes (parent and child) have the same memory image, same environment string, same open flies
- Usually child process then executes execve or a similar system call to change its memory image and run a new program
- The reason for the two step process is to allow the child to manipulate its file descriptors after the fork, but before the execve in order to accomplish redirection of standard input, output and error

- After a new child process is created, both processes will execute the next instruction following the fork() system call, therefore we have to distinguish the parent from the child.
    - Can do so by testing the returned value of fork()
    - Negative value: creation of the child process was unsuccessful
    - Fork will return a 0 to the newly created child process and it will return a positive value (the process ID) of the child process back to the parent process

In UNIX after a process is created:

- Parent and child have both their own distinct address space, and if either process changes its address, the change is not visible to the other process
- The child's initial address space is a copy of the parents
    - No writable memory is shared
- The child process inherits an exact copy of the parent process's memory, file descriptors and other resources

Process termination:
Exit in UNIX is voluntary and happens when the process has done its work

In UNIX a process and all of its children and further descendants together form a process group

- When a user sends a signal from the keyboard the signal is delivered to all members of the process group currently associated with the keyboard
- Processes in UNIX cannot disinherit their children

States:

1. Running state:
    a. Before the 'fork' command is executed: The parent process is in the running state and actively executing its instructions
    b. After the 'fork' command: both the parent and child are in the running state, executing the same set of instructions from the point of the command
        i. They have distinct address spaces (PID's)
2. New state:
    a. After the command, the child process will be in the new state as the operating system creates a copy of the parent process, including code, data, stack and other resources that were apart of the parent process.
    b. At this point the child process is an exact replica of the parent, but it has its own process ID
3. Ready state
    a. Once the parent and child move to this state, it means that they are ready to be executed but are waiting for the CPU to be assigned by the operating system scheduler. The operating system is in charge of determining which processes are allocated which times on the CPU, this may differ between systems.
4. Blocked state
    a. Blocked states occur when a process makes a request that is going to take a significant amount of time. This state indicates that the process is no longer ready to run, it is not running, but it is not quite done yet.
    b. Both the child or the parent process can enter the blocked state if they encounter operations that require blocking, eg waiting for a file to be retrieved.
5. Exit state
    a. When a process finally finishes, it moves to the exit state, here the process is just waiting to be cleaned up and terminated, and the return values are given back to the process
    b. The parent process will wait for the child process to terminate with a system call waitpid() and will give the return value back and release the resources associated with the child, once the child is finished