HW19: Understanding the fork system call in UNIX

The fork system call is a critical function for process creation in UNIX-like operating systems. It enables the implementation of multiprogramming by allowing a process, the parent, to create a new process, the child. The child is almost an exact duplicate of the parent at the time of the fork.

When fork is invoked, it creates a new process. The child process inherits the parent's address space, which includes the text (code), data, stack, and heap segments. However, modern systems employ a copy-on-write technique, which avoids the actual duplication of memory pages until one of the processes attempts to modify them. This method is efficient and helps to save system resources.

Upon successful completion, fork returns twice: In the parent process, it returns the PID (Process ID) of the newly created child process. In the child process, it returns 0, indicating that it is the child. If fork encounters an error, such as the inability to allocate necessary resources, it returns a negative value in the parent process.

Normally, the parent process will be put to ‘ready’ state from ‘running’ and the child process will be put to ‘running’ state from ‘ready’ immediately following the call. The CPU time of the operations above is allocated by the scheduler. The child process may be ‘blocked’ if it is waiting for I/O operations or other resources until it happens, then it will be put to ‘ready’ state waiting for the scheduler’s commands.

However, more generally speaking, the immediate states of both the parent and the child process is solely determined by the scheduler after the call. The decision will be based on the current load on the system, the algorithm used, and the priority of all processes. The exact states of each process cannot be guaranteed.