Marta Bukowska (mlb9559)

UNIX “fork” Command

The fork command in UNIX-based systems is a method for creating new processes. It is called by an existing (parent) process to create a new (child) process that is identical to the calling process, except for the process identifier (PID). This means that the child process has an exact copy of the parent process's memory segments, including code, data, heap, stack, and registers, as well as the process's environment and open file descriptors. The fork function takes no parameters. If successful, it returns 0 to the child process and the child’s PID to the parent process. If it fails, -1 is returned to the parent process. These return values allow us to determine if the fork was successful and the relationship between the two processes.

To understand the operating system’s (OS) role in process creation, it’s crucial to analyze the state transitions of the parent and child processes. For the parent process to call fork, it must be actively executing instructions and therefore have allocated CPU time by the OS scheduler, meaning it is in the "running" state. When the fork system call is executed, the OS begins by creating a new process control block (PCB) for the child process. It duplicates the parent's address space, assigns a PID, and copies the parent's execution context (registers, stack, etc.). During this phase, the child process is in the conceptual "new" state, where it doesn't fully exist yet, as it is still being set up. Once the creation and initialization are complete, the child process moves to the "ready" state, where it waits for CPU availability. After the fork, the child and parent processes execute independently.

The child process can transition from the "ready" state to "running" when it gets CPU time. It can also enter the "blocked" state if waiting for an event, or be moved to "blocked/suspended" or "ready/suspended" states if it needs to be swapped out of main memory. Typically, the child process needs to execute a new program. In this case, it calls execve (or a similar system call, with different parameters) to replace its memory image with a new program, effectively transforming it into a different process with the same PID. When the child finishes execution, it transitions to the "exit" state, awaiting cleanup by the OS. All of the above transitions are managed by the OS.

After the fork, the parent process may continue in the "running" state and proceed with its execution, or return to the "ready" state if the OS scheduler decides to run another process. It can also call wait or waitpid to wait for any or a particular child process to terminate. If it does so, the parent process is moved to the "blocked" state, where it remains until the child process exits. Once that happens, it returns to the "ready" state. The parent process can also experience the same state transitions as described for the child process.

The OS is also responsible for memory management after the fork. Although the child’s initial address space is a copy of the parent’s address space, the parent and child processes have distinct address spaces. Initially, these address spaces map to the same physical memory locations. However, when either process attempts to modify the "shared memory", the system called Copy-On-Write (COW) is used. The OS makes a copy of the memory segment for the modifying process. Consequently, changes are made only to this private copy, ensuring no memory is shared between the parent and child processes, and changes made by one do not affect the other.

The fork command is crucial in UNIX-based systems, as it enables the creation of new processes, allowing them to execute independently. The OS plays a vital role in this process by setting up the PCB, scheduling CPU time, ensuring state transitions, and handling memory management.

Sources:

- Tanenbaum, Andrew S., and Herbert Bos. Modern Operating Systems. 4th ed., Pearson, 2015, pp. 53-55, 90, 736-739.
- Sowmya, K. N. "A Study on Circumstances of Fork() and Exec() Process Model in LINUX System." Journal of Emerging Technologies and Innovative Research (JETIR), vol. 10, no. 3, March 2023. Available at: https://www.jetir.org/papers/JETIR2303723.pdf. Accessed 16 June 2024.