HW 19

Joo Hyun Lee
jhl504@nyu.edu
NYUCSBR Summer 24-week

One of the fundamental features of UNIX operating system is `fork()`. The `fork()` command is the primary method of creating a separate, duplicate process called a *child* process out of the original process called a *parent* process.

Although the child process is exactly same as the parent process and that the child process is made by copying the address space of the parent process, they are still separate processes, hence, have different memory spaces. The child process uses the same process counter, registers, and information of open files as the parent process. Thus, when the `fork()` is called, the system creates a new PCB for the child process by copying some parts of of PCB of the parent process. Also, it uses *copy on write*, so the child and parent processes can utilize variables independent from each other. This can be done because the operating systems copy internal data structures to manage the child process.

`fork()` does not take any parameter and returns a process ID (PID) of the child process in the parent process if the child process is successfully created and 0 in the child process. If it fails to create the child process, it returns -1.

Before the `fork()` command, the parent process must already be in *running* state and actively executing its instruction. Once the command is called, the child process is being created and is in *new* state. Once the new PID is assigned to the child process and the creation is done, the chlid process now enters *ready* state. After the `fork()` command, Both processes are in *running* state concurrently and resume the exact instruction right after the fork system call.

It is not predictable, however, to know which of the processes is executed first, creating a possibility that the child process becomes the orphan proess–a process that is still running but whose parent process no longer exists (either killed or exited). In order to avoid this, it is safe to place the parent process in *wait* state using the `wait()` command, which make the parent process placed in *wait* state. With the `wait()` call on the parent process, the child process is executed completely and moves to *exit* state, the parent process is returned to *running* state, executed till the end of the instruction, and becomes *terminated* state.