HW #13 FORK()

Gowtham Rajeshshekaran

gr2180

fork() is a UNIX command that is used to create **new** processes. The calling process is called the **parent** process and the newly created process is called the **child** process. The fork() system call takes no arguments and returns an integer value. When fork() is called, the kernel in OS creates an exact **duplicate** of the parent process (in **RUNNING** state) and the newly created child process (in **NEW** state) will have its own **address space** and unique process ID (**pid**). It also uses the same program counter, file descriptors, CPU registers, and the program text of the parent process. After the creation of the child process, both the processes will be in **READY** state and will run **concurrently** with the help of scheduler scheduling along with other **READY** processes. Both processes will start its execution from the **next instruction** after fork() in the program code. The parent or the child process that is picked by the scheduler will be moved to **RUNNING** state from **READY** state and it will be executed in the CPU while the other staying in the **READY** state waiting for its chance. The program can use the fork call's return value to tell whether the execution is in the parent or the child process.

- If the return value is **0**, the program executes in the **child** process.
- If the return value is **greater than 0**, the program executes in the **parent** process. In this case, the value returned to the parent is the **pid** of the child process.
- If the return value is **less than 0**, then the fork call is **unsuccessful** and no new child process will be created.

If the child process wants to execute a **different program** from that of the parent, **exec()** system call is used to replace the current program with a new executable, which is given as its first parameter.

The parent process can use **wait()** system call to suspend its execution, moving to the **BLOCKED** state until the child process completes its execution. On success, it returns the **pid** of the terminated child. Wait() is used to avoid the unpredictable **kernel scheduling** between these two processes by letting one process to finish first. Also, using wait() the parent process can obtain the **exit status** of the terminated child. After the child is terminated, the parent process will move to **READY** state from **BLOCKED** and waits for CPU time for its execution.