Name: Minjie Shen
NetID: ms10733
Homework # 13

The UNIX “fork” command: *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.*

In UNIX, there is only one system call to create a new process: fork. This call creates an exact clone of the calling process. When calling the “Fork” function, we are expected to see that there will be a copy of the main process (which is called a parent process) created as a child process.

“Fork” function takes in no value and returns an integer. When it returns a “0”, this means that it is in a child process; when it returns an integer that is greater than “0”, this means that it is in a parent process; when it returns an integer that is less than “0”, this means that the process is error and no new child process will be created.

After the fork, the two processes, the parent and the child, have the same memory image, the same environment strings, and the same open files. That is all there is. Usually, the child process then executes execve or a similar system call to change its memory image and run a new program. For example, when a user types a command, say, sort, to the shell, the shell forks off a child process and the child executes sort. The reason for this 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, standard output, and standard error.

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 them 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.