Briana Singleton
NYU ID: bfs254
HW 19
26 November 2021

The fork function is the only method used to create new processes in UNIX [1]. Fork operations initially produce an exact duplicate process of an existing process [1]. A new process begins in the "New" state. The "New" state allows the operating system to accumulate the data a new process needs and way to track the progress of that accumulation. To create a new process, an existing process makes a process creation system call [1] and switches from the "Running" state into the "Blocked" state to wait on the operating system's assistance and signal it is not finished but also no longer ready to run. A process creation system call tells the operating system to create a new process [1]. It also specifies which program code to run in the new process [1] and indicates the roles, either "child" or "parent," of the processes through varying return values [3]. Fork returns once with a value of negative one to the parent process to indicate creation failure [3]. Fork returns twice with a value of zero to the newly created child process and with a value of the child's process ID number to the parent process to indicate creation success [3].

As a newly created child is a clone of its parent, both have the same memory image, environment strings, open files, variables, registers, and other information [2], but they have different address spaces to prevent changes in memory one makes from affecting the other [1]. Their data is thus shared using two different approaches. The first is that the child's initial address space is a separate copy of the parent's address space with no writable memory shared between them because their address spaces are distinct [1]. The second and less expensive approach is that the child may directly access all of the parent's memory copy-on-write [1]. Meaning, if modifications are to be made, the necessary section of memory is explicitly copied and altered in a memory area exclusive to the process making the changes [1]. Still, there are some resources that are shared openly between parent and child [1]. One such resource is open files: any alterations made by either process are shared not only between parent and child, but also the rest of the system [1]. Another such resource is, depending on the UNIX system, program text as program text cannot be changed [1].

Once a child process has had its data prepared and both the child and parent processes have received their return values from the fork operation, fork has completed its function and the parent and child processes are able to move on to the "Ready" queue. Once the parent is selected from the "Ready" queue, it has two courses of action. The first is that it will continue execution in the "Running" state from the next instruction in its code following the fork call [4]. The second is that it will try to synchronize its actions with its child process by waiting for the child process to stop or terminate [4]. The parent process is able to wait on its child using the wait system call [4]. This will bring the parent into the "Suspended/Blocked" state as the parent may have no indication of how long the child will be on the "Ready" queue or how long the child will take to execute, especially considering interrupts as well as the child's own cycles between "Ready," "Running," and "Blocked." Once the child is selected from "Ready," it also has two possibilities as to what actions it can take. In the case that the child process is to execute the same code as its parent, the child will also continue its execution in the "Running" state from the next instruction in its code following the fork operation that created it [4]. However, in the more common case that the child process is to do a different task from its parent, the child will have to make an exec or similar system call to have its data overlayed by the executable file an exec operation will produce [2]. Doing this would expectedly move it from the "Running" state to the "Blocked" state in order to have its data overlayed by the executable called by the exec (or similar) function. The child process would then find itself back in the "Ready" queue cycling

Briana Singleton
NYU ID: bfs254
HW 19
26 November 2021

through “Running,” “Ready,” and, potentially, “Blocked” until termination with an exit system call to the operating system [1]. Following which, its parent, if it is suspended and waiting, can receive the child’s termination status via wait’s return value [4].

Briana Singleton
NYU ID: bfs254
HW 19
26 November 2021

Reference List

[1] A. S. Tanenbaum and H. Bos, "Processes and Threads," in *Modern Operating Systems*. T. Johnson, 4th ed. Upper Saddle River, NJ, USA: Pearson Prentice-Hall, 2015, ch. 2, sec. 2.1.2, pp. 89 – 91.

[2] A. S. Tanenbaum and H. Bos, "Case Study 1: UNIX, LINUX, and Android," in *Modern Operating Systems*. T. Johnson, 4th ed. Upper Saddle River, NJ, USA: Pearson Prentice-Hall, 2015, ch. 10, sec. 10.3.2, pp. 736 – 737, 742.

[3] M. Kerrisk. "fork(2)—Linux manual page." Man7.org. https://man7.org/linux/man-pages/man2/fork.2.html#RETURN_VALUE (accessed November 26, 2021).

[4] "CS330 Intro to Processes, Forks & Exec." Uregina.ca. https://www.cs.uregina.ca/Links/class-info/330/Fork/fork.html (accessed November 25, 2021).