fork1 starts by printing that it has started, sleeping for 3 seconds, then printing that it has begun the fork process. After forking it stores the returned PID (which is the child's PID) in a variable. If the PID is less than 0, the program prints an error message and exits. Otherwise, the parent prints out its information and variables before waiting for the child process to finish executing. The child process then prints its information and asks for a user-given return value. It then exits using the given return code, which is stored by the parent for the final printf statements before the parent exits. The system calls in the program are printf, wait, sleep, fork, perror, and exit.

fork2 loops, creating a total of 5 children. Each time it creates a child it prints the child's PID and then loops again, exiting after the loop. The children print their PID as well before sleeping for 3 seconds. After 3 seconds, each child wakes up, prints that it has woken along with its PID for reference, and then exits. The truly interesting thing about fork2 is that the order in which things are processed seems inconsistent. It is impossible to predict whether the parent will spawn more than one child before the first child executes, a pattern that continues throughout the execution of fork2. The system calls in fork2 are printf, fork, sleep, and exit.