Santa Clara University
Computer Science and Engineering
Advanced Operating System(COEN 383)
Project 6
Group 4:

Robin Lee
Anand Santosh Kulkarni
Justin Li
Nikhil Kavuru
Prahlad Kulkarni

In this assignment, we must simulate communication between parent and child processes using a pipe. A pipe is a virtual file that is created in memory and has both a write and a read file descriptor, but it is never written to the file system. Our simulation has a straightforward design. We forked 5 different child processes after creating 5 pipes in the main process. Then we assigned one pipe to each of the child's processes. The child process uses this pipe to deliver messages to the main process as required by the project, and the 'main' process reads from all of these child processes and publishes the output to "Output.txt" as soon as the pipe is read.

We've had to deal with the following issues:

---

1. After forking, all five pipes constructed were split among the five offspring.

Issue:

When the primary process forks a child process, the young process inherits the main process's entire memory and stack.
Challenge:

To complete our answer, we must first design 5 pipes and then allocate each pipe to one of the children. This brings up the same problem as in the previous section. However, pipes that do not belong to a specific kid are shared with it, making backpropagation of pipe closure impossible to detect for the parent.

Resolution:

For the unused pipe, each child must close both the read and write file descriptors.

2. When the child process closes the dedicated pipe's write file descriptor, the parent process is unable to detect it.

Issue:

each child process must use a "close" system call to close its pipe after simulation. A combination of "select" and "read" system calls can be used by the parent process to detect it. When a child process shuts its pipe, the "select" system call informs the parent that there is data at the pipe's end; however, when the parent reads that data, the number of bytes read is "0," indicating that the pipe has been closed by the child process.
Challenge:

When a pipe has no open write file descriptor, an EOF byte is inserted to it. When we fork a child after building a pipe, both the main and child processes have pipes that write file descriptors, breaching the preceding criterion. As a result, a child's end of a pipe does not indicate to the parent any EOF character in the pipe.
Resolution:

The unused file descriptor must be closed by both the parent and the child. For example, the pipe's write descriptor should be closed by the parent, whereas the pipe's read descriptor should be closed by the kid.

3. At the conclusion of the simulation, the termination of the fifth child.

Issue: The 5th child in this project has the unique need of reading input from the terminal and communicating it to the parent process. Another project needed was to stop the simulation after 30 seconds by closing the dedicated stream.
Challenge:

We were reading from stdin using "scanf" at first. "scanf" is now a blocking call that is based on the user's response. As a result, using "scanf" pauses the application from running and waits for a USER I/O operation. When we tried to terminate the child process at the end of the simulation, this reliance caused a problem. Because our 5th kid was waiting for the user's input, it was unable to verify the simulation time on a regular basis.

Resolution: We were able to overcome this problem by using "select" and "read" on "STDIN." As a result, whenever data from standard input has to be read, just the 5th child will do so. As a result, the 5th child can keep track of the simulation time in a non-blocking manner.