COEN383

Advanced Operating System

Project 6 Report

Group No. 8

Team Members:

Akaash Meghraj Trivedi (W1650955)
Anagha Viswanath (W1651360)
Michael Zarrabi (W1479183)
Nakul Pravin Thombare (W1650130)
Sai Sanjana Reddy Vatte (W1650167)

For this task, we needed to replicate communication between parent and child processes by employing a pipe. A pipe is essentially a virtual file that is generated in memory and possesses both read and write file descriptors but is not written in the file system. Our simulation follows a simple structure. Following the creation of 5 pipes in the main process, we initiated the forking of 5 distinct child processes. Each child process was assigned a specific pipe. The child processes utilize these pipes to transmit messages to the main process as outlined in the project requirements. The 'main' process, in turn, reads from all these child processes and outputs the results to "Output.txt" after reading from the pipe.

We faced the following issues:

1. If the child process closes the write file descriptor of its dedicated pipe, the parent process faces difficulty detecting it.

   Each child process needs to utilize a "close" system call to properly close its pipe during the simulation. The parent process can employ a combination of "select" and "read" system calls to identify the closure. When a child process closes its pipe, the "select" system call notifies the parent about available data at the pipe's end. However, upon reading the data, the parent observes zero bytes read, indicating the child process has closed the pipe.

   - Challenge: A pipe lacking an open write file descriptor receives an EOF byte. Since both the main and child processes possess pipes with write file descriptors after forking, this violates the previous condition. Consequently, the closure of a child's pipe does not convey an EOF character to the parent.

- Resolution: Both the parent and child must close the unused file descriptor. Specifically, the parent should close the write descriptor of the pipe, while the child should close the read descriptor.

2. After the forking process, the five constructed pipes were distributed among the five offspring.

When the main process forks a child process, the child inherits the entire memory and stack of the main process.
- Challenge: To address this, we need to design five pipes and assign each pipe to one of the children. This introduces a similar issue as in the previous section. Pipes that are not specifically allocated to a child are shared with it, complicating the detection of pipe closure backpropagation for the parent.
- Resolution: To resolve this, each child must close both the read and write file descriptors for the unused pipe.

3. Upon completing the simulation, the fifth child' termination.

In this project, the fifth child has a unique requirement to read input from the terminal and communicate it to the parent process. Another project objective was to conclude the simulation after 30 seconds by closing the dedicated stream.
- Challenge: Initially, we utilized "scanf" to read from stdin. However, "scanf" is a blocking call dependent on the user's response, causing the application to pause and await USER I/O operations. This posed an issue when attempting to terminate the fifth child process at the simulation's end, as it was waiting for user input, hindering regular verification of the simulation time.
- Resolution: To address this challenge, we successfully implemented "select" and "read" on "STDIN." Now, only the fifth child reads data from standard input when necessary. Consequently, the fifth child can monitor the simulation time in a non-blocking manner, resolving the issue.