Advanced Operating Systems
Project-3 Group 2

Group Members:

1. Aifaz Maknojia
2. Samarth Kulkarni
3. Samuel Karumanchi
4. Sashidhar Kancharla
5. Vedant Raj Kavi

Introduction

This assignment involved simulating communication between parent and child processes using pipes. A pipe is a virtual file created in memory, capable of both write and read operations, but it does not persist in the file system. The simulation design is straightforward: the main process creates five pipes and forks five separate child processes. Each child process is assigned a dedicated pipe to send messages to the parent process. The parent process reads these messages and writes the output to a file, "Output.txt," as soon as it receives the data.

Challenges and Resolutions

1. Parent Process Unable to Detect Pipe Closure by Child

•Context:

Each child process must close its pipe after completing its task using the close system call. The parent process should detect this closure using a combination of the select and read system calls. When a child closes its pipe, select indicates data availability, and read returns 0 bytes, signaling the pipe's closure. However, this behavior depends on all write file descriptors for the pipe being closed.

•Issue:

After forking a child, both the parent and the child processes possess open write file descriptors for the pipe, violating the condition for signaling an EOF. Consequently, the parent could not detect the closure of the pipe by the child.

•Resolution:

To address this, both parent and child processes must close their unused file descriptors. Specifically:

•The parent should close the write descriptor for the pipe.

•The child should close the read descriptor for the pipe.

2. Shared Pipes Between All Child Processes

•Context:

When the main process forks a child, its entire memory, including the pipes, is replicated in the child process. As a result, all five pipes are shared among all child processes.

•Issue:

Since unused pipes are shared across all child processes, the backpropagation of pipe closure to the parent becomes problematic. Pipes that do not belong to a specific child also remain open, preventing the detection of EOF by the parent.

•Resolution:

Each child must close both the read and write file descriptors for all pipes it does not use. This ensures that the parent process correctly detects the closure of the assigned pipe.

3. Terminating the Fifth Child Process After Simulation Ends

•Context:

The fifth child process had a unique property: it needed to read input from the terminal and communicate it back to the parent process. The simulation required termination after 30 seconds by closing the assigned pipe.

•Issue:

Initially, the fifth child used scanf to read from standard input. However, scanf is a blocking call and waits indefinitely for user input. This behavior prevented the process from continuously checking the simulation time and terminating properly.

•Resolution:

To resolve this, we replaced scanf with a combination of select and read on STDIN. This approach allows the fifth child to read from the standard input only when data is available. As a result, the process can continuously monitor the simulation time in a non-blocking manner and terminate correctly.

Conclusion

This simulation successfully demonstrated inter-process communication using pipes. The challenges encountered, including detecting pipe closures, managing shared pipes, and handling non-blocking input for the fifth child, were resolved through effective use of system calls like select and read. These solutions ensured the parent process could manage communication efficiently and the simulation terminated as expected.