CSEN 383: Advanced Operating Systems

Group 3 Project 6 Report

Members: Lydia Myla, Praful Mamidi, Joshua Kindarara, Samantha Lee, and Guru Sushma Kasa

Objective: In a multiplexed manner, your main process will read from multiple pipes, from the standard input (the terminal), and write to the file system.

Issues Encountered:

- **Reliable, bounded writes to the pipe using a shared buffer :** One practical challenge we faced while implementing writeToPipe() was safely sending variable-length messages from each child using a fixed-size shared buffer. All messages are first built into the global read_buffer, so we had to avoid overflowing it or writing beyond the valid text. To handle this, we compute the message length with strlen(read_buffer) and clamp it to BUFFER_SIZE - 1 before calling write(), so we never exceed the protocol size. We also respect the global is_timeout flag set by the SIGALRM handler: once the 30-second window ends, writeToPipe() simply returns and stops sending data. Together, the timeout guard and length clamping make the pipe writes robust for both snprintf-based messages (children 1–4) and fgets input from child 5.
- This is a very small minor issue, but we were confused as to how to include the various timestamps and show this in our output file. It took us a while but we were able to figure out how to display the stamps for both the parent and child and have it set up where multiple children were under one parent.
- Another issue was about writes colliding with each other. For example, if multiple children were writing simultaneously, messages could mix if written at the same time, especially if write() is not atomic. So to fix this and prevent it from happening, we limited message length to BUFFER_SIZE, and made sure each child was writing a complete message one at a time. Then we added fflush() after the parent writes to the file to maintain the correct order. Additionally, we used usleep(100), which gives the parent a tiny window (100 microseconds) to read the message before the child potentially loops back with **sleep(0)** again. This reduces the chance of multiple messages being buffered together.
- Overall, we didn't have many major issues encountered, mainly just dealing with minor bugs that were a result of user mistakes.