COEN383 Advanced Operating System Report Project : 6

Group No: 4
Patrick Lee
Piyush Joshi
Sudarshan Mehta
Manasa Madiraju

Problem Statement:

This assignment aims to practice UNIX I/O system calls in C by implementing a multiplexed process where the main process reads from multiple pipes, standard input, and writes to the file system. Five pipes are created, each connected to a child process which writes to its pipe. The first four child processes generate time-stamped messages at random intervals, while the fifth process prompts for user input, writes messages to its pipe, and terminates after 30 seconds. The parent process reads from the pipes using the select() system call, writes messages to an output file with dual time stamps, and terminates after all child processes finish. The output file, output.txt, contains a mixture of lines from child processes.

We have faced the following challenges during finding solution to above problem statement:

1. **Challenge:**
   The parent process encountered difficulty in detecting when a child process closed its dedicated pipe's write file descriptor. This was problematic as the parent process relied on a combination of the "select" and "read" system calls to identify when the child process closed its pipe. Despite the child process closing the pipe, the parent process received no indication of closure, resulting in a failure to detect the end of data transmission.

   Solution:
   To address this issue, both parent and child processes needed to close unused file descriptors. Specifically, the parent process should close the write descriptor for the pipe, while the child process should close the read descriptor. By ensuring both processes closed unnecessary descriptors, the parent process could accurately detect the closure of the pipe by the child process, as indicated by the "select" system call returning a read size of 0.

2. **Challenge:**
   All five pipes created were shared among all five children after forking. This occurred because when the main process forked a child process, the entire memory stack was replicated to the child process, including pipes that didn't belong to it. Consequently, it was impossible for the parent process to detect when a child process closed a pipe that didn't belong to it.

Solution:
To mitigate this issue, each child process needed to close both read and write file descriptors for any unused pipes. By ensuring that each child only operated on its designated pipe, the parent process can accurately monitor the closure of pipes associated with each child process.

3. **Challenge:**
   Terminating the fifth child after the end of the simulation posed a challenge due to its special requirement to read input from the terminal and communicate it back to the parent process. Initially, using "scanf" for input caused program execution to halt, waiting for user input, making it impossible for the child process to continuously monitor the simulation time and terminate appropriately.

   Solution:
   The challenge was resolved by implementing non-blocking input reading using "select" and "read" on "STDIN". This allowed the fifth child process to monitor the simulation time continuously while only reading from standard input when data was available, ensuring smooth termination after the designated simulation period.

4. **Challenge:**
   Efficient management of system resources, such as file descriptors and memory, is crucial to prevent resource exhaustion and optimize performance, especially in long-running simulations with multiple processes.

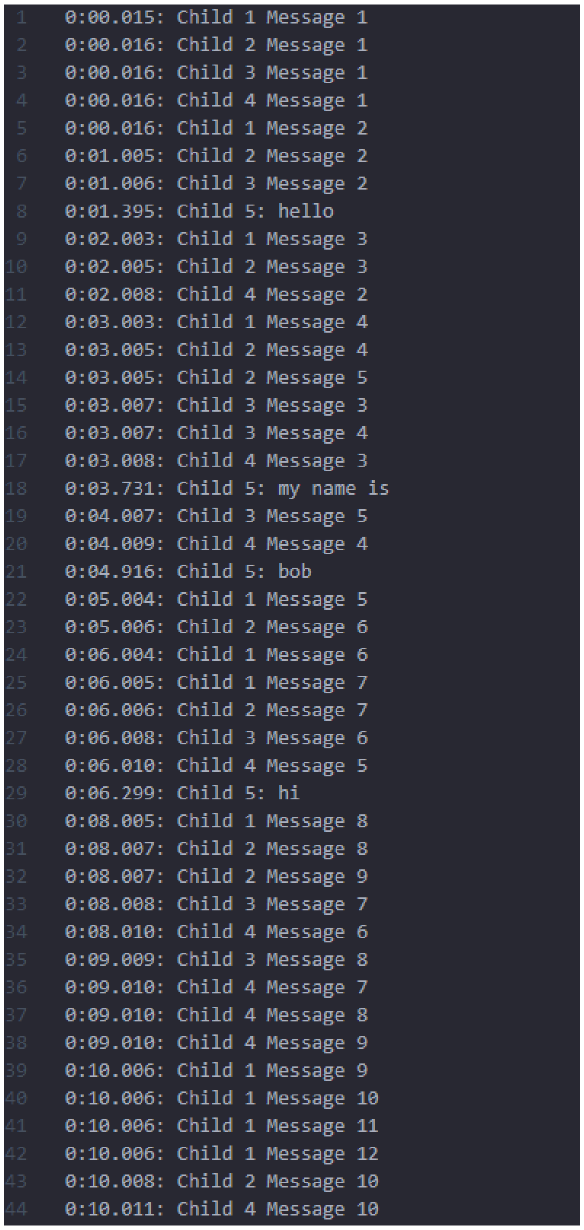
   Solution:
   Efficient management of system resources, particularly file descriptors, in the code is achieved by closing unused file descriptors in both parent and child processes. Each child process closes the file descriptors it doesn't need, ensuring that resources are released when they are no longer required. Additionally, the parent process closes file descriptors associated with pipes after reading or writing data, preventing resource exhaustion and optimizing performance by freeing up system resources promptly. This approach minimizes the risk of resource leaks and improves the scalability and efficiency of the program, particularly in long-running simulations involving multiple processes.

5. **Challenge:**
   Handling input from the terminal while ensuring non-blocking behavior and timely termination of the fifth child process presents a significant challenge. The initial approach using scanf() for input caused program execution to block, waiting for user input and hindering the child process's ability to monitor the simulation time accurately.

   Solution:
   We overcame this challenge by implementing non-blocking input reading using the select() and read() system calls on the standard input (STDIN). By employing

non-blocking input reading techniques, the fifth child process can continuously monitor the simulation time without being blocked by user input operations. This ensures that the child process remains responsive and can terminate smoothly after the designated simulation period, contributing to the overall efficiency and reliability of the program.

Output.txt screenshot:



```
0:00.015: Child 1 Message 1
0:00.016: Child 2 Message 1
0:00.016: Child 3 Message 1
0:00.016: Child 4 Message 1
0:00.016: Child 1 Message 2
0:01.005: Child 2 Message 2
0:01.006: Child 3 Message 2
0:01.395: Child 5: hello
0:02.003: Child 1 Message 3
0:02.005: Child 2 Message 3
0:02.008: Child 4 Message 2
0:03.003: Child 1 Message 4
0:03.005: Child 2 Message 4
0:03.005: Child 2 Message 5
0:03.007: Child 3 Message 3
0:03.007: Child 3 Message 4
0:03.008: Child 4 Message 3
0:03.731: Child 5: my name is
0:04.007: Child 3 Message 5
0:04.009: Child 4 Message 4
0:04.916: Child 5: bob
0:05.004: Child 1 Message 5
0:05.006: Child 2 Message 6
0:06.004: Child 1 Message 6
0:06.005: Child 1 Message 7
0:06.006: Child 2 Message 7
0:06.008: Child 3 Message 6
0:06.010: Child 4 Message 5
0:06.299: Child 5: hi
0:08.005: Child 1 Message 8
0:08.007: Child 2 Message 8
0:08.007: Child 2 Message 9
0:08.008: Child 3 Message 7
0:08.010: Child 4 Message 6
0:09.009: Child 3 Message 8
0:09.010: Child 4 Message 7
0:09.010: Child 4 Message 8
0:09.010: Child 4 Message 9
0:10.006: Child 1 Message 9
0:10.006: Child 1 Message 10
0:10.006: Child 1 Message 11
0:10.006: Child 1 Message 12
0:10.008: Child 2 Message 10
0:10.011: Child 4 Message 10
```