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

Group 4:
Robin Lee
Anand Santosh Kulkarni
Justin Li
Nikhil Kavuru
Pralhad Kulkarni

The issue we encountered are as follows :

Issue with Pipe Allocation:

- **Issue :** After forking, all five pipes constructed were split among the five offspring.
- **Challenge :** When the primary process forks a child process, the young process inherits the main process's entire memory and stack. Designing 5 pipes and allocating each pipe to one of the children poses a challenge as pipes not belonging to a specific child are shared with it, making backpropagation of pipe closure detection impossible for the parent.
- **Resolution :** Each child must close both the read and write file descriptors for the unused pipe.

Issue with Detecting Pipe Closure:

- **Issue :** When the child process closes the dedicated pipe's write file descriptor, the parent process is unable to detect it.
- **Challenge :** Each child process must use a "close" system call to close its pipe after simulation. The combination of "select" and "read" system calls can be used by the parent process to detect closure. However, when the parent reads data after the "select" call, the number of bytes read is "0," indicating that the pipe has been closed by the child process.
- **Resolution :** Both the parent and the child must close the unused file descriptor. For example, the pipe's write descriptor should be closed by the parent, whereas the pipe's read descriptor should be closed by the child.

Issue with Fifth Child Termination:

- **Issue :** At the conclusion of the simulation, the termination of the fifth child was problematic due to its unique input-reading requirement.

- **Challenge :** The fifth child reads input from the terminal and communicates it to the parent. Initially using "scanf," which is a blocking call, posed a problem as it waits for user input, causing a delay in checking the simulation time regularly.
- **Resolution :** Overcame the problem by using "select" and "read" on "STDIN" to read data from standard input in a non-blocking manner, allowing the fifth child to keep track of simulation time effectively.

Inaccurate Timestamps:

- **Issue:** The timestamps generated for each message in the child processes are based on the system time, which may not be precise enough for the intended granularity of 1000th of a second.
- **Challenge:** The gettimeofday() function provides time in seconds and microseconds, but the conversion to a timestamp might lose precision when represented as a floating-point number.
- **Resolution:** To achieve more accurate timestamps, consider using a high-resolution timer or a dedicated library for timestamp generation, ensuring that the timestamps reflect the intended precision.

Issue with Random Waiting Time:

- Issue: The implementation of random waiting times for the first four child processes between 0, 1, or 2 seconds could potentially result in a delay in the termination of the overall program.
- Challenge: The random_wait_time is generated using rand() % 4, which may lead to a maximum wait time of 3 seconds instead of the intended 2 seconds. This could cause the program to run for a more extended period than the specified 30 seconds.
- Resolution: Adjust the random number generation to ensure that the waiting time falls within the desired range (0, 1, or 2 seconds) to adhere to the specified program duration.