COEN-383
Advanced Operating System
Project 6 Report
Group 4
Rima Modak(W1650730),
Gouri Lalitha Priyanka Tummalapenta(W1650617),
RaghaSrilakshmi Sumithra Mounika Yalamarty(7700012596)

Project Title: UNIX I/O System Calls with Pipes and select()

Objective

The primary goal of this project is to gain hands-on experience with UNIX I/O system calls using the C programming language. The central process will simultaneously manage inputs from several pipes and the standard input device (the terminal) and then output the data to the file system.

The project involves the use of pipe-based inter-process communication. Pipes act as temporary, in-memory files with reading and writing capabilities without committing data to the physical file system.

The project's structure is as follows: the primary process initiates five pipes and subsequently generates five unique child processes. Each child is allocated a specific pipe through which it communicates with the primary process. The primary process then receives messages through these pipes, processes them, and records the data into a file named "Output.txt".

Challenges and Resolutions

While developing main.c, several challenges were encountered, particularly with the select() function call, file descriptor set handling, and the gettimeofday() function call. These issues are common in projects involving inter-process communication and time-sensitive operations.

1. select() Function Call

Issue:
The select() system call monitors multiple file descriptors, waiting until one or more of them becomes "ready" for some class of I/O operation (e.g., input possible). Common issues with select() include the proper setup of file descriptor sets for reading, writing, and error detection. If not correctly initialized and updated, select() might not behave as expected, leading to missed signals or infinite waits.

- **Problem Encountered:** Not resetting the file descriptor sets before each select() call or incorrectly calculating the maximum file descriptor value plus one, required for the first parameter of select(), led to issues.

Resolution:
We utilized the C macro FD_SETSIZE as a substitute for nfds, simplifying the calculation required for setting up select(). This adjustment ensured proper initialization and management of file descriptors.

2. gettimeofday() Function Call

Issue:
The gettimeofday() function is used to obtain the current time with precision up to microseconds. In main.c, this function is crucial for timestamping messages from child processes. A potential issue is

handling the wraparound of the microsecond field (tv_usec), which can lead to incorrect time calculations.

- **Problem Encountered:** Incorrect time calculations led to negative time differences or incorrect durations, affecting the logic dependent on accurate timing. The timeDiff function and other time-related calculations had to account for the possibility of overflow or underflow in time calculations, especially with microsecond precision.

Resolution:
We experimented with the getTime() function as an alternative to timeDiff for the parent process timestamp. Calculating the parent timestamp concerning the child start times was challenging, but careful handling of time fields and overflow conditions resolved this issue.

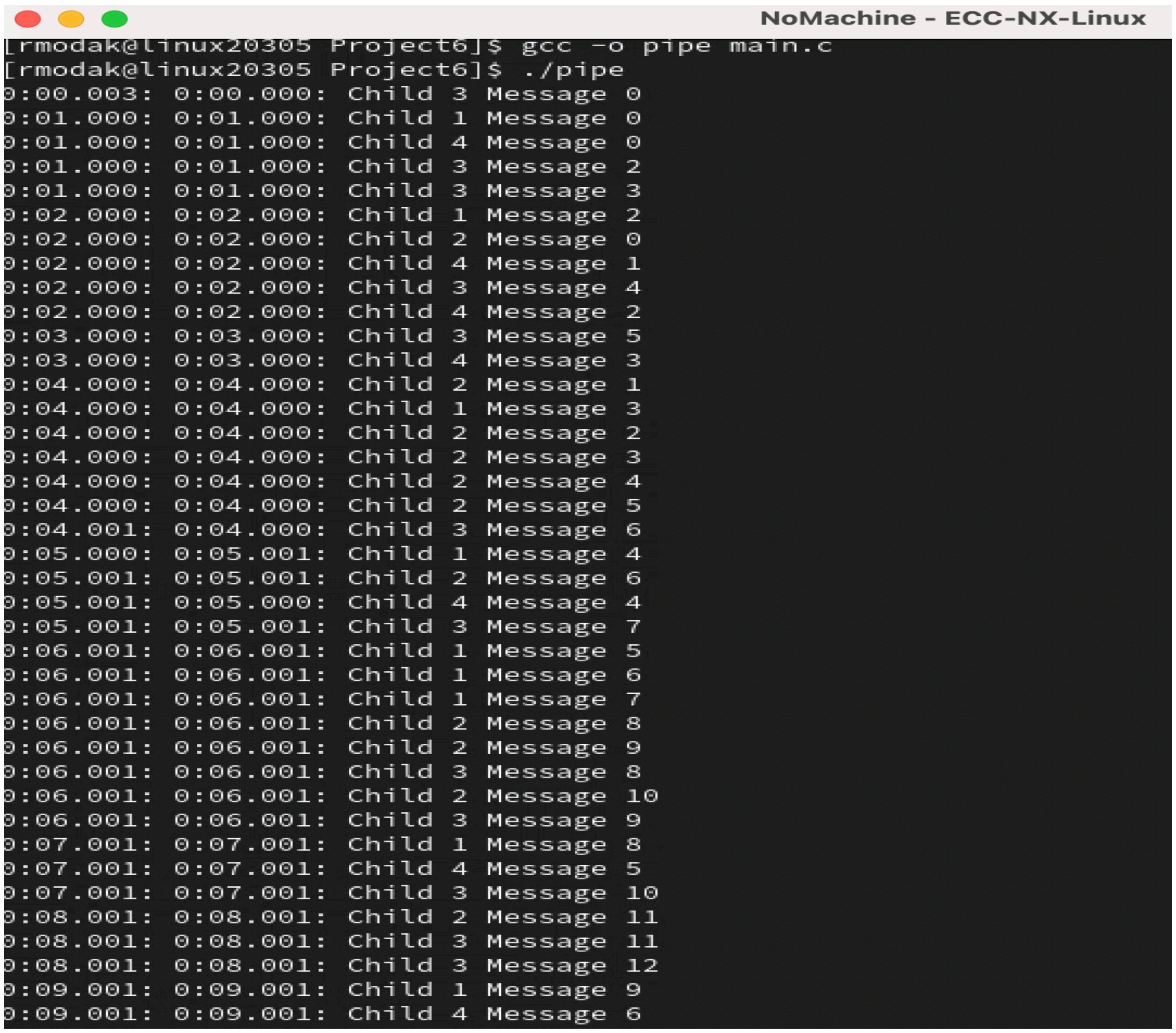
3. EOF Detection and Pipe Closure

Issue:
The parent process was unable to detect when the child process closed the dedicated pipe's write file descriptor.

- **Problem Encountered:** The parent process failed to recognize the closure of a child process's pipe due to both retaining the write file descriptor post-fork, preventing EOF detection.

Resolution:
To solve this, both parties had to close their unused ends: the parent closed the write end, and the child closed the read end. This ensured proper EOF signaling and closure detection, allowing the parent process to detect the end-of-file condition correctly.

Output:



```
[rmodak@linux20305 Project6]$ gcc -o pipe main.c
[rmodak@linux20305 Project6]$ ./pipe
0:00.003: 0:00.000: Child 3 Message 0
0:01.000: 0:01.000: Child 1 Message 0
0:01.000: 0:01.000: Child 4 Message 0
0:01.000: 0:01.000: Child 3 Message 2
0:01.000: 0:01.000: Child 3 Message 3
0:02.000: 0:02.000: Child 1 Message 2
0:02.000: 0:02.000: Child 2 Message 0
0:02.000: 0:02.000: Child 4 Message 1
0:02.000: 0:02.000: Child 3 Message 4
0:02.000: 0:02.000: Child 4 Message 2
0:03.000: 0:03.000: Child 3 Message 5
0:03.000: 0:03.000: Child 4 Message 3
0:04.000: 0:04.000: Child 2 Message 1
0:04.000: 0:04.000: Child 1 Message 3
0:04.000: 0:04.000: Child 2 Message 2
0:04.000: 0:04.000: Child 2 Message 3
0:04.000: 0:04.000: Child 2 Message 4
0:04.000: 0:04.000: Child 2 Message 5
0:04.001: 0:04.000: Child 3 Message 6
0:05.000: 0:05.001: Child 1 Message 4
0:05.001: 0:05.001: Child 2 Message 6
0:05.001: 0:05.000: Child 4 Message 4
0:05.001: 0:05.001: Child 3 Message 7
0:06.001: 0:06.001: Child 1 Message 5
0:06.001: 0:06.001: Child 1 Message 6
0:06.001: 0:06.001: Child 1 Message 7
0:06.001: 0:06.001: Child 2 Message 8
0:06.001: 0:06.001: Child 2 Message 9
0:06.001: 0:06.001: Child 3 Message 8
0:06.001: 0:06.001: Child 2 Message 10
0:06.001: 0:06.001: Child 3 Message 9
0:07.001: 0:07.001: Child 1 Message 8
0:07.001: 0:07.001: Child 4 Message 5
0:07.001: 0:07.001: Child 3 Message 10
0:08.001: 0:08.001: Child 2 Message 11
0:08.001: 0:08.001: Child 3 Message 11
0:08.001: 0:08.001: Child 3 Message 12
0:09.001: 0:09.001: Child 1 Message 9
0:09.001: 0:09.001: Child 4 Message 6
```

Conclusion

Developing main.c for Project 6 presented challenges primarily related to the select() function call, file descriptor set handling, and the gettimeofday() function call. These issues highlighted the complexities involved in writing concurrent, time-sensitive applications in C. Proper initialization, careful management of file descriptors, and accurate time calculations were critical to addressing these challenges.

By addressing these issues, we improved the robustness and reliability of the inter-process communication mechanism, leading to a successful implementation of the project objectives.