COEN383

Advanced Operating System

Project 6 Report and Issues faced

Group No. 5

Group Members:

1. Bharti Prakash - W1652174

2. Iniyan Chandran Ramachandran - W1651510

3. Ruthu Rajendra - W1653722

4. Siddhi Sanjay Powar - W1650216

5. Vaibhav Sachdeva - W1650084

Objective

The primary goal of this task 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.

To facilitate interaction between the parent and child processes, a pipe-based simulation has been developed. Pipes serve as temporary, in-memory files that come with both reading and writing capabilities, yet they do not commit any data to the physical file system.

The structure of the simulation is clear-cut: the primary process initiates five pipes and subsequently generates five unique child processes. Each child is allocated a particular pipe through which it communicates with the primary process, as per the project's specifications. Upon receiving messages through these pipes, the primary process efficiently processes and records the data into a file named "Output.txt".

When developing `main.c` for project6, several issues were encountered, particularly with the `select()` function call, file descriptor set handling, and the `gettimeofday()` function call. These challenges are common in projects involving inter-process communication and time-sensitive operations. Below, we detail these issues based on the provided context and reference materials.

Issues Encountered

1. `select()` Function Call

The `select()` system call is used to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O operation (e.g., input possible). A common issue with `select()` involves correctly setting up the file descriptor sets for reading, writing, and error detection. If not properly initialized and updated, `select()` might not behave as expected, leading to missed signals or infinite waits. Typical problems we faced were not resetting the file descriptor sets before each `select()` call or incorrectly calculating the maximum file descriptor value plus one, which is required for the first parameter of `select()`.

Resolution:

The example program in “select.c” provides the C macro “FD_SETSIZE” in place of “nfds” which proved to be a simple alternative to the otherwise required calculation. This helped us set up our select().

2. `gettimeofday()` Function Call

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 with `gettimeofday()` is handling the wraparound of the microsecond field (`tv_usec`), which can lead to incorrect time calculations. This is particularly relevant when calculating time differences or durations, as seen in the `timeDiff` function.The timeDiff function and other time-related calculations had to account for the possibility of overflow or underflow in time calculations, especially when dealing with microsecond precision. Incorrect calculations lead to negative time differences or incorrect durations, affecting the logic dependent on accurate timing. We had to resolve this by experimenting with getTime() function as an alternative to timeDiff for the parent process timestamp. Calculating the parent timestamp with respect to the child start times was challenging.

3. Parent process was unable to detect when the child process closes the dedicated pipe’s write file descriptor.

The problem arises when the parent process fails to recognize the closure of a child process's pipe due to both retaining the write file descriptor post-fork, preventing EOF detection. To solve this, both parties has to close their unused ends: the parent the write end, and the child the read end, ensuring proper EOF signaling and closure detection.

Conclusion

The development of `main.c` for project6 presented challenges primarily related to the `select()` function call, file descriptor set handling, and the `gettimeofday()` function call. These issues show the complexities involved in writing concurrent, time-sensitive applications in C. Proper initialization, careful management of file descriptors, and accurate time calculations are critical to addressing these challenges.