

CSEN 383 – Assignment 4 **Winter 2024**

Group 2

Chaitra Boggaram (W1651213)
Divyanth Chalicham (07700005995)
Shreya Chinthala (07700005606)
Ruchi Manikrao Dhole (W1652116)
Rohit Parthiban (07700006502)

Objective:

The objective of this project is to demonstrate proficiency in handling UNIX I/O, process management, and inter-process communication using pipes in a C programming environment.

Insights From Question:

- The main process creates five pipes and spawns five child processes

Child Process:

- Each child process is connected to a specific pipe.
- The first four child processes generate time-stamped messages every few seconds and write them to their respective pipes. Each message is timestamped to the nearest 1000th of a second and includes the child process number.
- Fifth child process reads input from the user via terminal and writes messages (with timestamps) to its pipe. It prompts for input repeatedly and terminates after 30 seconds.

Parents Process:

- The parent process reads from all pipes in a multiplexed manner using the select() system call to determine which pipes have input available.

Output.txt:

- Each line in the output file is time stamped by the parent process and includes the original timestamp from the child process.

Termination :

- Once all child processes have terminated, the parent process also terminates, ending the program.

Challenges Faced:

1. The parent process finds it difficult to identify if the child process shuts the write file descriptor of its dedicated pipe. Throughout the simulation, each child process must make use of the "close" system function to correctly close its pipe. To determine the closure, the parent process may use a mix of "select" and "read" system calls. The "select" system call alerts the parent process to data that is accessible at the pipe's end when a child process ends its pipe. But when the parent reads the data, it finds that there are 0 bytes read, which means the child process has closed the pipe. An EOF byte is sent to a pipe that does not have an open write file descriptor. This breaks the prior criterion because pipes with write file descriptors are present in both the parent and child processes after forking. As a result, a parent does not get an EOF character when their child's pipe is closed.

Solution:

The unused file descriptor has to be closed by both the parent and the child. In order to be more specific, the child should close the read descriptor of the pipe and the parent should close the write descriptor.

2. When the main process forks a child process, the child inherits the entire memory and stack of the main process. To address this, we need to design five pipes and assign each pipe to one of the children. This introduces a similar issue as in the previous section. Pipes that are not specifically allocated to a child are shared with it, complicating the detection of pipe closure backpropagation for the parent.

Solution:

To resolve this, each child must close both the read and write file descriptors for the unused pipe.

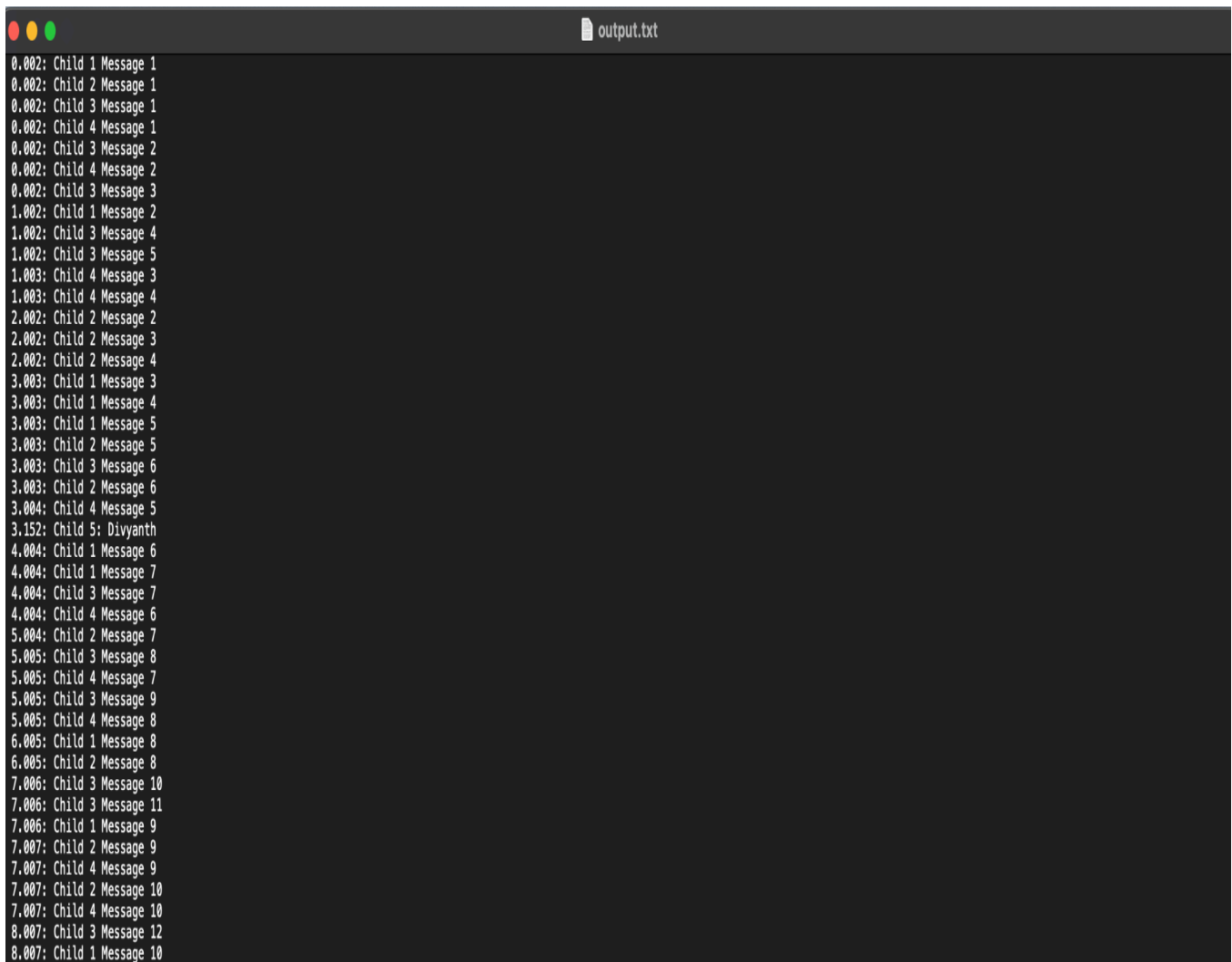
3. After the simulation ended, the fifth child was terminated. The fifth child in this project has a special need to read input from the terminal and relay

it to the parent process. Another goal for the project was to close the dedicated stream and end the simulation after 30 seconds. Initially, we read from stdin using "scanf". But "scanf" is a blocking call that waits for a response from the user, pausing the program to wait for USER I/O activities. Because it was awaiting user input, this created a problem when trying to halt the fifth child process at the conclusion of the simulation, impeding regular verification of the simulation time.

Solution:

To overcome this difficulty, we were able to effectively apply "read" and "select" on "STDIN." Currently, only the fifth child uses standard input to read data as needed. As a result, the problem is fixed since the fifth child monitors the simulation duration without interfering.

Output:



```
0.002: Child 1 Message 1
0.002: Child 2 Message 1
0.002: Child 3 Message 1
0.002: Child 4 Message 1
0.002: Child 3 Message 2
0.002: Child 4 Message 2
0.002: Child 3 Message 3
1.002: Child 1 Message 2
1.002: Child 3 Message 4
1.002: Child 3 Message 5
1.003: Child 4 Message 3
1.003: Child 4 Message 4
2.002: Child 2 Message 2
2.002: Child 2 Message 3
2.002: Child 2 Message 4
3.003: Child 1 Message 3
3.003: Child 1 Message 4
3.003: Child 1 Message 5
3.003: Child 2 Message 5
3.003: Child 3 Message 6
3.003: Child 2 Message 6
3.004: Child 4 Message 5
3.152: Child 5: Divyanth
4.004: Child 1 Message 6
4.004: Child 1 Message 7
4.004: Child 3 Message 7
4.004: Child 4 Message 6
5.004: Child 2 Message 7
5.005: Child 3 Message 8
5.005: Child 4 Message 7
5.005: Child 3 Message 9
5.005: Child 4 Message 8
6.005: Child 1 Message 8
6.005: Child 2 Message 8
7.006: Child 3 Message 10
7.006: Child 3 Message 11
7.006: Child 1 Message 9
7.007: Child 2 Message 9
7.007: Child 4 Message 9
7.007: Child 2 Message 10
7.007: Child 4 Message 10
8.007: Child 3 Message 12
8.007: Child 1 Message 10
```