# Advanced Operating Systems

## COEN 383 Project - 5

## Group 5

**Snehal Nikam**
**Sarthak Patel**
**Bharadwaj Parthasarathy**
**Raksha Narasegowda**
**Smit Patel**

### Objective :

The main purpose of this assignment is to practice making UNIX I/O system calls in C. In a multiplexed manner, your main process will read from multiple pipes, from the standard input (the terminal), and write to the file system.

To achieve the communication between parent and child processes, we've implemented a simulation using pipes. A pipe acts as a virtual file created in memory, equipped with both write and read file descriptors. Importantly, no data is written to the actual file system.

The simulation design is straightforward: in the main process, we establish five pipes and then spawn five distinct child processes. Each child process is assigned a specific pipe, leveraging it to send messages to the main process as dictated by project requirements. The 'main' process, in turn, reads from all these child processes and promptly writes the output to "Output.txt" upon receiving data from the corresponding pipes.

### Issues we have encountered :

1. **Parent process was unable to detect when the child process closes the dedicated pipe's write file descriptor.**

   **<u>Issue</u> :**

   To elaborate on this problem consider that every child process is required to terminate its pipe post-simulation using a "close" system call. The parent process can detect this by employing a combination of the "select" and "read" system calls. More precisely, when the child process closes its pipe, the "select" system call signals to the parent that there is data at the pipe's end. However, upon the parent reading this data, the number of bytes read is "0," signifying that the child process has closed the pipe.

   An EOF byte is added to a pipe only when the pipe doesn't have an open write file descriptor. If we fork a child after creating a pipe, both the main process and the child process possess the pipe's write file descriptor, violating the mentioned condition. Consequently, when the child closes the pipe, it doesn't signify the presence of an EOF character in the pipe to the parent.

**Resolution :**

Both the parent and the child need to close the unused file descriptor. For instance, the parent should close the write descriptor for the pipe, and the child should close the read descriptor for the pipe.

2. **All the five pipes created were shared between all the five children after forking.**

   Whenever the main process spawns a child process, the child process inherits its complete memory, including the stack.

   **Issue :**

   In our implementation, we have created five pipes and then allocated each pipe to an individual child. This introduces a challenge similar to the one discussed in the preceding section. However, in this case, pipes not explicitly assigned to a specific child are also accessible to that child, preventing the parent from identifying the closure of pipes through backpropagation.

   **Resolution :**

   Each child needs to close both the read and write file descriptors for the unused pipes to address this challenge.

3. **Terminating 5th child after the end of the simulation**

   The 5th child in this project has a distinct role—it needs to gather input from the terminal and then relay it back to the parent process. Additionally, the project requires terminating the simulation after 30 seconds by closing the dedicated pipe.

   **Issue :**

   Initially, we used "scanf" to read from terminal input(stdin). Since the "scanf" is a blocking call that relies on the user's response, this was causing the program to halt and wait for user input. This dependency became problematic when attempting to conclude the 5th child process at the end of the simulation. Since our 5th child was waiting for user input, it couldn't consistently check for the simulation time.

   **Resolution :**

   We resolved this issue by implementing "select" and "read" on "STDIN." Consequently, the 5th child only reads from stdin when there is data available, allowing it to continuously monitor the simulation time in a non-blocking manner.