**CSE344 MIDTERM REPORT**

**Ayşe Begüm NUR**

**1901042613**

1 - ) PARENT SERVER

The main role of the parent server is to manage command line arguments, manipulate the current working directory, and determine the number of child processes to be created based on these arguments. To receive "connect/tryconnect" requests from clients, a single server FIFO is used. The server reads these requests and handles them by either placing them in a request queue or forwarding them to available child processes, depending on the availability of child processes at that specific moment. Communication between the parent server and child servers is facilitated using pipes. To ensure that only one instance of the parent server is running at any given time, a semaphore is utilized. This semaphore enforces mutual exclusion, allowing only one instance of the parent server to execute critical sections of the code. By using a semaphore, conflicts and race conditions are avoided, ensuring the correct behavior of the parent server. In summary, the parent server is responsible for managing command line arguments, manipulating the working directory, creating child processes, handling client requests through the server FIFO, communicating with child servers using pipes, and enforcing mutual exclusion through the use of a semaphore. This design allows for efficient and controlled interaction between the parent server and its child processes, enabling effective handling of client requests in a concurrent and synchronized manner.

2 - ) CHILD SERVERS

Child processes establish communication with clients using FIFOs, specifically the client FIFO for writing responses and the child server FIFO for reading client requests. The child server initially receives the first request from a client and subsequently enters a loop to continuously wait for new requests. This loop continues until either the client chooses to disconnect or a SIGINT signal

(typically generated by pressing Ctrl+C) is received. To ensure the proper handling of concurrent requests and prevent race conditions, mutual exclusion is enforced for operations that involve modifying, reading, or listing file or directory contents. This is achieved through the use of a semaphore, which allows only one child process at a time to access critical sections of the code. The semaphore guarantees that conflicting operations do not occur simultaneously, ensuring data consistency and preventing data corruption. Additionally, a lock file is employed to prevent data corruption specifically while writing to the log file. The lock file serves as a synchronization mechanism, allowing only one process to access and modify the log file at any given time. This ensures that log entries are written correctly and avoids conflicts when multiple child processes attempt to write to the log file simultaneously. Also, the log file is protected against client requests for writing, reading or downloading.

Child processes notify the parent process of their availability to handle new clients by sending a signal, specifically SIGUSR1. This signal serves as a communication mechanism between child and parent processes, allowing the parent process to track the availability of child processes for handling incoming client requests effectively. In summary, child processes communicate with clients using FIFOs, enforce mutual exclusion using a semaphore, prevent data corruption while writing to the log file using a lock file, and signal their availability to the parent process using SIGUSR1. This design enables effective and synchronized handling of client requests while maintaining data integrity and consistency in a concurrent environment.

3 -) CLIENT PART

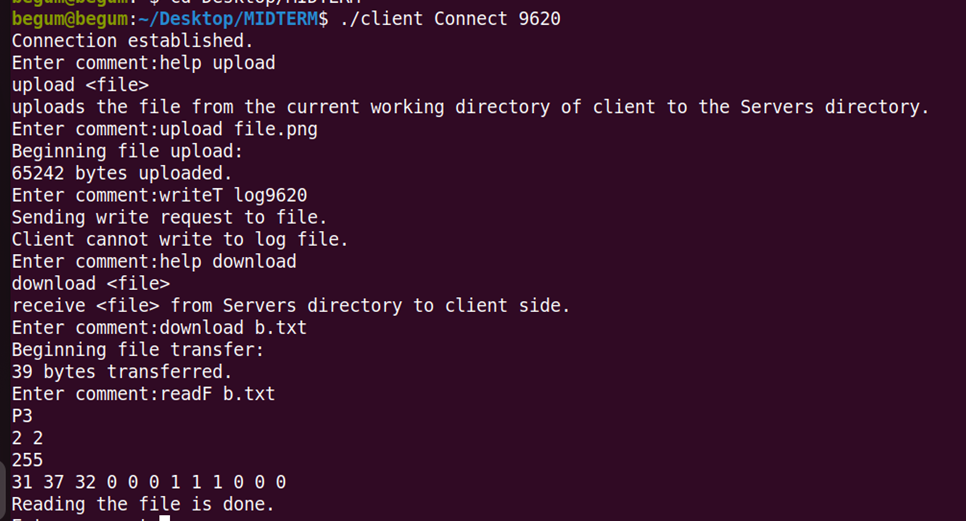
The client process establishes communication with the parent server by opening the parent server's FIFO for writing the connect/tryconnect request. Simultaneously, it opens its own client FIFO to receive the server's responses. The client process enters a loop where it repeatedly performs the following steps: Reads the server's response from the client FIFO. Prints the response to the screen for the user to see. Prompts the user for a new request from STDIN. Writes the user's request to the child server's FIFO. This loop continues until the client decides to quit the program. By following this approach, the client process can effectively communicate with the parent server, receive responses, display them, and send new requests for processing.

4 -) TEST CASES

Client side:

A screenshot of a computer

Description automatically generated



Server side:

A screenshot of a computer program

Description automatically generated with low confidence