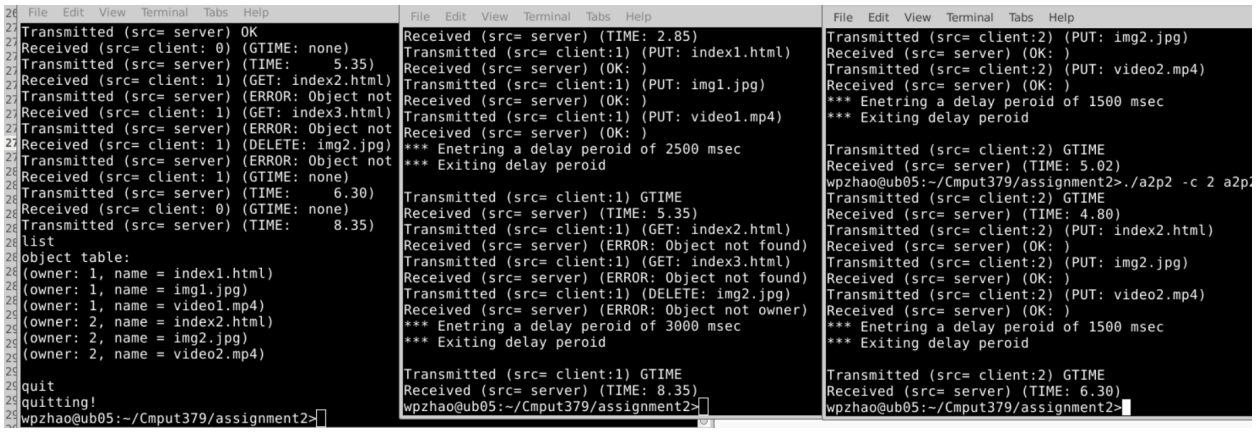
Cmput 379 Project Report

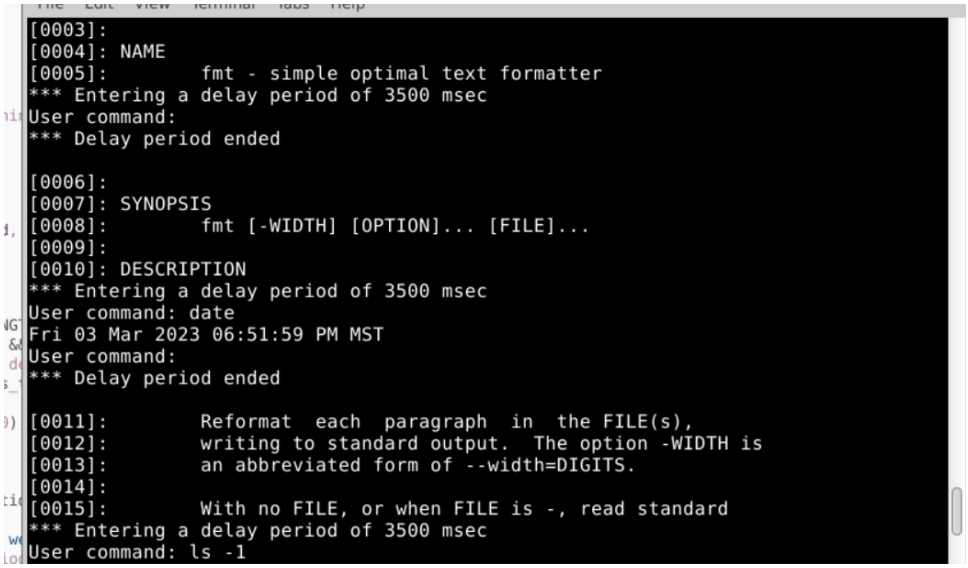
- **Objectives**: state the project objectives and value from your point of view (which may be different from the one mentioned above)
    - The objective of this project is to help us learn how to use threads and create a client-server program with FIFO files for communication. By implementing this project, we can gain a better understanding of how these concepts work and how they differ from each other.
- **Design Overview:** highlight in point-form the important features of your design
    - For part 1:
        - The program reads in a file and processes it line by line
        - The user can input commands while the program is processing the file
        - The program uses threads to run a process in the background while it is sleeping
        - The program handles errors such as a file not being found or a command failing to execute
        - The program exit when the user inputs the "quit" command
    - For part 2:
        - Global Variables: The program has three global integer arrays fifocs, fifosc, and fifo_names, that are used to store file descriptors for the fifos and their names. The FIFO names are represented as a 3x2 two-dimensional character array.
        - Enums: The program defines an enumerated data type PACKETTYPE that contains several packet types such as PUT, GET, DELETE, GTIME, TIME, OK, ERROR, and QUIT.
        - Structures: The program defines a structure named PACKET that contains three members: type, client_id, and message. The member type is of type PACKETTYPE, client_id is an integer, and message is an array of characters.
        - Functions: The program defines several functions such as print_packet(), send_packet(), rcv_packet(), and do_client(). The print_packet() function prints the contents of a PACKET structure, send_packet() sends a PACKET to a given file descriptor, rcv_packet() receives a PACKET from a given file descriptor, and do_client() executes the client side of the program.
        - In the do_client() it will process the file line by line and send a packet of the commands the client wants to do to the server if it matches the client id. The packet will be different types depending on the commands it wants to send.
        - The do_server() has two loops, one infinite loop until the user types in quit, and another loop that loops through all of the client to server FIFO files to pull and see if there is any that wants to PULLIN.

- **Project Status:** describe the status of your project (to what degree the programs work as specified, e.g., tested and working, working with some known issues, etc.) mention difficulties encountered in the implementation
    - Both parts of the assignment have been completed successfully. During testing in part 1, both the main function and execution of the user commands worked as intended. Creating the threads proved to be the most challenging aspect of the implementation for part 1, but after many trials and errors, I found that putting process commands in their own thread was a better design. The system can now call it to stop after the sleep thread is finished.
    - In part 2, testing showed that all the get, put, delete, delay, get time, list and quit commands worked as intended on the client's end. On the server's end, all of these commands also worked correctly, and the server was capable of handling up to 3 clients simultaneously.
    - The most challenging aspect of part 2 was figuring out how to implement the structure of the client-server communication process and how the FIFO (first in, first out) system would work. This required me to read textbooks to gain a better understanding of the structure. Another challenge was implementing the delete command that the client chooses, which required me to create a separate file for testing and completion.

- **Testing and Results:** comment on how you tested your implementation, and discuss the obtained results.
    - The testing was done on the lab machine with the sample data provided as well as mine own tests which consist of the same commands. There would be extra delete and add to see if there was an existing one or another client is trying to delete another client's data. Through the obtained results, we can see the program runs pretty fast, so in order to have better results I slowed down the time for pull by 10000 instead of 0 to show better of a times laps when running the results and to not put that much strain on the cpu for checking.



This is the result for part 2 as we can see the process is fifo compared to part 1's threads where there are 2 threads running. As seen below, the part 1 runs two threads as the same time



- **Acknowledgments:** acknowledge sources of assistance
    - Cmput 379 lab notes posted on eclass
    - Advanced Programming in the UNIX® Environment, Third Edition
    - Operating System Concepts, 10th Edition