CSE 344 System Programming

Midterm Project

Kahraman Arda KILIÇ

1901042648

1-Server-Client Communication

In this project, a multiple client-server structure was required. I implemented the communication between the client and server using the FIFO (named pipe) mechanism. There is a general server FIFO and a specific FIFO for each client. The client communicates with the server through its own FIFO. To establish a connection, the client sends a connection request to the server via the server FIFO. Other commands are sent through the client-specific FIFO.

The client first sends an access request to the server FIFO. If the client meets the necessary conditions (e.g., not exceeding the maximum client limit or being at the front of the queue), the server approves the client's connection and creates a child process using fork() on the server side. The child process then enters an infinite loop, waiting for requests from the client's FIFO. It processes the incoming requests accordingly and sends a response back to the client.

The FIFO mechanism used for communication between the client and server involves the use of structs (response and request).

2-Semaphores

To ensure mutual exclusion and avoid race conditions among clients, the use of mutual exclusion mechanisms was required. Therefore, I used semaphores in my project. There are a total of 3 named semaphores in my project, namely emptySlotForClients, totalConnectedClients, and connectionRequestSent.

- emptySlotForClients: This semaphore is used to control whether the maximum number of clients (maxClient) has been exceeded. Each time a client connects, the value of this semaphore is decreased using sem_wait(). When a client disconnects, the value is increased.
- totalConnectedClients: This semaphore assists in numbering the processes and ensuring uniqueness among them.
- connectionRequestSent: This semaphore is used to prevent connection requests from clients, who attempted to connect with a "Connect" request but were placed in the queue due to the lack of available slots, from interfering with each other when they send connection requests to the server to check if slots have become available.

Each semaphore is unlinked (sem_unlink()) on the server side first. Then, it is opened (sem_open()) with the desired values and set accordingly. Finally, after all the operations are completed, the semaphores are closed.

```
sem_t *emptySlotForClients;
sem_t *totalConnectedClients;
sem_t *connectionRequestSent;
```

3-File Locks

For the WriteT and Upload commands that involve writing to files on the server side, it was necessary to implement file locking to prevent concurrent access. Therefore, before writing to each file, a lock is applied to prevent other processes from accessing it. Once the writing operation is completed, the lock is released. Below are the functions I used to lock and unlock files:

```
int lock_file(FILE * filePointer) {
    int file = fileno(filePointer);
    struct flock lock;
    lock.l_type = F_WRLCK;
    lock.l_whence = SEEK_SET;
    lock.l_start = 0;
    lock.l_len = 0; // lock whole file
    lock.l_pid = getpid();

    fcntl(file, F_SETLKW, &lock);

    lock.l_type = F_RDLCK ;
    lock.l_whence = SEEK_SET;
    lock.l_start = 0;
    lock.l_len = 0; // lock whole file
    lock.l_pid = getpid();

    fcntl(file, F_SETLK, &lock);

    return 0;
}
```

```c
int unlock_file(FILE * filePointer) {
    int file = fileno(filePointer);
    struct flock lock;
    lock.l_type = F_UNLCK;
    lock.l_whence = SEEK_SET;
    lock.l_start = 0;
    lock.l_len = 0;  // unlock whole fil
    lock.l_pid = getpid();

    fcntl(file, F_SETLKW, &lock);
    fcntl(file, F_SETLK, &lock);

    return 0;
}
```

4- Connect / tryConnect

Each client sends a connection request to the server via the SERVER FIFO, and if the server accepts, it creates a child process for the client using fork().

Connect: This command ensures that if the maxClients value is exceeded, the client is queued. Firstly, the client sends a connection request to the server via the SERVER FIFO. If the server has reached the maxClients value, the client is added to the queue, and a response is sent to the client. The client examines this response to check if the connection is established or not. If the connection is not established, the client continuously sends connection requests to the server in parallel with the clients in the queue by incrementing and decrementing the connectionRequestSent semaphore. If a slot becomes available and the client PID that was first added to the queue matches the client PID that made the request, the client is dequeued and the connection is established. A response indicating the successful connection is sent to the client.

tryConnect: The connection request made in the Connect command is also made here. If the server has reached the maxClients value, the client receives a request indicating the failure of the connection. If the client is executed with the tryConnect command, it exits using exit().

5- readF / WriteT / Upload / Download

readF: First, the values in the command received from the client are parsed. Then, the necessary values in the request struct are set and sent to the server. Upon receiving the readF request, the server first sets the file path. Then, it opens the file for reading using open(). Afterwards, it reads the contents of the file. Since the size of the file can exceed the limit of transmission via FIFO in a single operation, it is sent in chunks. The buffer size is 4096 characters. The read contents are placed in the buffer and sent to the client. If there is nothing more to read, the readingFinished value is set to 1 and communicated to the client. Upon receiving this, the client stops further receives, and the server does not send any more responses, thus completing the operation.

writeT: Similar to readF, the command is prepared and a request is sent to the server. Then, on the server side, a temporary file is created. The contents of the original file, where the new text should be written, are copied into the temporary file. A count variable is used to track the line number. When the desired line is reached, the string sent by the client is written there. After writing the entire file to the temporary file, the old file is deleted, and the temporary file's name is replaced with the old file's name. During this process, the temporary file being written is locked.

download: The execution of this command involves reading the file requested for download on the server side and sending it to the client through buffers. The received buffers are then written to a file with the same name on the client side.

upload: The execution of this command is the reverse of the download operation. The content of the buffer sent by the client is read on the server side and written to a file with the same name.

6- KillServer

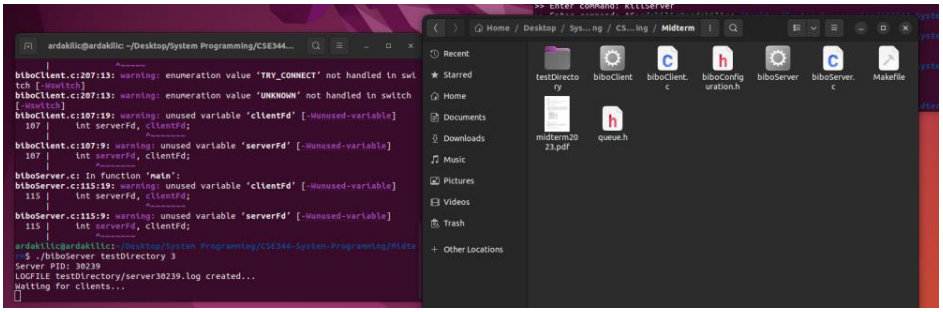
In the server, the SIGINT and SIGCHLD signals are handled. On the child side, the SIGINT and SIGTERM signals are handled. The server keeps track of the PIDs of each client added in an array called childPids. If a killServer command is received from a client, the server sends a SIGTERM signal to all the PIDs in the array. This causes all the processes to exit. Then, the server itself exits.

```c
if(req.type == KILLSERVER){
    printf("kill signal from client%02d\n",req.clientId);
    fprintf(logFilePointer,"kill signal from client%02d\n",req.clientId);
    for (int i = 0; i < currentIndexChildPids; i++) {
        if (kill(childPids[i], SIGTERM) == -1) {
            fprintf(logFilePointer,"Failed to send kill signal to PID %d\n", childPids[i]);
            printf("Failed to send kill signal to PID %d\n", childPids[i]);
        }
    }
    printf("bye...\n");
    fprintf(logFilePointer,"bye...");
    exit(EXIT_SUCCESS);
}
```

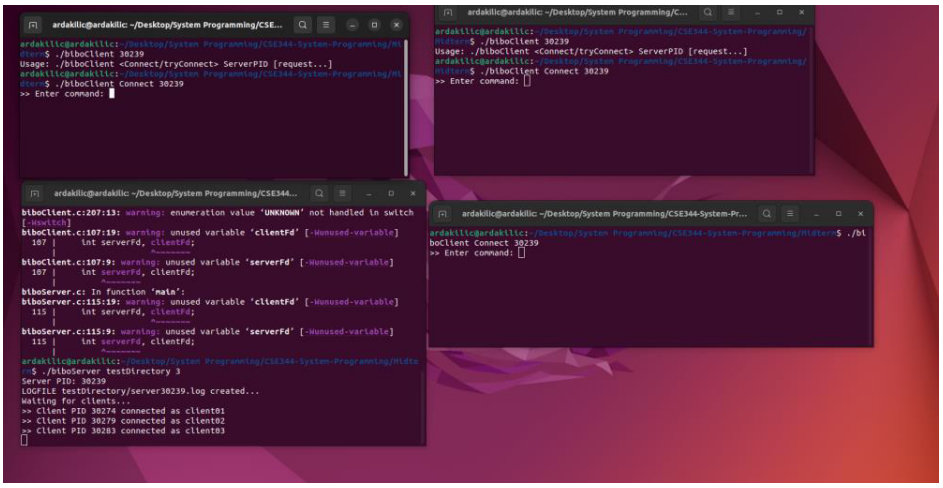
7- Tests

While testing this project, you need to have one server terminal. The number of clients is dependent on the user's request.

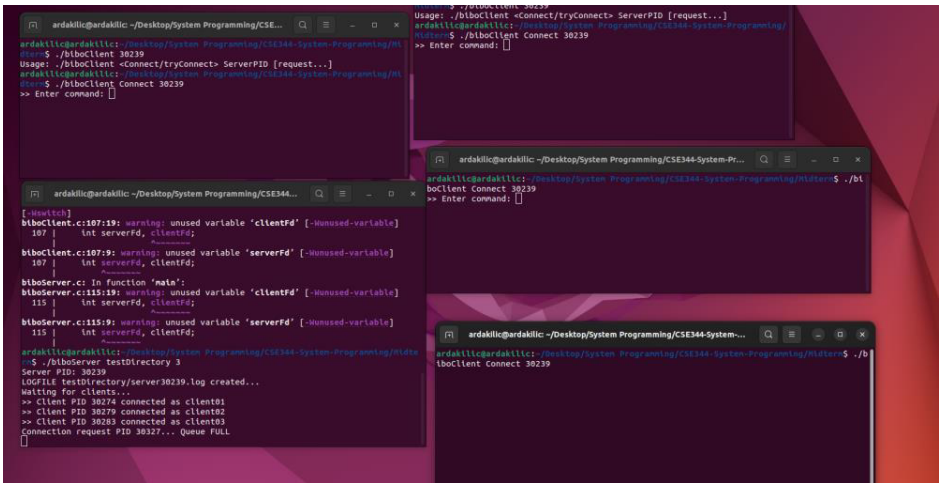
a. Running server

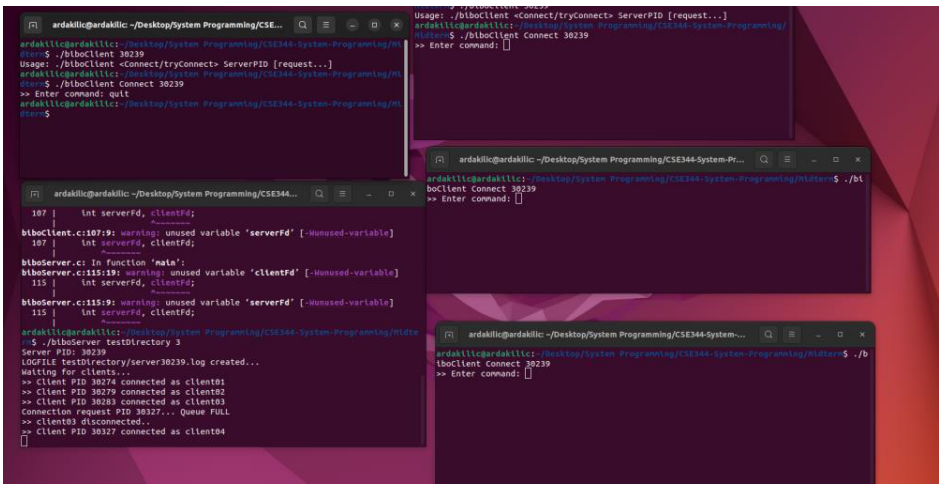


b. Connection from multiple clients

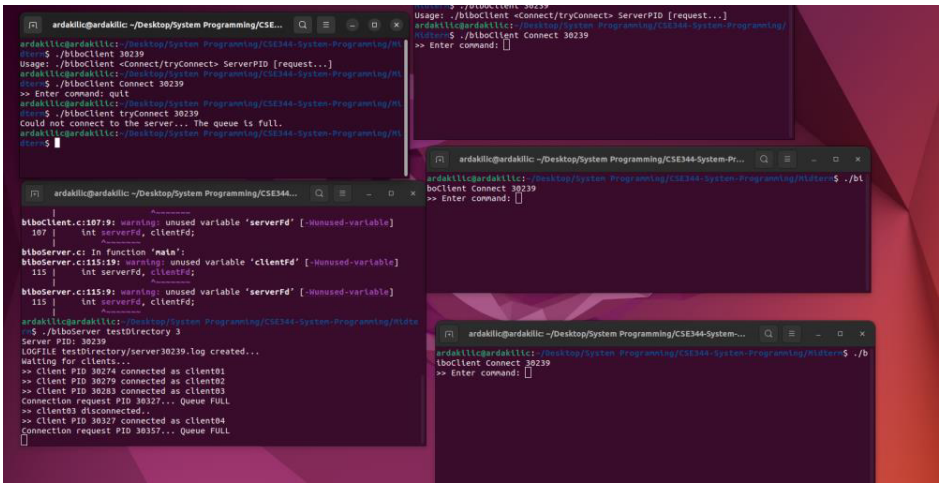


c. Queue handling in Connect command

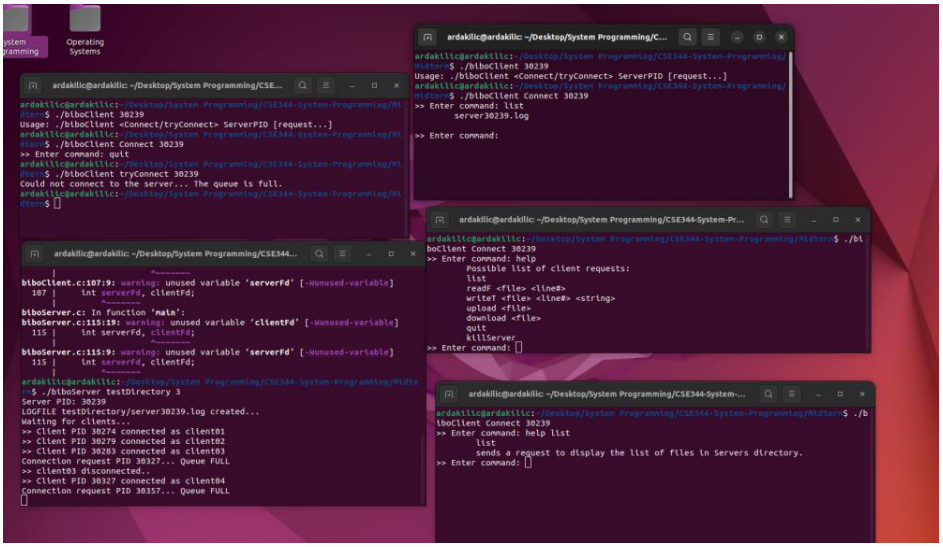




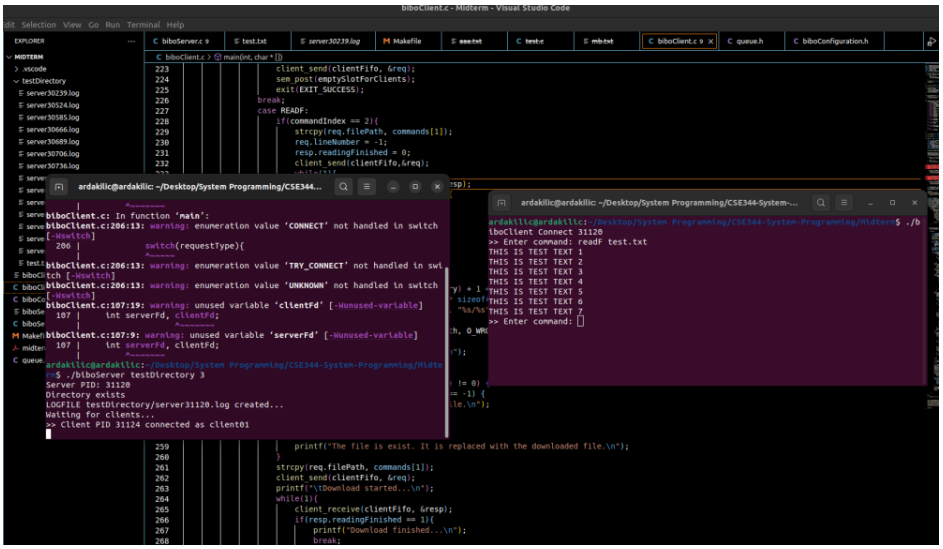
d. Queue handling in tryConnect command



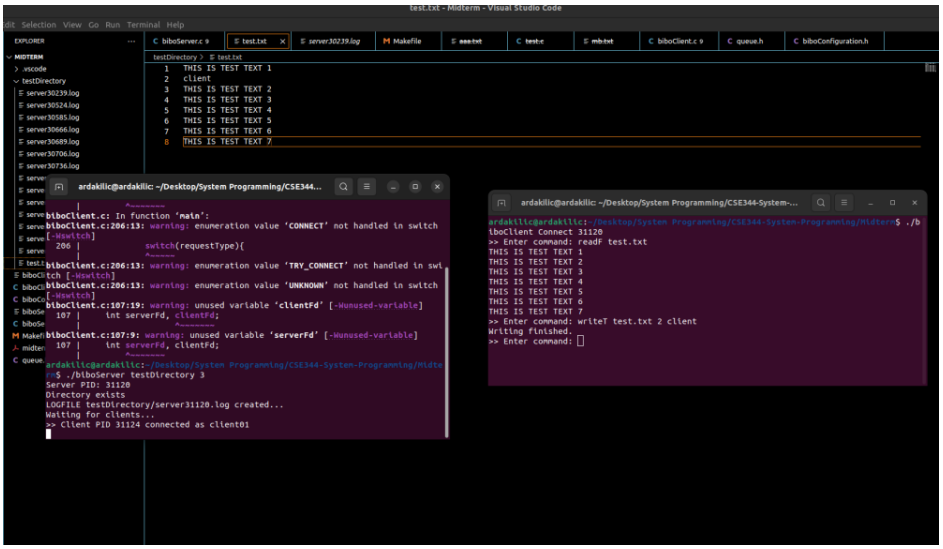
e. Help and list commands

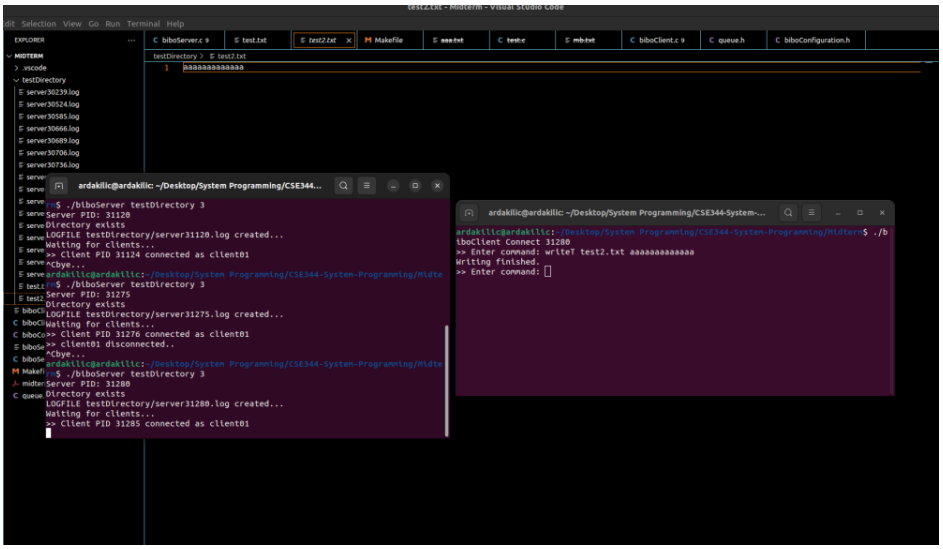


f. ReadF command

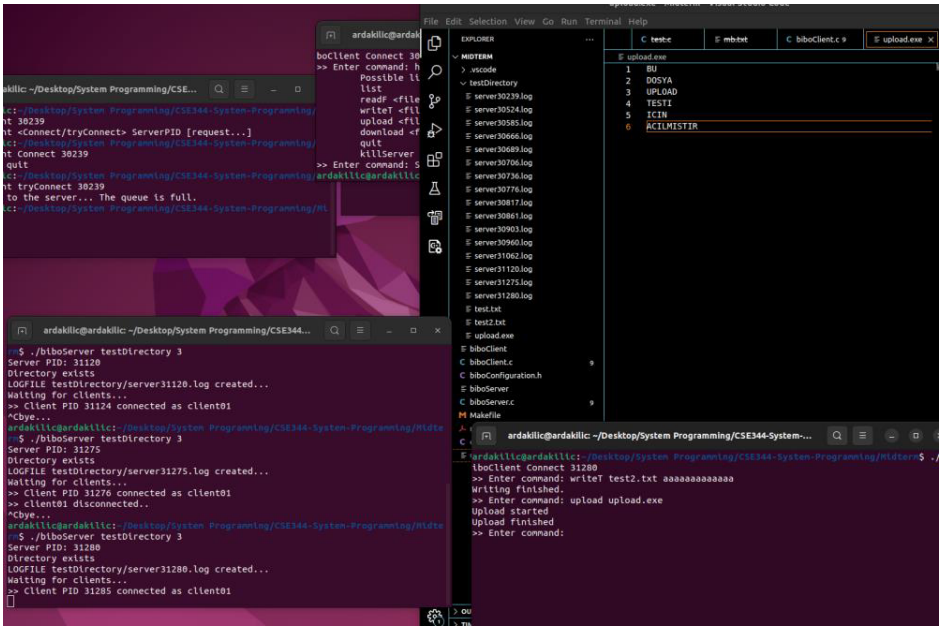


g. WriteT command

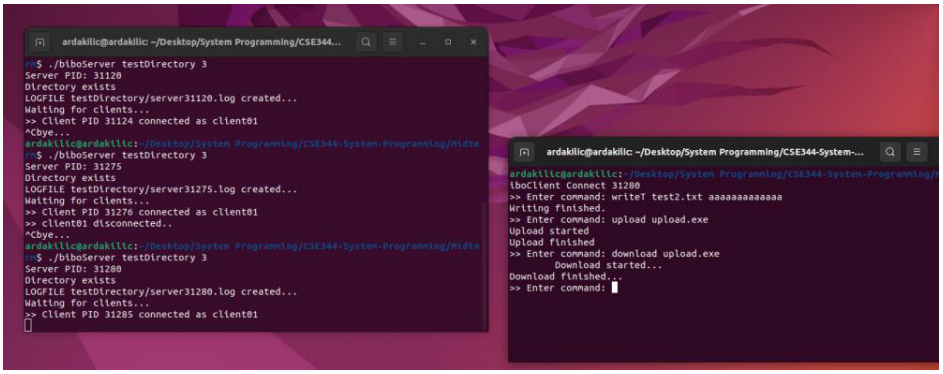




h. Upload



I. Download



i. KillServer