CSE 344
SYSTEM PROGRAMMING

HW4
REPORT

OGUZ MUTLU
1801042624

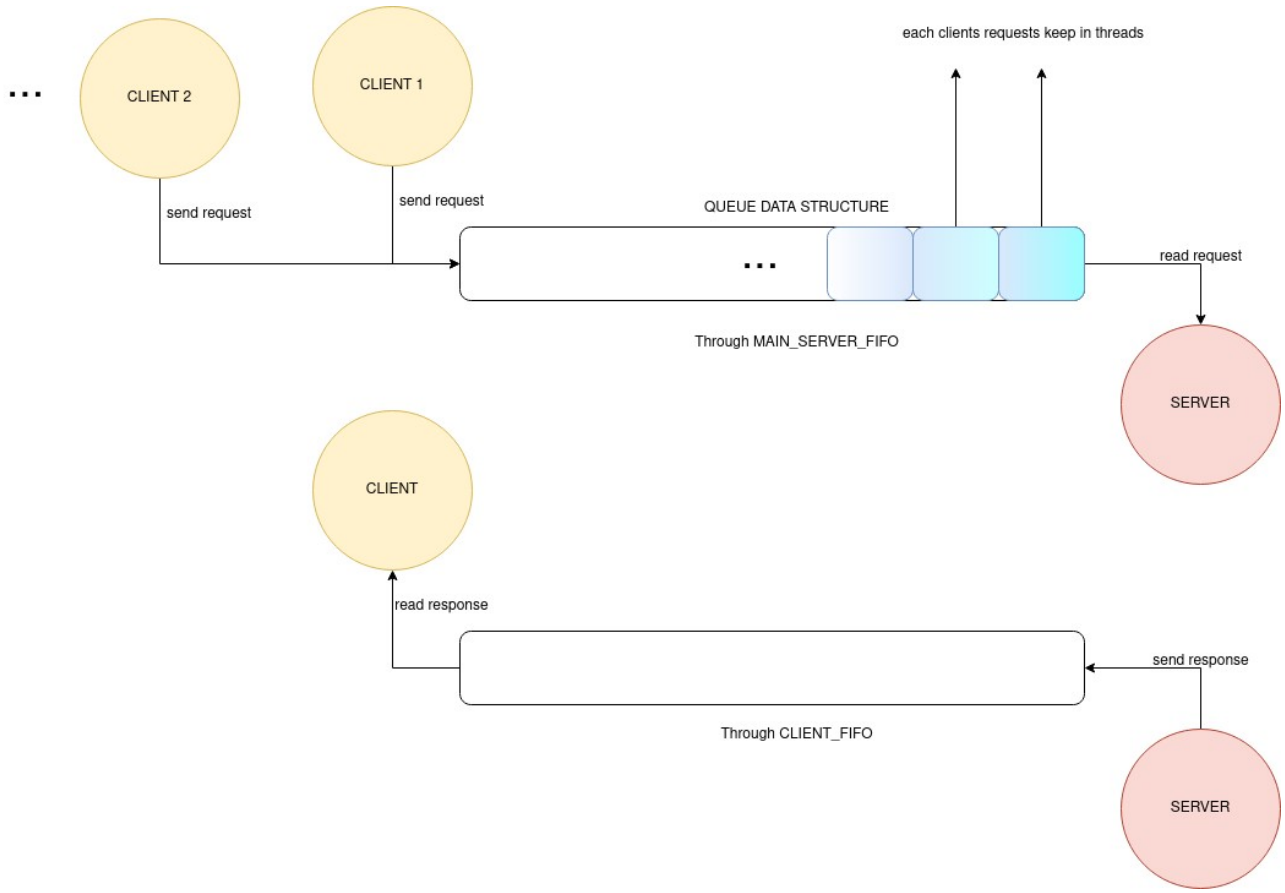
1. PROBLEM DEFINITION

Design and implement a file server that enables multiple clients to connect, with a threaded server-client model,
access and modify the contents of files in a specific directory
- in server side = biboServer <dirname> <max.#ofClients>
enters specified directory and create a log file for the clients and prompt it its PID,
- in client side the client program with Connect option request a spot from the Server Que with ServerPID and connects if a spot is available (if not the client should wait until a spot becomes available,

help, list, readF, writeT, upload, download, quit, killServer implement → implement commands.

• Program should be able to handle large files (i.e. > 10 MB).
•Program should be able to handle different file formats (i.e. text, binary).
• Program should use multiple processes for file access and synchronization.
• Signals should be handled properly on both client and Server sides

2. PROBLEM SOLUTION APPROACH



I did not implement homework,Since I couldn't complete the midterm exam, I had time to complete the homework and/or design it, but the system I designed would be like this;

Server Side Functionality:

1. The server starts by creating a thread pool of a fixed size specified as an argument.
2. It enters the specified directory and creates a log file for client communication.
3. The server waits for clients to connect by on a server FIFO
4. When a client connects, the server adds the client's PID and communication FIFO to a queue.
5. The server dequeues a client from the queue and assigns a thread from the thread pool to handle the client's requests.

6. The thread reads the client's requests from the server FIFO, and send request through the communication FIFO, performs the requested operations, and sends the response back to the client.
7. After completing a request, thread handle another client request.

Client Side Functionality:

1. The client program is started with the "connect" command and the server's PID as arguments.
2. The client creates its own communication FIFO using its PID.
3. The client sends a connection request to server FIFO.
4. The client waits for a response from the server indicating successful connection or waiting for a spot in the queue.
5. Multiple clients can run concurrently, each with its own communication FIFO and separate process IDs.

In all these approaches I want to use producer-consumer approach.

There are several similarities between producer-consumer and threaded server-client model

- Thread pool = In the server code, a fixed number of threads are created at the start to handle client requests, while in the producer-consumer pattern, a pool of threads is responsible for producing and consuming data.

- Queue = Both implementations use queue to manage the request and responses. In the server code, the queue stores the client requests, and the threads pick up requests from the queue to process.

-Synchronization: Both implementations require synchronization mechanisms to ensure thread safety and prevent race conditions. In the server code, mutexes and condition variables are used to synchronize access to the queue and coordinate thread execution.

Also I want to do producer consumer synchronization solution on my homework

```c
void getRequest(int client_id, const char* request, int max_client, struct request* req) {
    pthread_mutex_lock(&mutex);

    while (clientNumber >= max_client) {
        pthread_cond_wait(&cond, &mutex);
    }

    // read the client request to the queue
    char client_fifo[20];
    snprintf(client_fifo, CLIENT_FIFO_TEMP_LEN, CLIENT_FIFO_TEMP, client_id);
    int client_fd = open(client_fifo, O_RDONLY);
    read(client_fd, req, sizeof(struct request));
    close(client_fd);

    clientNumber--;

    pthread_mutex_unlock(&mutex);
}
```

I want to make producer-consumer synchronization solution on my getResponse and sendRequest function, by using mutexes and conditional variables as in the above code.

```c
void* consumeClient(void* arg)
{
    pthread_t t1;
    int client_fd;
    struct request* req = arg;
    char client_fifo[CLIENT_FIFO_TEMP_LEN];
    snprintf(client_fifo, CLIENT_FIFO_TEMP_LEN,CLIENT_FIFO_TEMP, (long) req->pid);
    client_fd = open(client_fifo, O_RDONLY);

    while (read(client_fd, req, sizeof(struct request)) > 0) {
        printf("Received request %s\n", req->data);
    }
    close(client_fd);
    unlink(client_fifo);
    pthread_exit(NULL);

}
```

Consumer thread, which takes and reads request from the communication FIFO, and exits the thread when read operation done.

3. NOTES

I aimed to do as I described in the report, but I could not reach it.The code I sent is not a working code, so you can only think of it as a report sent, best regards