CSE344

HW#4

Report

1901042676

Mustafa Mercan

Assignment:

This assignment bears a striking resemblance to the midterm assignment. As a result, my file structures and the operations performed show similarities with minor differences. However, unlike the midterm assignment, instead of using a child process for each client, we are required to create a thread structure for each incoming request from the client, up to the specified pool size. Afterwards, we need to process these requests here and generate responses.

Server Side:

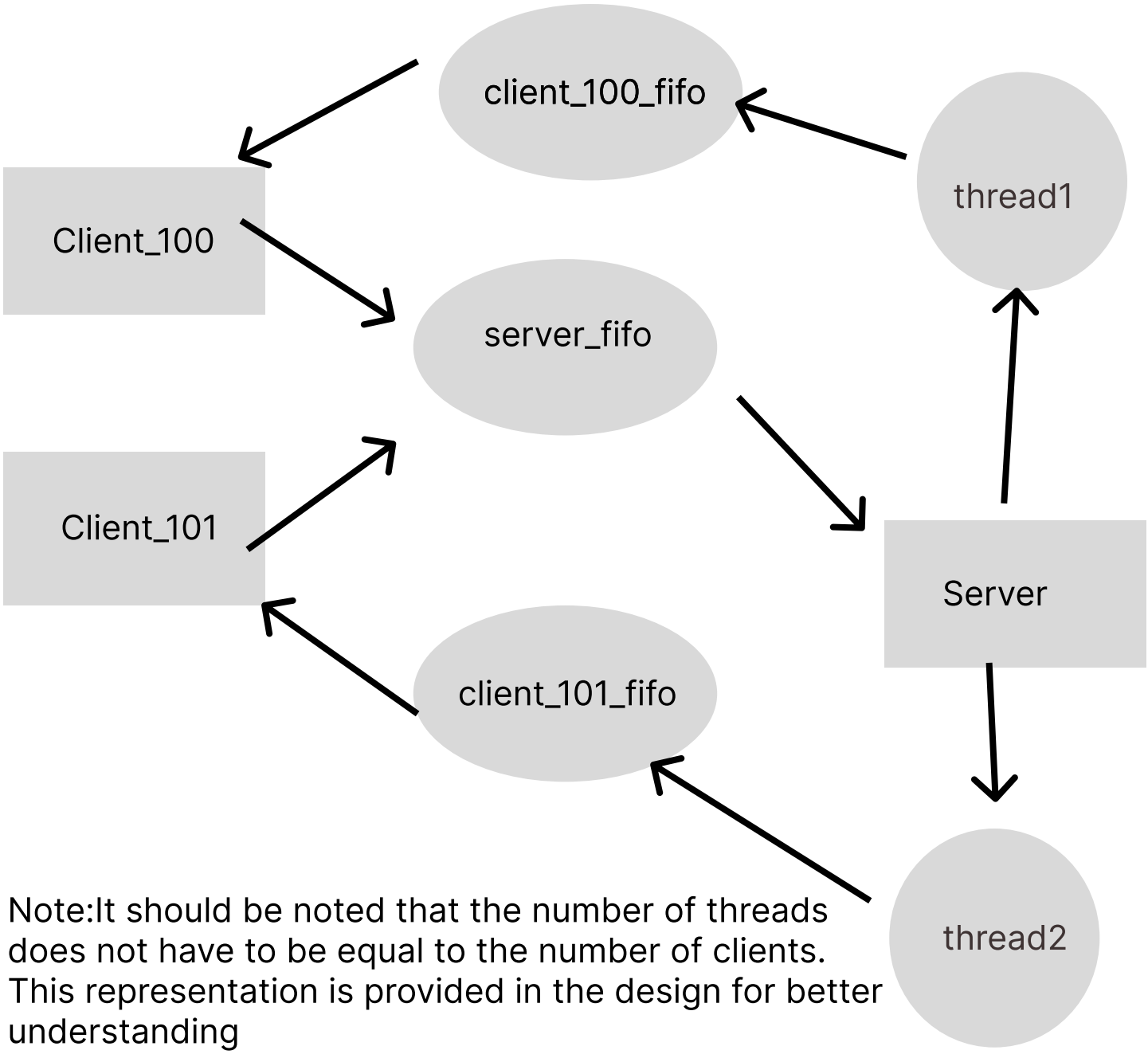
The server part should take the poolsize as a parameter, which is different from the previous assignment. This poolsize determines the number of threads to be created. Other than that, all other parts remain the same.

Client Side:

The operations to be performed on the client side are the same as in the previous project.

Approach to the Problem:

Since I was familiar with the subject from the previous assignment, I didn't have difficulty designing what was necessary for this assignment. My file structure remained unchanged. I only modified the parameters and variables used within the functions. Prior to these changes, I created the necessary design for the project as shown below.



Note:It should be noted that the number of threads does not have to be equal to the number of clients. This representation is provided in the design for better understanding

By using a thread structure, we can handle a large number of clients connecting to the server while consuming fewer resources and generating responses for more clients. In the previous assignment, creating a child process for each client required much more resources.

After creating the design in this way, I developed a new system for the communication between the client and the server, as it was different from my previous assignment. Then, I made changes to the other functions accordingly to fit this structure, and thus completed a significant part of my project.

After this stage, I performed the necessary signal processing as follows:

Server Side:

```
signal(SIGUSR1,decrease_client);
signal(SIGUSR2,quit_signal);
signal(SIGINT, quit_signal);
```

If a client terminates the process by using Ctrl + C instead of the 'quit' command, the SIGUSR1 signal allows the pending client in the queue to connect to the server.

If the 'killServer' parameter is entered by the client, the SIGUSR2 signal allows terminating the server and ending the operations of connected clients

Finally, the SIGINT signal performs the same operation when the server generates the Ctrl + C signal.

Client Side:

```
signal(SIGUSR1,queue_connect);
signal(SIGINT, sigint_signal);
```

With the SIGUSR1 signal, I handle the 'connect/tryConnect' status based on the condition received from the server. I either keep the client waiting or provide the appropriate output to allow it to continue processing.

With the SIGINT signal, I capture the Ctrl + C condition and notify the server. This allows the clients in the queue to connect to the server.

```
typedef struct{
    int target_pid; // server pid
    char path[BUFFER_SIZE]; // client fifo path
    int pid; // client pid
    char request[BUFFER_SIZE]; // requst command
    bool connection;// connect or not connect
    char response[RESPONSE_SIZE]; // reponse buff
    char folder_path[BUFFER_SIZE]; // execute folder path
    char connect_type[BUFFER_SIZE]; // connect type

}client_request;
```

Finally, using fifos and the above-mentioned struct structure, I establish communication between clients and the server.

With the following structure, I handle the requests coming to the threads according to their respective states correctly.

And every time a client connects or disconnects, I record the logs for that server in the 'logs' folder. Finally, I unlink the created fifos, completing my project.

```c
void execute_task(client_request client)
{

    char **commands = str_split(client.request,' ');

    if(strcmp(client.request,"help") == 0)
    {
        char availableComments[] = "Available comments are :help, list, readF, writeT, upload, download, quit, killServer\n";
        strcpy(client.response,availableComments);
        lock_and_write(client.path,client);
    }
    else if(strcmp(client.request,"list") == 0)
    {
        list_command_handle(client);
    }
    else if(strcmp(commands[0],"readF") == 0)
    {
        readF_handle(client);
    }
    else if(strcmp(commands[0],"writeT") == 0)
    {
        writeT_handle(client);
    }
    else if(strcmp(commands[0],"upload") == 0)
    {
        upload_handle(client);
    }
    else if(strcmp(commands[0],"download") == 0)
    {
        download_handle(client);
    }
    else if(strcmp(commands[0],"quit") == 0)
    {
        log_info(client,"disconnect");

        if(queue_size != 0)
        {
            kill(client_queue_wait[0].pid,SIGUSR1);
            int i = 0;
            for(i = 0; i < queue_size - 1; i++)
            {
                client_queue_wait[i] = client_queue_wait[i+1];
            }
            queue_size--;
        }
        current_client--;
        printf("client %d is disconnected server\n",client.pid);
    }
```