**Faculty of Computing and Technology**

**University of Kelaniya**

**B.Sc. (Honours) in Computer Science**

**CSCI 21023- Data Communication and Networking**

**Assignment 02**

**File Transfer System (Client to Server)**

**Team Members: -**

Punsarani W.S. CS/2020/026

Gunathilaka G.H.N. CS/2020/021

**Introduction**

In the realm of networked systems, efficient and scalable communication between devices is paramount. This project focuses on implementing a Concurrent File Transfer System, where a client can request files from a server, and the server can seamlessly handle multiple requests concurrently. The communication is facilitated through sockets, allowing for a robust and versatile solution. We are using C programming language to implement this communication. There are some key features of this program.

1. **Concurrency:** The server is designed to handle multiple client requests concurrently, ensuring that the system remains responsive and scalable.
2. **File Transfer:** Clients can request files from the server, and the server responds by providing the requested file. This establishes a reliable mechanism for file sharing between devices on a network.
3. **Socket Programming:** The communication between the client and server is achieved through socket programming. Sockets provide a standardized and efficient way for devices to communicate over a network.
4. **Dynamic IP Address and Port Configuration:** The client program takes the IP address of the server and the port number as inputs, providing flexibility in configuring the communication parameters.
5. **Request-Response Protocol:** A well-defined protocol governs the communication between the client and server. Requests for files are made by clients, and the server responds with the requested files or appropriate messages.

**Program Codes**

**Server-Side Code: -**

#include <stdio.h>

#include <stdlib.h>

#include <winsock2.h>

#pragma comment(lib, "ws2\_32.lib")

#define PORT 13920

#define BUFFER\_SIZE 1024

void send\_file(SOCKET client\_socket, const char \*filename) {

    FILE \*file = fopen(filename, "r");

    if (!file) {

        printf("File not found: %s\n", filename);

        send(client\_socket, "File not found", sizeof("File not found"), 0);

        return;

    }

    char buffer[BUFFER\_SIZE];

    int read\_bytes;

    while ((read\_bytes = fread(buffer, 1, sizeof(buffer), file)) > 0) {

        send(client\_socket, buffer, read\_bytes, 0);

    }

    fclose(file);

}

void handle\_client(SOCKET client\_socket) {

    char filename[BUFFER\_SIZE];

    int received\_bytes;

    // Receive the filename from the client

    if ((received\_bytes = recv(client\_socket, filename, sizeof(filename), 0)) <= 0) {

        printf("Client disconnected\n");

        closesocket(client\_socket);

        return;

    }

    // Null-terminate the received filename

    filename[received\_bytes] = '\0';

    // Send the file content to the client

    send\_file(client\_socket, filename);

    // Close the client socket when done

    closesocket(client\_socket);

}

int main() {

    WSADATA wsa;

    SOCKET server\_socket, client\_socket;

    struct sockaddr\_in server\_address, client\_address;

    int client\_addr\_len = sizeof(client\_address);

    HANDLE thread\_handle;

    // Initialize Winsock

    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {

        printf("Failed to initialize Winsock\n");

        return 1;

    }

    // Create socket

    if ((server\_socket = socket(AF\_INET, SOCK\_STREAM, 0)) == INVALID\_SOCKET) {

        printf("Socket creation failed\n");

        return 1;

    }

    // Initialize server address struct

    server\_address.sin\_family = AF\_INET;

    server\_address.sin\_addr.s\_addr = INADDR\_ANY;

    server\_address.sin\_port = htons(PORT);

    // Bind the socket

    if (bind(server\_socket, (struct sockaddr\*)&server\_address, sizeof(server\_address)) == SOCKET\_ERROR) {

        printf("Binding failed\n");

        return 1;

    }

    // Listen for incoming connections

    if (listen(server\_socket, SOMAXCONN) == SOCKET\_ERROR) {

        printf("Listening failed\n");

        return 1;

    }

    printf("Server listening on port %d\n", PORT);

    while (1) {

        // Accept a connection from a client

        if ((client\_socket = accept(server\_socket, (struct sockaddr\*)&client\_address, &client\_addr\_len)) == INVALID\_SOCKET) {

            printf("Acceptance failed\n");

            continue;

        }

        printf("Accepted connection from %s:%d\n", inet\_ntoa(client\_address.sin\_addr), ntohs(client\_address.sin\_port));

        // Create a new thread to handle the client

        thread\_handle = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)handle\_client, (LPVOID)client\_socket, 0, NULL);

        CloseHandle(thread\_handle);  // Close the thread handle to avoid resource leak

    }

    // Close the server socket

    closesocket(server\_socket);

    // Cleanup Winsock

    WSACleanup();

    return 0;

}

**Client-Side Code: -**

#include <stdio.h>

#include <winsock2.h>

#pragma comment(lib, "ws2\_32.lib")

#define PORT 13920

#define BUFFER\_SIZE 1024

void receive\_file(SOCKET server\_socket, const char \*filename) {

    FILE \*file = fopen(filename, "w");

    if (!file) {

        printf("Error creating file: %s\n", filename);

        return;

    }

    char buffer[BUFFER\_SIZE];

    int received\_bytes;

    while ((received\_bytes = recv(server\_socket, buffer, sizeof(buffer), 0)) > 0) {

        fwrite(buffer, 1, received\_bytes, file);

    }

    fclose(file);

}

int main() {

    WSADATA wsa;

    SOCKET client\_socket;

    struct sockaddr\_in server\_address;

    char filename[BUFFER\_SIZE];

    char buffer[BUFFER\_SIZE];

    // Initialize Winsock

    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {

        printf("Failed to initialize Winsock\n");

        return 1;

    }

    // Create socket

    if ((client\_socket = socket(AF\_INET, SOCK\_STREAM, 0)) == INVALID\_SOCKET) {

        printf("Socket creation failed\n");

        return 1;

    }

    // Initialize server address struct

    server\_address.sin\_family = AF\_INET;

    server\_address.sin\_addr.s\_addr = inet\_addr("192.168.8.139");  // Change this to the server's IP address

    server\_address.sin\_port = htons(PORT);

    // Connect to the server

    if (connect(client\_socket, (struct sockaddr\*)&server\_address, sizeof(server\_address)) == SOCKET\_ERROR) {

        printf("Connection failed\n");

        return 1;

    }

    printf("Connected to server\n");

    // Get the filename from the user

    printf("Enter the filename to request from the server: ");

    fgets(filename, sizeof(filename), stdin);

    filename[strcspn(filename, "\n")] = '\0';  // Remove the newline character

    // Send the filename to the server

    send(client\_socket, filename, strlen(filename), 0);

    // Receive the file content from the server

    receive\_file(client\_socket, filename);

    // Display the content of the received file

    FILE \*file = fopen(filename, "r");

    if (!file) {

        printf("Error opening file: %s\n", filename);

        closesocket(client\_socket);

        WSACleanup();

        return 1;

    }

    printf("Content of %s:\n", filename);

    while (fgets(buffer, sizeof(buffer), file) != NULL) {

        printf("%s", buffer);

    }

    fclose(file);

    // Close the client socket

    closesocket(client\_socket);

    // Cleanup Winsock

    WSACleanup();

    return 0;

}

**Sample Text File: -**

Sample.txt

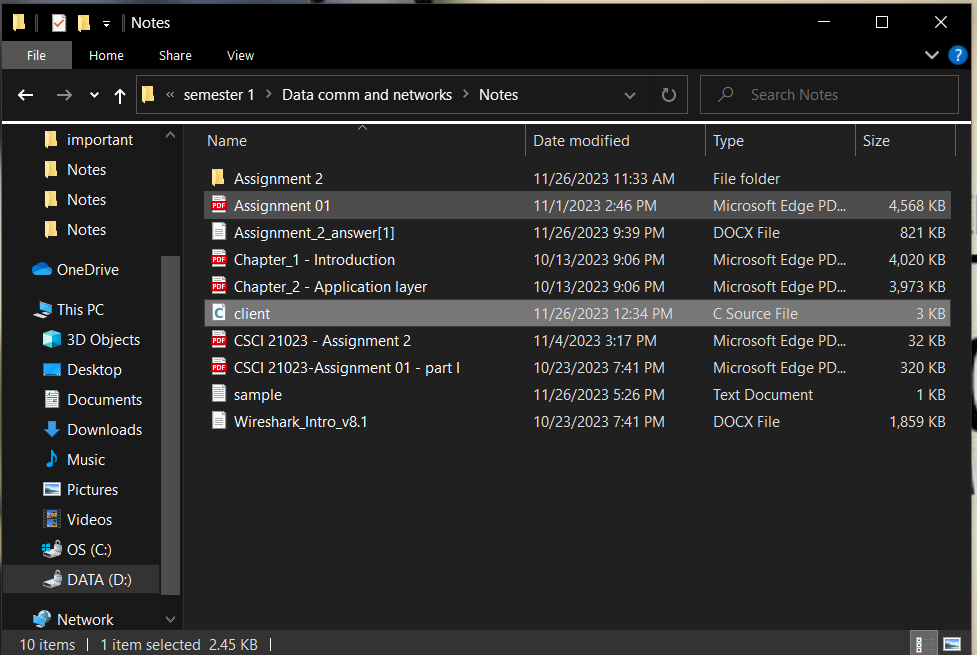
“Underneath the moonlit sky, a gentle breeze whispered through the trees. Stars adorned the night canvas, each telling a tale of distant galaxies. In this tranquil moment, the world felt like a story waiting to be written, and every heartbeat resonated with the rhythm of the universe.”

**Instructions to run the program**

1.Before running the program you must do some changes to the code.

1. Replace the “192.168.8.139” in the client code line 47, with the actual IP address of the server which it runs.
2. Change the port numbers of both codes into a port number that is not already in use by other applications and is within the range of valid port numbers (0-65535).
3. Save the server.c file and the sample.txt text file in the same folder of one computer and client.c file in another computer.

Client computer:-

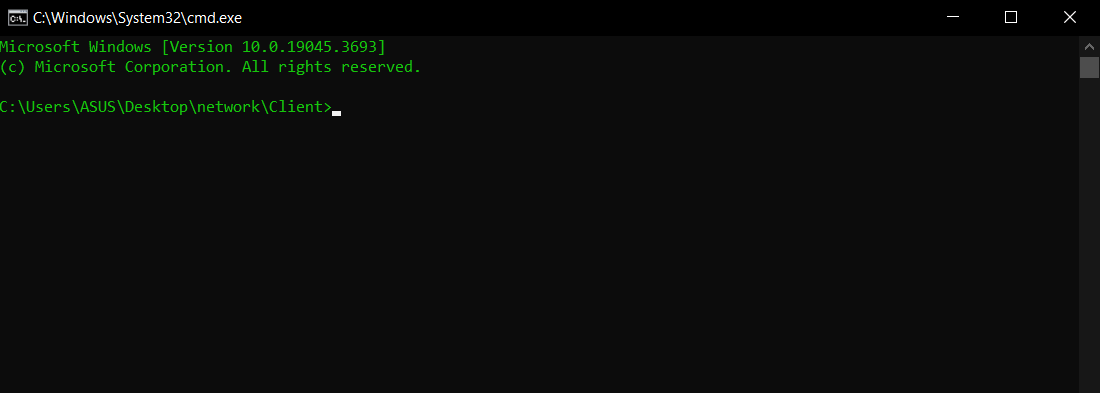
****

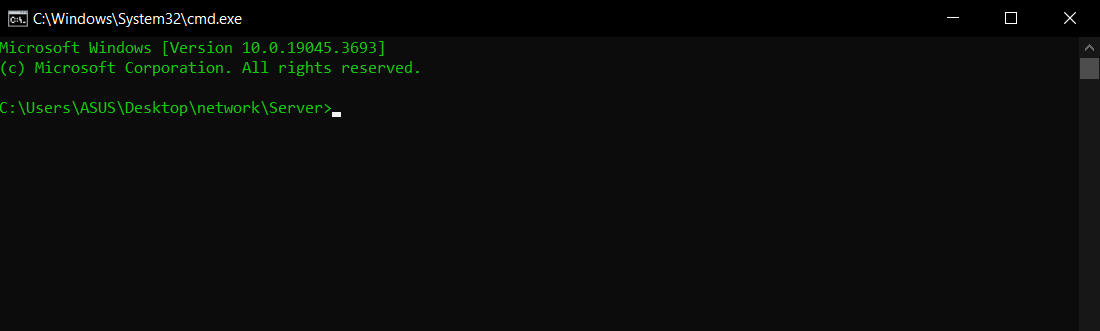
Server computer: -

A screenshot of a computer

Description automatically generated

2.Open a command prompt on each computer in the directory where the server code and client code are saved.

****

****

3.Compile the codes

Server Computer:-

gcc server.c -o server -lws2\_32

A screenshot of a computer

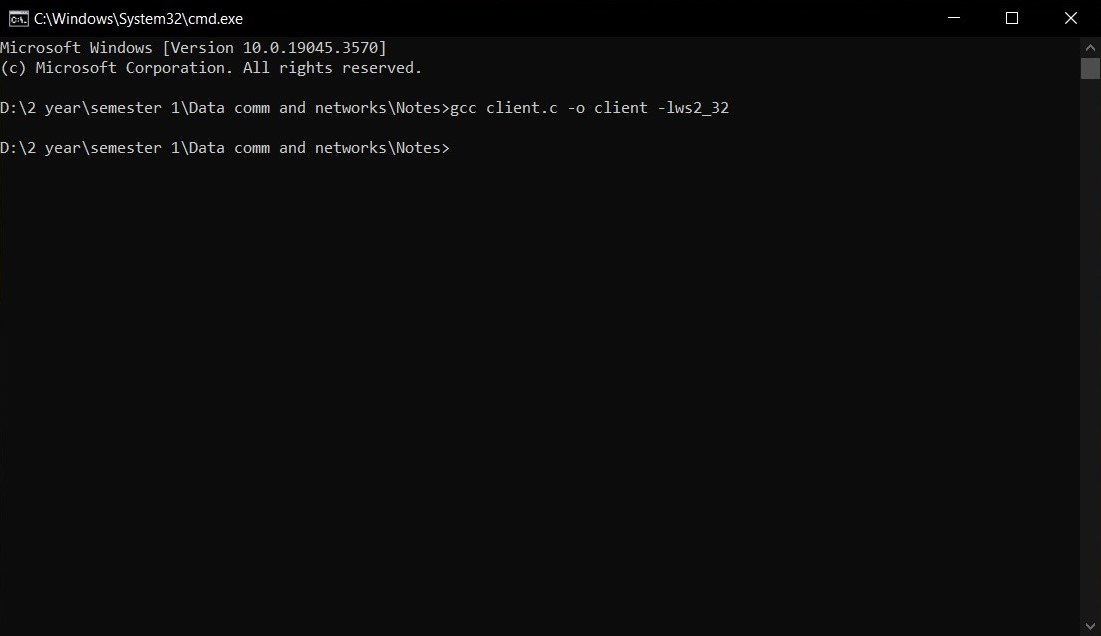
Description automatically generated

A screenshot of a computer

Description automatically generated

Client Computer: -

gcc client.c -o client -lws2\_32



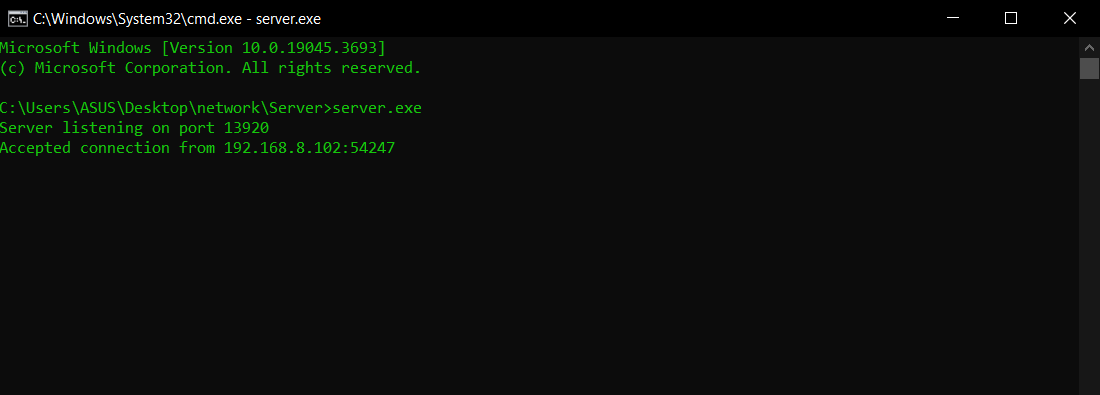
**A screenshot of a computer

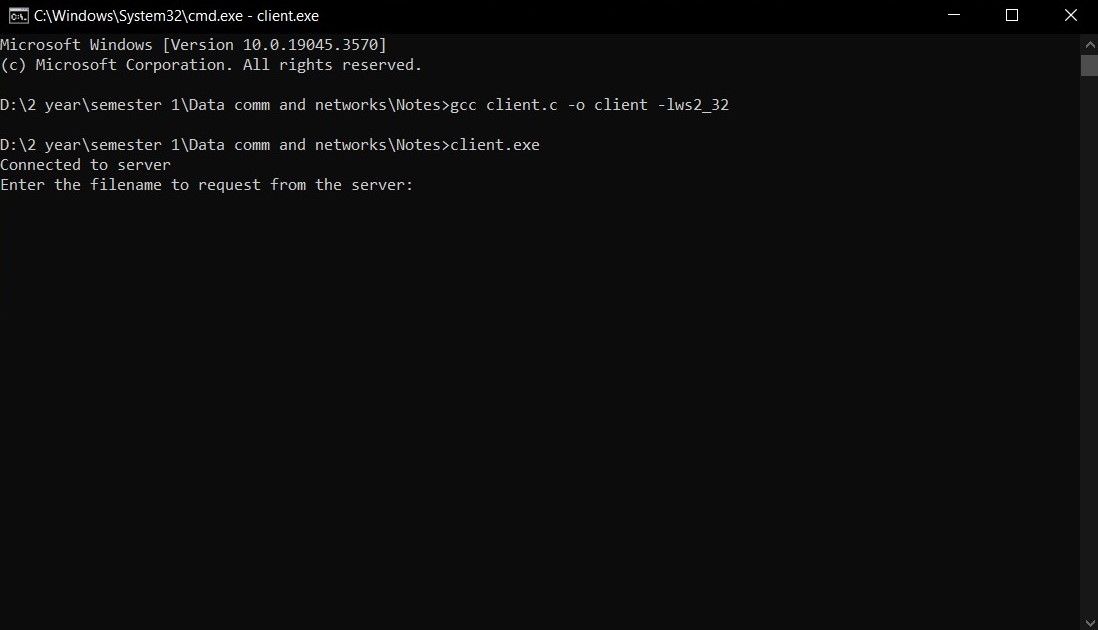
Description automatically generated**

4.Run the codes

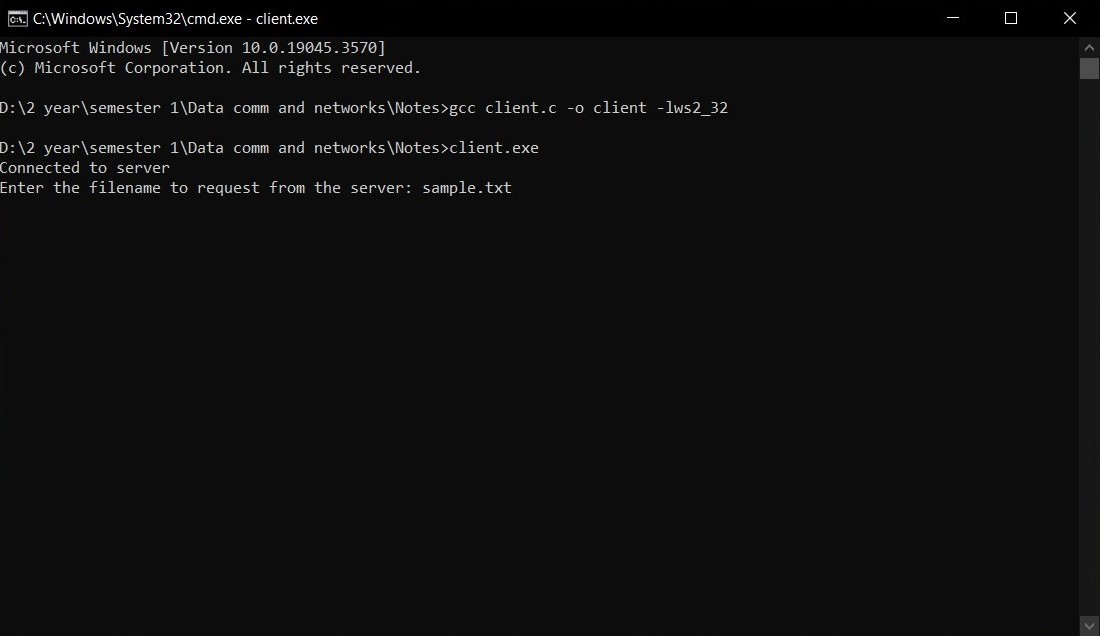
Server :- server.exe

Client :- client.exe





5.Enter the file name client want to get from server (In this case enter sample.txt).



**Output of the program**

