PROJECT REPORT

ORANGE PROJECT

DONE

Janus

Author(s):

Famura

Contents

1. Project Description

The Janus project is designed as a flexible and extensible file transfer system implemented in C++. It enables reliable data transmission between devices using different communication modes, specifically supporting both parallel and linear (sequential) transfer strategies. The system is structured to allow users to select their preferred transfer mode and connection direction (send or receive) at runtime, providing a user-friendly and interactive experience.

It made for being infrastructure to other projects. Thus, it lacks crucial features like frontend. The project did not developed to be used raw but still can be used in the current state. The project uses Client – Server connection type. Client sends the data & Server receives it.

2. Project Technologies

2.1. Module(s)

- **Windows (Windows 10):**
  The entire project is designed to run in Windows systems. Did not tested in lower Windows versions like Windows 7. But tested in Windows 11 and running flawlessly.

2.2. Coding Language(s)

- **C++ (C++14 Standard) – Visual Studio 2022:**
  The entire project is written in C++ and is set to compile with the C++14 standard. This provides modern language features and compatibility with Visual Studio 2022.

2.3. Libraries & External Additions

- **Winsock2.h:**
  Provides the Windows Sockets API for network communication (TCP/IP).
- **targetver.h & stdafx.h:**
  Provides required coding for socket programming.
- **thread.h:**
  Using “Posix threads” for the parallel side of the communication.
- **shlobj.h:**
  Used for accessing special Windows folders (e.g., Downloads directory via SHGetKnownFolderPath).

3. Project Architecture

As stated before, the project divides into two parts: Parallel part and Linear part. All of these are controlled by the main file.

3.1. Linear

The linear part works in two different modes that TCP connection consists of client and server.

3.1.1. Client

1. Initialize Everything

```
SOCKET clientSocket = INVALID_SOCKET;
u_short port = 71;
WSADATA wsaData;
int wsaerr;
WORD wVersionRequested = MAKEWORD(2, 2);
wsaerr = WSAStartup(wVersionRequested, &wsaData);
clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
sockaddr_in clientService;
clientService.sin_family = AF_INET;

string serverIP;
cout << "Enter server IP address: ";
cin >> serverIP;
InetPton(AF_INET, serverIP.c_str(), &clientService.sin_addr.s_addr);
```

Client's and servers port numbers must be same and port 71 might be blocked on some devices. Change it if needed.

2. Try to Handshake With Server

```
if (connect(clientSocket, (SOCKADDR*)&clientService, sizeof(clientService)) == SOCKET_ERROR) {
    cout << "Error at connect(): " << WSAGetLastError() << endl;
    closesocket(clientSocket);
    WSACleanup();
    return;
}
else {
    cout << "Connected to server!" << endl;
}
```

3. Take File Path

File path is taken from the user as normal string. Example file path is like this: "C:\My Files\Wallpapers\daft-punk-wallpaper-preview.jpg".

```
string filepath, Sonne;
cout << "Enter your file path: " << endl;
cin >> Sonne;
getline(cin, filepath);
//cout << filepath << endl;
filepath = escapeFilePath(Sonne + filepath);
```

But, when adding this to calculation directly, C++ recognizes "\M" structure as a function in string (not only "\M", every "\" feature). Thus, "filepath" is fixed with a function:

```
static std::string escapeFilePath(const std::string& filePath) {
    std::string escapedPath;
    for (char ch : filePath) {
        if (ch == '\\') {
            escapedPath += "\\\\";
        }
        else {
            escapedPath += ch;
        }
    }
    return escapedPath;
}
```

The function only adds more backlashes to string so it wouldn't recognize the structure as a function of string class.

4. Send the Data

Client calls the send file and passes the process:

```
sendFile(clientSocket, filepath);
```

4.1. Open the File

```
std::ifstream inFile(filePath, std::ios::binary);
if (!inFile.is_open()) {
    cout << "Failed to open file: " << filePath << endl;
    return;
}
```

4.2. Take the Filename & Send It

```
string fileName = filePath.substr(filePath.find_last_of("/\\") + 1);

size_t fileNameSize = fileName.size();
send(clientSocket, reinterpret_cast<const char*>(&fileNameSize), sizeof(fileNameSize), 0);
send(clientSocket, fileName.c_str(), fileNameSize, 0);
```

Because filename is at the end of the path, we take after last backlash in the path. Then we send this filename to server, so the server can write filename to open and write the received file. Also, file extension is send with the name of the file.

4.3. Send the Filesize

```
inFile.seekg(0, ios::end);
streamsize fileSize = inFile.tellg();
inFile.seekg(0, ios::beg);

send(clientSocket, reinterpret_cast<const char*>(&fileSize), sizeof(fileSize), 0);
```

Since it is TCP and quality of the connection is preferred, the file size is captured and send so that server can check it and realize if an error is occurred.

4.4. Send the Data

```
auto start = std::chrono::high_resolution_clock::now();
char buffer[1024];
while (inFile.read(buffer, sizeof(buffer))) {
    send(clientSocket, buffer, sizeof(buffer), 0);
}
// Send any remaining bytes
send(clientSocket, buffer, inFile.gcount(), 0);
```

Finally, the main event. The file is sent here, and it is getting sent in chunks. It uses chunks because sending all of the package together makes the file resend if any little mistake is made in the sending. In this way, only that chunk needs to resend if there is any problem. The resends are handled by TCP automatically because of the nature of the TCP.

Since last chunk cannot be determined in size, it is getting sent individually.

4.5. Wait for the End Signal

Since client closes the socket without waiting, I added a wait command. Client will wait for the server's approval.

```
unsigned char endMsg = 0;
int endRead = recv(clientSocket, reinterpret_cast<char*>(&endMsg), 1, 0);
if (endRead == 1 && endMsg == 0xFF) {
    cout << "End-of-file message (0xFF) received." << endl;
}
else {
    cout << "End-of-file message not received or incorrect." << endl;
}
```

3.1.2. Server

1. Initialize Everything

Since in Server – Client relationship Server is handling most of the work, Server's initializer is longer and a lot more complicated. Winsock library is used here and allows Server to do this connection creation. Again be careful with the port number 71.

```
static void server() {
    WSADATA wsaData;
    int wsaerr;
    WORD wVersionRequested = MAKEWORD(2, 2);
    wsaerr = WSAStartup(wVersionRequested, &wsaData);
    if (wsaerr != 0) {
        std::cout << "The Winsock dll not found" << endl;
        return;
    }

    SOCKET ServerSocket = INVALID_SOCKET;
    ServerSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (ServerSocket == INVALID_SOCKET) {
        cout << "Error at socket():" << WSAGetLastError() << endl;
        return;
    }

    sockaddr_in service;
    u_short port = 71;
    service.sin_family = AF_INET;
    service.sin_addr.s_addr = INADDR_ANY;
    service.sin_port = htons(port);
```

2. Create Connection and Wait for Handshake

The connection is created with bind, then Servers starts to listen for any possible clients. Then when client connects, it will try to adjust this connection request. If no error occurs it accepts the connection and data transfer could go underway.

```
if (bind(ServerSocket, (SOCKADDR*)&service, sizeof(service)) == SOCKET_ERROR) {
    cout << "Error at bind():" << WSAGetLastError() << endl;
    closesocket(ServerSocket);
    WSACleanup();
    return;
}

if (listen(ServerSocket, 1) == SOCKET_ERROR) {
    cout << "Error at listen():" << WSAGetLastError() << endl;
    closesocket(ServerSocket);
    WSACleanup();
    return;
}

SOCKET acceptSocket = accept(ServerSocket, NULL, NULL);
if (acceptSocket == INVALID_SOCKET) {
    cout << "Error at accept():" << WSAGetLastError() << endl;
    closesocket(ServerSocket);
    WSACleanup();
    return;
}
```

3. Receive File

```
receiveFile(acceptSocket);
```

Again like in the client, the work is called through receive function.

3.1. Receive the Filename

```
size_t fileNameSize;
if (recv(acceptSocket, reinterpret_cast<char*>(&fileNameSize), sizeof(fileNameSize), 0) <= 0) {
    std::cerr << "Error receiving file name size." << std::endl;
    return;
}

std::cout << "Received file name size: " << fileNameSize << std::endl;

// Receive the file name
std::vector<char> fileNameBuffer(fileNameSize);
if (recv(acceptSocket, fileNameBuffer.data(), fileNameSize, 0) <= 0) {
    std::cerr << "Error receiving file name." << std::endl;
    return;
}
std::string fileName(fileNameBuffer.begin(), fileNameBuffer.end());
```

3.2. Open the File

```
string uniqueFilePath = getUniqueFilePath(fileName);
ofstream outFile(uniqueFilePath, std::ios::binary);
if (!outFile.is_open()) {
    cout << "Failed to open file: " << uniqueFilePath << endl;
    return;
}
```

After receiving filename, the code creates the file immediately so that it can write it later. It uses a function called getUniqueFilepath to create. It is like this:

```
static std::string getUniqueFilePath(const std::string& fileName) {
    PWSTR path = NULL;
    HRESULT result = SHGetKnownFolderPath(FOLDERID_Downloads, 0, NULL, &path);

    if (SUCCEEDED(result)) {
        char downloadsPath[MAX_PATH];
        wcstombs(downloadsPath, path, MAX_PATH);
        CoTaskMemFree(path);

        std::string directory = std::string(downloadsPath) + "\\";
        std::string filePath = directory + fileName;
        std::string newFilePath = filePath;
        int counter = 1;

        while (ifstream(newFilePath).good()) {
            size_t lastDot = filePath.find_last_of('.');
            if (lastDot == std::string::npos) {
                newFilePath = filePath + "_" + std::to_string(counter);
            }
            else {
                newFilePath = filePath.substr(0, lastDot) + "_" + std::to_string(counter) + filePath.substr(lastDot);
            }
            counter++;
        }

        return newFilePath;
    }
    else {
        std::cerr << "Error retrieving Downloads folder path. Saving file with original name in the current directory." << std::endl;
        return fileName;  // Fallback if Downloads folder cannot be found
    }
}
```

It reaches to downloads folder in the computer and checks if there is a file named and has the same extension as it. If it cannot reach, it folds and if finds a file has the same name and extension, changes name of the current file with adding next number to the name. Then returns filepath at the end.

3.3. Receive the Filesize

```
std::streamsize fileSize;
if (recv(acceptSocket, reinterpret_cast<char*>(&fileSize), sizeof(fileSize), 0) <= 0) {
    std::cerr << "Error receiving file size." << std::endl;
    return;
}

std::cout << "Received file size: " << fileSize << std::endl;
```

3.4. Receive the File in Chunks and Write

```
char buffer[1024];
std::streamsize bytesReceived = 0;

while (bytesReceived < fileSize) {
    int bytesRead = recv(acceptSocket, buffer, sizeof(buffer), 0);
    if (bytesRead > 0) {
        outFile.write(buffer, bytesRead);
        bytesReceived += bytesRead;
    }
    else if (bytesRead == 0) {
        break;
    }
    else {
        cout << "Receive error: " << WSAGetLastError() << endl;
        break;
    }
}

cout << "File received successfully! Saved as: " << uniqueFilePath << endl;
```

3.5. Send the End Signal

```
unsigned char endMsg = 0xFF;
send(acceptSocket, reinterpret_cast<const char*>(&endMsg), 1, 0);
```

3.2. Parallel

In this part, only differences from the Linear part is covered.

3.2.1. Key Differences from Linear

The differences are listed as on the bottom:

Feature	Linear	Parallel
Transfer Method	Single TCP socket	Multiple TCP sockets (one per thread)
Thread Usage	Mostly single-threaded	Uses multiple threads (one per chunk)
File Handling	Full file sent in sequence	File split into multiple chunks

Feature	Linear	Parallel
Speed	Slower (due to sequential send)	Faster (parallel transfer per chunk)
Network Port Usage	Single port (default 71)	Multiple ports (base 41000 + index)
Error Handling	Simple, handled by TCP	TCP still handles chunks, but more socket-level complexity
Complexity	Easy to implement	Complex, requires synchronization and chunk tracking
Use Case	Simple and robust transfers	Large files and high-speed networks

3.2.2. How it Works

1. File Split into Chunks

Before sending, the client reads the file and splits it into *N* chunks (equal to the number of threads specified by the user)

```
static void splitFileIntoChunks(const std::string& filePath, int threadCount, std::vector<std::vector<char>>& chunks) {
    std::ifstream inFile(filePath, std::ios::binary);
    if (!inFile.is_open()) {
        std::cerr << "Failed to open file: " << filePath << std::endl;
        return;
    }

    // Dosya boyutunu al
    inFile.seekg(0, std::ios::end);
    std::streamsize fileSize = inFile.tellg();
    inFile.seekg(0, std::ios::beg);

    // Her parçanın boyutunu hesapla
    std::streamsize chunkSize = fileSize / threadCount;
    std::streamsize lastChunkSize = chunkSize + (fileSize % threadCount); // Son parça, kalan byte'ları da alır

    // Parçaları oluştur
    chunks.resize(threadCount);
    for (int i = 0; i < threadCount; ++i) {
        std::streamsize currentChunkSize = (i == threadCount - 1) ? lastChunkSize : chunkSize;
        chunks[i].resize(currentChunkSize);
        inFile.read(chunks[i].data(), currentChunkSize);
    }

    inFile.close();
}
```

Each chunk is stored in a vector<char>, and their sizes are tracked separately. The final chunk may be slightly larger if the file size is not perfectly divisible.

2. Metadata Exchange

Like in linear mode, the client first connects on port 71 and sends:

- File name and its size

- Full file size
- Number of threads to be used
- Size of each chunk
- Size of the last chunk (in case it's different)

Once this metadata is received, the server replies with a "R" ready signal, and both sides proceed to open **individual connections** for each chunk.

3. Multiple Socket Connections

Client and server open additional TCP sockets on a base port (default: 41000). For example:

- Thread 0 → Port 41000
- Thread 1 → Port 41001
- ...
- Thread N → Port 41000 + N

Each thread establishes its own connection and handles exactly one chunk.

4. Parallel Data Transfer

On the client side:

```
thread(sendChunk, sockets[z], ref(chunks[z])).detach();
```

```
thread(receiveChunk, sockets[i], ref(chunk)).detach();
```

Each thread sends its chunk over its own socket. On the server side:

Each chunk is received in parallel and stored in memory.

5. Reassembling the File

After all chunks are received:

- The server writes them in sequence to form the full file.
- The file is saved with a unique name to prevent overwriting.

This part is like linear mode, but instead of streaming bytes, the system merges already received memory chunks:

```
for (const auto& chunk : chunks) {
    outFile.write(chunk.data(), chunk.size());
}
```

6. Switch Off Everything

```
for (auto& sock : sockets) {
    closesocket(sock);
}

inFile.close();
closesocket(clientSocket);
WSACleanup();
```

4. Project Analysis

4.1. Test

The tests were made on Windows computers. The system is tested on the same computer locally, same computer using Wi-fi and two different computers. All tests were successful.

4.2. Upgradable Things

The Janus only uses TCP protocol. It needs to be developed further if it needs to be used on UDP or maybe in a more detailed way like USB, Telemetry or even ethernet connection.

5. Project Comments

5.1. Possible Usages

The project can be used on daily usage as well. But it mostly designed to used on other projects like Theia.

5.1.1. To-Do's When Using

Do not lose the concept of the limitations.

5.2. Creator Comments

This was kinda the first Orange project of mine. And thus holds a special place in my heart.

Hooah

-Famura

6. Credits

- Famura Only Project

7. References

- Microsoft. (n.d.). Winsock2 Reference. Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2
- ISO/IEC. (2014). ISO/IEC 14882:2014 – Programming Languages – C++ (C++14 Standard). International Organization for Standardization. https://www.iso.org/standard/64029.html
- Microsoft. (n.d.). SHGetKnownFolderPath function (shlobj_core.h). Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/api/shlobj_core/nf-shlobj_core-shgetknownfolderpath
- Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer Networks (5th ed.). Pearson.
- Stevens, W. R., Fenner, B., & Rudoff, A. M. (2004). Unix Network Programming, Volume 1: The Sockets Networking API (3rd ed.). Addison-Wesley.
- Microsoft. (n.d.). InetPton function (ws2tcpip.h). Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/api/ws2tcpip/nf-ws2tcpip-inetpton

8. Disk Contents

- Video explaining the project
- Release of the project
- Project source code (whole Visual Studio Project) contains this report on .docx format
- This report in PDF format
- 11th Dimension by Julian Casablancas