PROJE RAPORU

TURUNCU PROJE

TAMAMLANDI

# Janus

*Yazar(lar):*

Famura

# İçerikler

# 1. Proje Tanımı

Janus projesi, C++ ile geliştirilmiş, esnek ve genişletilebilir bir dosya aktarım sistemi olarak tasarlanmıştır. Farklı iletişim modlarını destekleyerek cihazlar arasında güvenilir veri iletimi sağlar. Özellikle parallel (paralel) ve linear (sıralı) aktarım stratejilerini destekler. Sistem, çalışma zamanında kullanıcının tercih ettiği aktarım modunu ve bağlantı yönünü (gönderme veya alma) seçmesine olanak tanır ve bu sayede kullanıcı dostu, etkileşimli bir deneyim sunar.

Bu proje, diğer projeler için bir altyapı olarak geliştirilmiştir. Bu nedenle frontend gibi kritik bazı özellikleri barındırmıyor. Ham haliyle kullanılmak üzere tasarlanmamıştır ancak mevcut haliyle de çalışabilir durumdadır. Proje, Client–Server bağlantı modelini kullanır. Client veriyi gönderir, Server ise veriyi alır.

# 2. Proje Teknolojileri

## 2.1. Modül(ler)

- **Windows (Windows 10):**
  Tüm proje Windows sistemleri için tasarlanmıştır. Windows 7 gibi daha eski sürümlerde test edilmemiştir. Windows 11 üzerinde test edilmiş ve sorunsuz çalışmaktadır.

## 2.2. Programlama Dili(leri)

- **C++ (C++14 Standard) – Visual Studio 2022:**
  Projenin tamamı C++ ile yazılmıştır ve C++14 standardına göre derlenmektedir. Bu sayede modern dil özellikleri kullanılmış ve Visual Studio 2022 ile uyumluluk sağlanmıştır.

## 2.3. Kütüphaneler & Harici Eklentiler

- **Winsock2.h:**
  TCP/IP tabanlı ağ iletişimi için Windows Sockets API'sini sağlar.
- **targetver.h & stdafx.h:**
  Socket programlama için gerekli temel yapılandırmaları içerir.
- **thread.h:**
  İletişimin paralel kısmında "POSIX threads" kullanımı için gereklidir.
- **shlobj.h:**
  Windows'taki özel klasörlere erişim sağlamak için kullanılır (örneğin SHGetKnownFolderPath ile Downloads klasörü).

# 3. Proje Mimarisi

Daha önce belirtildiği gibi proje iki ana bölümden oluşur: **Linear** ve **Parallel**. Bu bölümlerin tamamı ana dosya (main file) tarafından kontrol edilir.

## 3.1. Linear

Linear yapı, TCP bağlantısının doğal yapısına uygun olarak **Client** ve **Server** olmak üzere iki farklı modda çalışır.

### 3.1.1. Client

1. Başlatma (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 ve Server port numaraları aynı olmalıdır. Varsayılan port olan **71**, bazı cihazlarda engellenmiş olabilir. Gerekirse değiştirilmelidir.

2. Server ile Handshake Denemesi

```
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. Dosya Yolu Alma

Dosya yolu, kullanıcıdan normal bir string olarak alınır. Örnek bir dosya yolu şu şekildedir: “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);
```

Ancak bu yol doğrudan işleme sokulduğunda, C++ \M gibi yapıları string içinde özel karakter olarak algılar (sadece \M değil, tüm \ yapıları). Bu nedenle dosya yolu özel bir fonksiyonla düzeltilir.

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

Bu fonksiyon, string içerisine ekstra ters eğik çizgiler (\\) ekleyerek C++’ın bu yapıları özel karakter olarak algılamasını engeller.

4. Veriyi Gönderme

Client, dosya gönderme fonksiyonunu çağırır ve işlem şu adımlarla ilerler:

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

4.1. Dosyayı Açma

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

4.2. Dosya Adını Alma ve Gönderme

```
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);
```

Dosya adı, dosya yolunun en sonunda yer aldığı için, yol içerisindeki son ters eğik çizgiden (\) sonraki kısım alınır. Bu dosya adı server'a gönderilir. Böylece server, alınan veriyi doğru dosya adı ve uzantı ile oluşturabilir. Dosya uzantısı da dosya adıyla birlikte gönderilir.

4.3. Dosya Boyutunu Gönderme

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

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

TCP bağlantısı güvenilirliği önceliklendirdiği için dosya boyutu önceden gönderilir. Server bu bilgi sayesinde aktarım sırasında bir hata oluşup oluşmadığını kontrol edebilir.

4.4. Veriyi Gönderme

```
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);
```

Asıl aktarım bu aşamada gerçekleşir. Dosya **parçalar (chunks)** halinde gönderilir. Bunun sebebi, dosyanın tamamını tek seferde göndermenin, küçük bir hatada tüm dosyanın yeniden gönderilmesine yol açmasıdır.

Parça bazlı aktarım sayesinde yalnızca sorunlu parça yeniden gönderilir. TCP'nin doğası gereği bu yeniden gönderme işlemleri otomatik olarak yönetilir.

Son parça boyutu önceden kesin olarak belirlenemediği için ayrı şekilde gönderilir.

4.5. Bitiş Sinyalini Bekleme

Client, socket'i doğrudan kapattığı için, server'dan onay almadan bağlantıyı sonlandırmaması adına bir bekleme mekanizması eklenmiştir. Client, server'dan gelen onay sinyalini bekler.

```
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. Başlatma (Initialize Everything)

Server, Client–Server ilişkisinde daha fazla iş yükü taşıdığı için başlatma süreci daha uzun ve karmaşıktır. Winsock kütüphanesi kullanılarak bağlantı oluşturulur. Burada da port numarası **71** konusunda dikkatli olunmalıdır.

```
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. Bağlantı Oluşturma ve Handshake Bekleme

bind fonksiyonu ile bağlantı oluşturulur, ardından server olası client'ları dinlemeye başlar. Client bağlantı talebinde bulunduğunda, bu istek değerlendirilir. Herhangi bir hata oluşmazsa bağlantı kabul edilir ve veri aktarımı başlar.

```
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. Dosya Alma

`receiveFile(acceptSocket);` Client'ta olduğu gibi, dosya alma işlemi de bir fonksiyon üzerinden yürütülür.

3.1. Dosya Adını Alma

```
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. Dosyayı Açma

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

Dosya adı alındıktan sonra, server dosyayı hemen oluşturur. Bunun için getUniqueFilepath adlı bir fonksiyon kullanılır.

Bu fonksiyon, bilgisayardaki **Downloads** klasörüne ulaşır ve aynı isimde ve uzantıda bir dosya olup olmadığını kontrol eder. Eğer ulaşamazsa işlem başarısız olur. Aynı isimde bir dosya bulunursa, dosya adına bir numara eklenerek benzersiz bir isim oluşturulur ve bu dosya yolu geri döndürülür.

### 3.3. Dosya Boyutunu Alma

```
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
    }
}
```

```
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. Dosyayı Parçalar Halinde Alma ve Yazma

```
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. Bitiş Sinyali Gönderme

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

## 3.2. Parallel

Bu bölümde yalnızca Linear yapıdan farklı olan noktalar ele alınmıştır.

### 3.2.1. Linear Yapıdan Temel Farklar

| **Özellik** | **Linear** | **Parallel** |
|---|---|---|
| Aktarım Yöntemi | Tek TCP socket | Birden fazla TCP socket (thread başına bir tane) |
| Thread Kullanımı | Çoğunlukla tek thread | Çoklu thread (parça başına bir thread) |
| Dosya İşleme | Dosya sıralı şekilde gönderilir | Dosya parçalara bölünür |
| Hız | Daha yavaş | Daha hızlı |
| Port Kullanımı | Tek port (varsayılan 71) | Birden fazla port (41000 + index) |
| Hata Yönetimi | TCP tarafından yönetilir | TCP + socket karmaşıklığı |
| Karmaşıklık | Basit | Daha karmaşık |
| Kullanım Alanı | Basit ve güvenilir | Büyük dosyalar, hızlı ağlar |

### 3.2.2. Nasıl Çalışıyor

1. Dosyanın Parçalara Bölünmesi

Gönderim öncesinde Client, dosyayı kullanıcı tarafından belirlenen thread sayısına eşit olacak şekilde **N parçaya** böler.

Her parça vector<char> içinde tutulur ve boyutları ayrı ayrı takip edilir. Dosya boyutu tam bölünemezse, son parça biraz daha büyük olabilir.

```cpp
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();
}
```

2. Metadata Alışverişi

Linear modda olduğu gibi Client, önce port **71** üzerinden bağlanır ve şu bilgileri gönderir:

- Dosya adı ve uzantısı
- Toplam dosya boyutu
- Kullanılacak thread sayısı
- Her bir parçanın boyutu
- Son parçanın boyutu

Server bu bilgileri aldıktan sonra "R" (ready) sinyalini gönderir ve iki taraf da parça bazlı bağlantıları açmaya başlar.

3. Çoklu Socket Bağlantıları

Client ve Server, varsayılan **41000** taban portu üzerinden ek TCP socket'leri açar. Örnek olarak:

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

Her thread kendi bağlantısını kurar ve yalnızca kendisine ait parçayı yönetir.

4. Paralel Veri Aktarımı

Client tarafında her thread kendi parçasını kendi socket'i üzerinden gönderir.

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

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

Server tarafında ise her parça paralel olarak alınır ve bellekte tutulur.

5. Dosyanın Yeniden Birleştirilmesi

Tüm parçalar alındıktan sonra server:

- Parçaları sırayla yazarak tam dosyayı oluşturur.
- Dosyayı benzersiz bir isimle kaydeder.

Bu yapı Linear moda benzer, ancak byte akışı yerine bellekteki hazır parçalar birleştirilir.

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

6. Her Şeyi Kapatma

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

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

# 4. Proje Analizi

## 4.1. Test

Testler Windows bilgisayarlar üzerinde yapılmıştır. Aynı bilgisayar üzerinde yerel testler, Wi-Fi üzerinden testler ve iki farklı bilgisayar arasında yapılan testlerin tamamı başarılı olmuştur.

## 4.2. Geliştirilebilir Alanlar

Janus yalnızca TCP protokolünü kullanmaktadır. UDP, USB, telemetri veya Ethernet gibi farklı iletişim yöntemleri için daha da geliştirilebilir.

# 5. Proje Yorumları

## 5.1. Olası Kullanım Alanları

Proje günlük kullanım için de uygundur ancak esas olarak **Theia** gibi diğer projelerde altyapı olarak kullanılmak üzere tasarlanmıştır.

### 5.1.1. Kullanım Sırasında Yapılması Gerekenler

Sistemin ve kullanılan ağın sınırlamalarının farkında olunmalıdır.

## 5.2. Geliştirici Yorumları

Bu proje benim ilk **Turuncu projem** sayılır. Bu yüzden bende ayrı bir yeri vardır.

Hooah

-Famura

# 6. Credits

- Sadece Famura Projesi

## 7. Referanslar


- 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 İçeriği

- Projeyi anlatan video
- Proje release'i
- Tüm Visual Studio proje kaynak kodları
- Bu raporun PDF versiyonu
- 11th Dimension by Julian Casablancas