Uploading a file with HTTP POST

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <winsock2.h>
#include <windows.h>

#pragma comment(lib, "ws2_32.lib")

const int PORT = 8080;
const std::string UPLOAD_DIR = "./";

void handleFileUpload(SOCKET clientSocket)
{
    char buffer[8192];
    std::string requestBody;
    bool isPostMethod = false;
    bool isFileSection = false;
    std::string boundary;

    // Read HTTP request
    int bytesRead;
    while ((bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0)) > 0)
    {
        requestBody.append(buffer, bytesRead);

        // Find the boundary from the content type header
        if (!boundary.empty())
        {
            if (requestBody.find(boundary) != std::string::npos)
            {
                isFileSection = true;
                break;
            }
        }
        else
        {
            size_t boundaryPos = requestBody.find("boundary=");
            if (boundaryPos != std::string::npos)
            {
                size_t start = boundaryPos + 9;
                size_t end = requestBody.find("\r\n", start);
                boundary = "--" + requestBody.substr(start, end - start);
            }
```

Generally server have max buffer

From OS Kernel socket, data is copied to this process memory

```
    }
}

if (isFileSection)
{
    // Extract the file content
    size_t fileStartPos = requestBody.find("\r\n\r\n") + 4;
    size_t fileEndPos = requestBody.find(boundary, fileStartPos);
    std::string fileContent = requestBody.substr(fileStartPos, fileEndPos - fileStartPos);

    // Extract the filename
    size_t filenamePos = requestBody.find("Content-Disposition: form-data; name=\"file\"; filename=\"");
    filenamePos = requestBody.find("\"", filenamePos + 1) + 1;
    size_t filenameEndPos = requestBody.find("\"", filenamePos);
    std::string filename = requestBody.substr(filenamePos, filenameEndPos - filenamePos);

    // Save the file
    std::ofstream file(UPLOAD_DIR + filename, std::ios::binary);
    if (file)
    {
        file.write(fileContent.c_str(), fileContent.size());
        file.close();
    }
}

std::cout << "File Created In HDD Successfully" << std::endl;

// Send response with CORS headers
std::string response =
    "HTTP/1.1 200 OK\r\n"
    "Content-Type: text/plain\r\n"
    "Access-Control-Allow-Origin: *\r\n" // Allow any origin
    "Access-Control-Allow-Methods: POST, GET, OPTIONS\r\n"
    "Access-Control-Allow-Headers: Content-Type\r\n"
    "\r\n"
    "File uploaded successfully!";

send(clientSocket, response.c_str(), response.size(), 0);
```

File handling

OS system call to write to Disk

Response header string

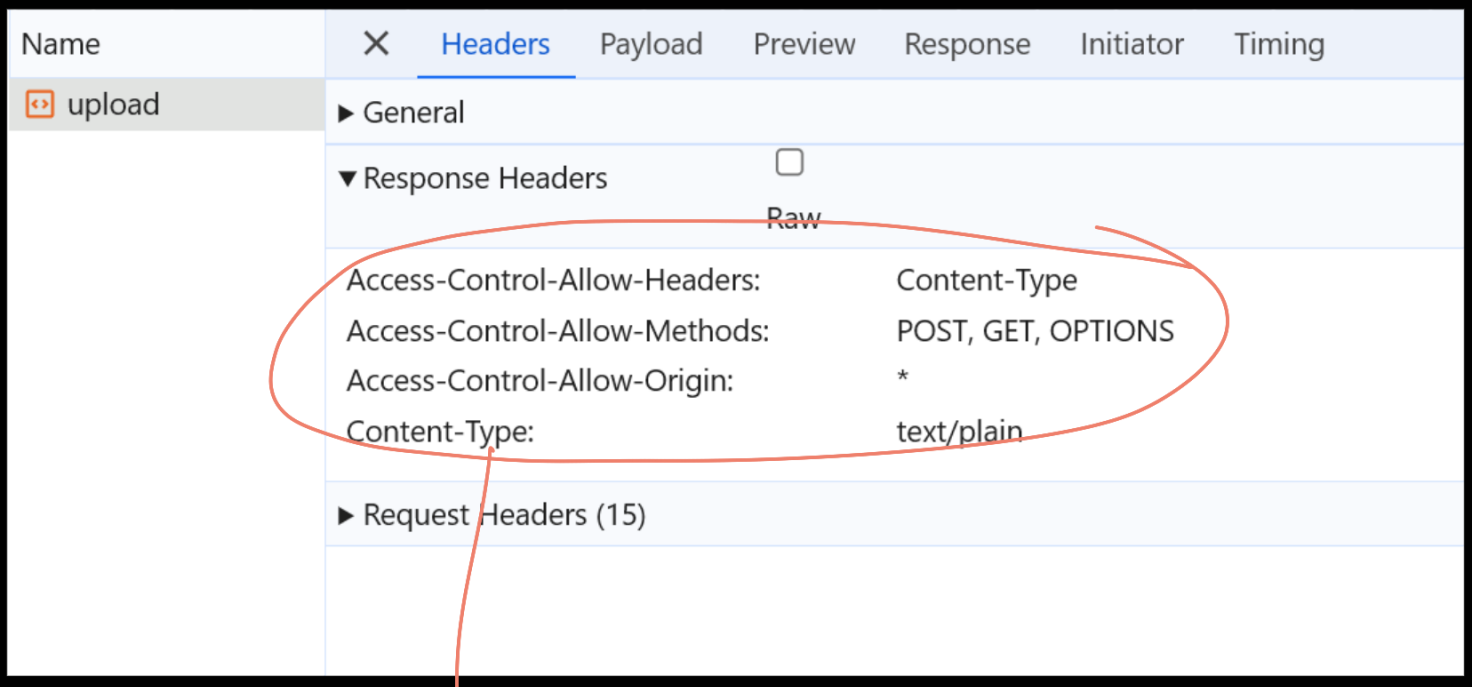
Separator

Response body

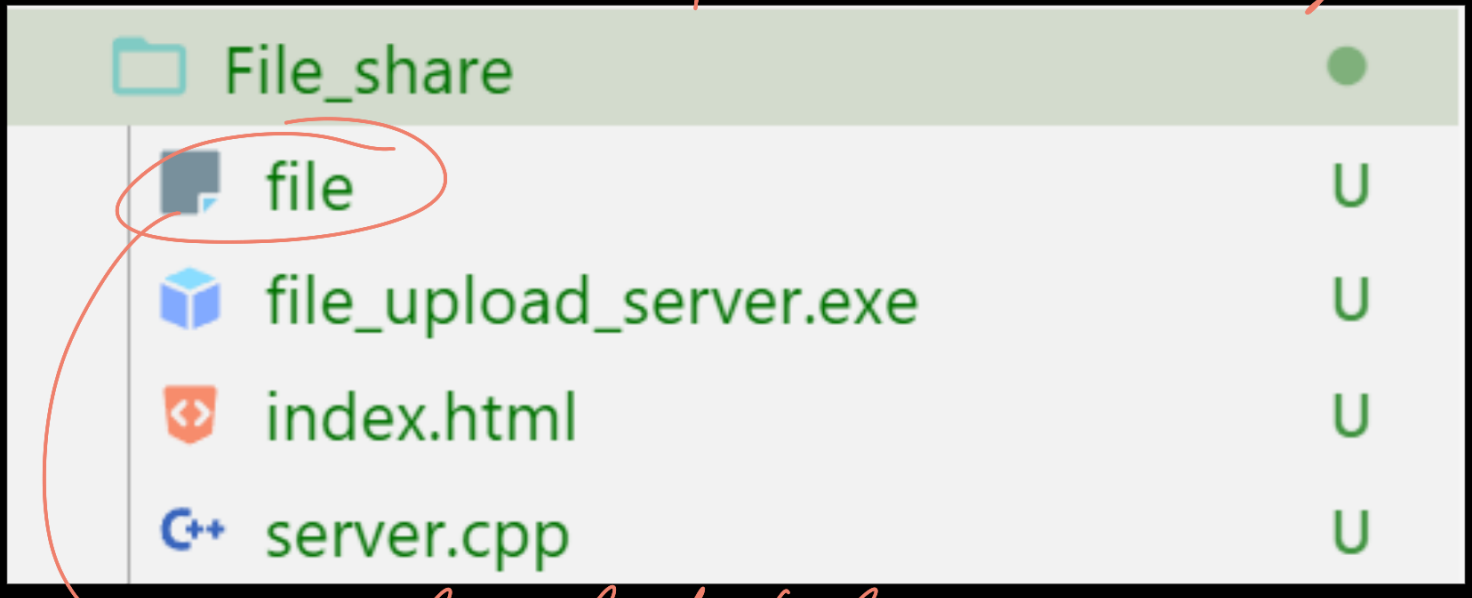
```cpp
int main()
{
    // Initialize Winsock
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        std::cerr << "WSAStartup failed!" << std::endl;
        return -1;
    }

    // Create socket
    SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (serverSocket == INVALID_SOCKET)
    {
        std::cerr << "Error creating socket" << std::endl;
        WSACleanup();
        return -1;
    }

    // Set up server address
    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    // Bind the socket
    if (bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR)
    {
        std::cerr << "Error binding socket" << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return -1;
    }

    // Listen for incoming connections
    if (listen(serverSocket, 5) == SOCKET_ERROR)
    {
        std::cerr << "Error listening on socket" << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return -1;
    }

```

```cpp
    std::cout << "Server started on http://localhost:" << PORT << std::endl;
    std::cout << "Waiting for connections..." << std::endl;

    // Accept client connections and handle them
    while (true) ——→ Loop (Infinite)
    {
        SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);
        if (clientSocket == INVALID_SOCKET)
        {
            std::cerr << "Error accepting client" << std::endl;
            continue;
        }

        std::cout << "Connection established!" << std::endl;

        // Handle the request
        handleFileUpload(clientSocket);
    }

    // Clean up
    closesocket(serverSocket);
    WSACleanup();
    return 0;
}
```

↳ Blocking call



Response header string



Uploaded file

```
3020 492.076093  127.0.0.1  127.0.0.1  TCP   56 50605 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3021 492.076240  127.0.0.1  127.0.0.1  TCP   56 8080 → 50605 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 S
3022 492.076374  127.0.0.1  127.0.0.1  TCP   44 50605 → 8080 [ACK] Seq=1 Ack=1 Win=327424 Len=0
3023 492.078158  127.0.0.1  127.0.0.1  TCP   660 50605 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=616 [TCP segment of
3024 492.078341  127.0.0.1  127.0.0.1  TCP   44 8080 → 50605 [ACK] Seq=1 Ack=617 Win=2160640 Len=0
3025 492.078959  127.0.0.1  127.0.0.1  HTTP  236 POST /upload HTTP/1.1  (text/plain)
3026 492.079075  127.0.0.1  127.0.0.1  TCP   44 8080 → 50605 [ACK] Seq=1 Ack=809 Win=2160384 Len=0
3027 492.082159  127.0.0.1  127.0.0.1  TCP   242 8080 → 50605 [PSH, ACK] Seq=1 Ack=809 Win=2160384 Len=198 [TCP segment
3028 492.082265  127.0.0.1  127.0.0.1  TCP   44 50605 → 8080 [ACK] Seq=809 Ack=199 Win=327168 Len=0
3029 492.082340  127.0.0.1  127.0.0.1  HTTP  44 HTTP/1.1 200 OK  (text/plain)
3030 492.082392  127.0.0.1  127.0.0.1  TCP   44 50605 → 8080 [ACK] Seq=809 Ack=200 Win=327168 Len=0
3031 492.082890  127.0.0.1  127.0.0.1  TCP   44 50605 → 8080 [FIN, ACK] Seq=809 Ack=200 Win=327168 Len=0
3032 492.083024  127.0.0.1  127.0.0.1  TCP   44 8080 → 50605 [ACK] Seq=200 Ack=810 Win=2160384 Len=0
3033 492.338516  127.0.0.1  127.0.0.1  TCP   56 50606 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3034 492.338578  127.0.0.1  127.0.0.1  TCP   56 8080 → 50606 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 S
3035 492.338660  127.0.0.1  127.0.0.1  TCP   44 50606 → 8080 [ACK] Seq=1 Ack=1 Win=327424 Len=0
```

Issues with above approach

What if we upload 1 GB file from frontend, also it's not 1 client, multiple client

1. Memory Usage

The server stores the entire HTTP request body in memory as a `std::string` (`requestBody`). For a 1 GB file:

- The server will attempt to allocate at least 1 GB of memory, which can lead to:
  - **Memory Exhaustion**: If the system doesn't have sufficient memory, the server will crash.
  - **Performance Issues**: Even if sufficient memory exists, using it all at once will severely degrade system performance.

---

2. Socket Buffer Limits

The `recv` function reads data in chunks of 8192 bytes, appending it to `requestBody`. However:

- **Network Bottleneck**: Transmitting a large file over the network will take significant time, especially if the network connection is slow.
- **Buffer Overflow**: If the network buffer fills up due to slow processing or high memory consumption, the transfer may stall or fail.

3. File Handling

The code writes the file content to disk only after the entire file is received:

- **I/O Bottleneck**: Writing 1 GB of data to disk in a single operation can overwhelm the I/O subsystem.
- **File Truncation**: If the process crashes or the connection is interrupted during upload, the file will be incomplete or corrupted.

---

4. Multipart Parsing

The current implementation processes the entire request body as a single string before extracting the file:

- **Inefficient Parsing**: Parsing 1 GB of data in memory can be extremely slow and error-prone.
- **Potential Data Loss**: If the request isn't received completely, the parsing will fail.

5. Timeouts

Large file uploads can take significant time:

- **Client Timeouts**: If the client application has a timeout for uploads, it might abort the operation.
- **Server Timeouts**: If the server doesn't process data fast enough, the connection may timeout.

* File upload is never implemented in this way, multi-part upload is one way to solve this