**Text file transfer from one server to another server**

**Server code**

#include <iostream>

#include <fstream>

#include <cstring>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

const int PORT = 8080;

const int BUFFER\_SIZE = 1024;

void receive\_file(const char\* server\_ip) {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Create socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = inet\_addr(server\_ip);

address.sin\_port = htons(PORT);

// Bind the socket to the network address and port

if (bind(server\_fd, (struct sockaddr\*)&address, sizeof(address)) < 0) {

perror("bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("listen");

exit(EXIT\_FAILURE);

}

std::cout << "Server listening on " << server\_ip << ":" << PORT << std::endl;

// Accept an incoming connection

if ((new\_socket = accept(server\_fd, (struct sockaddr\*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("accept");

exit(EXIT\_FAILURE);

}

std::ofstream outfile("received\_file.txt", std::ios::binary);

int bytes\_read;

while ((bytes\_read = read(new\_socket, buffer, BUFFER\_SIZE)) > 0) {

outfile.write(buffer, bytes\_read);

}

outfile.close();

close(new\_socket);

close(server\_fd);

std::cout << "File received successfully." << std::endl;

}

int main() {

const char\* server\_ip = "127.0.0.1"; // Change this to the appropriate server IP

receive\_file(server\_ip);

return 0;

}

**Client code:**

#include <iostream>

#include <fstream>

#include <cstring>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

const int PORT = 8080;

const int BUFFER\_SIZE = 1024;

void send\_file(const char\* server\_ip, const char\* file\_path) {

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = {0};

// Create socket file descriptor

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

std::cerr << "Socket creation error" << std::endl;

return;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 and IPv6 addresses from text to binary form

if (inet\_pton(AF\_INET, server\_ip, &serv\_addr.sin\_addr) <= 0) {

std::cerr << "Invalid address/ Address not supported" << std::endl;

return;

}

// Connect to the server

if (connect(sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) {

std::cerr << "Connection Failed" << std::endl;

return;

}

std::ifstream infile(file\_path, std::ios::binary);

if (!infile) {

std::cerr << "File not found: " << file\_path << std::endl;

return;

}

while (!infile.eof()) {

infile.read(buffer, BUFFER\_SIZE);

int bytes\_read = infile.gcount();

send(sock, buffer, bytes\_read, 0);

}

infile.close();

close(sock);

std::cout << "File sent successfully." << std::endl;

}

int main() {

const char\* server\_ip = "127.0.0.1"; // Change this to the appropriate server IP

const char\* file\_path = "file\_to\_send.txt"; // Change this to the appropriate file path

send\_file(server\_ip, file\_path);

return 0;

}