**Text file transfer:**

**Server:**

#include <iostream>

#include <fstream>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#define PORT 65432

#define BUFFER\_SIZE 1024

void receiveFile(int new\_socket) {

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

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

if (!outfile.is\_open()) {

std::cerr << "Failed to open file for writing" << std::endl;

return;

}

int bytesReceived = 0;

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

outfile.write(buffer, bytesReceived);

}

outfile.close();

std::cout << "File received from client and saved as 'received\_from\_client.txt'" << std::endl;

}

void sendFile(int new\_socket) {

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

std::ifstream infile("send\_to\_client.txt", std::ios::binary);

if (!infile.is\_open()) {

std::cerr << "Failed to open file for reading" << std::endl;

return;

}

while (infile.read(buffer, BUFFER\_SIZE)) {

send(new\_socket, buffer, infile.gcount(), 0);

}

// Send any remaining bytes

if (infile.gcount() > 0) {

send(new\_socket, buffer, infile.gcount(), 0);

}

infile.close();

std::cout << "File 'send\_to\_client.txt' sent to client" << std::endl;

}

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

// Creating socket file descriptor

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

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Define the server address

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

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");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

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

perror("listen");

close(server\_fd);

exit(EXIT\_FAILURE);

}

std::cout << "Server listening on port " << PORT << std::endl;

// Accept a connection

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

perror("accept");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Receive a file from the client

receiveFile(new\_socket);

// Send a file to the client

sendFile(new\_socket);

close(new\_socket);

close(server\_fd);

    return 0;

}

**Client:**

#include <iostream>

#include <fstream>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 65432

#define BUFFER\_SIZE 1024

void sendFile(int sock) {

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

std::ifstream infile("send\_to\_server.txt", std::ios::binary);

if (!infile.is\_open()) {

std::cerr << "Failed to open file for reading" << std::endl;

return;

}

while (infile.read(buffer, BUFFER\_SIZE)) {

send(sock, buffer, infile.gcount(), 0);

}

// Send any remaining bytes

if (infile.gcount() > 0) {

send(sock, buffer, infile.gcount(), 0);

}

infile.close();

std::cout << "File 'send\_to\_server.txt' sent to server" << std::endl;

}

void receiveFile(int sock) {

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

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

if (!outfile.is\_open()) {

std::cerr << "Failed to open file for writing" << std::endl;

return;

}

int bytesReceived = 0;

while ((bytesReceived = read(sock, buffer, BUFFER\_SIZE)) > 0) {

outfile.write(buffer, bytesReceived);

}

outfile.close();

std::cout << "File received from server and saved as 'received\_from\_server.txt'" << std::endl;

}

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

// Creating socket file descriptor

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

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

return -1;

}

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, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

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

return -1;

}

// Connect to the server

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

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

return -1;

}

// Send a file to the server

sendFile(sock);

// Receive a file from the server

receiveFile(sock);

close(sock);

    return 0;

}