

**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ  
 НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу  
Кафедра системного проектування**

**Звіт**

**про виконання практичної роботи №1  
з дисципліни «Комп’ютерні мережі»**

Виконав:  
студент III курсу, групи ДА-21  
Сімков Микита Дмитрович

Прийняв:

к.н. Гіоргізова-Гай В. Ш.

Київ – 2025

**ПОСТАНОВКА ЗАДАЧІ**

Мета роботи:

* Ознайомлення з роботою мережевих транспортних протоколів TCP іUDP;
* Ознайомлення з сокетами і елементами API сокетів;
* Написання програмних модулів клієнт і сервер для організації обміну даними за протоколами TCP і UDP.

Завдання

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варіанта | N | Тип1 | M | Тип2 |
| 20 | 5 | int | 2 | long |

**ХІД РОБОТИ**

Огляд основних функцій бібліотеки Winsock:

**WSAStartup**

Функція WSAStartup виконує ініціалізацію бібліотеки Winsock перед її використанням у програмному забезпеченні. Вона приймає параметр wVersionRequested, який визначає версію використовуваного протоколу (наприклад, 2.2), та покажчик lpWSAData, у який записується інформація про ініціалізовану бібліотеку. У разі успішної ініціалізації функція повертає 0, в іншому випадку — код помилки.

**socket**

Функція socket створює новий сокет для мережевої комунікації. Вона приймає такі параметри: сімейство протоколів (af, наприклад, AF\_INET для TCP/IP), тип сокету (SOCK\_STREAM для TCP або SOCK\_DGRAM для UDP) та протокол транспортного рівня (IPPROTO\_TCP або IPPROTO\_UDP). У разі успішного створення повертається дескриптор сокету, інакше — INVALID\_SOCKET.

**bind**

Функція bind прив’язує створений сокет до конкретної локальної IP-адреси та порту. Для цього вона використовує дескриптор сокету (s), структуру sockaddr\_in, яка містить IP-адресу та порт (name), а також розмір цієї структури (namelen). Якщо прив’язка виконана успішно, функція повертає 0, в іншому випадку — код помилки.

**listen**

Функція listen переводить сокет у режим очікування вхідних підключень, що необхідно для реалізації серверної частини мережевої взаємодії. Вона отримує дескриптор сокету (s) та параметр backlog, який визначає максимальну кількість підключень у черзі очікування. У разі успіху повертається 0, при помилці — відповідний код.

**accept**

Функція accept використовується сервером для прийому вхідних підключень від клієнтів. Вона приймає дескриптор прослуховуючого сокету (s), структуру sockaddr\_in (addr), що містить інформацію про клієнта, та її розмір (addrlen). У разі успішного підключення повертається новий дескриптор сокету, який використовується для подальшої взаємодії з клієнтом.

**connect**

Функція connect застосовується на стороні клієнта для встановлення з'єднання з сервером. Вона приймає дескриптор сокету (s), структуру sockaddr\_in (name), що містить IP-адресу сервера, а також її розмір (namelen). У разі успіху повертається 0, в іншому випадку — код помилки.

**send / recv**

Функції send та recv забезпечують передачу (send) і прийом (recv) даних через встановлене TCP-з'єднання. Вони використовують дескриптор сокету (s), буфер (buf), розмір передаваних або отримуваних даних (len) та додаткові флаги (flags). Повертається кількість переданих або отриманих байт, а у разі виникнення помилки — SOCKET\_ERROR.

**sendto / recvfrom**

Функції sendto та recvfrom застосовуються для обміну даними в безз'єднувальних UDP-з'єднаннях. Вони використовують ті самі аргументи, що й send/recv, але додатково містять параметри, що визначають адресу отримувача або відправника (addr) та її розмір (addrlen). У разі успішного виконання повертається кількість переданих або отриманих байт, при помилці — SOCKET\_ERROR.

**closesocket**

Функція closesocket закриває сокет та вивільняє всі пов’язані з ним ресурси. Вона приймає єдиний аргумент — дескриптор сокету (s). Якщо завершення виконано успішно, повертається 0, інакше — код помилки.

**WSACleanup**

Функція WSACleanup завершує роботу бібліотеки Winsock, вивільняючи всі ресурси, що використовувалися під час її роботи. Вона не потребує аргументів і повертає 0 у разі успішного виконання або код помилки у випадку виникнення проблем.

**Лістинг коду**

client-tcp.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#pragma comment(lib, "ws2\_32.lib")

#define SERVER\_IP "127.0.0.1"

#define SERVER\_PORT 8001

void send\_int\_data(SOCKET sock, int \*data, size\_t size) {

    size\_t bytes\_sent = 0;

    size\_t total\_size = size \* sizeof(int);

    char buffer[16];

    for (size\_t i = 0; i < size; i++) {

        int value = htonl(data[i]);

        send(sock, (char\*)&value, sizeof(int), 0);

    }

    printf("Sent int data:\n");

    for (size\_t i = 0; i < size; i++) {

        printf("%d ", data[i]);

    }

    printf("\n");

    while (bytes\_sent < total\_size) {

        int received = recv(sock, buffer, sizeof(buffer), 0);

        if (received <= 0) {

            perror("recv failed");

            closesocket(sock);

            WSACleanup();

            exit(EXIT\_FAILURE);

        }

        bytes\_sent += received;

        printf("Received: %.\*s\n", received, buffer);

    }

}

void send\_long\_data(SOCKET sock, long long \*data, size\_t size) {

    size\_t bytes\_sent = 0;

    size\_t total\_size = size \* sizeof(long long);

    char buffer[16];

    for (size\_t i = 0; i < size; i++) {

        long long value = htonll(data[i]);

        send(sock, (char\*)&value, sizeof(long long), 0);

    }

    printf("Sent long data:\n");

    for (size\_t i = 0; i < size; i++) {

        printf("%lld ", data[i]);

    }

    printf("\n");

    while (bytes\_sent < total\_size) {

        int received = recv(sock, buffer, sizeof(buffer), 0);

        if (received <= 0) {

            perror("recv failed");

            closesocket(sock);

            WSACleanup();

            exit(EXIT\_FAILURE);

        }

        bytes\_sent += received;

        printf("Received: %.\*s\n", received, buffer);

    }

}

int main() {

    WSADATA wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

        printf("WSAStartup failed\n");

        return 1;

    }

    SOCKET sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock == INVALID\_SOCKET) {

        perror("Socket creation failed");

        WSACleanup();

        exit(EXIT\_FAILURE);

    }

    struct sockaddr\_in server\_addr;

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

    server\_addr.sin\_port = htons(SERVER\_PORT);

    inet\_pton(AF\_INET, SERVER\_IP, &server\_addr.sin\_addr);

    printf("Connecting to %s port %d\n", SERVER\_IP, SERVER\_PORT);

    if (connect(sock, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) == SOCKET\_ERROR) {

        perror("Connection failed");

        closesocket(sock);

        WSACleanup();

        exit(EXIT\_FAILURE);

    }

    int message1[] = {10, 20, 30, 40, 50};

    long long message2[] = {10000000000LL, 20000000000LL};

    send\_int\_data(sock, message1, sizeof(message1) / sizeof(int));

    send\_long\_data(sock, message2, sizeof(message2) / sizeof(long long));

    printf("Closing socket\n");

    closesocket(sock);

    WSACleanup();

    return 0;

}

server-tcp.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#pragma comment(lib, "ws2\_32.lib")

#define SERVER\_IP "127.0.0.1"

#define SERVER\_PORT 8001

void receive\_int\_data(SOCKET client\_sock, int \*buffer, size\_t size) {

    size\_t total\_size = size \* sizeof(int);

    size\_t bytes\_received = 0;

    while (bytes\_received < total\_size) {

        int received = recv(client\_sock, (char\*)buffer + bytes\_received, total\_size - bytes\_received, 0);

        if (received <= 0) {

            perror("recv failed");

            closesocket(client\_sock);

            WSACleanup();

            exit(EXIT\_FAILURE);

        }

        bytes\_received += received;

    }

    printf("Received int data:\n");

    for (size\_t i = 0; i < size; i++) {

        buffer[i] = ntohl(buffer[i]);

        printf("%d ", buffer[i]);

    }

    printf("\n");

    send(client\_sock, "OK", 2, 0);

}

void receive\_long\_data(SOCKET client\_sock, long long \*buffer, size\_t size) {

    size\_t total\_size = size \* sizeof(long long);

    size\_t bytes\_received = 0;

    while (bytes\_received < total\_size) {

        int received = recv(client\_sock, (char\*)buffer + bytes\_received, total\_size - bytes\_received, 0);

        if (received <= 0) {

            perror("recv failed");

            closesocket(client\_sock);

            WSACleanup();

            exit(EXIT\_FAILURE);

        }

        bytes\_received += received;

    }

    printf("Received long data:\n");

    for (size\_t i = 0; i < size; i++) {

        buffer[i] = ntohll(buffer[i]);

        printf("%lld ", buffer[i]);

    }

    printf("\n");

    send(client\_sock, "OK", 2, 0);

}

int main() {

    WSADATA wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

        printf("WSAStartup failed\n");

        return 1;

    }

    SOCKET server\_sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (server\_sock == INVALID\_SOCKET) {

        perror("Socket creation failed");

        WSACleanup();

        exit(EXIT\_FAILURE);

    }

    struct sockaddr\_in server\_addr;

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

    server\_addr.sin\_port = htons(SERVER\_PORT);

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

    if (bind(server\_sock, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) == SOCKET\_ERROR) {

        perror("Bind failed");

        closesocket(server\_sock);

        WSACleanup();

        exit(EXIT\_FAILURE);

    }

    if (listen(server\_sock, 1) == SOCKET\_ERROR) {

        perror("Listen failed");

        closesocket(server\_sock);

        WSACleanup();

        exit(EXIT\_FAILURE);

    }

    printf("Waiting for a connection on %s:%d...\n", SERVER\_IP, SERVER\_PORT);

    struct sockaddr\_in client\_addr;

    int client\_size = sizeof(client\_addr);

    SOCKET client\_sock = accept(server\_sock, (struct sockaddr\*)&client\_addr, &client\_size);

    if (client\_sock == INVALID\_SOCKET) {

        perror("Accept failed");

        closesocket(server\_sock);

        WSACleanup();

        exit(EXIT\_FAILURE);

    }

    printf("Client connected!\n");

    int int\_data[5];

    long long long\_data[2];

    receive\_int\_data(client\_sock, int\_data, 5);

    receive\_long\_data(client\_sock, long\_data, 2);

    printf("Closing connection\n");

    closesocket(client\_sock);

    closesocket(server\_sock);

    WSACleanup();

    return 0;

}

server-udp.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#pragma comment(lib, "ws2\_32.lib")

int main(void) {

    WSADATA wsaData;

    int iResult;

    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

    if (iResult != 0) {

        printf("WSAStartup failed: %d\n", iResult);

        return 1;

    }

    SOCKET sock = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

    if (sock == INVALID\_SOCKET) {

        printf("Socket creation failed: %d\n", WSAGetLastError());

        WSACleanup();

        return 1;

    }

    struct sockaddr\_in serverAddr;

    memset(&serverAddr, 0, sizeof(serverAddr));

    serverAddr.sin\_family = AF\_INET;

    serverAddr.sin\_port = htons(8001);

    serverAddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    if (bind(sock, (struct sockaddr \*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

        printf("Bind failed: %d\n", WSAGetLastError());

        closesocket(sock);

        WSACleanup();

        return 1;

    }

    printf("Start on %s port %d\n", "localhost", 8001);

    while (1) {

        char recvBuf[20];

        struct sockaddr\_in clientAddr;

        int clientAddrLen = sizeof(clientAddr);

        printf("Waiting to receive message\n");

        int recvLen = recvfrom(sock, recvBuf, sizeof(recvBuf), 0,

                               (struct sockaddr \*)&clientAddr, &clientAddrLen);

        if (recvLen == SOCKET\_ERROR) {

            printf("recvfrom failed: %d\n", WSAGetLastError());

            break;

        }

        printf("Received %d bytes from %s:%d\n", recvLen,

               inet\_ntoa(clientAddr.sin\_addr), ntohs(clientAddr.sin\_port));

        if (recvLen > 0) {

            int sent = sendto(sock, recvBuf, recvLen, 0,

                              (struct sockaddr \*)&clientAddr, clientAddrLen);

            if (sent == SOCKET\_ERROR) {

                printf("sendto failed: %d\n", WSAGetLastError());

                break;

            }

        }

    }

    closesocket(sock);

    WSACleanup();

    return 0;

}

client-udp.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdint.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#pragma comment(lib, "ws2\_32.lib")

int main(void) {

    WSADATA wsaData;

    int iResult;

    iResult = WSAStartup(MAKEWORD(2,2), &wsaData);

    if (iResult != 0) {

        printf("WSAStartup failed: %d\n", iResult);

        return 1;

    }

    SOCKET sock = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

    if (sock == INVALID\_SOCKET) {

        printf("Socket creation failed: %d\n", WSAGetLastError());

        WSACleanup();

        return 1;

    }

    struct sockaddr\_in serverAddr;

    memset(&serverAddr, 0, sizeof(serverAddr));

    serverAddr.sin\_family = AF\_INET;

    serverAddr.sin\_port = htons(8001);

    serverAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    uint32\_t msg1\_values[5] = {1, 2, 3, 4, 5};

    char message1b[20];

    for (int i = 0; i < 5; i++) {

        uint32\_t val = htonl(msg1\_values[i]);

        memcpy(message1b + i \* sizeof(uint32\_t), &val, sizeof(uint32\_t));

    }

    int message1Size = sizeof(message1b);

    printf("Sending message1 (size %d bytes)\n", message1Size);

    int sent = sendto(sock, message1b, message1Size, 0, (struct sockaddr \*)&serverAddr, sizeof(serverAddr));

    if (sent == SOCKET\_ERROR) {

        printf("sendto failed: %d\n", WSAGetLastError());

        closesocket(sock);

        WSACleanup();

        return 1;

    }

    char recvbuf[20];

    int addrLen = sizeof(serverAddr);

    printf("Waiting to receive...\n");

    int recvLen = recvfrom(sock, recvbuf, sizeof(recvbuf), 0, (struct sockaddr \*)&serverAddr, &addrLen);

    if (recvLen == SOCKET\_ERROR) {

        printf("recvfrom failed: %d\n", WSAGetLastError());

        closesocket(sock);

        WSACleanup();

        return 1;

    }

    printf("Received %d bytes\n", recvLen);

    uint64\_t msg2\_values[2] = {10000000000ULL, 20000000000ULL};

    char message2b[16];

    for (int i = 0; i < 2; i++) {

        uint64\_t val = htonll(msg2\_values[i]);

        memcpy(message2b + i \* sizeof(uint64\_t), &val, sizeof(uint64\_t));

    }

    int message2Size = sizeof(message2b);

    printf("Sending message2 (size %d bytes)\n", message2Size);

    sent = sendto(sock, message2b, message2Size, 0, (struct sockaddr \*)&serverAddr, sizeof(serverAddr));

    if (sent == SOCKET\_ERROR) {

        printf("sendto failed: %d\n", WSAGetLastError());

        closesocket(sock);

        WSACleanup();

        return 1;

    }

    printf("Waiting to receive...\n");

    recvLen = recvfrom(sock, recvbuf, sizeof(recvbuf), 0, (struct sockaddr \*)&serverAddr, &addrLen);

    if (recvLen == SOCKET\_ERROR) {

        printf("recvfrom failed: %d\n", WSAGetLastError());

        closesocket(sock);

        WSACleanup();

        return 1;

    }

    printf("Received %d bytes\n", recvLen);

    printf("Closing socket\n");

    closesocket(sock);

    WSACleanup();

    return 0;

}

Результат роботи tcp клієнта та сервера:

A black screen with white text

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

Рисунок 1 — Результат виконання TCP клієнту і серверу на однаковій машині.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Рисунок 2 — Результат виконання UDP клієнту і серверу на однаковій машині.

**ВИСНОВКИ**

У межах виконання практичної роботи було досліджено основи мережевої взаємодії на прикладі транспортних протоколів TCP та UDP, а також реалізовано клієнт-серверні додатки для передачі даних цими протоколами.

Було детально проаналізовано роботу основних функцій API сокетів, зокрема socket, bind, listen, accept, connect, send, recv, sendto, recvfrom, closesocket та інших. Проведені експерименти продемонстрували ключові відмінності між TCP та UDP.

**TCP** забезпечує надійність передачі даних завдяки механізму встановлення з'єднання (handshake), гарантує порядок доставки пакетів і компенсує можливі втрати шляхом повторної передачі. **UDP**, натомість, працює без встановлення з'єднання, що робить його швидшим, але менш надійним, оскільки втрати пакетів не компенсуються.

Тестові запуски програмного забезпечення були виконані на одній машині. Отримані результати підтверджують успішну реалізацію як TCP, так і UDP клієнтів і серверів. Це дозволило не лише засвоїти принципи роботи з мережевими протоколами, а й значно розширити практичні навички роботи з сокетами.