Звіт

Із виконання лабораторної роботи № 2 на тему:

**«Взаємодія розподілених процесів через механізм сокетів»**

**«Синхронізація множин чисел»**

Виконав студент ІІ курсу

факультету комп’ютерних наук та кібернетики

Київського національного університету імені Тараса Шевченка

Групи К-27

**Шевченко Максим**

Викладач: Бондарчук Юрій Васильович

Викладач практичних занять:

Махно Михайло Федорович

Варіант № 30:

Синхронізація множин чисел

Суть задачі полягає у тому, що студент повинен реалізувати дві програми: програма-клієнт та програма-сервер за допомогою механізму сокетів.

Сокети, які називаються Windows Sockets або API Winsock є програмним інтерфейсом до протоколу TCP який забезпечує дотримування послідовності повідомлень, надійне з’єднання. Та протоколу UDP який є лише простою дейтаграмною розсилкою та ненадійним з’єднанням. Вони базуються на Berkley Sockets API, тому є практично у всіх ОС.

Відмінності прослідковуються лише на рівні розширень. Технологія сокетів дуже схожа на роботу із послідовними файлами, або для Windows – іменованими каналами: Сокет потрібно відкрити, зчитувани (прослуховувати ) з нього, писати і в кінці роботи закрити.

Що передається через сокет знає лише програма. Для інтерфейсу сокетів це лише група байт(повідомлення), і яка його структура, призначення тощо має бути передбачена протоколом взаємодіючих процесів.

Механізм сокетів, як взаємодію мінімум двох мережевих процесів, зручно описувати в термінології клієнт/сервер. Так програма (хост), яка ініціює сесію з іншою програмою – сервером, виступає клієнтом, і мета взаємодії визначається клієнтом через його запити до програми-сервера.

**Опис виконання задачі**

Для виконання задачі перш за все була встановлена середовище програмування Microsoft Visual Studio 2017 як найбільш оптимізоване середовище для роботи із сокетами на мові програмування С++.

Були ініціалізовані наступні бібліотеки:

#include <iostream>

Стандартна бібліотека С++, надає можливість використовувати базові функції введення та виведення значень у консоль.

#include "pch.h"

Службовий .h файл середовища програмування, має нічого всередині себе.

#include <WinSock2.h>

Бібліотека сокетів яка надає можливість ініціалізувати їх та маніпулювати з ними(прослуховувати, приймати та відсилати дані, тощо)

#include <fstream>

Бібліотека роботи із файлами. В даному випадку використовується для створення текстового файлу у якому ведеться логування усіх операції, здійснених сервером та клієнтом.

#include <time.h>

Дана бібліотека була використана для того щоб вводити у файл поточний час змін.

#include <time.h>

Дана бібліотека використана для виводу рядків на консоль.

Також були ініціалізовані службові дані для роботи із сокетами:

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

Дана строка надає можливість нехтування помилок та використання незахищенних операцій копіювання (str\_cpy).

#define DEFAULT\_PROTOCOL 0

Визначає стандартний протокол

#define MAX 255

Визначає максимальну кількість байтів які можна передавать між сервером та клієнтом.

Код сервера має наступний вигляд:

int main()

{

WSADATA wsaData;

string serverSet = "2 3 3 5 9 55 66 ";

ofstream f;

f.open("server\_log.txt");

time\_t rawtime;

struct tm \* timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo) << "Server START\n\n";

//ініціалізація інтерфейсу

int iResult = WSAStartup(0x0202, &wsaData);

if (iResult != 0)

{

cout << "ERROR at WSAStartup\n";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "ERROR at WSAStartup\n\n";

f.close();

return 1;

}

else

{

cout << "WSAData created\n";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "WSAData created\n\n";

}

//створення сокету

SOCKET serverSocket;

serverSocket = socket(PF\_INET, SOCK\_STREAM, DEFAULT\_PROTOCOL);

if (serverSocket == 0)

{

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "ERROR: Socket is not created\n\n";

cout << "ERROR: Socket is not created\n";

f.close();

return 1;

}

else

{

cout << "Socket is created\n";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Socket is created\n\n";

}

//встановлення асоціації локальної адреси із сокетом

sockaddr\_in service;

service.sin\_family = PF\_INET;

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

service.sin\_port = htons(1055);

iResult = bind(serverSocket, (SOCKADDR\*)&service, sizeof(service));

if (iResult != 0)

{

cout << "ERROR: bind() failed\n";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "ERROR: bind() failed\n\n";

f.close();

return 1;

}

else

{

cout << "Socket is binded\n";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Socket is binded\n\n";

}

//прослуховування порта на з'єднання

iResult = listen(serverSocket, 1);

if (iResult != 0)

{

cout << "ERROR listening on socket\n";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "ERROR listening on socket\n\n";

f.close();

return 1;

}

else

{

cout << "Listening on socket\n";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Listening on socket\n\n";

}

//вибір клієнтів із черги на приєднання до сервера

SOCKET acceptSocet;

cout << "Waiting for client to connect\n";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Waiting for client to connect\n\n";

acceptSocet = SOCKET\_ERROR;

while (acceptSocet == SOCKET\_ERROR)

{

acceptSocet = accept(serverSocket, NULL, NULL);

}

cout << "Client connected\n";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Client connected\n\n";

serverSocket = acceptSocet;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//робота з клієнтом

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_

int iBytesSent;

int iBytesRecv;

char recvbuf[MAX];

string s;

while (1)

{

iBytesRecv = SOCKET\_ERROR;

while (iBytesRecv == SOCKET\_ERROR)

{

iBytesRecv = recv(serverSocket, recvbuf, MAX, 0);

if (iBytesRecv != SOCKET\_ERROR)

{

cout << "\nReceived " << iBytesRecv << " bytes" << endl;

cout << recvbuf << endl;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Received " << iBytesRecv << " bytes" << endl << recvbuf << endl << endl;

break;

}

}

s = string(recvbuf);

if (s == "Who")

{

char sendbuf1[MAX] = "Maksym Shevcenko K-27 || 30. Synchronization of sets of numbers";

iBytesSent = send(serverSocket, sendbuf1, MAX, 0);

cout << "\nServer sent " << iBytesSent << " bytes" << endl;

cout << sendbuf1 << endl;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Server sent " << iBytesSent << " bytes" << endl << sendbuf1 << endl << endl;

continue;

}

if (s == "Synchronization")

{

char sendbuf2[MAX] = "Server is ready for Synchronization";

iBytesSent = send(serverSocket, sendbuf2, MAX, 0);

cout << "\nServer sent " << iBytesSent << " bytes" << endl;

cout << sendbuf2 << endl;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Server sent " << iBytesSent << " bytes" << endl << sendbuf2 << endl << endl;

cout << "\nserverSet before Synchronization = " << serverSet << endl;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "serverSet before Synchronization = " << serverSet << endl << endl;

iBytesRecv = SOCKET\_ERROR;

while (iBytesRecv == SOCKET\_ERROR)

{

iBytesRecv = recv(serverSocket, recvbuf, MAX, 0);

if (iBytesRecv != SOCKET\_ERROR)

{

cout << "\nReceived " << iBytesRecv << " bytes" << endl;

cout << recvbuf << endl;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Received " << iBytesRecv << " bytes" << endl << recvbuf << endl << endl;

break;

}

}

s = string(recvbuf);

string sets;

string buf;

while (serverSet.find(' ') != string::npos)

{

buf.clear();

buf = serverSet.substr(0, serverSet.find(' ') + 1);

if (s.find(buf) != string::npos)

{

s.erase(s.find(buf), buf.length());

}

sets += buf;

serverSet.erase(0, serverSet.find(' ') + 1);

}

s += sets;

serverSet = s;

cout << "\nserverSet after Synchronization = " << serverSet << endl;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "serverSet after Synchronization = " << serverSet << endl << endl;

char sendbuf3[MAX];

strcpy(sendbuf3, serverSet.c\_str());

iBytesSent = send(serverSocket, sendbuf3, MAX, 0);

cout << "\nServer sent " << iBytesSent << " bytes" << endl;

cout << sendbuf3 << endl;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Server sent " << iBytesSent << " bytes" << endl << sendbuf3 << endl << endl;

continue;

}

if (s == "End of work")

{

char sendbuf1[MAX] = "Server EoW";

iBytesSent = send(serverSocket, sendbuf1, MAX, 0);

cout << "\nServer sent " << iBytesSent << " bytes" << endl;

cout << sendbuf1 << endl;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Server sent " << iBytesSent << " bytes" << endl << sendbuf1 << endl << endl;

break;

}

char sendbuf4[MAX] = "Wrong command";

iBytesSent = send(serverSocket, sendbuf4, MAX, 0);

cout << "\nServer sent " << iBytesSent << " bytes" << endl;

cout << sendbuf4 << endl;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Server sent " << iBytesSent << " bytes" << endl << sendbuf4 << endl << endl;

}

if (closesocket(serverSocket) != 0)

{

cout << "ERROR: socket is not closed";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "ERROR: socket is not closed";

return 1;

}

WSACleanup();

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo) << "Server END\n\n";

f.close();

return 0;

}

Код клієнта має наступний вигляд:

int main()

{

WSADATA wsaData;

string clientSet = "1 2 2 3 3 3 4 5 6 7 ";

ofstream f;

f.open("client\_log.txt");

time\_t rawtime;

struct tm \* timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo)<<"Client START\n\n";

//ініціалізація інтерфейсу

int iResult = WSAStartup(0x0202, &wsaData);

if (iResult != 0)

{

cout << "ERROR at WSAStartup\n";

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "ERROR at WSAStartup\n\n";

f.close();

return 1;

}

else

{

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "WSAData created\n\n";

cout << "WSAData created\n";

}

//створення сокету

SOCKET clientSocket;

clientSocket = socket(PF\_INET, SOCK\_STREAM, DEFAULT\_PROTOCOL);

if (clientSocket == 0)

{

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "ERROR: Socket is not created\n\n";

cout << "ERROR: Socket is not created\n";

f.close();

return 1;

}

else

{

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Socket is created\n\n";

cout << "Socket is created\n";

}

//під'єднання до сервера

sockaddr\_in clientService;

clientService.sin\_family = PF\_INET;

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

clientService.sin\_port = htons(1055);

iResult = connect(clientSocket, (SOCKADDR\*) &clientService, sizeof(clientService));

if (iResult != 0)

{

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "connection ERROR\n\n";

cout << "connection ERROR\n";

f.close();

return 1;

}

else

{

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Socket is connected\n\n";

cout << "Socket is connected\n";

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_

//робота із сервером

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

int iBytesSent;

int iBytesRecv;

char recvbuf[MAX];

char sendbuf[MAX];

string str;

while (1)

{

cout << '>';

getline(cin, str);

strcpy(sendbuf, str.c\_str());

//++++++++++++++++++++++++++++++++++++++++

iBytesSent = send(clientSocket, sendbuf, MAX, 0);

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Client sent " << iBytesSent << " bytes" << endl << sendbuf << endl << endl;

cout << "Client sent " << iBytesSent << " bytes" << endl;

iBytesRecv = SOCKET\_ERROR;

while (iBytesRecv == SOCKET\_ERROR)

{

iBytesRecv = recv(clientSocket, recvbuf, MAX, 0);

if (iBytesRecv != SOCKET\_ERROR)

{

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Received " << iBytesSent << " bytes" << endl << recvbuf << endl << endl;

cout << "\nReceived " << iBytesRecv << " bytes" << endl;

cout << recvbuf << endl;

break;

}

}

//Synchronization

if (str == "Synchronization")

{

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "clientSet before Synchronization = " << clientSet << endl << endl;

cout << "\nclientSet before Synchronization = " << clientSet << endl;

char sendbuf3[MAX];

strcpy(sendbuf3, clientSet.c\_str());

iBytesSent = send(clientSocket, sendbuf3, MAX, 0);

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Client sent " << iBytesSent << " bytes" << endl << sendbuf3 << endl << endl;

cout << "\nClient sent " << iBytesSent << " bytes" << endl;

cout << sendbuf3 << endl;

iBytesRecv = SOCKET\_ERROR;

while (iBytesRecv == SOCKET\_ERROR)

{

iBytesRecv = recv(clientSocket, recvbuf, MAX, 0);

if (iBytesRecv != SOCKET\_ERROR)

{

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "Received " << iBytesSent << " bytes" << endl << recvbuf << endl << endl;

cout << "\nReceived " << iBytesRecv << " bytes" << endl;

cout << recvbuf << endl;

clientSet = recvbuf;

break;

}

}

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "clientSet after Synchronization = " << clientSet << endl << endl;

cout << "\nclientSet after Synchronization = " << clientSet << endl;

}

if (str == "End of work")

{

break;

}

}

if (closesocket(clientSocket) != 0)

{

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo);

f << "ERROR: socket is not closed";

cout << "ERROR: socket is not closed";

f.close();

return 1;

}

WSACleanup();

time(&rawtime);

timeinfo = localtime(&rawtime);

f << asctime(timeinfo) << "Client END\n\n";

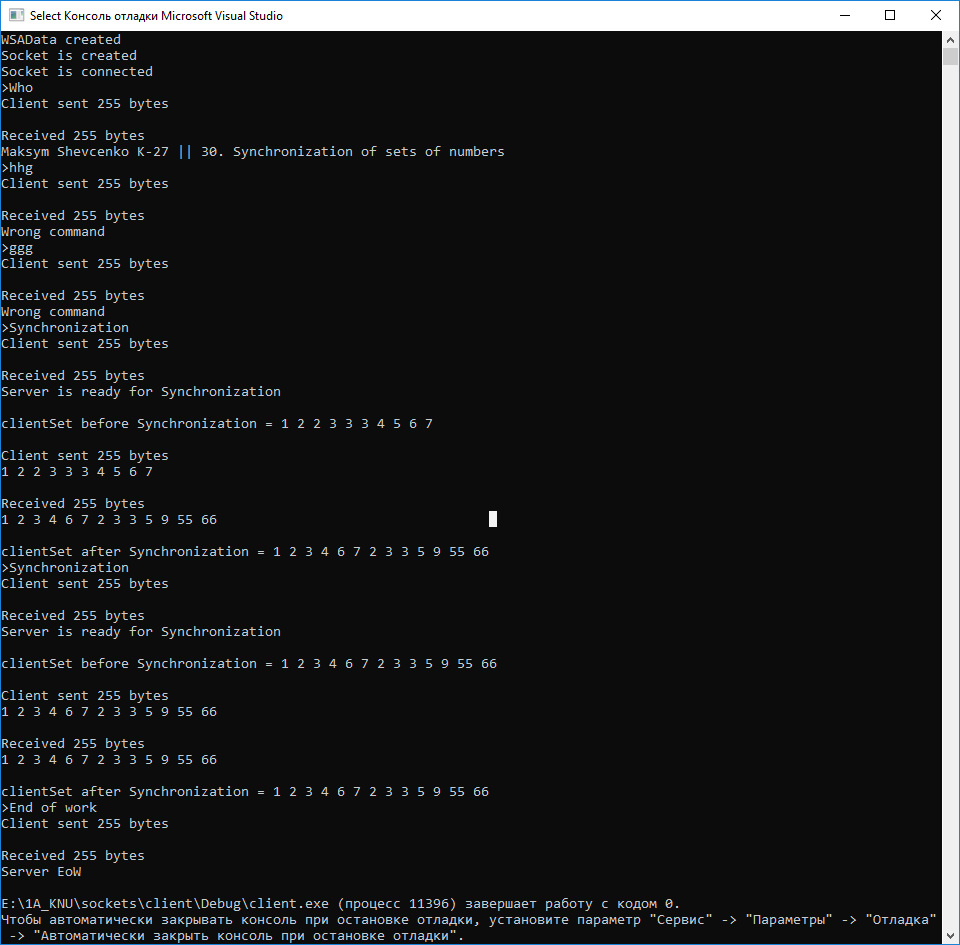
f.close();

return 0;

}

**Приклад виконання програми**

Клієнт:



Сервер:

