МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТ імені ВОЛОДИМИРА ДАЛЯ

Факультет інформаційних технологій та електроніки Кафедра комп’ютерних наук та інженерії Освітньо-кваліфікаційний рівень бакалавр Спеціальність 123 «Комп'ютерна інженерія»

(шифр і назва)

КУРСОВИЙ ПРОЕКТ (РОБОТА)

З дисципліни: Системне програмне забезпечення (назва дисципліни)

на тему: Розроблення програмних додатків за технологією «клієнт-сервер». Метод комунікації : поштова скринька.

здобувача вищої освіти 3 курсу , групи КІ-19д

(прізвище та ініціали) (підпис)

Керівник проекту

доц.Деркач М.В.

(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Результати захисту:

Національна шкала Кількість балів: Оцінка: ECTS

Члени комісії:

(прізвище та ініціали) (підпис)

(прізвище та ініціали) (підпис)

(прізвище та ініціали) (підпис)

Дата

Сєвєродонецьк – 2021

Східноукраїнський національний університет імені Володимира Даля

Факультет ІТЕ Кафедра КНІ Спеціальність 123 Комп’ютерна інженерія

ЗАТВЕРДЖУЮ:

Зав. кафедри

КНІ

І.С.Скарга-Бандурова

« » 20 р.

**З А В Д А Н Н Я**

**НА КУРСОВУ РОБОТУ**

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розроблення програмних додатків за технологією

«клієнт-сервер». Метод комунікації : поштова скринька.

затверджена наказом по інституту від « » 2020 р. №

1. Термін подання закінченого проекту (роботи) 17 тиждень
2. Початкові дані до проекту (роботи) Розробити окремі додатки: додаток- сервер і два додатки-клієнти. Функції додатку-серверу :а) створення поштової скриньки; б) отримання даних від додатків-клієнтів; в) вивід на екран отриманих даних. Функції додатку-клієнту 1: а) відкрити поштову скриньку; б)підготувати і передати серверу такі дані: ім'я комп’ютера, ім'я користувача, версію операційної систем. Функції додатку-клієнту 2: а) відкрити поштову скриньку; б)підготувати і передати серверу такі дані: відсоток фізичної пам’яті, що використовується; відсоток віртуальної пам'яті, що використовується. За необхідності синхронізації процесів вибраний засіб синхронізації обґрунтувати.
3. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробленню) структура та оформлення пояснювальної записки повинна відповідати вимогам методичних вказівок до курсової роботи і ДСТУ. Основна частина записки повинна вміщувати: постановку задачі; короткі теоретичні відомості з теми курсової роботи (сутність, область застосування, переваги и недоліки архітектури «клієнт-сервер»; огляд методу комунікації через поштову скриньку); розробку та опис алгоритму роботи додатків; результати застосування додатків у вигляді копій екрану. У додатку до записки навести роздруківку програмних додатків.
4. Перелік графічного матеріалу (з точною вказівкою обов'язкових креслень) графічні матеріали до курсової роботи не вимагаються
5. Дата видачі завдання 4 тиждень

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів курсового проектування | Строк виконання  етапів | Примітка |
| 1 | Підготовчий етап. Уточнення постановки задачі. Аналітичний огляд науково-технічної  літератури | 24.09.2020 –  27.09.2020 |  |
| 2 | Проектний етап. Розроблення алгоритмів роботи програми. Обґрунтування необхідності і  вибір засобів синхронізації | 28.09.2020 –  09.10.2020 |  |
| 3 | Реалізаційний етап. Реалізація розроблених алгоритмів засобами вибраної мови  програмування, остаточне налагодження | 10.10.2020 –  30.11.2020 |  |
| 4 | Оформлювальний етап. Оформлення  пояснювальної записки відповідно до вимог методичних вказівок і ДСТУ | 01.12.2020 –  14.12.2020 |  |
| 5 | Заключний етап. Захист курсової роботи. | до  24.12.2020 |  |

Здобувач вищої освіти

( підпис ) (прізвище та ініціали)

Керівник проекту (роботи) Деркач М.В.

**РЕФЕРАТ**

Пояснювальна записка до курсової роботи містить: 30 сторінок, 5 рисунків, 3 додатка, 5 джерел.

**Мета роботи:** розробка програм на платформі Win32 для дослідження взаємодії між процесами через метод поштової скриньки.

**Об’єкт розробки** – програмне забезпечення, що містить програму виконуючу функції сервера і дві програми, виконуючи функції клієнтів. Розробка програм виконується на платформі Win32 у середовищі Microsoft Visual Studio, з використанням для між процесорної взаємодії метод поштової скриньки (mailslots).

Архітектура «клієнт-сервер» застосовується для організації обміну інформації між двома об’єктами.

У результаті розробки курсової роботи було визначено переваги та недоліки архітектури «клієнт-сервер»; зроблено огляд системних засобів комунікації і синхронізації процесів. Розроблено графічний інтерфейс користувача. Налагоджена програма.

WIN32 API, MAILSLOTS, ПОШТОВА СКРИНЬКА, МЬЮТЕКС, БУФЕР, ПРОЦЕСИ, МІЖПРОЦЕСОРНІ ВЗАЄМОДІЇ, АЛГОРИТМ, ПРОГРАМА.

# ЗМІСТ

[**ВСТУП 5**](#_Toc73948352)

[**1.** **АНАЛІЗ І ПОСТАНОВКА ЗАДАЧІ 6**](#_Toc73948353)

[1.1 Визначення ООП 6](#_Toc73948354)

[1.2 Основні поняття Об'єктно-Орієнтованого Програмування 6](#_Toc73948355)

[1.3 Регулярні вирази 12](#_Toc73948356)

[1.4 Технічне завдання на розробку 13](#_Toc73948357)

[**2.** **ПОБУДОВА ТА РЕАЛІЗАЦІЯ ПРОГРАМИ 14**](#_Toc73948358)

[2.1 Вибір засобів розробки 14](#_Toc73948359)

[2.2 Реалізація базового класу “TextStats” 15](#_Toc73948360)

[2.3 Реалізація похідного класу “Csymbol” 15](#_Toc73948361)

[2.4 Реалізація похідного класу “CsymbolNonSpace” 16](#_Toc73948362)

[2.5 Реалізація похідного класу “Cletter” 16](#_Toc73948363)

[2.6 Реалізація похідного класу “Cwords” 17](#_Toc73948364)

[2.7 Реалізація похідного класу “Csentence” 18](#_Toc73948365)

[2.8 Реалізація похідного класу “Cnumbers” 18](#_Toc73948366)

[2.9 Реалізація похідного класу “CtheNumbers” 19](#_Toc73948367)

[2.10 Реалізація класу “CText” 19](#_Toc73948368)

[**2.10.1 Реалізація конструктору класу** 20](#_Toc73948369)

[**2.10.2 Реалізація деструктору** 20](#_Toc73948370)

[**2.10.3 Реалізація функції “loadFile”** 20](#_Toc73948371)

[**2.10.4 Реалізація функції “calculate”** 21](#_Toc73948372)

[**2.10.5 Реалізація функції “print”** 22](#_Toc73948373)

[2.11 Реалізація головної функції програми 22](#_Toc73948374)

[2.12 Приклади результатів 22](#_Toc73948375)

[ВИСНОВКИ 24](#_Toc73948376)

[**ВИКОРИСТАНІ ДЖЕРЕЛА: 25**](#_Toc73948377)

[**Додаток А 26**](#_Toc73948378)

**ВСТУП**

Зі зростанням розвитку технологій та поширеності Інтернету робота з віддаленою інформацією стала звичним явищем. Технологія клієнт-сервер широко застосовується під час роботи з віддаленою інформацією в мережі. Термін «клієнт-сервер» застосовується до архітектури програмного забезпечення, що описує розподіл процесу виконання за принципом взаємодії двох програмних процесів, один із яких у цій моделі називався «клієнтом», а інший – «сервером». Ця модель утворює єдину систему, що забезпечує розподілені обчислення, аналіз та представлення даних. Клієнт відправляє запит на сервер, де він обробляється, і готовий результат зворотно відправляється клієнтові. У можливості сервера також входить одночасне обслуговування кількох клієнтів.

Клієнт-серверну архітектуру можна визначити, як концепцію інформаційної мережі в якій основна частина її ресурсів зосереджена на серверах, які обслуговують своїх клієнтів. Така архітектура визначає такі компоненти:

* **Клієнт**,який виконує надсилання запиту на сервер для можливості надання даних або виконання певної групи дій;
* **Сервер**, який виконує роботу за клієнтськими запитами, вирішує певного кола завдань та надає користувачам доступ до певних системних ресурсів;
* **Мережа**, яка забезпечує взаємодію між клієнтами та серверами.

Зазвичай клієнт та сервер розташовані на різних обчислювальних машинах і взаємодіють між собою через обчислювальну мережу за допомогою мережевих протоколів, але можуть бути розташовані також і на одній машині.

Архітектура клієнт-сервер визначає лише загальні принципи взаємодії між комп'ютерами, деталі взаємодії визначають різні протоколи. Дана концепція розділяє на клієнтські пристрої, яким завжди щось треба і серверні, які дають те, що треба.

В результаті виконання даної курсової роботи необхідно:

* визначити метод налагодження міжпроцесової взаємодії, та розробити алгоритми побудови додатків сервера і клієнтів;
* розробити та налагодити сервер та клієнти;
* провести аналіз роботи розроблених програм.

**ТЕОРЕТИЧНА** **ЧАСТИНА**

**Клієнт**-**серверна** **модель**

Поділ додатка на окремі задачі, що розміщуються на різних платформах для більшої ефективності як правило означає, що програма представлення даних знаходиться на машині користувача (на клієнті), а програма управління даними та самі дані - на сервері. Залежно від програми та програмного забезпечення вся обробка даних може здійснюватися на клієнтській машині або розподілятися між клієнтом та сервером. Сервер з'єднується зі своїми клієнтами через мережу. Серверне програмне забезпечення приймає запити від клієнтського програмного забезпечення та повертає йому результати (Рис. 1.1).



Рисунок 1.1 - "Клієнт-серверна модель"

Існують концепції побудови системи клієнт-сервер:

* Слабкий клієнт – потужний сервер. Вся обробка інформації перенесена на сервер, а у клієнта права доступу обмежені. Сервер відправляє відповідь, яка не вимагає додаткової обробки. Клієнт взаємодіє з користувачем, складає та відправляє запит, приймає результат і виводить інформацію на екран.
* Потужний клієнт – концепція, в якій частина обробки інформації надається клієнтові. У такому випадку сервер виступає сховищем даних, а вся робота по обробці та подання інформації переноситься на комп'ютер клієнта.

У будь-якому додатку з використанням клієнт-серверної моделі виділяються такі логічні компоненти:

* Рівень уявлення (інтерфейс користувача);
* Рівень бізнес-логіки (обробка);
* Рівень даних;
* Службові функції.

Рівень інтерфейсу користувача містить все необхідне для безпосереднього спілкування з користувачем. Рівень бізнес-логіки зазвичай містить програми, а рівень даних - власне дані, з якими відбувається робота. Службові функції здійснюють зв'язок між функціями перших трьох груп.

Перевагами цієї моделі є:

* відсутність дублювання коду програми-сервера програмами-клієнтами;
* коли обчислення виконуються на сервері, вимоги до комп'ютерів, де встановлено клієнт знижаються;
* на сервері простіше організувати контроль повноважень, що дозволить доступ до даних лише клієнтам із відповідними правами доступу;
* всі дані зберігаються на сервері, який захищений набагато краще;
* зменшення навантаження на мережу з огляду на те, що клієнт в основному передає серверу команди, а той уже їх виконує.

Недоліками є:

* високе навантаження на серверне обладнання та канал зв'язку до нього;
* непрацездатність сервера може зробити непрацездатною всю обчислювальну мережу;
* висока вартість серверного обладнання та його обслуговування (може знадобитись спеціаліст для налаштування).

Як і будь-яка інша технологія, клієнт-серверна архітектура має свої переваги та недоліки. Звідси можна з'ясувати, що ця архітектура має ряд істотних переваг у порівнянні з традиційною архітектурою інформаційних систем. Вища продуктивність, покращення безпеки та цілісності даних, можливість завдання бізнес-правил, також, існують можливості вдосконалення клієнт-серверних систем шляхом переходу до багатоланкової архітектури.

**Міжпроцесова взаємодія**

Операційна система Windows надає механізми для полегшення зв'язку та обміну даними між програмами. У сукупності дії, що здійснюються за допомогою цих механізмів, називаються міжпроцесними комунікаціями (IPC). Деякі форми IPC полегшують розподіл праці між кількома спеціалізованими процесами. Інші форми IPC полегшують розподіл праці між декількома комп'ютерами в мережі. Загалом, IPC - обмін даними між потоками одного або різних процесів. Реалізується за допомогою механізмів, що надаються ядром ОС або процесом, що використовують механізми ОС і реалізують нові можливості IPC.

ОС Windows підтримує такі механізми IPC:

* буфер обмена Clipboard;
* модель компонентних об'єктів COM;
* копіювання даних Data;
* динамічний обмін даними DDE;
* File Mapping – файли, відображенні в пам'яті;
* поштові скриньки Mailslots;
* канали Pipes;
* RPC – виклик віддаленої процедури;
* сокети Windows.

Оскільки кожен окремий запит користувача може призвести до запуску кількох процесів в операційній системі, процес може вимагати зв’язку один з одним. Кожен підхід протоколу IPC має свої переваги та обмеження, тому одна програма може використовувати всі методи IPC.

**Метод Windows Sockets**

Windows sockets або WinSock - це інтерфейс програмного програмування (API), створений для реалізації додатків у мережі на основі протоколу TCP/IP. З'єднуючи разом два сокети, можна передавати дані між різними процесами (локальними чи віддаленими). Реалізація сокетів забезпечує інкапсуляцію протоколів мережного та транспортного рівнів.

Windows sockets беруть свій початок від сокетів, які вперше запропоновані Berkeley Software Distribution (BSD – програмне забезпечення Каліфорнійського університету). Програма, яка використовує сокети Windows, може взаємодіяти із сокетами інших ОС.

Усі ресурси, відкритого зв'язку, в UNIX і Windows ідентифікуються дескрипторами. Ці дескриптори, або описувачі (handles), можуть вказувати на файл, пам'ять або на будь-який інший канал зв'язку, а фактично вказують на внутрішню структуру даних, що використовується операційною системою. Сокет, будучи таким самим ресурсом, теж представляється дескриптором. Отже, для сокетів життєвий цикл дескриптора можна поділити на три фази: відкриття (створення) сокету, отримання із сокету або відправлення сокету і зрештою закриття сокету. Інтерфейс IPC для взаємодії між різними процесами побудований поверх методів введення-виводу. Вони полегшують для сокетів відправлення та отримання даних. Кожен цільовий об'єкт задається адресою сокета, отже, цю адресу можна вказати в клієнті, щоб встановити з'єднання з метою. Сервер, зі свого боку, прослуховує порт із заданим номером і створює для цього серверний сокет. Коли клієнт запитує з'єднання з сервером, сервер може дозволити йому і створити новий сокет, який буде кінцевою точкою зв'язку. Завдяки цьому, сокет, за яким відбувалося прослуховування, не використовується для передачі даних і може перебувати в режимі прослуховування далі, підключаючи нових клієнтів.

Існують два основних типи сокетів - потокові сокети та дейтаграмні. Потоковий сокет - це сокет із встановленим з'єднанням, що складається з потоку байтів, який може бути двонаправленим, тобто через цю кінцеву точку додаток може і передавати, і отримувати дані. Дейтаграмні сокети – сокети без організації з'єднань, тобто ніякого явного з'єднання між ними не встановлюється повідомлення надсилається зазначеному сокету і, відповідно, може виходити від зазначеного сокета. Потокові сокети в порівнянні з дейтаграмними дійсно дають більш надійний метод, але для скорочення накладних витрат слід використовувати дейтаграмні сокети.

В дійсності, сокет можна представляти як конкатенацію IP-адреси і порту. Оскільки IP-адреса унікальна в Інтернеті, а номери портів унікальні на окремій машині, номери сокетів також унікальні у всьому Інтернеті. Ця характеристика дозволяє процесу спілкуватися через мережу з іншим процесом лише на підставі номера сокету. Спосіб використання сокетів залежить від того, де вони використовуються: на клієнтської або серверної частини. Клієнтська частина створює з'єднання шляхом створення сокету і викликом з'єднує функції з певною адресною інформацією. До того як сокет не з'єднається, він не буде пов'язаний з адресою. Коли з'єднання викликано, WinSock вибере IP адресу і номер порту для з'єднання і зв'яже з ними сокет до того, як клієнт фактично з'єднається з сервером.

**ПРАКТИЧНА** **ЧАСТИНА**

**Постановка задачі**

Завданням даної курсової роботи є розроблення програмних додатків за технологією «клієнт-сервер».. Метод комунікації між процесами, який необхідно використовувати при виконанні роботи – сокети.

До складу програмного забезпечення, розробленого відповідно до завдання, повинні входити:

* додаток, що виконує функції сервера;
* два додатки, що виконують функції клієнтів.

Додаток-сервер повинна виконувати наступні функції:

1. створення сокетів;
2. отримання даних від додатків-клієнтів;
3. вивід на екран отриманих даних.

Додаток-клієнт 1 повинна виконувати наступні функції:

1. відкрити сокет;
2. встановити зв'язок с сервером;
3. підготувати і передати серверу такі дані: кількість кнопок у миші; наявність коліщатка прокрутки.

Додаток-клієнт 2 повинна виконувати наступні функції:

1. відкрити сокет;
2. встановити зв'язок с сервером;
3. підготувати і передати серверу такі дані: розмір файлу підкачки у байтах; кількість вільних байтів у файлі підкачки.

**Реалізація додатку-сервера**

При обробці повідомлення WM\_CREATE спочатку створюється текстове поле «EDIT» із встановленими параметрами:

case WM\_CREATE: {

hwndEdit = CreateWindow( // Створюємо доч. вікно для виведення даних від процесів

TEXT("EDIT"), NULL,

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL |

ES\_LEFT | ES\_MULTILINE | ES\_AUTOVSCROLL,

0, 0, 330, 518, hWnd, NULL, hInst, NULL);

**Реалізація функції “ServerStart”**

Алгоритм запуску сервера представлений на рисунку 2.1:

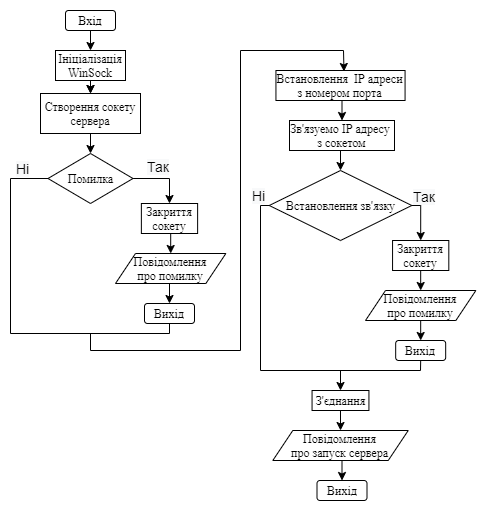


Рисунок 2.1 - Алгоритм запуску сервера.

**Реалізація функцій підключення клієнтів**

Функція “AcceptClient”, для підключення клієнту до серверу:

BOOL AcceptClient(int j)

{

int sockaddr\_len = sizeof(sockaddr[j]);

sock[j] = accept(srv\_socket, (LPSOCKADDR)&sockaddr[j], (int FAR\*)&sockaddr\_len);

if(sock[j]!= INVALID\_SOCKET)

if(!WSAAsyncSelect(sock[j], hWindow, WSA\_NETEVENT, FD\_READ|FD\_CLOSE))

return TRUE;

closesocket(sock[j]);

return FALSE;

}

Функція “WndProc\_OnWSAAccept”, для підключення двох клієнтів до серверу, та визначення адреси підключення:

void WndProc\_OnWSAAccept(HWND hWnd, LPARAM lParam)

{

//при помилці скасовуємо надходження повідомлень у головне вікно програми

if (WSAGETSELECTERROR(lParam)) {

MessageBoxA(hWnd, "Accept error", "Error", MB\_OK);

WSAAsyncSelect(srv\_socket, hWnd, 0, 0);

return;

}

if (ClientNum == 1) { //Функція підключення клієнту до серверу

MessageBoxA(hWnd,"Number of clients >2\r\n","Connection is invalid", MB\_OK);

return;

}

ClientNum++;

if (!AcceptClient(ClientNum)) {

MessageBoxA(hWnd, "Client connection error", "Error", MB\_OK);

return;

}

//додаємо клієнта

sprintf\_s(szBuf," Added client %i\r\n Address: IP=%s Port=%u\r\n \0",

ClientNum + 1,inet\_ntoa(sockaddr[ClientNum].sin\_addr),

htons(sockaddr[ClientNum].sin\_port));

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)szBuf);

}

**Реалізація функції “SendToClient”**

Алгоритм відправлення даних з сервера до клієнту представлений на рисунку 2.2:

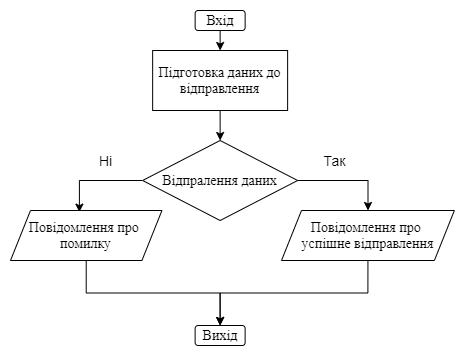


Рисунок 2.2 - Алгоритм відправлення даних до клієнту.

**Реалізація функції “WndProc\_OnWSANetEvent”**

Алгоритм отримання даних з клієнтів до серверу представлений на рисунку 2.3:

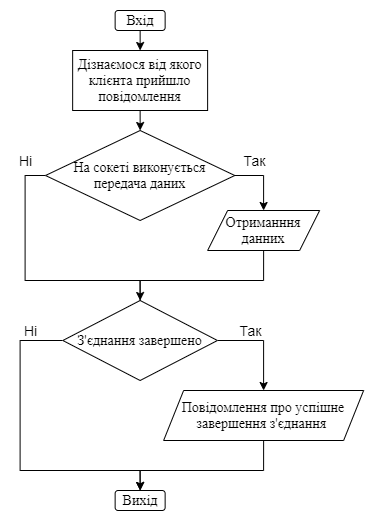


Рисунок 2.2 - Алгоритм отримання даних від клієнту.

**Реалізація функції “ClientOff”**

Функція “ClientOff” відключає клієнтів від сервера:

void ClientOff(HWND hWnd, int j)

{

if(j > ClientNum) return;

sprintf\_s(szBuf, " Disable Client %i?", j + 1);

if(IDYES == MessageBoxA(hWnd, szBuf, "Question", MB\_YESNO | MB\_ICONQUESTION)) {

WSAAsyncSelect(sock[j], hWindow, 0, 0);

closesocket(sock[j]);

return;

}

}

**Реалізація функції “ServerStop”**

Функція “ServerStop” вимикає сервер:

void ServerStop(HWND hWnd)

{

WSAAsyncSelect(srv\_socket, hWnd, 0, 0);

srv\_socket = INVALID\_SOCKET;

closesocket(srv\_socket);

WSACleanup();

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)" Server stopped");

}

**Реалізація додатків-клієнтів**

У додатку клієнта можна виділити наступні етапи:

* формування віконного інтерфейсу;
* підключення клієнта до серверу;
* формування даних клієнтом для відправки;
* відправка повідомлень;

**Реалізація функції отримання інформації першого клієнта**

char MoseWheel[20];

int MouseButtons = GetSystemMetrics(SM\_CMOUSEBUTTONS);

GetSystemMetrics(SM\_MOUSEWHEELPRESENT) ? strcpy(MoseWheel, " is present") : strcpy(MoseWheel, " is missing");

sprintf(szBuf, " The number of buttons in the mouse: %d \r\n Scroll wheel: %s\r\n ", MouseButtons, MoseWheel);

Функція передає серверу такі дані: кількість кнопок у миші; наявність коліщатка прокрутки.

**Реалізація функції отримання інформації другого клієнта**

MEMORYSTATUSEX status;

status.dwLength = sizeof(status);

GlobalMemoryStatusEx(&status);

char buffer[2][\_MAX\_U64TOSTR\_BASE10\_COUNT];

sprintf(buffer[0], "%llu", status.ullTotalPageFile);

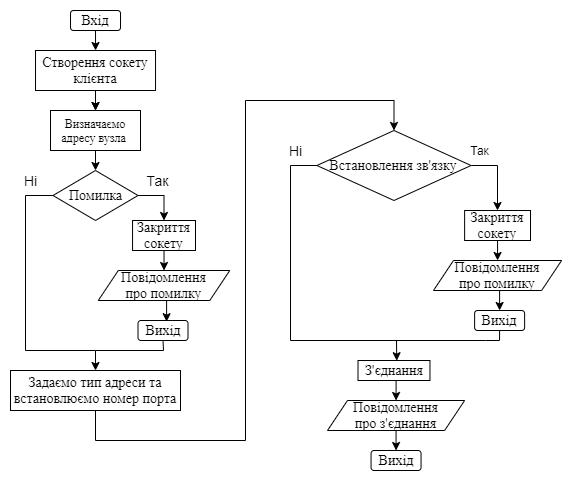
sprintf(buffer[1], "%llu", status.ullAvailPageFile);

sprintf(szBuf, " Pagefile size: %s byte. \r\n Pagefile free size: %s byte.\r\n", buffer[0], buffer[1]);

Функція передає серверу такі дані: розмір файлу підкачки у байтах; кількість вільних байтів у файлі підкачки.

**Реалізація функції встановлення зв’язку клієнта з сервером**

Алгоритм встановлення зв’язку клієнта з сервером представлений на рисунку 2.4:

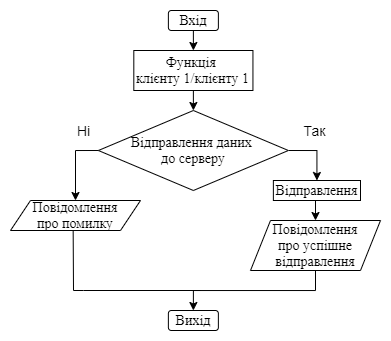


%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20value%3D%22%D0%A2%D0%B0%D0%BA%22%20style%3D%22text%3Bhtml%3D1%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3Bresizable%3D0%3Bpoints%3D%5B%5D%3Bautosize%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22407.06%22%20y%3D%22460%22%20width%3D%2230%22%20height%3D%2220%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3C%2Froot%3E%3C%2FmxGraphModel%

3E Рисунок 2.4 - Алгоритм встановлення зв’язку клієнта з сервером.

**Реалізація функції відправлення даних від клієнту до серверу**

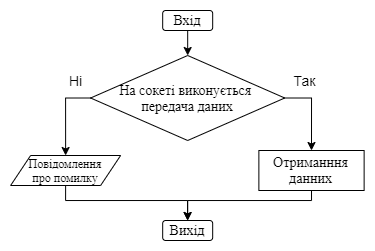
Алгоритм відправлення даних від клієнту до серверу представлений на рисунку 2.5:



3E Рисунок 2.5 - Алгоритм відправлення даних до серверу.

**Реалізація функції отримання даних з клієнтів до серверу**

Алгоритм отримання та вивід даних з клієнтів до серверу представлений на рисунку 2.6:



3E Рисунок 2.6 - Алгоритм отримання даних від серверу.

**Результати роботи програми**

Розробка та налагодження програми виконувалася у середовищі Microsoft Visual Studio. Було розроблено один додаток-сервер і два додатки клієнта, які були перевірені на практиці. В результаті виконання програми від клієнтів 1 і 2 за вибором користувача передаються дані серверу.

Клієнти формують інформацію для подальшої передачі її до серверу. Також є можливість власноруч добавляти інформацію в текстове поле для її передачі.

Робота програми відображена рисунку 2.6.

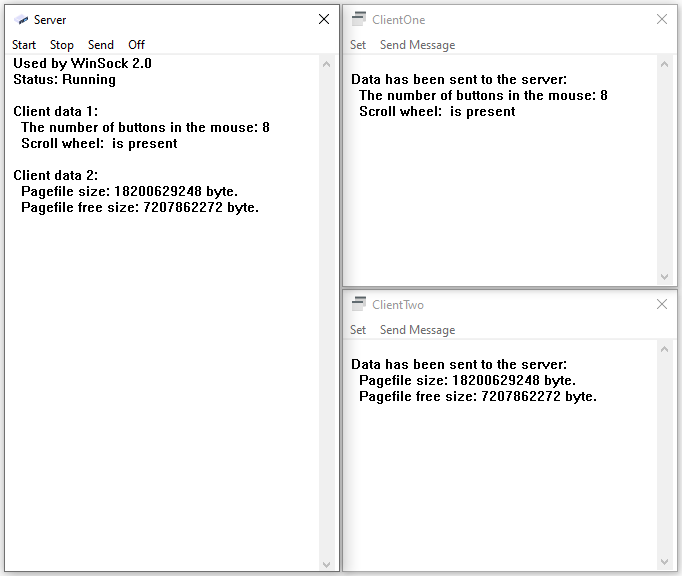
****

Рисунок 2.6 - Скріншот роботи програми.

**ВИСНОВКИ**

ВИКОРИСТАНІ ДЖЕРЕЛА:

1. Регулярні вирази [Електронний ресурс] // slideshare.net. Режим доступу: www. URL: <https://www.slideshare.net/bulgakovdmitriy/c-37604265>
2. Регулярні вирази [Електронний ресурс] // Вікіпедія. Режим доступу: www. URL: <https://ru.wikipedia.org/wiki/Регулярные_выражения>
3. Основи регулярних виразів [Електронний ресурс] // ravesli.com. Режим доступу: www. URL:
4. <https://ravesli.com/regulyarnye-vyrazheniya-osnovy/#toc-7>
5. ООП [Електронний ресурс] // Вікіпедія. Режим доступу: www. URL: <https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование>
6. Основні принципи ООП [Електронний ресурс] // habr.com. Режим доступу: www. URL: <https://habr.com/ru/post/87205/>
7. Модифікатори доступу ООП [Електронний ресурс] // javarush.ru. Режим доступу: www. URL: <https://javarush.ru/groups/posts/1988-modifikatorih-dostupa-private-protected-default-public>
8. ООП [Електронний ресурс] // programming.in.ua. Режим доступу: www. URL: <http://programming.in.ua/programming/basisprogramming/25-oop.html>
9. Основні поняття ООП [Електронний ресурс] // works.doklad.ru. Режим доступу: www. URL: <https://works.doklad.ru/view/o_odpt9kmPo/all.html>
10. Основні принципи ООП [Електронний ресурс] // studme.org. Режим доступу: www. URL: <https://studme.org/341972/informatika/osnovnye_printsipy>
11. Абстракція даних [Електронний ресурс] // Вікіпедія. Режим доступу: www. URL: <https://ru.wikipedia.org/wiki/Абстракция_данных>
12. Основи ООП [Електронний ресурс] // schoolboyprog10.blogspot.com. Режим доступу: www. URL: <https://schoolboyprog10.blogspot.com/p/blog-page_44.html>
13. Спадкування [Електронний ресурс] // metanit.com. Режим доступу: www. URL: <https://metanit.com/cpp/tutorial/5.10.php>
14. Управління доступом. Інкапсуляція [Електронний ресурс] // metanit.com. Режим доступу: www. URL: <https://metanit.com/cpp/tutorial/5.4.php>
15. Конструктори і ініціалізація об'єктів [Електронний ресурс] // metanit.com. Режим доступу: www. URL: <https://metanit.com/cpp/tutorial/5.2.php>

Додаток **А**

Лістинг коду додатка-сервера

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#define SERV\_PORT 5000 // Порт сервера

#define WSA\_ACCEPT (WM\_USER+0)

#define WSA\_NETEVENT (WM\_USER+1)

#define MAX\_LOADSTRING 100

#pragma comment(lib, "WS2\_32.lib")

#include "Server.h"

#include <windows.h>

#include <windowsx.h>

#include <winsock.h>

#include <commctrl.h>

#include <stdlib.h>

#include <stdio.h>

#include <string>

char szBuf[512];

static HWND hwndEdit;

char mess[2048];

char\* m\_mess = mess;

int err = 0;

int ClientNum = -1;

WSADATA wsaData; //сведения о конкр. реализации интерфейса Windows Sockets

WORD wVersionRequested = MAKEWORD(1, 1); //Номер требуемой версии Windows Sockets

SOCKET srv\_socket = INVALID\_SOCKET; // Сокет сервера

SOCKET sock[2]; // Сокеты клиентов

SOCKADDR\_IN sockaddr[2]; // Адреса клиентов

DWORD cbWritten;

HINSTANCE hInst; // current instance

TCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

TCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

HWND hWindow;

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPWSTR lpCmdLine, \_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place code here.

MSG msg;

HACCEL hAccelTable;

// Initialize global strings

LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadString(hInstance, IDC\_SERVER, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance(hInstance, nCmdShow))

{

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_SERVER));

// Main message loop:

while (GetMessage(&msg, NULL, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_SERV));

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_SERVER);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SERV));

return RegisterClassEx(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

HWND hWnd;

hInst = hInstance;

// Store instance handle in our global variable

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPED | WS\_SYSMENU,

662, 200, 350, 575, nullptr, nullptr, hInstance, nullptr);

if (!hWnd) return FALSE;

hWindow = hWnd;

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

void ServerStart(HWND hWnd)

{

//Функція WSAStartup ініціалізує WinSock

err = WSAStartup(wVersionRequested, &wsaData);

if (err) {

MessageBoxA(hWnd, "WSAStartup Error", "ERROR", MB\_OK | MB\_ICONSTOP);

return;

}

if (srv\_socket != INVALID\_SOCKET) {

MessageBoxA(hWnd, "Socket already created", "Info", MB\_OK | MB\_ICONINFORMATION);

return;

}

//Cтворюємо сокет сервера для роботи з потоком даних

srv\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (srv\_socket == INVALID\_SOCKET) {

MessageBoxA(hWnd, "Socket creation error", "Error", MB\_OK | MB\_ICONERROR);

return;

}

//Встановлюємо адресу IP та номер порту

SOCKADDR\_IN srv\_address;

srv\_address.sin\_family = AF\_INET;

srv\_address.sin\_port = htons(SERV\_PORT);

srv\_address.sin\_addr.s\_addr = INADDR\_ANY; //вик. адресу за замовчуванням (тобто будь-яку)

//Зв'язуємо адресу IP із сокетом

if (SOCKET\_ERROR == bind(srv\_socket, (LPSOCKADDR)&srv\_address, sizeof(srv\_address))) {

closesocket(srv\_socket);

MessageBoxA(hWnd, "Port binding error", "Error", MB\_OK | MB\_ICONSTOP);

return;

}

//очікуємо на встановлення зв'язку

if (listen(srv\_socket, 4) == SOCKET\_ERROR) {

closesocket(srv\_socket);

MessageBoxA(hWnd, "Communication setup pending error", "Error", MB\_OK);

return;

}

//при спробі з'єднання головне вікно отримає повідомлення WSA\_ACCEPT

int rc = WSAAsyncSelect(srv\_socket, hWnd, WSA\_ACCEPT, FD\_ACCEPT);

if (rc) {

closesocket(srv\_socket);

MessageBoxA(hWnd, "WSAAsyncSelect error", "Error", MB\_OK);

return;

}

// Выводим сообщение о запуске сервера

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)" Server started");

}

BOOL AcceptClient(int j)

{

int sockaddr\_len = sizeof(sockaddr[j]);

sock[j] = accept(srv\_socket, (LPSOCKADDR)&sockaddr[j], (int FAR\*) & sockaddr\_len);

if (sock[j] != INVALID\_SOCKET)

if (!WSAAsyncSelect(sock[j], hWindow, WSA\_NETEVENT, FD\_READ | FD\_CLOSE))

return TRUE;

closesocket(sock[j]);

return FALSE;

}

void WndProc\_OnWSAAccept(HWND hWnd, LPARAM lParam)

{

// при помилці скасовуємо надходження повідомлень у головне вікно програми

if (WSAGETSELECTERROR(lParam)) {

MessageBoxA(hWnd, "Accept error", "Error", MB\_OK);

WSAAsyncSelect(srv\_socket, hWnd, 0, 0);

return;

}

if (ClientNum == 1) {

MessageBoxA(hWnd, "Number of clients >2\r\n", "Connection is invalid", MB\_OK);

return;

}

ClientNum++;

if (!AcceptClient(ClientNum)) {

MessageBoxA(hWnd, "Client connection error", "Error", MB\_OK);

return;

}

//додаємо клієнта

sprintf\_s(szBuf, " Added client %i\r\n Address: IP=%s Port=%u\r\n \0", ClientNum + 1,

inet\_ntoa(sockaddr[ClientNum].sin\_addr), htons(sockaddr[ClientNum].sin\_port));

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)szBuf);

}

void SendToClient(int j)

{

if (j > ClientNum) return;

cbWritten = SendMessageA(hwndEdit, WM\_GETTEXTLENGTH, 0, 0);

SendMessageA(hwndEdit, WM\_GETTEXT, (WPARAM)cbWritten, (LPARAM)szBuf);

szBuf[cbWritten] = '\0';

if (send(sock[j], szBuf, strlen(szBuf), 0) != SOCKET\_ERROR) {

sprintf\_s(szBuf,szBuf);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)szBuf);

} else

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)" Error sending message \r\n");

}

void WndProc\_OnWSANetEvent(HWND hWnd, WPARAM wParam, LPARAM lParam)

{

char szTemp[256], szMess[256];

int Number;

//дізнаємося від якого клієнта надійшло повідомлення, => Number

if (sock[0] == (SOCKET)wParam) Number = 0;

else if (sock[1] == (SOCKET)wParam) Number = 1;

//якщо на сокеті виконується передача даних, приймаємо та відображаємо їх

if (WSAGETSELECTEVENT(lParam) == FD\_READ) {

int rc = recv((SOCKET)wParam, szTemp, 256, 0);

if (rc) {

szTemp[rc] = '\0';

sprintf(m\_mess, "%s \r\n Client data %i:\r\n%s", m\_mess, Number + 1, szTemp);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)m\_mess);

}

}

//если соединение завершено, выводим сообщение об этом

if (WSAGETSELECTEVENT(lParam) == FD\_CLOSE) {

WSAAsyncSelect(sock[Number], hWindow, 0, 0);

closesocket(sock[Number]);

sprintf\_s(szTemp, " Client %i finished", Number + 1);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)szTemp);

}

}

void ClientOff(HWND hWnd, int j)

{

if (j > ClientNum) return;

sprintf\_s(szBuf, " Disable Client %i?", j + 1);

if (IDYES == MessageBoxA(hWnd, szBuf, "Question", MB\_YESNO | MB\_ICONQUESTION)) {

WSAAsyncSelect(sock[j], hWindow, 0, 0);

closesocket(sock[j]);

return;

}

}

void ServerStop(HWND hWnd)

{

WSAAsyncSelect(srv\_socket, hWnd, 0, 0);

srv\_socket = INVALID\_SOCKET;

closesocket(srv\_socket);

WSACleanup();

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)" Server stopped");

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

int wmId, wmEvent;

PAINTSTRUCT ps;

HDC hdc;

switch (message)

{

case WM\_CREATE: {

hwndEdit = CreateWindow( // Створюємо доч. вікно для виведення даних від процесів

TEXT("EDIT"), NULL,

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL |

ES\_LEFT | ES\_MULTILINE | ES\_AUTOVSCROLL,

0, 0, 330, 518, hWnd, NULL, hInst, NULL);

err = WSAStartup(wVersionRequested, &wsaData);

if (err) {

MessageBoxA(hWnd, "WSAStartup Error", "ERROR", MB\_OK | MB\_ICONSTOP);

return FALSE;

}

WSACleanup();

sprintf(m\_mess, " Used by %s\r\n Status: %s\r\n",

wsaData.szDescription, wsaData.szSystemStatus);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)m\_mess);

}

break;

case WM\_COMMAND: {

wmId = LOWORD(wParam);

wmEvent = HIWORD(wParam);

switch (wmId)

{

case ID\_START:

ClientNum = -1;

ServerStart(hWnd);

break;

case ID\_STOP:

ServerStop(hWnd);

break;

case ID\_SEND\_CLIENTONE:

SendToClient(0);

break;

case ID\_SEND\_CLIENTTWO:

SendToClient(1);

break;

case ID\_OFF\_CLIENTONE:

ClientOff(hWnd, 0);

break;

case ID\_OFF\_CLIENTTWO:

ClientOff(hWnd, 1);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

EndPaint(hWnd, &ps);

break;

case WM\_DESTROY:

WSACleanup();

PostQuitMessage(0);

break;

case WSA\_ACCEPT:

WndProc\_OnWSAAccept(hWnd, lParam);

break;

case WSA\_NETEVENT:

WndProc\_OnWSANetEvent(hWnd, wParam, lParam);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

Додаток **Б**

Лістинг коду першого клієнта

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#define SERV\_PORT 5000 // Порт сервера

#define WSA\_NETEVENT (WM\_USER+1)

#define MAX\_LOADSTRING 100

#pragma comment(lib, "WS2\_32.lib")

#include "framework.h"

#include "ClientOne.h"

#include <winsock.h>

#include <stdio.h>

#include <string>

static PHOSTENT phe;

char szBuf[1024];

int flag = 0;

static HWND hwndEdit;

char mess[2048];

char\* m\_mess = mess;

char szHostName[128] = "localhost"; //имя хоста

int err = 0;

WSADATA wsaData; //сведения о конкретной реализации интерфейса Windows Sockets

WORD wVersionRequested = MAKEWORD(1, 1); //Номер требуемой версии Windows Sockets

SOCKET cln\_socket = INVALID\_SOCKET; // Сокет сервера

SOCKADDR\_IN dest\_sin; // Адрес сервера

DWORD cbWritten;

HINSTANCE hInst; // current instance

TCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

TCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPWSTR lpCmdLine, \_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place code here.

MSG msg;

HACCEL hAccelTable;

// Initialize global strings

LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadString(hInstance, IDC\_CLIENTONE, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance(hInstance, nCmdShow)){

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_CLIENTONE));

// Main message loop:

while (GetMessage(&msg, NULL, 0, 0)) {

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_CLIENT));

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_CLIENTONE);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_CLIENT));

return RegisterClassEx(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

HWND hWnd;

hInst = hInstance;

// Store instance handle in our global variable

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPED | WS\_SYSMENU,

1000, 200, 350, 290, nullptr, nullptr, hInstance, nullptr);

if (!hWnd) return FALSE;

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

BOOL SetConnection(HWND hWnd)

{

// створюємо сокет

cln\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (cln\_socket == INVALID\_SOCKET) {

MessageBoxA(hWnd, "Socket error", "Error", MB\_OK | MB\_ICONSTOP);

return FALSE;

}

// Визначаємо адресу вузла

phe = gethostbyname(szHostName);

if (phe == NULL) {

closesocket(cln\_socket);

MessageBoxA(hWnd, "Host address not defined", "Error", MB\_OK | MB\_ICONSTOP);

return FALSE;

}

dest\_sin.sin\_family = AF\_INET; // Задаємо тип адреси

dest\_sin.sin\_port = htons(SERV\_PORT); // Встановлюємо номер порту

memcpy((char FAR\*) & (dest\_sin.sin\_addr), phe->h\_addr, phe->h\_length);// Копіюємо адресу вузла

// Встановлюємо з'єднання

if (connect(cln\_socket, (PSOCKADDR)&dest\_sin, sizeof(dest\_sin)) == SOCKET\_ERROR) {

closesocket(cln\_socket);

MessageBoxA(hWnd, "Connection error", "Error", MB\_OK | MB\_ICONSTOP);

return FALSE;

}

// при спробі з'єднання головне вікно отримає повідомлення WSA\_ACCEPT

if (WSAAsyncSelect(cln\_socket, hWnd, WSA\_NETEVENT, FD\_READ | FD\_CLOSE)){

MessageBoxA(hWnd, "WSAAsyncSelect error", "Error", MB\_OK);

return FALSE;

}

// Виводимо повідомлення про встановлення з'єднання з вузлом

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)" Connection is set\r\n ");

return TRUE;

}

void SendMsg(HWND hWnd)

{

char MoseWheel[20];

int MouseButtons = GetSystemMetrics(SM\_CMOUSEBUTTONS);

GetSystemMetrics(SM\_MOUSEWHEELPRESENT) ? strcpy(MoseWheel, " is present") : strcpy(MoseWheel, " is missing");

sprintf(szBuf, " The number of buttons in the mouse: %d \r\n Scroll wheel: %s\r\n ", MouseButtons, MoseWheel);

if (send(cln\_socket, szBuf, strlen(szBuf), 0) != SOCKET\_ERROR) {

sprintf(m\_mess, "\r\n Data has been sent to the server: \r\n%s", szBuf);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)m\_mess);

} else {

sprintf(m\_mess, "%s \r\n Error sending message \r\n ", m\_mess);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)m\_mess);

}

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

int wmId, wmEvent;

PAINTSTRUCT ps;

HDC hdc;

switch (message)

{

case WM\_CREATE: {

hwndEdit = CreateWindow( // Создаем доч.окно для вывода данных от процессов

TEXT("EDIT"), NULL,

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL |

ES\_LEFT | ES\_MULTILINE | ES\_AUTOVSCROLL,

0, 0, 330, 230, hWnd, NULL, hInst, NULL);

err = WSAStartup(wVersionRequested, &wsaData);

if (err) {

MessageBoxA(hWnd, "WSAStartup Error", "ERROR", MB\_OK | MB\_ICONSTOP);

return FALSE;

}

sprintf(m\_mess, " Used by %s \r\n Status: %s\r\n",

wsaData.szDescription, wsaData.szSystemStatus);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)m\_mess);

}

break;

case WM\_COMMAND: {

wmId = LOWORD(wParam);

wmEvent = HIWORD(wParam);

switch (wmId)

{

case ID\_SET:

SetConnection(hWnd);

break;

case ID\_SENDMESSAGE:

SendMsg(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT: {

hdc = BeginPaint(hWnd, &ps);

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY: {

WSACleanup();

PostQuitMessage(0);

}

break;

case WSA\_NETEVENT: {

// якщо на сокеті виконується передача даних, приймаємо та відображаємо їх

if (WSAGETSELECTEVENT(lParam) == FD\_READ) {

int rc = recv(cln\_socket, szBuf, sizeof(szBuf), 0);

if (rc) {

szBuf[rc] = '\0';

sprintf(m\_mess, "%s\r\n\_\_\_\_\_\_\_\_\_\_Server data\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\r\n%s\r\n", m\_mess, szBuf);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)m\_mess);

}

}

// якщо з'єднання завершено, виводимо повідомлення про це

if (WSAGETSELECTEVENT(lParam) == FD\_CLOSE)

MessageBoxA(hWnd, "Server closed", "Server", MB\_OK);

}

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

Додаток **В**

Лістинг коду другого клієнта

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#define SERV\_PORT 5000 // Порт сервера

#define WSA\_NETEVENT (WM\_USER+1)

#define MAX\_LOADSTRING 100

#pragma comment(lib, "WS2\_32.lib")

#include "framework.h"

#include "ClientTwo.h"

#include <winsock.h>

#include <stdio.h>

#include <string>

static PHOSTENT phe;

char szBuf[1024];

int flag = 0;

static HWND hwndEdit;

char mess[2048];

char\* m\_mess = mess;

char szHostName[128] = "localhost"; //имя хоста

int err = 0;

WSADATA wsaData; //сведения о конкретной реализации интерфейса Windows Sockets

WORD wVersionRequested = MAKEWORD(1, 1); //Номер требуемой версии Windows Sockets

SOCKET cln\_socket = INVALID\_SOCKET; // Сокет сервера

SOCKADDR\_IN dest\_sin; // Адрес сервера

DWORD cbWritten;

HINSTANCE hInst; // current instance

TCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

TCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPWSTR lpCmdLine, \_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place code here.

MSG msg;

HACCEL hAccelTable;

// Initialize global strings

LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadString(hInstance, IDC\_CLIENTTWO, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance(hInstance, nCmdShow)) {

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_CLIENTTWO));

// Main message loop:

while (GetMessage(&msg, NULL, 0, 0)) {

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_CLIENT));

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_CLIENTTWO);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_CLIENT));

return RegisterClassEx(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

HWND hWnd;

hInst = hInstance;

// Store instance handle in our global variable

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPED | WS\_SYSMENU,

1000, 485, 350, 290, nullptr, nullptr, hInstance, nullptr);

if (!hWnd) return FALSE;

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

BOOL SetConnection(HWND hWnd)

{

// створюємо сокет

cln\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (cln\_socket == INVALID\_SOCKET) {

MessageBoxA(hWnd, "Socket error", "Error", MB\_OK | MB\_ICONSTOP);

return FALSE;

}

// Визначаємо адресу вузла

phe = gethostbyname(szHostName);

if (phe == NULL) {

closesocket(cln\_socket);

MessageBoxA(hWnd, "Host address not defined", "Error", MB\_OK | MB\_ICONSTOP);

return FALSE;

}

dest\_sin.sin\_family = AF\_INET; // Задаємо тип адреси

dest\_sin.sin\_port = htons(SERV\_PORT); // Встановлюємо номер порту

memcpy((char FAR\*) & (dest\_sin.sin\_addr), phe->h\_addr, phe->h\_length); // Копіюємо адресу вузла

// Встановлюємо з'єднання

if (connect(cln\_socket, (PSOCKADDR)&dest\_sin, sizeof(dest\_sin)) == SOCKET\_ERROR) {

closesocket(cln\_socket);

MessageBoxA(hWnd, "Connection error", "Error", MB\_OK | MB\_ICONSTOP);

return FALSE;

}

// при спробі з'єднання головне вікно отримає повідомлення WSA\_ACCEPT

if (WSAAsyncSelect(cln\_socket, hWnd, WSA\_NETEVENT, FD\_READ | FD\_CLOSE)) {

MessageBoxA(hWnd, "WSAAsyncSelect error", "Error", MB\_OK);

return FALSE;

}

// Виводимо повідомлення про встановлення з'єднання з вузлом

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)" Connection is set\r\n ");

return TRUE;

}

void SendMsg(HWND hWnd)

{

MEMORYSTATUSEX status;

status.dwLength = sizeof(status);

GlobalMemoryStatusEx(&status);

char buffer[2][\_MAX\_U64TOSTR\_BASE10\_COUNT];

sprintf(buffer[0], "%llu", status.ullTotalPageFile);

sprintf(buffer[1], "%llu", status.ullAvailPageFile);

sprintf(szBuf, " Pagefile size: %s byte. \r\n Pagefile free size: %s byte.\r\n",

buffer[0], buffer[1]);

if (send(cln\_socket, szBuf, strlen(szBuf), 0) != SOCKET\_ERROR) {

sprintf(m\_mess, "\r\n Data has been sent to the server: \r\n%s", szBuf);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)m\_mess);

} else {

sprintf(m\_mess, "%s \r\n Error sending message \r\n ", m\_mess);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)m\_mess);

}

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

int wmId, wmEvent;

PAINTSTRUCT ps;

HDC hdc;

switch (message)

{

case WM\_CREATE: {

hwndEdit = CreateWindow( // Создаем доч.окно для вывода данных от процессов

TEXT("EDIT"), NULL,

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL |

ES\_LEFT | ES\_MULTILINE | ES\_AUTOVSCROLL,

0, 0, 330, 230, hWnd, NULL, hInst, NULL);

err = WSAStartup(wVersionRequested, &wsaData);

if (err) {

MessageBoxA(hWnd, "WSAStartup Error", "ERROR", MB\_OK | MB\_ICONSTOP);

return FALSE;

}

sprintf(m\_mess, " Used by %s \r\n Status: %s\r\n",

wsaData.szDescription, wsaData.szSystemStatus);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)m\_mess);

}

break;

case WM\_COMMAND: {

wmId = LOWORD(wParam);

wmEvent = HIWORD(wParam);

switch (wmId)

{

case ID\_SET:

SetConnection(hWnd);

break;

case ID\_SENDMESSAGE:

SendMsg(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_PAINT: {

hdc = BeginPaint(hWnd, &ps);

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY: {

WSACleanup();

PostQuitMessage(0);

}

break;

case WSA\_NETEVENT: {

// если на сокете выполняется передача данных, принимаем и отображаем их

if (WSAGETSELECTEVENT(lParam) == FD\_READ) {

int rc = recv(cln\_socket, szBuf, sizeof(szBuf), 0);

if (rc) {

szBuf[rc] = '\0';

sprintf(m\_mess, "%s\r\n\_\_\_\_\_\_\_\_\_\_Server data\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\r\n%s\r\n", m\_mess, szBuf);

SendMessageA(hwndEdit, WM\_SETTEXT, 0, (LPARAM)m\_mess);

}

}

// если соединение завершено, выводим сообщение об этом

if (WSAGETSELECTEVENT(lParam) == FD\_CLOSE)

MessageBoxA(hWnd, "Server closed", "Server", MB\_OK);

}

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}