

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет "Запорізька політехніка"

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни
"Системне програмування"

для студентів спеціальності 123 "Комп'ютерна інженерія"
всіх форм навчання

Частина 3

2024

Методичні вказівки до виконання лабораторних робіт з дисципліни "Системне програмування" для студентів спеціальності 123 "Комп'ютерна інженерія" всіх форм навчання. Частина 3. / Укл. М.М. Хохлов, О.В. Зелік – Запоріжжя: НУ "Запорізька політехніка", 2024. – 24 с.

Укладачі: М.М. Хохлов, ст..викл.
О.В. Зелік, зав. навч. лаб.

Рецензент: С.Ю. Скрупський, к.т.н., доцент

Відповідальний за випуск: М.М. Хохлов, ст. викладач

Затверджено:
на засіданні кафедри
"Комп'ютерні системи та мережі"
Протокол № 7 від 22 березня 2024

Рекомендовано до видання НМК ФКНТ
Протокол № 8 від 27 березня 2024

ЗМІСТ

Вступ	4
1 Лабораторна робота № 6 - Створення сервера.....	5
1.1 Теоретичні відомості	5
1.2 Завдання.....	7
1.3 Контрольні запитання.....	12
2 Лабораторна робота № 7 - HTTP-сервер	13
2.1 Теоретичні відомості	13
2.2 Завдання для виконання	15
2.3 Контрольні запитання.....	23
Рекомендована література.....	24

ВСТУП

Лабораторні роботи спрямовані на засвоєння базових знань з системного програмування при використанні системних викликів ОС Windows (WinAPI). Вивчаються основні концепції роботи з системними об'єктами. Вивчається створення програм з використанням мови програмування C/C++.

У лабораторних роботах розглядається створення програм для реалізації задач взаємодії з операційною системою на системному рівні, доступу до файлової системи, роботи багатопоточних програм та їх синхронізація, організація обміну даними між процесами; вивчаються питання створення об'єктів синхронізації, створення потоків, засобів передавання даних між процесами та комп'ютерними системами в мережі.

Відповідно до графіку, студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою.

Для одержання заліку з кожної роботи студент здає викладачу оформлений звіт, а також демонструє на екрані комп'ютера результати виконання лабораторної роботи.

Звіт має містити:

- титульний аркуш;
- тему та мету роботи; завдання до роботи;
- лаконічний опис теоретичних відомостей;
- результати виконання лабораторної роботи;
- змістовний аналіз отриманих результатів, висновки та відповіді до контрольних запитань.

Звіт виконують на білому папері формату A4 (210x297 мм) або подають в електронному вигляді. Під час співбесіди при захисті лабораторної роботи студент повинний виявити знання про зміст роботи та методи виконання кожного етапу роботи, а також вміти продемонструвати результати роботи на конкретних прикладах та правильно аналізувати отримані результати.

1 ЛАБОРАТОРНА РОБОТА № 6 - СТВОРЕННЯ СЕРВЕРА

Мета роботи: навчитися створенню програми сервера на базі орієнтованих на з'єднання сокетів. Технологія клієнт-сервер.

1.1 Теоретичні відомості

Технологія клієнт-сервер

Існує наступний порядок дій клієнта та сервера під час під'єднання програми-клієнта до програми-сервера:

Сервер	Клієнт
1. Створення сокету, що очікує запити на з'єднання від клієнтів	
	2. Створення клієнтського сокету та спроба з'єднання з сервером
3. Дозвіл на спробу з'єднання	
4. Передавання/прийом даних	4. Прийом/передавання даних
5. Знищення з'єднання	5. Знищення з'єднання

Для створення сокету, що очікуватиме на з'єднання спочатку необхідно зв'язати створений сокет з локальною адресою комп'ютера. Для цього використовується функція **bind** (Берклі):

int bind(

SOCKET s, // ідентифікатор сокету

const struct sockaddr* name, // адреса структури

параметрів сокету

int namelen); // довжина структури параметрів

де **sockaddr* name** – структура, що описує правила з'єднання з сокетом **s**. Зазвичай ця структура має таке наповнення:

```

SOCKADDR_IN socketaddr;
socketaddr.sin_family = AF_INET;
socketaddr.sin_addr.s_addr = INADDR_ANY;
socketaddr.sin_port = PORT_ADDR;

```

де константа `INADDR_ANY` – зазначає, що отримати з'єднання може клієнт з будь-якою мережною адресою у мережі Internet, якщо спробує під'єднатись до порту номер якого зазначено, як `socketaddr.sin_port`.

Після пов'язування сокету з локальною адресою сокет можна перевести у режим очікування на з'єднання. Це здійснюється за допомогою функції **listen** (Берклі):

```

int listen(
    SOCKET s,      // ідентифікатор сокету
    int backlog); // максимальна кількість черги для клієнтів,
                  // що очікують на з'єднання (значення
                  // рівне SOMAXCONN – максимально можлива)

```

Очікування на з'єднання задається за допомогою функції **accept** (Берклі):

```

SOCKET accept(
    SOCKET s,      // ідентифікатор сокету
    struct sockaddr* addr, // буфер для структури параметрів
    сокету
    int* addrlen); // довжина буферу addr

```

Слід відзначити, що для організації обміну з клієнтом необхідно створити додаткову змінну типу **SOCKET** для подальшого її використання в операціях прийому/передавання й вивільнення базового (серверного) сокету для під'єднання інших клієнтів. Тобто під час з'єднання з клієнтом на сервері створюється окремий сокет для обміну даними з поточним клієнтом інформація про якого знаходиться у структурній

змінній **struct sockaddr** **addr**, яку також необхідно додатково створити.

1.2 Завдання

У середовищі розробки необхідно створити проект консольної Windows програми у своєму робочому каталозі згідно інструкцій відповідного середовища. Набрати наведену далі програму 1.1 зробивши відповідні зміни.

Програма 1.1 – lab6.cpp

```
#include "stdafx.h"
#include "windows.h"

#include "winsock.h"
#pragma comment(lib, "ws2_32.lib")

#define DEFAULT_PORT          5001
#define DEFAULT_PROTO        SOCK_STREAM
#define DEFAULT_BUFFER_LEN   4096
#define WINSOCK_VERSION      0x0101

char Buffer[DEFAULT_BUFFER_LEN]={0};

int main(int argc, char* argv[])
{
    int retval=0, i=0;
    sockaddr_in service;

    SOCKET ListenSocket=INVALID_SOCKET;
    SOCKET AcceptSocket=INVALID_SOCKET;

    SOCKADDR_IN socketclientaddr;
    int length=sizeof(SOCKADDR);

    char senddata[]="Welcome to my server!\r\n";

    WSADATA wsaData; // Відомості щодо реалізації
```

```

// Ініціалізація Winsock
if (WSAStartup(WINSOCK_VERSION, &wsaData) !=
    NO_ERROR) {
    printf("Error at WSAStartup()\n");
} else {
    // Створення SOCKET для прослуховування
    // вхідних запитів на під'єднання
    ListenSocket = socket(AF_INET, DEFAULT_PROTO,
                          IPPROTO_TCP);
    if (ListenSocket == INVALID_SOCKET) {
        printf("Error at socket(): %ld\n",
               WSAGetLastError());
        WSACleanup();
        return 1;
    }
    // Структура sockaddr_in, що описує параметри
    // обмежень під'єднань:
    // - множина дозволених протоколів;
    service.sin_family = AF_INET;
    // - множина IP адрес;
    // (будь-яка IP адреса)
    service.sin_addr.s_addr = INADDR_ANY;
    // - номер порт для під'єднання до сокету.
    service.sin_port = htons(DEFAULT_PORT);
    // зв'язування сокету з локальною адресою
    if (bind( ListenSocket, (SOCKADDR*)&service,
             sizeof(service)) == SOCKET_ERROR) {
        printf("bind() failed.\n");
        closesocket(ListenSocket);
        return 1;
    }
    // Прослуховування вхідних запитів на
    // під'єднання до створеного сокету
    if (listen( ListenSocket, 1 ) ==
        INVALID_SOCKET) {
        printf("Error listening on socket.\n");
    } else {
        printf("Listening on socket...\n");
        // Створення сокету для прийому запитів
        printf("Waiting for client connections.\n");
        AcceptSocket = accept( ListenSocket,

```

```

        (LPSOCKADDR) &socketclientaddr,
        (LPINT) &length);
if (AcceptSocket != INVALID_SOCKET) {
    // Надсилання запрошення клієнту
    send(AcceptSocket, senddata,
        sizeof(senddata), 0);
    printf("Client to accept.\n");
    printf("Received data:\n");
    do{
        // Приймання даних від клієнта поки
        // він не від'єднається
        retval=recv(AcceptSocket, Buffer,
            sizeof(Buffer), 0);
        // Перевірка розриву з'єднання
        if(!retval) break;
        // Виведення до екрану отриманих даних
        for(i=0;i<retval;i++)
            printf("%c", Buffer[i]);
    }while(Buffer[0]!='Q');
    // Припинення обміну
    if (!retval)
        printf("\nConnection is closed.\n");
    else
        printf("\nClient disconnected.\n");
    closesocket(AcceptSocket);
    }
    closesocket(ListenSocket);
}
WSACleanup();
}
return 0;
}

```

Завдання 1. Відлагодити програму. При відсутності помилок робоча програма має вигляд рис.1.1:

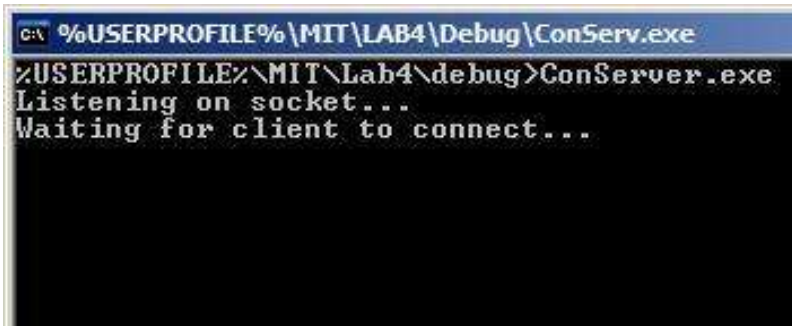


Рисунок 1.1 – Програма сервер

В окремому вікні виконати запуск командного процесору "cmd.exe" й виконати програму визначення стану відкритих портів локального комп'ютера "netstat.exe -a" (рис.1.2). За правильної роботи програми у списку "Активних підключень" має бути присутнім відкритий **TCP** порт номер якого збігається зі значенням DEFAULT_PORT в основній програмі.

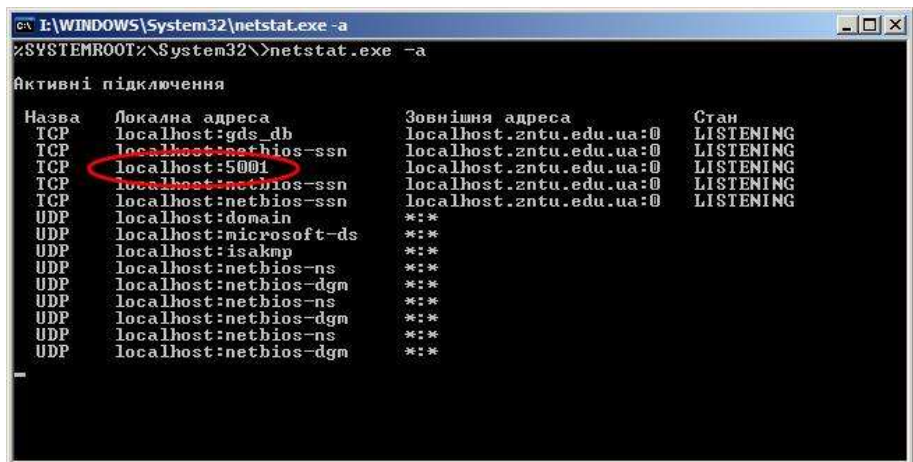


Рисунок 1.2 – Перегляд активних підключень програмою netstat.exe

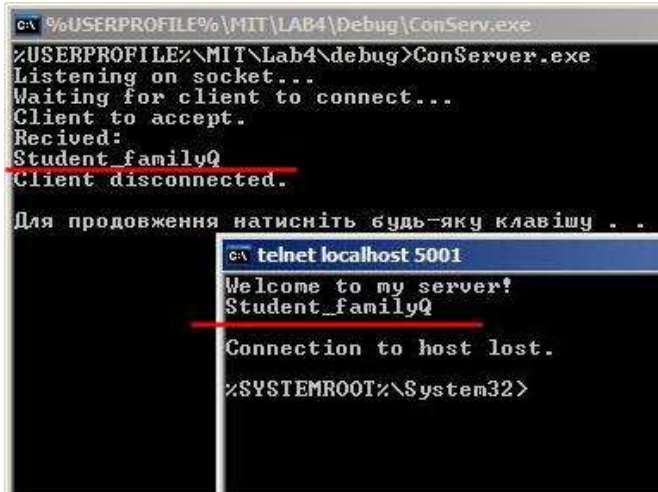


Рисунок 1.3 – Прийом даних створеним сервером

Для під'єднання до отриманого сервера необхідно в командному рядку виконати команду "**telnet.exe localhost 5001**". Для перевірки дієздатності програми на можливість прийому даних надрукуйте Ваше прізвище у вікні "**telnet**" та перегляньте результат прийому даних у вікні сервера (рис.1.3). Для закінчення обміну натисніть <Q> у вікні "**telnet**".

Завдання 2 – Внести необхідні зміни до програми згідно варіанту.

Перший варіант: при введенні першої літери Вашого прізвища програма-сервер повинна надсилати до вікна "**telnet**" наступні літери Вашого прізвища та **IP** адресу комп'ютера з якого було здійснено під'єднання до сервера у форматі "десятькове-з-крапкою".

Другий варіант: при введенні першої літери Вашого імені програма-сервер повинна надсилати до вікна "**telnet**" наступні літери Вашого імені та **DNS ім'я** комп'ютера з якого було здійснено під'єднання до сервера.

Примітка. Для визначення інформації щодо клієнту використовується **struct sockaddr** `addr` з функції **accept**.

Завдання 3. Організація обробки помилок.

1. Для організації оброблення помилок роботи програми сервера необхідно доповнити програму командами отримання значення помилок у гілках програми, що відповідають виникненню помилок при роботі.

2. Для отримання коду помилки можна скористатись інформацією з лабораторної роботи №1.

3. Створити умови для генерації помилок по чергово в усіх гілках обробки помилок та занотувати до звіту.

1.3 Контрольні запитання

1. Технологія клієнт-сервер.
2. Алгоритм організації сервера.
3. Створення сокету, що очікує на з'єднання. Прототип функції, призначення та склад аргументів.
4. Структурний тип **sockaddr_in**. Елементи структури.
5. Зв'язування сокету з локальною адресою. Прототип функції, призначення та склад аргументів.
6. Організація приймання запиту на з'єднання. Прототип функції, призначення та склад аргументів.
7. Функції приймання даних через орієнтований сокет. Прототип функцій, призначення та склад аргументів.
8. Функції передавання даних через орієнтований сокет. Прототип функцій, призначення та склад аргументів.
9. Особливості організації приймання даних без втрат.

2 ЛАБОРАТОРНА РОБОТА № 7 - HTTP-СЕРВЕР

Мета роботи: вивчити роботу з асинхронними сокетами на базі WinSock на прикладі HTTP сервера.

2.1 Теоретичні відомості

Для роботи з асинхронними сокетами у операційній системі Windows використовується механізм передавання повідомлень у разі зміни стану зазначеного сокету.

Робота з асинхронними сокетами.

Для створення асинхронного сокету спочатку необхідно створити сокет, а потім перевести його у асинхронний режим роботи за допомогою функції WinSock – **WSAAsyncSelect**:

```
int WSAAsyncSelect(
    SOCKET s,           // ідентифікатор сокету
    HWND hWnd,         // ідентифікатор вікна програми
    unsigned int wParam, // повідомлення, при мережних подіях
    long lEvent);       // бітова маска мережних подій
```

де **hWnd** – ідентифікатор вікна програми, щодо обробки мережних повідомлень для зазначеного сокету **s**. **wParam** – номер повідомлення, що повідомляє віконну функцію (програму) про появу деякої події на зазначеному сокеті, яка потребує обробки (номер повідомлення має бути значення отриманим в результаті додавання до повідомлення **WM_USER** деякого константного числа). **lEvent** – бітова маска для визначення подій у разі виникнення яких на сокеті **s** буде надсилатися повідомлення **wParam** до віконної функції вікна **hWnd**. Ознаки бітової маски наведено у таблиці 2.1.

Таблиця 2.1 – Ознаки бітової маски мережних подій

FD_READ	Отримувати повідомлення, якщо треба прийняти дані.
FD_WRITE	Отримувати повідомлення, якщо треба передати дані.
FD_ACCEPT	Отримувати повідомлення, якщо є запит на під'єднання до сокету.
FD_CONNEC T	Отримувати повідомлення, якщо є підтвердження запиту на під'єднання до сокету.
FD_CLOSE	Отримувати повідомлення, якщо сокет було зачинено.

Таким чином основна робота сервера буде перенесена до віконної функції у наступному вигляді:

```

case WM_ASYNC_SOCKET: // Повідомлення про мережну подію
    async_socket = (SOCKET)wParam;
    WSAEvent = WSAGETSELECTEVENT (lParam);
    switch (WSAEvent) {
        case FD_CLOSE:
            shutdown(async_socket, SD_BOTH);
            closesocket(async_socket);
            return 0;
        case FD_READ:
            recv_len=recv(async_socket, buf, MAX_DATA, 0);
            return 0;
        case FD_WRITE:
            send_len=send(async_socket, buf, MAX_DATA, 0);
            return 0;
        case FD_ACCEPT:
            client_socket = accept(async_socket,NULL,NULL);
            return 0;
    } break;

```

Слід зазначити, що у разі якщо сокет знаходиться у стані прийому або передавання даних, то звичайне зачинення не дієздатне, бо сокет знаходиться у заблокованому стані. З цієї причини необхідно при знищенні (зачиненні) сокету спочатку використовувати функцію, що забороняє прийом та передачу даних через сокет. Вимкнення режимів прийому-передавання сокету здійснюється за допомогою функції **shutdown**:

```
int shutdown(
    SOCKET s,      // ідентифікатор сокету
    int how); // ознака вимкнення операцій прийому-
передавання
```

де параметр **how** може приймати одне з наступних значень:
SD_RECEIVE – заборона прийому даних;
SD_SEND – заборона передавання даних;
SD_BOTH – заборона обох режимів: і прийому, і передавання.

2.2 Завдання для виконання

Завдання 1 – HTTP сервер на базі асинхронних сокетів.

У середовищі розробки необхідно створити каркасний (шаблонний) проект віконної Windows програми у своєму робочому каталозі. Зробити відповідні зміни у програмі згідно наведених далі інструкцій:

- додати у заголовочний частині програми під'єднання бібліотеки WinSocket:

```
#include "winsock2.h"
#pragma comment(lib, "ws2_32.lib")
```

- створити додаткові глобальні змінні:

```
// Global Variables for server:
#define PORT_ADDR      80
#define QUEUE_SIZE     5
#define WM_ASYNC_CLIENT WM_USER+1
#define DEFAULT_PROTOCOL SOCK_STREAM
#define MAX_DATA       1024
```

```

int    bStartServer=0;
BYTE   buffer[MAX_DATA]={0};
TCHAR  log_str[MAX_DATA]={0};

char *reply="HTTP/1.0\r\nAccept: */*"
        "\r\n\r\n";
char *head("<html><title>HTTP
Server</title><body><p>";
char *end_head("</p></body></html>";

```

```

WSADATA wsaData;
SOCKET   servsocket=INVALID_SOCKET,
        clientsocket=INVALID_SOCKET;

```

- описати прототипи додаткових функцій:

```

// Declarations of functions sockets:
BOOL    StartServer(HWND m_hWnd);
void    StopServer(HWND m_hWnd);
BOOL    CreateSocket();
BOOL    LinkSocketPort();
BOOL    LinkWindowSocket(HWND m_hWnd);
BOOL    ListenSocket();
BOOL    StartWinSock();
BOOL    StopWinSock();
void    CloseSocket();

```

- у віконній функції створити наступні локальні змінні:

```

int i=0;
RECT rect;
SOCKET currentsock=INVALID_SOCKET;
WORD   WSAEvent=0;
TCHAR  szStatus [2][MAX_STRING]={
        "HTTP Server is stopped.",
        "HTTP Server is running..."};

```

- скорегувати обробку системного повідомлення

WM_PAINT відповідно до наведеного фрагменту:

```

case WM_PAINT:
    hdc = BeginPaint(hWnd, &ps);
    // TODO: Add any drawing code here...
    GetClientRect(hWnd, &rect);
    DrawText(hdc, szStatus[bStartServer],
        strlen(szStatus[bStartServer]),

```

```

        &rect, DT_CENTER);
if (*log_str){
    rect.top+=50;
    DrawText(hdc, log_str, strlen(log_str),
        &rect, DT_LEFT);
}
EndPaint(hWnd, &ps);
break;

```

- додати обробку повідомлення миші, для керування режимом роботи сервера:

```

case WM_RBUTTONDOWN:
    bStartServer=(bStartServer+1)%2;
    if (bStartServer)
        StartServer(hWnd);
    else
        StopServer(hWnd);
    InvalidateRect(hWnd, NULL, TRUE);
    break;

```

- додати обробку повідомлення системи, щодо даного асинхронного сокету, яке було задане користувачем:

```

case WM_ASYNC_CLIENT:
    // Parse the async_socket events:
    currentsock = wParam;
    WSAEvent = WSAGETSELECTEVENT(lParam);
    switch (WSAEvent) {
    case FD_CLOSE:
        sprintf(log_str,
            "Client close socket.");
        InvalidateRect(hWnd, NULL, TRUE);
        break;
    case FD_READ:
        i=recv(currentsock, (char*)buffer,
            MAX_DATA, 0);
        if (currentsock!=INVALID_SOCKET) {
            buffer[i]=0;
            sprintf(log_str,
                "Read socket data.: %s",
                buffer);
            InvalidateRect(hWnd, NULL, TRUE);
        }
        sprintf((char*)buffer,

```

```

        "%s%s HOME Page %s",
        reply, head, end_head);
    send(currentsock, (char*)buffer,
        strlen((char*)buffer), 0);
    closesocket(currentsock);
    break;
case FD_ACCEPT:
    clientsocket=accept(currentsock, NULL,
        NULL);
    if (clientsocket!=INVALID_SOCKET) {
        wsprintf(log_str,
            "Accept socket client.");
        InvalidateRect(hWnd, NULL, TRUE);
    }
    break;
}
break;

```

- скорегувати обробку системного повідомлення **WM_DESTROY**, щодо закінчення роботи програми:

```

case WM_DESTROY:
    if (bStartServer) StopServer(hWnd);
    PostQuitMessage(0);
    break;

```

- додати реалізації наступних функцій:

```

BOOL StartServer(HWND m_hWnd)
{
    if (!StartWinSock()) return FALSE;
    if (!CreateSocket()) return FALSE;
    if (!LinkSocketPort()) return FALSE;
    if (!LinkWindowSocket(m_hWnd))
        return FALSE;
    if (!ListenSocket()) return FALSE;

    wsprintf(log_str,
        "Waiting for accept clients");
    InvalidateRect(m_hWnd, NULL, TRUE);
    return TRUE;
}

```

```

void StopServer(HWND m_hWnd)
{

```

```

    CloseSocket();
    StopWinSock();
    wsprintf(log_str, "Server closed socket.");
    InvalidateRect(m_hWnd, NULL, TRUE);
}

BOOL StartWinSock()
{
    if(WSAStartup(WINSOCK_VERSION, &wsaData))
    {
        MessageBox(NULL,
                    "Error initialize Winsock!",
                    szTitle, MB_OK|MB_ICONERROR);
        WSACleanup();
        return FALSE;
    }
    return TRUE;
}

BOOL StopWinSock()
{
    if(WSACleanup())
    {
        MessageBox(NULL, "Error WSACleanup!",
                    szTitle, MB_OK|MB_ICONERROR);
        return FALSE;
    }
    return TRUE;
}

BOOL CreateSocket()
{
    servsocket = socket(PF_INET,
                        DEFAULT_PROTOCOL,
                        IPPROTO_TCP);
    if (servsocket==INVALID_SOCKET)
    {
        MessageBox(NULL, "Error create socket!",
                    szTitle, MB_OK|MB_ICONERROR);
        return FALSE;
    }
    return TRUE;
}

```

```

}

void CloseSocket()
{
    if (clientsocket!=INVALID_SOCKET)
    {
        shutdown(clientsocket,SD_BOTH);
        closesocket(clientsocket);
    }

    if (servsocket!=INVALID_SOCKET)
    {
        shutdown(servsocket,SD_BOTH);
        closesocket(servsocket);
    }
}

BOOL LinkSocketPort()
{
    SOCKADDR_IN socketaddr;
    socketaddr.sin_family = AF_INET;
    socketaddr.sin_addr.s_addr = INADDR_ANY;
    socketaddr.sin_port = htons(PORT_ADDR);

    if(bind(servsocket,
            (LPSOCKADDR)&socketaddr,
            sizeof(socketaddr)) ==
        SOCKET_ERROR)
    {
        MessageBox(NULL,"Error bind socket!",
                    szTitle,MB_OK|MB_ICONERROR);
        return FALSE;
    }
    return TRUE;
}

BOOL LinkWindowSocket(HWND m_hWnd)
{
    if(WSAAsyncSelect(servsocket,m_hWnd,
                      WM_ASYNC_CLIENT,
                      FD_ACCEPT|FD_READ|FD_CLOSE) ==
        SOCKET_ERROR)

```

```

{
    MessageBox(NULL, "Error AsyncSelect!",
                szTitle, MB_OK|MB_ICONERROR);
    return FALSE;
}
return TRUE;
}

BOOL ListenSocket()
{
    if(listen(servsocket, QUEUE_SIZE) ==
        SOCKET_ERROR)
    {
        MessageBox(NULL, "Error listen socket!",
                    szTitle, MB_OK|MB_ICONERROR);
        return FALSE;
    }
    return TRUE;
}

```

Примітка. Активація HTTP сервера здійснюється за допомогою натискання правої клавіші маніпулятора миші у вікні програми.

Перевірка сервера здійснюється за допомогою браузеру
рис.2.1.

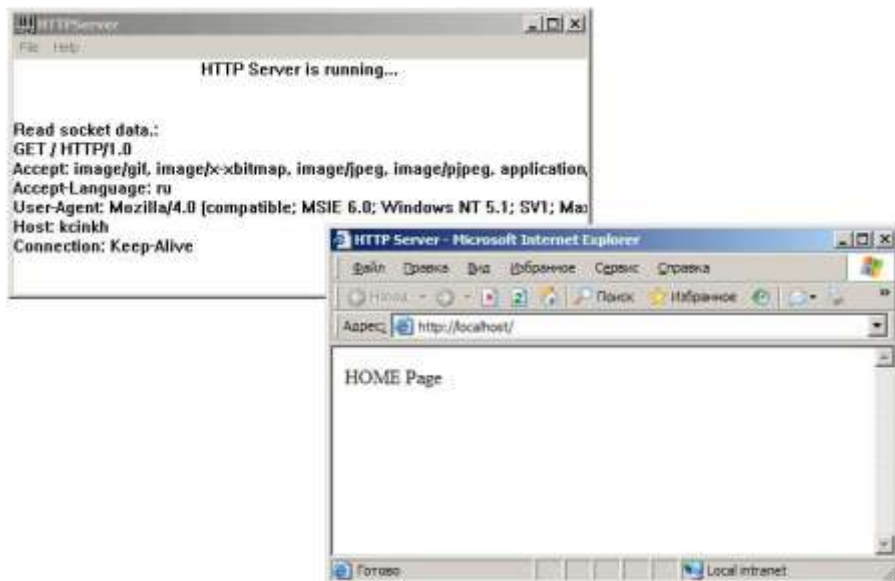


Рисунок 2.1 – Перевірка HTTP сервера

Завдання 2. Організація обробки помилок.

1. Для організації оброблення помилок роботи програми сервера необхідно доповнити програму командами отримання значення помилок у гілках програми, що відповідають виникненню помилок при роботі.

2. Для отримання коду помилки можна скористатись інформацією з лабораторної роботи №1.

3. Створити умови для генерації помилок по чергово в усіх гілках обробки помилок.

Завдання 3.

Змінити код програми таким чином, щоб у вікні браузеру замість тексту "HOME Page" відображались П.І.Б. та № групи, а заголовок вікна містив номер у списку групи.

Завдання 4*

Допрацювати програму таким чином щоб при запиті "http://localhost/прізвище.html" сервер повертав ім'я розробника.

Завдання 5*

Допрацювати програму таким чином щоб у відповідь на запити "http://localhost/filename.html" сервер надсилав зміст локального файлу **filename.html**, на запит "http://localhost/" - зміст локального файлу **index.html**

2.3 Контрольні запитання

1. Створення асинхронних сокетів.
2. Прототипи та призначення функцій. Аргументи, що передаються, та значення, що повертаються. (**WSAAsyncSelect**, **shutdown**)
3. Типи мережних подій. Вибір необхідних та їх обробка.
4. Аргументи мережних подій. Макрос **WSAGETSELECTEVENT**.
5. Приймання та передавання даних через асинхронні сокети.
6. Завершення роботи з асинхронними сокетами.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Геннадій Галісеєв. Системне програмування. – Університет "Україна". 2019 – 113 с.
2. Johnson M. Hart. Windows System Programming. Fourth Edition. – Addison-Wesley Microsoft Technology Series. 2010 – 609 с.
3. Lewis Van Winkle. Hands-On Network Programming with C: Learn socket programming in C and write secure and optimized network code – Packt Publishing, 2019 – 761с. ISBN: 978-1789349863
4. Костенко А.В., Костирко В.С., Плеша М.І. Крос-платформне програмування : навч. посіб. – Центр. спілка спожив. т-в України, Львів. 2019. - 247 с.
5. Рисований О.М. Системне програмування. Графічний інтерфейс користувача (GUI). Навч. посібник. – Х.: НТУ “ХПІ”, 2018. – 160 с.
6. Рисований О.М. Системне програмування: підручник для студентів напрямку “Комп’ютерна інженерія” вищих навчальних закладів в 2-х томах. Том 2. – Видання четверте: виправлено та доповнено – Х.: “Слово”, 2015. – 378 с.
7. Шевцов В.А. Операційні системи. - К.: Видавнича група BHV, 2005 – 576 с.
8. Andrew S. Tanenbaum, Herbert Bos. Modern Operating Systems. Fifth Edition – Hoboken, New Jersey 07458 – 2024. – 1185 с. ISBN: 978-1-292-45966-0
9. Windows Internals, Seventh Edition, Part 1: System architecture, processes, threads, memory management, and more by Pavel Yosifovich, Alex Ionescu, Mark E. Russinovich, and David A. Solomon. – Microsoft Press, 2017. – ISBN: 978-0-7356-8418-8.
10. Windows Internals, Seventh Edition, Part 2 by Andrea Allievi, Alex Ionescu, Mark E. Russinovich, and David A. Solomon. – Microsoft Press, 2021. – ISBN: 978-0-13-546240-9.