# Лабораторна робота № 11.
# Тема: ДАТАГРАМНИЙ ПРОТОКОЛ

**1. Програма SERVERD**

У деяких випадках доцільно використати протокол негарантованої доставки UDP, тому що він, наприклад, допускає одночасне розсилання пакетів всім вузлам мережі (у режимі broadcast).

Якщо вузли обмінюються даними з використанням датаграмного протоколу UDP, їм не потрібно створювати канал даних, тому процедура ініціалізації виходить простіше.

Сервер UDP повинен створити сокет за допомогою функції socket і прив'язати до нього адреса IP, викликавши функцію bind. Клієнт UDP виконує створення й ініціалізацію сокетів аналогічним образом за допомогою все тих же функцій socket та bind.

Такі відомі з попередніх програм функції, як connect, listen та accept, у додатках UDP використати не потрібно.

Для обміну даними програми UDP викликають функції sendto та recvfrom, аналогічні функціям send та recv, але одна відмінність, що мають - при виклику цих функцій їм необхідно задавати додаткові параметри, що мають відношення до адрес вузлів. Функції sendto потрібно вказати адресу, по якому буде відправлений пакет даних, а функції recvfrom - покажчик на структуру, у яку буде записана адреса відправника пакета.

Нижче наведені вихідні тексти програм SERVERD та CLIENTD, які виконують ті ж завдання, що й тільки що розглянуті додатки SERVER та CLIENT, але при цьому вони передають дані за допомогою датаграмного протоколу UDP.

Вихідний текст додатка SERVERD наведений у роздруківці 5.7.

Роздруківка 5.7. Файл serverd/serverd.c

```
#include <windows.h>
#include <windowsx.h>
#include <winsock.h>
#include <commctrl.h>
#include "resource.h"
// Опис функцій
// Функція головного вікна
LRESULT WINAPI
WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);
// Функція для обробки повідомлення WM_CREATE
BOOL WndProc_OnCreate(HWND hWnd, LPCREATESTRUCT IpCreateStruct);
// Функція для обробки повідомлення WM_DESTROY
void WndProc_OnDestroy(HWND hWnd);
// Функція для обробки повідомлення WM_COMMAND
void WndProc_OnCommand (HWND hWnd, int id, HWND hwndCtl, UINT
codeNotify);
// Функція для обробки повідомлення WM_SIZE
void WndProc_OnSize(HWND hWnd, UINT state, int cx, int cy);
// Запуск сервера
void ServerStart(HWND hWnd);
// Останов сервера
void ServerStop(HWND hWnd);
// Обробка повідомлення WSA_NETEVENT
void WndProc_OnWSANetEvent(HWND hWnd, UINT msg, WPARAM wParam, LPARAM
lParam); // Порт сервера
#define SERV_PORT 5000
#define IDS_STATUSBAR 802
// Визначення кодів повідомлень
#define WSA_NETEVENT (WM_USER + 1)
// Глобальні змінні
// Ідентифікатор додатка
HINSTANCE hInst;
// Назва додатка
```

```
char szAppName[] = "WServerUDP";
// Заголовок головного вікна додатка
char szAppTitle[] = "Windows Socket UDP Server Demo";
// Ідентифікатор органа Statusbar
HWND hwndSb;
// Сокет сервера
SOCKET srv_socket;
// Функція WinMain
int APIENTRY
WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
int nCmdShow)
{
WNDCLASSEX wc;
HWND hWnd;
MSG msg;
hInat = hInstance;
// Перевіряємо, чи не був цей Програма запущений раніше
hWnd = FindWindow(szAppName, NULL);
if(hWnd)
{
// Якщо вікно додатка було згорнуто в піктограму,
// відновлюємо його
if(IsIconic(hWnd))
ShowWindow(hWnd, SW_RESTORE);
// Висуваємо вікно додатка на передній план
SetForegroundWindow(hWnd);
return FALSE;
}
// Реєструємо клас вікна
memset(&wc, 0, sizeof(wc));
wc.cbSize = sizeof(WNDCLASSEX);
wc.hIconSm = LoadImage(hInst, MAKEINTRESOURCE(IDI_APPICON_SM),
IMAGE_ICON, 16, 16, 0);
wc.style = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc = (WNDPROC)WndProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = hInst;
wc.hIcon = LoadImage(hInst,
MAKEINTRESOURCE(IDI_APPICON), IMAGE_ICON, 32, 32, 0);
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH) (COLOR_WINDOW +1);
wc.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
wc.lpszClasaName = szAppName;
// Викликаємо функцію RegisterClassEx, що виконує
// реєстрацію вікна
if(!RegisterClassEx(&wc))
i?(!RegisterClass((LPWNDCLASS)&wc.style))
return FALSE;
InitCommonControls();
// Створюємо головне вікно програми
hWnd = CreateWindow(szAppName, szAppTitle, WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInst, NULL);
if(!hWnd) return(FALSE);
// Відображаємо вікно
ShowHindow(hWnd, nCmdShow);
UpdateWindow(hWnd);
// Запускаємо цикл обробки повідомлень
while(GetMessage(&msg, NULL, 0, 0))
{
```

```
TranslateMessage(&msg);
DispatchMessage(&msg);
}
return msg.wParam;
}
// Функція WndProc
LRESULT WINAPI WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
switch(msg)
{
// Викликаємо оброблювач повідомлення WSA_NETEVENT
case WSA_NETEVENT:
WndProc_OnWSANetEvent(hWnd, msg, wParam, lParam);
break;
HANDLE_MSG(hWnd, WM_CREATE, WndProc_OnCreate);
HANDLE_MSG(hWnd, WM_COMMAND, WndProc_OnCommand);
HANDLE_MSG(hWnd, WM_SIZE, WndProc_OnSize);
HANDLE_MSG(hWnd, WM_DESTROY, WndProc_OnDestroy);
default:
return(DefWindowProc(hWnd, msg, wParam, lParam));
}
}
// Функція WndProc_OnCreate
BOOL WndProc_OnCreate(HWND hWnd, LPCREATESTRUCT lpCreateStruct)
{
int rc;
WSADATA WSAData;
char szTemp[128];
// Ініціалізація й перевірка версії Windows Sockets
rc = WSAStartup(MAKEWORD(1, 1), &WSAData);
if(rc != 0)
{
MessageBox(NULL, "WSAStartup Error", "Error", MB_OK);
return FALSE;
}
// Відображаємо опис і версію системи Windows Sockets
//у вікні органа керування Statusbar
wsprintf(szTemp, "Server use %s %s",
WSAData.szDescription,WSAData.szSystemStatus);
hwndSb = CreateStatusWindow(WS_CHILD | WS_VISIBLE | WS_BORDER |
SBARS_SIZEGRIP, szTemp, hWnd, IDS_STATUSBAR);
return TRUE;
}
// Функція WndProc_OnDestroy
#pragma warning(disable: 4098)
void WndProc_OnDestroy(HWND hWnd)
{
// Звільнення ресурсів, отриманих для
// роботи з Windows Sockets WSACleanup();
// Завершення циклу обробки повідомлень
PostQuitMessage(0);
return FORWARD_WM_DESTROY(hWnd, DefWindowProc);
}
// Функція WndProc_OnSize
#pragma warning(disable: 4098)
void
WndProc_OnSize(HWND hWnd, UINT state, int cx, int cy)
{
SendMessage(hwndSb, WM_SIZE, cx, cy);
return FORWARD_WH_SIZE(hWnd, state, cx, cy, DefWindowProc);
```

```
}
// Функція WndProc_OnCommand
#pragma warning(disable: 4098)
void WndProc_OnCommand(HWND hWnd, int id, HWND hwndCtl, UINT codeNotify)
{
switch(id)
{
case IDM_EXIT:
// Знищення головного вікна додатка
DestroyWindow(hWnd);
break;
case IDM_START:
// Запуск сервера
ServerStart(hWnd);
break;
case IDM_STOP:
// Останов сервера
ServerStop(hWnd);
break;
default:
MessageBox(NULL, "Unknown command", "Error", MB_OK);
}
return FORWARD_WM_COMMAND(hWnd, id, hwndCtl, codeNotify, DefWindowProc);
}
// Функція ServerStart
void ServerStart(HWND hWnd)
{
struct sockaddr_in srv_address;
int rc;
// Створюємо сокет сервера для роботи з потоком даних
srv_socket = socket(AF_INET, SOCK_DGRAM, 0);
if(srv_socket == INVALID_SOCKET)
{
MessageBox(NULL, "socket Error", "Error", MB_OK);
return;
}
// Установлюємо адресу IP і номер порту
srv_address.sin_family = AF_INET;
srv_address.sin_addr.s_addr = INADDR_ANY;
srv_address.sin_port = htons(SERV_PORT);
// Зв'язуємо адреса IP із сокетом
if(bind(srv_socket, (LPSOCKADDR)&srv_address, sizeof(srv_address)) ==
SOCKET_ERROR)
{
// При помилці закриваємо сокет
closesocket(srv_socket);
MessageBox(NULL, "bind Error", "Error", MB_OK);
return;
}
// Якщо на даному сокете почнеться передача даних від
// клієнта, у головне вікно додатка надійде
// повідомлення WSA_NETEVENT.
rc = WSAAsyncSelect(srv_socket, hWnd, WSA_NETEVENT, FD_READ);
if(rc > 0)
{
closesocket(srv_socket);
MessageBox(NULL, "WSAAayncSelect Error", "Error", MB_OK);
return;
}
// Виводимо у вікна Statusbar повідомлення про запуск сервера
```

```
SendMessage(hwndSb, SB_SETTEXT, 0, (LPARAM)"Server started");
}
// Функція ServerStop
void ServerStop(HWND hWnd)
{
// Скасовуємо прихід будь-яких повідомлень у головну функцію
// вікна при виникненні будь-яких події, зв'язаних
// с системою Windows Sockets
WSAAsyncSelect(srv_socket, hWnd, 0, 0);
// Якщо сокет був створений, закриваємо його
if(srv_socket != INVALID_SOCKET)
{
closesocket(srv_socket);
}
// Виводимо у вікна Statusbar повідомлення про зупинку сервера
SendMessage(hwndSb, SB_SETTEXT, 0, (LPARAM)"Server stopped");
}
// Функція WndProc_OnWSANetEvent
void WndProc_OnWSANetEvent(HWND hWnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
char szTemp[256];
int rc;
SOCKADDR_IN addr;
int nAddrSize;
char szBuf[80];
LPSTR lpAddr;
if(WSAGETSELECTEVENT(lParam) == FD_READ)
{
// Приймаємо дані
rc = recvfrom((SOCKET)wParam, szTemp, 256, 0, (PSOCKADDR)&addr,
&nAddrSize);
if(rc)
{
szTemp[rc] = '\0';
strcpy(szBuf, "Received from ");
// Перетворюємо адреса IP вилученого клієнта
// у текстовий рядок
lpAddr = inet_ntoa(addr.sin_addr);
strcat(azBuf, lpAddr);
// Відображаємо адресу вилученого клієнта
// й отриману від нього рядок
MesaageBox(NULL, szTemp, szBuf, MB_OK);
}
return;
}
}
```

Програма SERVERD багато в чому нагадує Програма SERVER, тому можна розглянути тільки відмінності.

Перша відмінність полягає в тім, що при запуску сервера тип створюваного сокету вказується як SOCK_DGRAM:

**srv_socket = socket(AF_INET, SOCK_DGRAM, 0);**

Далі виконується ініціалізація сокету і його прив'язка до адреси, для чого викликається функція bind. Ця операція, як й у випадку протоколу TCP, необов'язкова. Після виконання прив'язки можна приступати до одержання пакетів даних від клієнта. Для того щоб не виконувати очікування пакетів у циклі, Програма використає функцію

WSAAsyncSelect, указуючи з її допомогою, що при одержанні пакетів даних головне вікно додатка повинне одержувати повідомлення з кодом WSA_NETEVENT:

```
rc = WSAAsyncSelect(srv_socket, hWnd, WSA_NETEVENT, FD_READ);
```

На цьому ініціалізація сервера завершується.
Оброблювач повідомлення WSA_NETEVENT читає отриманий пакет за допомогою функції recvfrom:

```
SOCKADDR_IN addr;
int nAddrSize;
rc = recvfrom((SOCKET)wParam, szTemp, 256, 0, (PSOCKADDR)&addr, &nAddrSize);
```

Як передостанній параметр цієї функції передається адреса структури типу SOCKADDR_IN, куди функція записує адресу вузла, що надіслав пакет. Останній параметр функції recvfrom повинен містити розмір зазначеної структури. Нижче наведені можливі коди помилок для функції recvfrom.

| **Код помилки** | **Опис** |
|---|---|
| **WSANOTINITIALISED** | Перед використанням функції необхідно викликати функцію **WSAStartup** |
| **WSAENETDOWN** | Збій у мережі |
| **WSAEFAULT** | Занадто мале значення параметра, що визначає розмір буфера для прийому даних |
| **WSAEINTR** | Робота функції була скасована за допомогою функції WSACancelBlockingCall |
| **WSAEINPROGRESS** | Виконується функція, що блокує, інтерфейсу Windows Sockets |
| **WSAEINVAL** | Сокет не був підключений функцією bind |
| **WSAENOTSOCK** | Зазначений дескриптор не є дескриптором сокету |
| **WSAESHUTDOWN** | Сокет був закритий функцією shutdown |
| **WSAEWOULDBLOCK** | Сокет відзначений як що не блокує, але запитана операція приведе до блокування |
| **WSAEMSGSIZE** | Розмір датаграми занадто великий, тому відповідний блок даних не міститься в буфер. Прийнятий блок даних був обрізаний |
| **WSAECONNABORTED** | Збій через занадто велику затримку або з іншої причини |
| **WSAECONNRESET** | Скидання з'єднання вилученим вузлом |

При обміні даними з використанням протоколу UDP на кожен виклик функції sendto повинен доводитися один виклик функції recvfrom. Якщо ж передаються дані через канал з використанням протоколу TCP, на один виклик функції send може доводитися кілька викликів функції recv.
Для відображення адреси вузла, що послав пакет UDP, Програма перетворить ця адреса в символьний рядок за допомогою функції inet_ntoa:

```
lpAddr = inet_ntoa(addr.sin_addr);
```

Ця функція записує отриманий рядок у статичну область пам'яті, що належить системі Windows Sockets, тому для подальшого використання необхідно скопіювати рядок до наступного виклику будь-якої функції програмного інтерфейсу Windows Sockets.

### 2. Програма CLIENTD

Вихідні тексти додатка CLIENTD, призначеного для спільної роботи з додатком SERVERD, представлені в роздруківці 5.8. Тому що ця програма дуже схоже на програму CLIENT, будуть описані тільки основні відмінності.

Роздруківка 5.8. Файл clientd/clientd.c

```
#include <windows.h>
#include <windowsx.h>
#include <winsock.h>
#include <commctrl. h>
#include "resource.h"
// Опис функцій
// Функція головного вікна
WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);
// Функція для обробки повідомлення WM_CREATE
BOOL WNDProc_OnCreate(HWND hWnd, LPCREATESTRUCT IpCreateStruct);
// Функція для обробки повідомлення WM_DESTROY
void WndProc_OnDestroy(HWND hWnd);
// Функція для обробки повідомлення WM_COMMAND
void WndProc_OnCommand(HWND hWnd, int id, HWND hwndCtl, UINT codeNotify);
// Функція для обробки повідомлення WM_SIZE
void WndProc_OnSize(HWND hWnd, UINT state, int cx, int cy);
// Установка з'єднання
void SetConnection(HWND hWnd);
// Передача повідомлення
void SendMsg(HWND hWnd);
// Порт сервера
#define SERV_PORT 5000
#define IDS_STATUSBAR 802
// Глобальні змінні
// Ідентифікатор додатка
HINSTANCE hInst;
// Назва додатка
char szAppName[] = "WClientUDP";
// Заголовок головного вікна додатка
char azAppTitle[] = "Windows Socket UDP Client Demo";
// Ідентифікатор органа керування Statusbar
HWND hwndSb;
// Сокет клієнта
SOCKET srv_socket;
// Адреса сервера
SOCKADDR_IN dest_sin;
// Функція WinMain
int APIENTRY
WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
int nCmdShow)
{
WNDCLASSEX wc;
HWND hWnd;
HSG msg;
hInst = hInstance;
// Перевіряємо, чи не був цей Програма запущений раніше
hWnd = FindWindow(szAppName, NULL);
if(hWnd)
{
// Якщо вікно додатка було згорнуто в піктограму,
// відновлюємо його
if(IsIconic(hWnd))
ShowWindow(hWnd, SW_RESTORE);
// Висуваємо вікно додатка на передній план
SetForegroundWindow(hWnd);
return FALSE;
}
// Реєструємо клас вікна
memset(&wc, 0, sizeof(wc));
wc.cbSize = sizeof(WNDCLASSEX);
```

```
wc.hIconSm = LoadImage(hInet, MAKEINTRESOURCE(IDI_APPICON_SM),
IMAGE_ICON, 16, 16, 0);
wc.style = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc = (WNDPROC)WndProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = hInst;
wc.hIcon = LoadImage(hInst, MAKEINTRESOURCE(IDI_APPICON), IMAGE_ICON, 32,
32, 0);
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH) (COLOR_WINDOW+1);
wc.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
wc.lpszClassName = szAppName;
// Викликаємо функцію RegisterClassEx, що виконує
// реєстрацію вікна
if(!RegisterClassEx(&wc))
if(!RegisterClass((LPWNDCLASS)&wc.style))
return FALSE;
InitCommonControls();
// Створюємо головне вікно додатка
hWnd = CreateWindow(szAppName, szAppTitle, WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInst, NULL);
if(!hWnd) return(FALSE);
// Відображаємо вікно
ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);
// Запускаємо цикл обробки повідомлень
while(GetMessage(&msg, NULL, 0, 0))
{
TranslateMessage(&msg);
DispatchMessage(&msg);
}
return msg.wParam;
}
// Функція WndProc
LRESULT WINAPI
WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
switch(msg)
{
HANDLE_MSG(hWnd, WM_CREATE, WndProc_OnCreate);
HANDLE_MSG(hWnd, WM_COMMAND, WndProc_OnCommand);
HANDLE_MSG(hWnd, WM_SIZE, WndProc_OnSize);
HANDLE_MSG(hWnd, WM_DESTROY, WndProc_OnDestroy);
default:
return(DefWindowProc(hWnd, msg, wParam, lParam));
}
}
// Функція WndProc_OnCreate
BOOL WndProc_OnCreate(HWND hWnd, LPCREATESTRUCT lpCreateStruct)
{
int rc;
WSADATA WSAData;
char szTemp[128];
// Ініціалізація й перевірка версії Windows Sockets
rc = WSAStartup(MAKEWORD(1, 1), &WSAData);
if(rc != 0)
{
MessageBox(NULL, "WSAStartup Error", "Error", MB_OK);
return FALSE;
```

```
}
// Відображаємо опис і версію системи Windows Sockets
// у вікні органа керування Statusbar
wsprintf(szTemp, "Server use %s %s", WSAData.szDescription,
WSAData.szSystemStatus);
hwndSb = CreateStatusWindow(WS_CHILD | WS_VISIBLE | WS_BORDER |
SBARS_SIZEGRIP, szTemp, hWnd, IDS_STATUSBAR);
return TRUE;
}
// Функція WndProc_OnDestroy
#pragma warning(disable: 4098)
void WndProc_OnDestroy(HWND hWnd)
{
// Звільнення ресурсів, отриманих для
// роботи з Windows Sockets WSACleanup();
// Завершення циклу обробки повідомлень
PostQuitMessage(0);
return FORWARD_WM_DESTROY(hWnd, DefWindowProc);
}
// Функція WndProc_OnSize
#pragma warning(disable: 4098)
void WndProc_OnSize(HWND hWnd, UINT state, int cx, int cy)
{
SendMessage(hwndSb, WM_SIZE, cx, cy);
return FORWARD_WM_SIZE(hWnd, state, cx, cy, DefWindowProc);
}
// Функція WndProc_OnCommand
#pragma warning(disable: 4098)
void WndProc_OnCommand(HWND hWnd, int id, HWND hwndCtl, UINT codeNotify)
{
switch(id)
{
case IDM_EXIT:
// Знищення головного вікна додатка
DestroyWindow(hWnd);
break;
case IDM_CONNECT:
// Установка з'єднання із сервером
SetConnection(hWnd);
break;
case IDM_SEND:
// Посилка повідомлення серверу
SendMsg(hWnd);
break;
default:
MessageBox(NULL, "Unknown command", "Error", MB_OK);
}
return FORWARD_WM_COMMAND(hWnd, id, hwndCtl, codeNotify, DefWindowProc);
}
// Функція SetConnection
void SetConnection(HWND hWnd)
{
PHOSTENT phe;
// Створюємо сокет
srv_socket = socket(AF_INET, SOCK_DGRAM, 0);
if(srv_socket == INVALID_SOCKET)
{
MessageBox(NULL, "socket Error", "Error", MB_OK);
return;
}
```

```
// Зв'язуємо адреса IP із сокетом
dest_sin.sin_family = AF_INET;
dest_sin.sin_addr.s_addr = INADDR_ANY;
dest_sin.sin_port = 0;
if(bind(srv_socket, (LPSOCKADDR)&dest_sin, sizeof(dest_sin)) ==
SOCKET_ERROR)
{
// При помилці закриваємо сокет
closesocket(srv_socket);
MessageBox(NULL, "bind Error", "Error", MB_OK);
return;
}
// Установлюємо адресу IP і номер порту
dest_sin.sin_family = AF_INET;
// Визначаємо адреса вузла
// Адреса локального вузла для налагодження
phe = gethostbyname("localhost");
// Адреса вилученого вузла
// phe = gethostbyname("maxsinev");
if(phe == NULL)
{
closesocket(srv_socket);
MessageBox(NULL, "gethostbyname Error", "Error", MB_OK);
return;
}
// Копіюємо адресу вузла
memcpy((char FAR *)&(dest_sin.sin_addr), phe->h_addr, phe->h_length);
// Інший спосіб вказівки адреси вузла
// dest_sin.sin_addr.s_addr = inet_addr("200.200.200.201");
// Копіюємо номер порту
dest_sin.sin_port = htons(SERV_PORT);
// У випадку успіху виводимо повідомлення про установку
// з'єднання з вузлом
SendMessage(hwndSb, SB_SETTEXT, 0, (LPARAM)"Connected");
}
// Функція SendMsg
void SendMsg(HWND hWnd)
{
char szBuf[80];
lstrcpy(szBuf, "Test string");
// Посилаємо повідомлення
sendto(srv_socket, szBuf, lstrlen(szBuf), 0, (PSOCKADDR)&dest_sin,
sizeof(dest_sin));
}
```

Функція SetConnection створює сокет типу SOCK_DGRAM, тому що передача даних буде виконуватися з використанням протоколу UDP:

**srv_socket = socket(AF_INET, SOCK_DGRAM, 0);**

Далі виконується прив'язка сокету до адреси за допомогою функції bind. При цьому вказується нульове значення порту й адреса INADDR_ANY, тому що на даному етапі ці параметри не мають значення.

Потім функція SetConnection записує адресу сервера в структуру dest_sin. Ця адреса буде потрібно для передачі повідомлень серверу.

При використанні протоколу UDP й якщо не створений канал між додатками, для передачі даних варто використати функцію sendto:

**sendto(srv_socket, szBuf, lstrlen(szBuf), 0, (PSOCKADDR)&dest_sin, sizeof(dest_sin));**

Як передостанній параметр цієї функції потрібно передати адресу заповненої структури, що містить адреса вузла, куди буде посилати пакет даних. Через останній параметр функції sendto необхідно передати розмір зазначеної структури.
Приведем список можливих кодів помилок для функції sendio:

| Код помилки | Опис |
|---|---|
| **WSANOTINITIALISED** | Перед використанням функції необхідно викликати функцію WSAStartup |
| **WSAENETDOWN** | Збій у мережі |
| **WSAEACCES** | Не був установлений прапор широкомовної адреси |
| **WSAEINTR** | Робота функції була скасована за допомогою функції WSACancelBlockingCall |
| **WSAEINPROGRESS** | Виконується функція, що блокує, інтерфейсу Windows Sockets |
| **WSAEFAULT** | Неправильно зазначена адреса буфера, що містить передані дані |
| **WSAENETRESET** | Необхідно скинути з'єднання |
| **WSAENOBUFS** | Відбулося зациклення буферів |
| **WSAENOTSOCK** | Зазначений дескриптор не є дескриптором сокета |
| **WSAESHUTDOWN** | Сокет був закритий функцією shutdown |
| **WSAEWOULDBLOCK** | Сокет відзначений як не блокує, але запитана операція приведе до блокування |
| **WSAEMSGSIZE** | Розмір датаграми більше, ніж це допускається даною реалізацією інтерфейсу Windows Sockets |
| **WSAECONNABORTED** | Збій через занадто велику затримку або з іншої причини |
| **WSAECONNRESET** | Скидання з'єднання вилученим вузлом |
| **WSAEADDRNOTAVAIL** | Зазначена адреса недоступна |
| **WSAEAFNOSUPPORT** | Даний тип сокету не може працювати із зазначеним сімейством адрес |
| **WSAEDESTADDRREQ** | Необхідно вказати адресу одержувача датаграми |
| **WSAENETUNREACH** | Тепер і з даного вузла неможливо одержати доступ до мережі. |

Відмітимо, що клієнт може створити канал зв'язку із сервером, викликавши функцію connect, і передавати по цьому каналі пакети UDP, користуючись функціями send і recv. Цей спосіб зручний тим, що при передачі пакета не потрібно щораз указувати адресу одержувача.

**3. Індивідуальні завдання на роботу**

1. Індивідуальне завдання для кожного студента. Взяти з попередньої роботи для передачі пакети двох типів: а)в одному фрагменті; б)у фрагментах, число яких дорівнює числу, обумовленому останніми двома цифрами в заліковій книжці студента.
У якості пересилають даних, що, використати на робочій станції:
№1 – текстові файли у форматі MS DOS;
№2 - файли об'єктних модулів у форматі C;
№3 – упаковані за допомогою архіватора «arj.exe» файли;
№4 – упаковані за допомогою архіватора «pkzip.exe» файли;
№5 – виконавчі «exe»-файли;
№6 – командні «bat»-файли;
№7 –БД-файлы у форматах Access, Interbase;
№8 – графічні файли у форматах BMP, GIF, JPG.
Файли кожного студента повинні бути індивідуальними.
2. Індивідуальне завдання для кожного студента. Організувати роботу системи «сервер-сервер-клієнт-сервер» з використанням циклічного виклику функції accept для підготовлених індивідуальних пакетів, що пересилають. Одна із програм «клієнт» або «сервер» розміщається на робочій станції з номером «№-1», де «№» - номер власної робочої станції (при «№-1», рівному нулю, номер іншої станції приймається рівним «8»). Клієнт спочатку звертається до сервера по відомому з попередньої роботи індивідуальному сокету

студента через широкомовну адресу станції. Після відповіді сервера клієнт використає для одержання чергових пакетів повна адреса сервера. Сервер використає для відповіді адреса клієнта, отримана із запиту клієнта.

3. Індивідуальне завдання для кожного студента. Модифікувати роботу системи «сервер-сервер-клієнт-сервер» з п. 2.2.3.2., об'єднавши й клієнта й сервера в одній програмі. Зробити запуск системи на: а)двох різних робітників станціях - «№» й «№ - 1», де «№» - номер власної робочої станції (при «№ - 1», рівному нулю, номер іншої станції приймається рівним «8»); б)однієї робочої станції зі зверненням програми через мережу самої до себе.

4. Індивідуальне завдання для кожного студента. Організувати роботу системи «клієнт-сервер» з використанням виклику функції WSAAsyncSelect для підготовлених індивідуальних пакетів, що пересилають. Одна із програм «клієнт» або «сервер» розміщається на робочій станції з номером «№ - 1», де «№» - номер власної робочої станції (при «№ - 1», рівному нулю, номер іншої станції приймається рівним «8»). Клієнт спочатку звертається до сервера по відомому з попередньої роботи індивідуальному сокету студента через широкомовну адресу станції. Після відповіді сервера клієнт використає для одержання чергових пакетів повна адреса сервера. Сервер використає для відповіді адреса клієнта, отримана із запиту клієнта.

5. Загальне завдання для всіх студентів. За результатами лабораторної роботи підготувати повний протокол, що включає формулювання пунктів завдання, коротке опис реалізації кожного з пунктів завдання із фрагментами коду, висновки по кожному із пунктів завдання.

## Теоретичні питання.

1. Призначення та параметри функції `connect`?
2. Призначення та параметри функції `send`?
3. Призначення та параметри функції `recv`?
4. Призначення та параметри функції `accept`?
5. Які можуть виникати помилки при виклику команди `send`?
6. Які можуть виникати помилки при виклику команди `recv`?
7. Призначення та параметри функції `listen`?
8. Які можуть виникати помилки при виклику команди `accept`?
9. Які можуть виникати помилки при виклику команди `listen`?
10. Як здійснюється передача та прийом даних?