 

**Lab 5 i lab 6.**

**Zajmiemy się dokładniej komunikacją za pomocą tzw. gniazd**. Sama idea gniazd została opracowana na Uniwersytecie Kalifornijskim w Berkley w celu zapewnienia możliwości komunikacyjnych w systemie Unix. Opracowany tam interfejs programowania nosi nazwę "Berkley socket interface" i jest stopniowo z mniejszymi lub większymi zmianami przejmowany przez kolejne systemy operacyjne. Największa zaleta jaką daje korzystanie z gniazd to w miarę prosty i przejrzysty interfejs niezależny od warstwy komunikacyjnej (oczywiście stosunkowo najmniej wygodnie korzysta się z interfejsu gniazd w czystym C, w Javie czy C# jest jeszcze prościej).

**1.Podstawy biblioteki Winsock**

Gniazda zaadaptowano do systemu Windows pod nazwą **Winsock**. Za pomocą tej biblioteki można tworzyć aplikacje korzystające z wielu protokołów sieciowych. Dzięki prostemu interfejsowi, z perspektywy programisty korzystanie z różnych protokołów wygląda niemal identycznie.

Przy nawiązywaniu połączeń jedna strona jest "serwerem", który oczekuje połączenia, druga strona jest "klientem", który nawiązuje połączenie z "serwerem".

Poniżej zaprezentowano kod prostego serwera i klienta, którzy wymieniają się wiadomościami. Wykorzystano protokół TCP/IP.

Serwer w pętli oczekuje na klientów i po nawiązaniu połączenia tworzy nowy wątek, który zajmuje się odbieraniem komunikatów i odsyłaniem ich z powrotem (to tylko przykład!). Klient wymaga oczywiście parametru, którym jest nazwa serwera. Klient nawiązuje połączenie i wysyła do serwera komunikat, po czym czeka na jego echo.

Kod programu serwera:

// prosty moduł serwera

// komunikacji za pomocą Winsock

// użycie: server.exe

#include <winsock2.h>

#include <stdio.h>

#include <stdlib.h>

#define DEFAULT\_PORT 5000

#define DEFAULT\_BUFFER 4096

// tylko Visual C++

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

// funkcja: wątek do komunikacji z klientem

DWORD WINAPI ClientThread(LPVOID lpParam)

{

SOCKET sock = (SOCKET)lpParam;

char szBuf[DEFAULT\_BUFFER];

int ret,

nLeft,

idx;

// serwer będzie oczekiwał na informacje od klienta

while(1)

{

// najpierw odbierz dane

ret = recv(sock, szBuf, DEFAULT\_BUFFER, 0);

if (ret == 0)

break;

else if (ret == SOCKET\_ERROR)

{

printf("błąd funkcji recv(): %d\n", WSAGetLastError());

break;

}

szBuf[ret] = '\0';

printf("RECV: '%s'\n", szBuf);

// następnie odeślij te dane, poporcjuj jeśli trzeba

// (niestety send() może nie wysłać wszystkiego)

nLeft = ret;

idx = 0;

while(nLeft > 0)

{

ret = send(sock, &szBuf[idx], nLeft, 0);

if (ret == 0)

break;

else if (ret == SOCKET\_ERROR)

{

printf("błąd funkcji send(): %d\n", WSAGetLastError());

break;

}

nLeft -= ret;

idx += ret;

}

}

return 0;

}

int main(int argc, char \*argv[])

{

WSADATA wsd;

SOCKET sListen,

sClient;

int iAddrSize;

HANDLE hThread;

DWORD dwThreadID;

struct sockaddr\_in local, client;

struct hostent \*host = NULL;

// inicjuj Winsock 2.2

if (WSAStartup(MAKEWORD(2,2), &wsd) != 0)

{

printf("Błąd ładowania Winsock 2.2!\n");

return 1;

}

// twórz gniazdo do nasłuchu połączeń klientów

sListen = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_IP);

if (sListen == SOCKET\_ERROR)

{

printf("Błąd funkcji socket(): %d\n", WSAGetLastError());

return 1;

}

// wybierz interfejs (warstwę komunikacyjną)

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

local.sin\_family = AF\_INET;

local.sin\_port = htons(DEFAULT\_PORT);

if (bind(sListen, (struct sockaddr \*)&local, sizeof(local)) == SOCKET\_ERROR)

{

printf("Błąd funkcji bind(): %d\n", WSAGetLastError());

return 1;

}

// nasłuch

host = gethostbyname("localhost");

if (host == NULL)

{

printf("Nie udało się wydobyć nazwy serwera\n");

return 1;

}

listen(sListen, 8);

printf("Serwer nasłuchuje.\n");

printf("Adres: %s, port: %d\n", host->h\_name, DEFAULT\_PORT);

// akceptuj nadchodzące połączenia

while (1)

{

iAddrSize = sizeof(client);

sClient = accept(sListen, (struct sockaddr \*)&client, &iAddrSize);

if (sClient == INVALID\_SOCKET)

{

printf("Błąd funkcji accept(): %d\n", WSAGetLastError());

return 1;

}

printf("Zaakceptowano połączenie: serwer %s, port %d\n",

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

hThread = CreateThread(NULL, 0, ClientThread,

(LPVOID)sClient, 0, &dwThreadID);

if (hThread == NULL)

{

printf("Błąd funkcji CreateThread(): %d\n", WSAGetLastError());

return 1;

}

CloseHandle(hThread);

}

closesocket(sListen);

WSACleanup();

return 0;

}

Kod programu klienta:

// prosty moduł klienta

// komunikacji za pomocą Winsock

// użycie: klient.exe -s:IP

#include <winsock2.h>

#include <stdio.h>

#include <stdlib.h>

#define DEFAULT\_COUNT 5

#define DEFAULT\_PORT 5000

#define DEFAULT\_BUFFER 4096

#define DEFAULT\_MESSAGE "Wiadomość testowa, Systemy Operacyjne 2002"

// tylko Visual C++

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

char szServer[128], szMessage[1024];

// funkcja sposob\_uzycia

void sposob\_uzycia()

{

printf("Klient.exe -s:IP\n");

ExitProcess(1);

}

void WalidacjaLiniiPolecen(int argc, char \*\*argv)

{

int i;

if (argc < 2)

{

sposob\_uzycia();

}

for (i=1; i<argc; i++)

{

if (argv[i][0] == '-')

{

switch (tolower(argv[i][1]))

{

case 's':

if (strlen(argv[i]) > 3)

strcpy(szServer, &argv[i][3]);

break;

default:

sposob\_uzycia();

break;

}

}

}

}

int main(int argc, char \*argv[])

{

WSADATA wsd;

SOCKET sClient;

char szBuffer[DEFAULT\_BUFFER];

int ret, i;

struct sockaddr\_in server;

struct hostent \*host = NULL;

// linia poleceń

WalidacjaLiniiPolecen(argc, argv);

// inicjuj Winsock 2.2

if (WSAStartup(MAKEWORD(2,2), &wsd) != 0)

{

printf("Błąd ładowania Winsock 2.2!\n");

return 1;

}

strcpy(szMessage, DEFAULT\_MESSAGE);

// twórz gniazdo do nasłuchu połączeń klientów

sClient = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_IP);

if (sClient == INVALID\_SOCKET)

{

printf("Błąd funkcji socket(): %d\n", WSAGetLastError());

return 1;

}

// wybierz interfejs

server.sin\_addr.s\_addr = inet\_addr(szServer);

server.sin\_family = AF\_INET;

server.sin\_port = htons(DEFAULT\_PORT);

// jeśli adres nie był w postaci xxx.yyy.zzz.ttt

// to spróbuj go wydobyć z postaci słownej

if (server.sin\_addr.s\_addr == INADDR\_NONE)

{

host = gethostbyname(szServer);

if (host == NULL)

{

printf("Nie udało się wydobyć nazwy serwera: %s\n", szServer);

return 1;

}

CopyMemory(&server.sin\_addr, host->h\_addr\_list[0], host->h\_length);

}

if (connect(sClient, (struct sockaddr \*)&server, sizeof(server)) == SOCKET\_ERROR)

{

printf("Błąd funkcji connect(): %d\n", WSAGetLastError());

return 1;

}

// wysyłaj i odbieraj dane

for (i=0; i<DEFAULT\_COUNT; i++)

{

ret = send(sClient, szMessage, strlen(szMessage), 0);

if (ret == 0)

break;

else if (ret == SOCKET\_ERROR)

{

printf("Błąd funkcji send(): %d\n", WSAGetLastError());

return 1;

}

printf("Wysłano %d bajtów\n", ret);

ret = recv(sClient, szBuffer, DEFAULT\_BUFFER, 0);

if (ret == 0)

break;

else if (ret == SOCKET\_ERROR)

{

printf("Błąd funkcji recv(): %d\n", WSAGetLastError());

return 1;

}

szBuffer[ret] = '\0';

printf("RECV [%d bajtów]: '%s'\n", ret, szBuffer);

}

closesocket(sClient);

WSACleanup();

return 0;

}

Komentarz:  
Struktura obu programów jest dość podobna. Oba tworzą odpowiednie gniazda, przy czym:

* serwer najpierw przypisuje gniazdu adres (za pomocą funkcji **bind()**), a następnie wchodzi w tryb nasłuchu (**listen()**), gdzie zatrzymuje się czekając na połączenia klientów (**accept()**).
* klient nawiązuje połączenie z serwerem za pomocą funkcji **connect()**.

Oba programy wymieniają się krótkimi informacjami za pomocą funkcji **send()** i **recv()**. Serwer nasłuchuje na zadanym porcie, zaś po nawiązaniu połączenia z klientem tworzy **nowe gniazdo i nowy wątek**, po czym komunikuje się przez nowo utworzone gniazdo w nowym wątku. Dzięki temu serwer może obsługiwać wielu klientów jednocześnie, bez względu na to jak długo trwałoby obsługiwanie pojedynczego klienta.

Zadania:

Część A.

* zapoznać się z konstrukcją obu programów,
* zapoznać się z opisami nowych funkcji systemowych używanych w obu programach
* uruchomić oba programy na jednym komputerze
* uruchomić oba programy w sieci lokalnej
* uruchamiać jednocześnie więcej niż jednego klienta do tego samego serwera

Część B.

Użyć mechanizmu gniazd do przesyłania plików binarnych. Zaprogramować odpowiednie programy serwera i klienta. Serwer nasłuchuje na zadanym porcie, po nawiązaniu połączenia z klientem wysyła mu

* nazwę oraz pozostałe atrybuty jakiegoś pliku binarnego
* zawartość tego pliku

Klient odtwarza plik wedle informacji serwera.