



Politechnika Łódzka

Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki

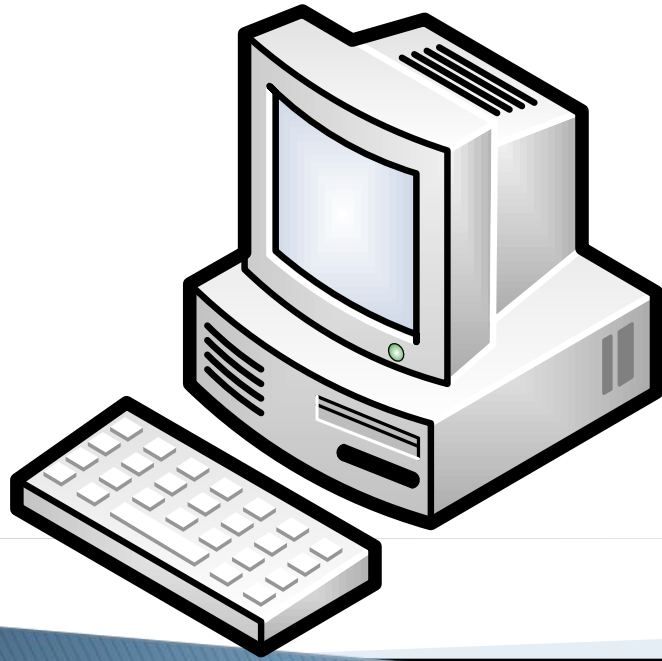
Programowanie sieciowe

Wykład 6
Komunikacja z użyciem protokołu UDP



Instytut Informatyki Stosowanej
Politechniki Łódzkiej

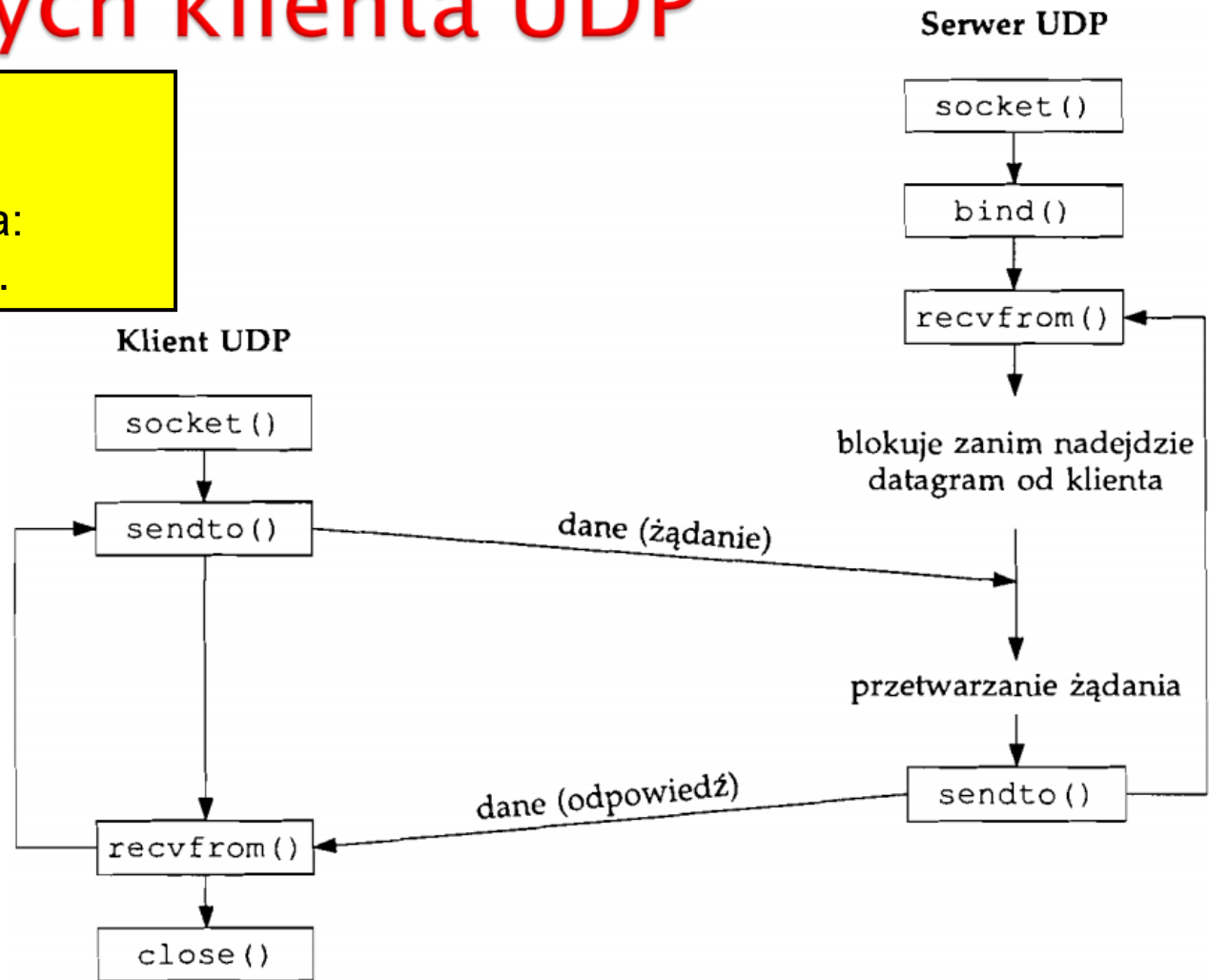
Opracował: dr hab. inż. Radosław Wajman



Klient UDP

Kolejność wykonywania funkcji gniazдовых klienta UDP

Protokół UDP jest bezpołączeniowy:
Nie używana jest funkcja:
accept, connect, listen.



Klient: Tworzenie gniazda [1]

`socket` ▶ `sendto` ▶ `recvfrom` ▶ `close`

```
int socket(int family, int type, int proto);
```

Funkcja tworzy nowe gniazdo w systemie i konfiguruje je do zadanego protokołu.

- ▶ *family* – rodzina protokołów:
 - **AF_INET** – protokół IPv4,
 - **AF_INET6** – protokół IPv6
- ▶ *type* – rodzaj gniazda:
 - **SOCK_STREAM** – gniazdo strumieniowe,
 - **SOCK_DGRAM** – gniazdo datagramowe,
 - **SOCK_RAW** – gniazdo surowe (raw),
- ▶ *proto* – protokół (dla *type*=**SOCK_RAW**):
 - **0** – Domyślny protokół (**SOCK_STREAM**=TCP, **SOCK_DGRAM**=UDP),
- ▶ **Wynik:** **uchwyt gniazda**, lub:
 - **INVALID_SOCKET**, kod błędu z *WSAGetLastError* (Windows),
 - **-1**, kod błędu z *errno* (Unix)

Klient: Zapisywanie danych do gniazda

socket ▶ **sendto** ▶ recvfrom ▶ close



```
int sendto(int sock, const char* buf, int len, int flags,
           sockaddr *to, int tolen);
```

Funkcja zapisuje dane do bufora nadawczego gniazda

- ▶ *sock* - uchwyt gniazda (zwrócony przez **socket** lub **accept**),
- ▶ *buf* - wskaźnik do bufora zawierającego dane do wysłania,
- ▶ *buflen* - ilość bajtów do wysłania,
- ▶ *flags* - flagi, domyślnie 0,
- ▶ *to* - wskaźnik na strukturę **sockaddr** przechowującą adres i port odbiorcy,
- ▶ *tolen* - długość, w bajtach, struktury **sockaddr** odbiorcy.
- ▶ **Wynik:** ilość wysłanych bajtów (**blocking**) lub ilość wysłanych bajtów \leq *buflen* (**nonblocking**) lub:
 - **SOCKET_ERROR**, kod błędu z *WSAGetLastError* (Windows),
 - **-1**, kod błędu z *errno* (Unix)
- ▶ **Blocking:** **sendto** czeka, aż w buforze nadawczym będzie wystarczająco dużo miejsca na wysłanie *buflen* bajtów
- ▶ **Nonblocking:** **sendto** zapisuje do bufora tyle, ile może (nie mniej niż 1) i zwraca ilość zapisanych bajtów. W przypadku braku miejsca w buforze, **sendto** zwraca **SOCKET_ERROR** a kod błędu = **WSAEWOULDBLOCK/EOULDBLOCK**.

Klient: Wczytywanie danych z gniazda

socket ► sendto ► **recvfrom** ► close



Odbieranie
danych!

```
int recvfrom(int sock, char *buf, int buflen, int flags,  
             sockaddr *from, int fromlen);
```

Funkcja odczytuje dane z bufora odbiorczego gniazda

- *sock* – uchwyt gniazda (zwrócony przez **socket** lub **accept**),
- *buf* – wskaźnik do bufora docelowego,
- *buflen* – ilość bajtów do odczytania,
- *flags* – flagi, domyślnie 0,
- *from* – wskaźnik na strukturę **sockaddr** dla adresu i port nadawcy,
- *fromlen* – długość, w bajtach, struktury **sockaddr** nadawcy.

- **Wynik:** $1 \leq \text{ilość wysłanych bajtów} \leq \text{buflen}$, lub:
 - **0** – gdy połączenie zostało zerwane lub zdalnie zamknięte,
 - **SOCKET_ERROR**, kod błędu z *WSAGetLastError* (Windows),
 - **-1**, kod błędu z *errno* (Unix)

UWAGA! Błąd **WSAEMSGSIZE/EMSGSIZE** oznacza, że bufor danych przeznaczony do odbioru był mniejszy niż odebrany datagram.

- **Blocking:** **read** czeka, aż w buforze odbiorczym będzie minimum (domyślnie 1) bajtów do pobrania
- **Nonblocking:** **read** wczytuje tyle danych, ile zostało odebrano (nie mniej niż 1) i zwraca ilość wczytanych bajtów. W przypadku braku danych w buforze, **read** zwraca **SOCKET_ERROR** a kod błędu = **WSAEWOULDBLOCK/EWOULDBLOCK**.

Klient: Zamykanie połączenia

socket ► sendto ► recvfrom ► **close**

```
int closesocket(int sock);           // windows  
int close(int sock);                 // unix
```

Funkcja zamyka gniazdo a wraz z nim połączenie. Wszystkie aktywne operacje związane z gniazdem są anulowane.

- *sock* – uchwyt gniazda (zwrócony przez **socket** lub **accept**),
- **Wynik:** **0** gdy gniazdo zostało zamknięte, lub:
 - **SOCKET_ERROR**, kod błędu z *WSAGetLastError* (Windows),
 - **-1**, kod błędu z *errno* (Unix)

Klient UDP: Implementacja w C++

```
int main(int argc, char* argv[])
{
    WSADATA data;
    int result;
    char datagram[1024];

    result = WSAStartup(MAKESOCK(2, 0), &data);
    assert(result == 0);

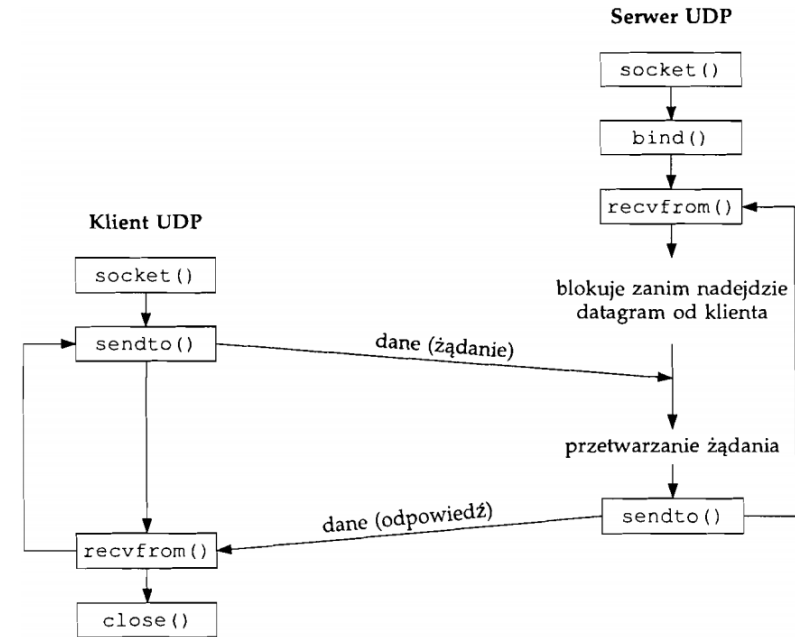
    SOCKET sock = socket(AF_INET, SOCK_DGRAM, 0);
    assert(sock != INVALID_SOCKET);

    sockaddr_in addr;
    int sa_size = sizeof(sockaddr_in);
    addr.sin_family = AF_INET;
    addr.sin_port = htons(3302);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    result = sendto(sock, datagram, 0, 0, (sockaddr*)&addr, sizeof(sockaddr_in));
    result = recvfrom(sock, datagram, sizeof(datagram), 0, (sockaddr*)&addr, &sa_size);

    printf(datagram);

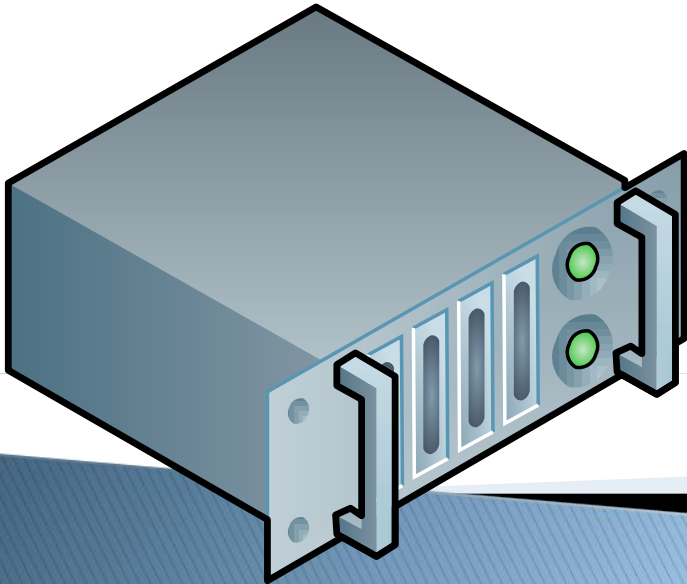
    closesocket(sock);
    return 0;
}
```



Klient UDP: Implementacja w C#

```
1  using System;
2  using System.Net;
3  using System.Net.Sockets;
4  using System.Text;
5
6  class UDPClient
7  {
8      public static void Main(string[] args) throws Exception
9      {
10         Boolean done = false;
11         Boolean exception_thrown = false;
12
13         Socket sending_socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
14         IPAddress send_to_address = IPAddress.Parse("127.0.0.1");
15         IPEndPoint sending_end_point = new IPEndPoint(send_to_address, 3301);
16         string received_data;
17
18         while (!done)
19         {
20             Console.WriteLine("Enter text to send, blank line to quit");
21             string text_to_send = "hello";
22             byte[] send_buffer = Encoding.ASCII.GetBytes(text_to_send);
23             sending_socket.Send(send_buffer, send_buffer.Length, sending_end_point);
24
25             byte[] receive_byte_array = listener.Receive(sending_end_point);
26             string received_data = Encoding.ASCII.GetString(receive_byte_array, 0, receive_byte_array.Length);
27             Console.WriteLine(received_data);
28         }
29     }
30 }
```

Server UDP



Serwer UDP: Implementacja w C++

```
int main(int argc, char* argv[])
{
    WSADATA data;
    int result, counter = 0;
    char line1[100], line2[100], line3[100];
    char datagram[1024];

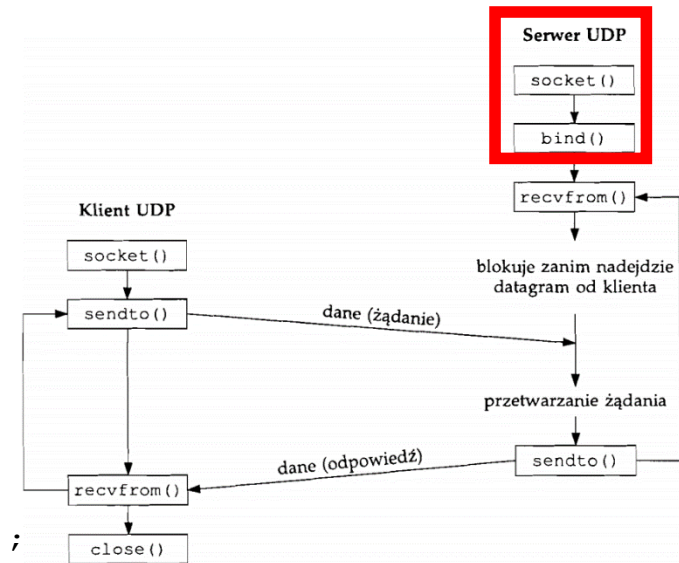
    result = WSAStartup(MAKESOCK(2, 0), &data);
    assert(result == 0);

    SOCKET server_socket = socket(AF_INET, SOCK_DGRAM, 0);
    assert(server_socket != INVALID_SOCKET);

    sockaddr_in service;
    service.sin_family = AF_INET;
    service.sin_port = htons(3302);
    service.sin_addr.s_addr = INADDR_ANY;
    result = bind(server_socket, (sockaddr*)&service, sizeof(sockaddr_in));
    assert(result != SOCKET_ERROR);

    Tutaj będzie główna pętla programu serwera UDP

    closesocket(server_socket);
    return 0;
}
```



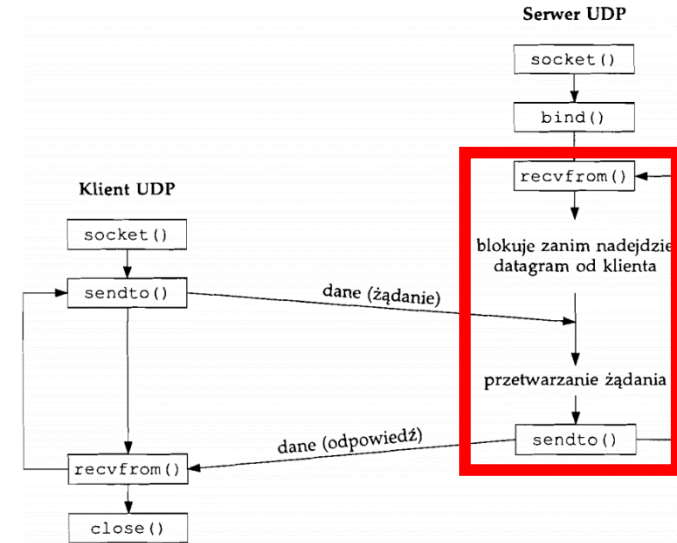
Serwer UDP: Implementacja w C++

główna pętla programu serwera

```
while(true)
{
    sockaddr_in client;
    int sa_size = sizeof(sockaddr_in);
    recvfrom(server_socket, datagram, sizeof(datagram), 0,
            (sockaddr*)&client, &sa_size);

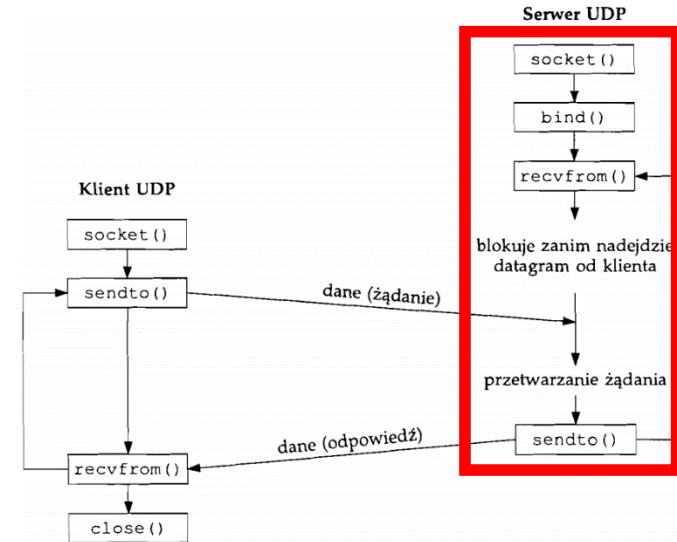
    time_t curr_time;
    time(&curr_time);
    tm *t = gmtime(&curr_time);
    counter++;

    sprintf(line1, "Data %02d/%02d/%04d\r\n", t->tm_mday, t->tm_mon + 1, t->tm_year + 1900);
    sprintf(line2, "Godzina %02d:%02d:%02d\r\n", t->tm_hour, t->tm_min, t->tm_sec);
    sprintf(line3, "Jestes klientem #%d\r\n", counter);
    sprintf(datagram, "%s%s%s", line1, line2, line3);
    sendto(server_socket, datagram, sizeof(datagram), 0,
            (sockaddr*)&client, sizeof(sockaddr_in));
}
```



Serwer UDP: Implementacja w C#

```
1  using System;
2  using System.Net;
3  using System.Net.Sockets;
4  using System.Text;
5
6  public class UDPServer
7  {
8      private const int listenPort = 3301;
9      public static void Main(string[] args) throws Exception
10     {
11         bool done = false;
12         UdpClient listener = new UdpClient(listenPort);
13         IPEndPoint groupEP = new IPEndPoint(IPAddress.Any, 3301);
14         while (!done)
15         {
16             listener.Receive(ref groupEP);
17
18             DateTime now = DateTime.Now;
19             StringBuilder sb = new StringBuilder();
20             sb.AppendLine(string.Format("Data: {0:00}/{1:00}/{2:0000}",
21                                     now.Day, now.Month, now.Year));
22             sb.AppendLine(string.Format("Czas: {0:00}/{1:00}/{2:00}",
23                                     now.Hour, now.Minute, now.Second));
24             sb.AppendLine(string.Format("Jestes klientem #{0}",
25                                     counter));
26             byte[] bufor = Encoding.ASCII.GetBytes(sb.ToString());
27             listener.Send(bufor, bufor.Length, groupEP);
28         }
29         listener.Close();
30     }
31 }
```



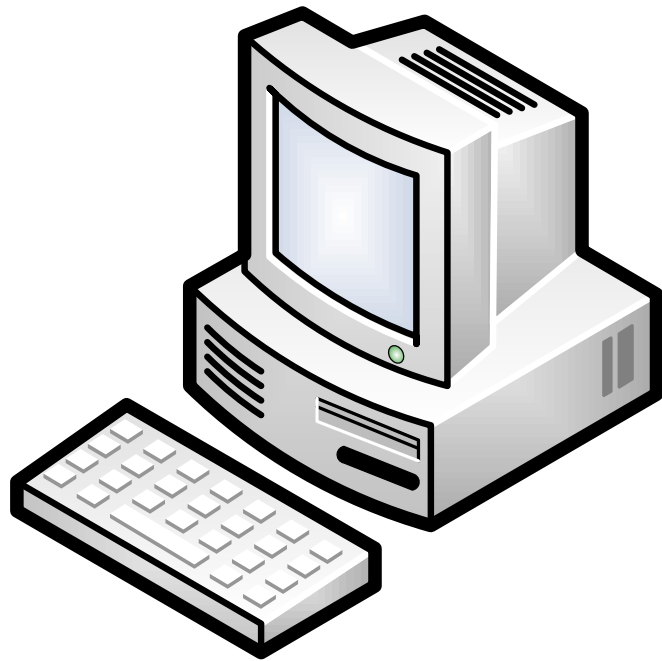
Serwer UDP: Implementacja w JAVA

```
public class UDPRecevier
{
    public static void main(String[] args) throws IOException
    {
        DatagramSocket ds = new DatagramSocket();
        ds.bind(new InetSocketAddress("0.0.0.0", 3301))
        byte[] receive = new byte[65535];

        DatagramPacket DpReceive = null;
        while (true)
        {
            DpReceive = new DatagramPacket(receive, receive.length);
            int bytesRead = ds.receive(DpReceive);
            System.out.println("Client "
                + DpReceive.getAddress().toString() + ":" + DpReceive.getPort() + "- "
                + new String(receive, 0, bytesRead));
        }
    }
}
```

```
public class UDPSender
{
    public static void Send(String mesg)
    {
        DatagramSocket ds = new DatagramSocket();
        byte buf[] = mesg.getBytes(Charset.forName("UTF-8"));
        DatagramPacket DpSend = new DatagramPacket(buf, buf.length, InetAddress.getLocalHost(), 3301);
        ds.send(DpSend);
        ds.close();
    }
}
```

Połączenia multicastowe i broadcastowe

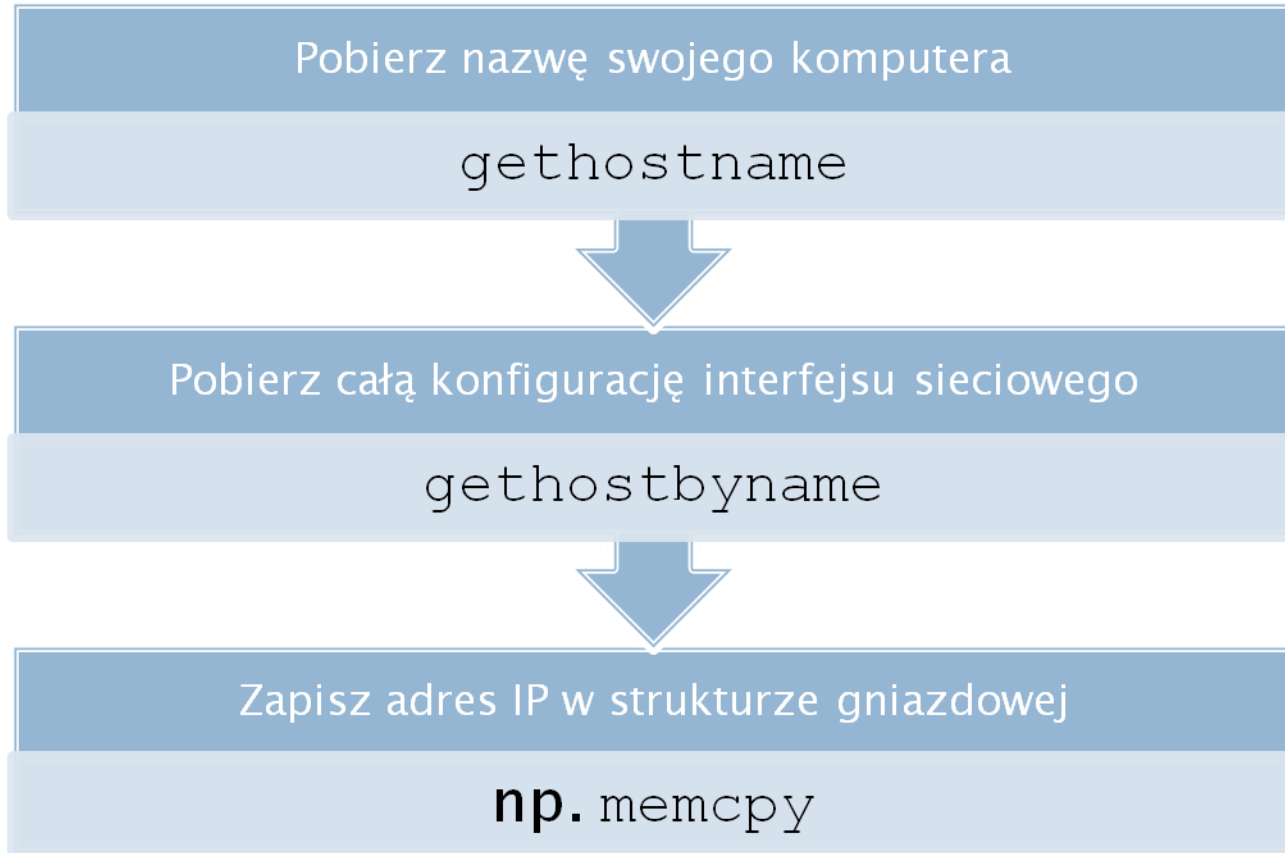


Pobieranie swojego adresu IP

Techniki rozsyłania grupowego

- Multicast
- Broadcast

Swój adres IP – algorytm



Pobieranie nazwy komputera

`gethostname` ► `gethostbyname` ► `memcpy`

```
int gethostname(__out char *name, __in int namelen);
```

Funkcja zwraca podstawową nazwę hosta lokalnego komputera.

name – wskaźnik do bufora, w którym zostanie zapisana nazwa lokalnego komputera,

namelen – długość (w bajtach) bufora wskazywanego przez paramter *name*.

Wynik: **jeśli brak błędu 0**, lub:
`SOCKET_ERROR`, kod błędu z *WSAGetLastError* (Windows),

Jeśli nazwa komputera nie została zdefiniowana, **getbyname** musi zakończyć się sukcesem i zwróci nazwę zgodną z działaniem funkcji **gethostbyname**
(DNS)

Pobierz konfigurację interfejsu sieciowego

gethostname ► **gethostbyname** ► memcpy

```
struct hostent* FAR gethostbyname(__out char *name);
```

Funkcja zwraca konfigurację interfejsu sieciowego zgodnego z nazwą komputera podaną jako parametr.

name – wskaźnik do bufora zawierającego nazwę komputera.

Wynik: jeśli brak błędu wskaźnik do struktury **hostent**, lub: NULL, kod błędu z *WSAGetLastError* (Windows),

Możliwe błędy:

WSAHOST_NOT_FOUND – host nie znaleziony

WSANO_DATA – nie znaleziono danych

Struktura hostent

gethostname ► **gethostbyname** ► memcpy

```
typedef struct hostent {  
    char FAR          * h_name;  
    char FAR FAR      **h_aliases;  
    short              h_addrtype;  
    short              h_length;  
    char FAR FAR      **h_addr_list;  
} HOSTENT;
```

Struktura przechowuje informacje o konfiguracji hosta.

W aplikacji nie wolno jej modyfikować, zwalniać obiektu struktury ani dowolnego jej pola

TYLKO jedna kopia struktury jest alokowana na wątek

Każde wywołanie funkcji **gethostbyname** lub **gethostbyaddr** nadpisuje dane w strukturze **hostent**

Struktura hostent

gethostname ► **gethostbyname** ► memcpy

```
typedef struct hostent {  
    char FAR          * h_name;  
    char FAR FAR      **h_aliases;  
    short              h_addrtype;  
    short              h_length;  
    char FAR FAR      **h_addr_list;  
} HOSTENT;
```

Pola:

h_name – oficjalna nazwa hosta (PC).

– W przypadku istnienia DNS nazwa jest nazwą **Fully Qualified Domain Name (FQDN)** zwracaną przez serwer DNS.

h_aliases – tablica ewentualnych aliasów,

h_addrtype – typ zwracanego adresu

h_length – długość adresu w bajtach

h_addr_list – lista adresów hosta zwracane w sieciowej kolejności bajtów.

Zdefiniowano makro *h_addr* jako odwołanie do
h_addr_list[0]

Skopiowanie adresu

gethostname ► gethostbyname ► **memcpy**

```
memcpy(&yyy.s_addr, *xxx->h_addr_list[0], 4);
```

lub po prostu

```
yyy.s_addr = *(u_long *) xxx->h_addr_list[0];
```


Pobranie adresu hosta (C#)

```
private IPAddress LocalIPAddress()
{
    if (!System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable())
    {
        return null;
    }

    IPEndPoint host = Dns.GetHostEntry(Dns.GetHostName());

    return host.AddressList.FirstOrDefault(ip => ip.AddressFamily == AddressFamily.InterNetwork);
}
```

```
string localIP;
using (Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, 0))
{
    socket.Connect("8.8.8.8", 65530);
    IPEndPoint endPoint = socket.LocalEndPoint as IPEndPoint;
    localIP = endPoint.Address.ToString();
}
```

Pobranie adresu hosta (C#)

```
public string GetLocalIPv4()
{
    string output = "";
    foreach (NetworkInterface item in NetworkInterface.GetAllNetworkInterfaces())
    {
        if (item.OperationalStatus == OperationalStatus.Up)
        {
            IPInterfaceProperties adapterProperties = item.GetIPProperties();
            if (adapterProperties.GatewayAddresses.FirstOrDefault() != null)
            {
                foreach (UnicastIPAddressInformation ip in adapterProperties.UnicastAddresses)
                {
                    if (ip.Address.AddressFamily == System.Net.Sockets.AddressFamily.InterNetwork)
                    {
                        output = ip.Address.ToString();
                    }
                }
            }
        }
    }
    return output;
}
```

Uzyskanie adresu, który ma zdefiniowany adres bramy

Pobranie adresu hosta (JAVA)

```
import java.net.InetAddress;

class GetIPAdres {
    public static void main() {
        InetAddress iAddress = InetAddress.getLocalHost();
        String currentIp = iAddress.getHostAddress();
        System.out.println("Current IP address : " +currentIp);
    }
}
```

Pobranie adresu hosta (JAVA)

```
private static InetAddress getLocalHostLANAddress() throws UnknownHostException {
    try {
        InetAddress candidateAddress = null;
        // Iterate all NICs (network interface cards)...
        for (Enumeration ifaces = NetworkInterface.getNetworkInterfaces(); ifaces.hasMoreElements();) {
            NetworkInterface iface = (NetworkInterface) ifaces.nextElement();
            // Iterate all IP addresses assigned to each card...
            for (Enumeration inetAddrs = iface.getInetAddresses(); inetAddrs.hasMoreElements();) {
                InetAddress inetAddr = (InetAddress) inetAddrs.nextElement();
                if (!inetAddr.isLoopbackAddress()) {
                    if (inetAddr.isSiteLocalAddress()) {
                        return inetAddr; // Found non-loopback site-local address
                    }
                    else if (candidateAddress == null) {
                        // Found non-loopback address, but not necessarily site-local.
                        // Store it as a candidate to be returned if site-local address is not subsequently found...
                        candidateAddress = inetAddr;
                    }
                }
            }
        }

        if (candidateAddress != null) {
            return candidateAddress;
        }

        // At this point, we did not find a non-loopback address.
        InetAddress jdkSuppliedAddress = InetAddress.getLocalHost();
        if (jdkSuppliedAddress == null) {
            throw new UnknownHostException("The JDK InetAddress.getLocalHost() method unexpectedly returned null.");
        }
        return jdkSuppliedAddress;
    }
    catch (Exception e) {
        UnknownHostException unknownHostException = new UnknownHostException("Failed to determine LAN address: " + e);
        unknownHostException.initCause(e);
        throw unknownHostException;
    }
}
```

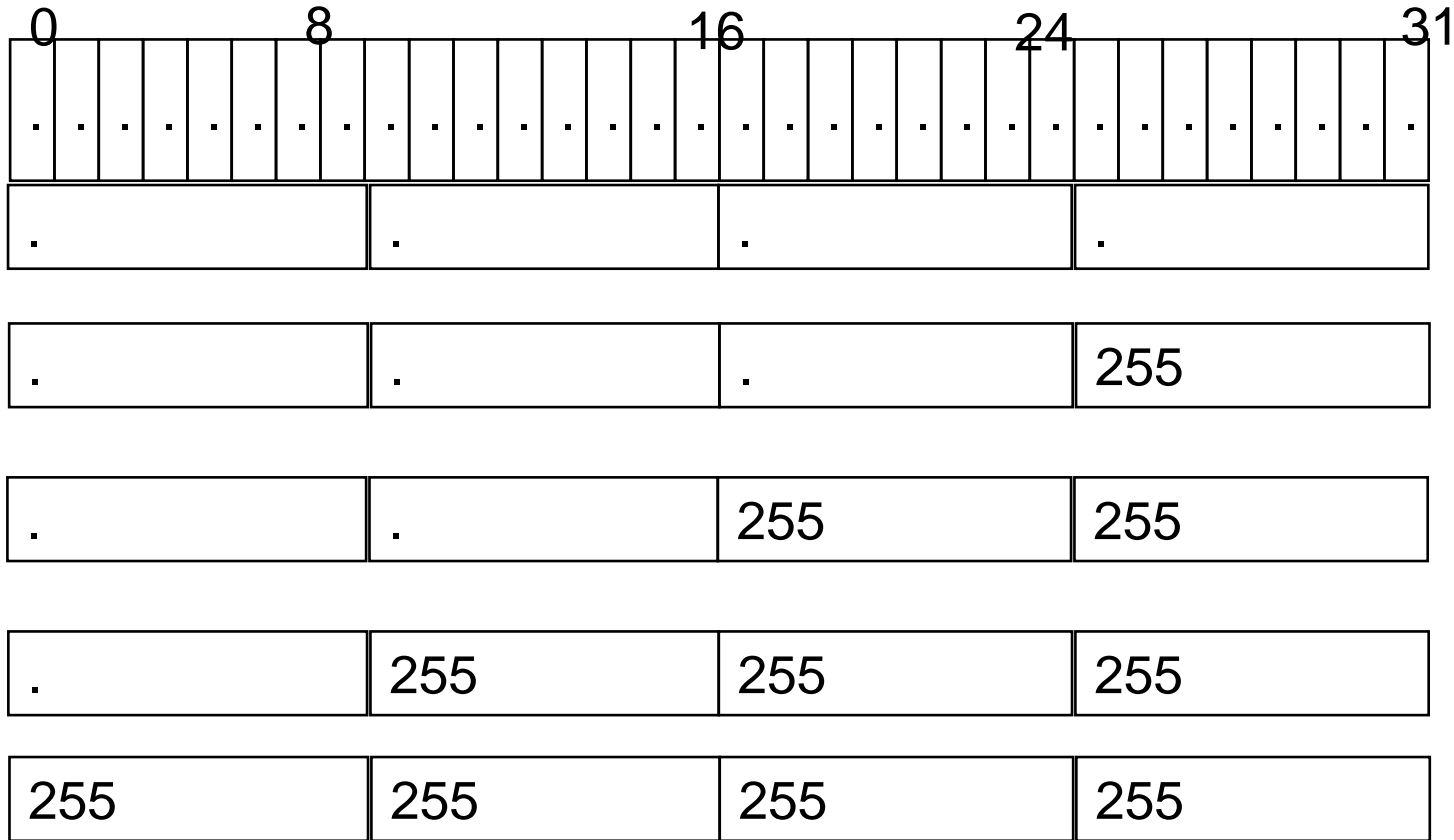
Rozgłaszanie (broadcasting)

Rozsyłanie pakietów do wszystkich komputerów w danej sieci lokalnej

→ Obciążenie komputerów

→ Obciążenie sieci

Adresy rozgłoszeniowe



Adres rozgłoszeniowy (broadcast)

Adres rozgłoszeniowy umożliwia przesłanie pakietu datagramowego (UDP) do wszystkich komputerów w danej sieci.

Adres rozgłoszeniowy (broadcast)

Przykład:

- ▶ Komputery: 192.168.0.1; 192.168.0.2; 192.168.0.3;
- ▶ Maska sieci: 255.255.255.0
- ▶ Adres sieci: **192.168.0.0**;
adres rozgłoszeniowy: **192.168.0.255**

Wysłanie datagramu UDP na adres IP=192.168.0.255 spowoduje odebranie go na wszystkich komputerach w sieci 192.168.0.0.

Czyli, jeśli są komputery z IP z zakresu 192.168.0.1 – 192.168.0.254 i nasłuchują na gnieździe umożliwiającym komunikację rozgłoszeniową, to otrzymają datagram.

```
SOCKET sock = socket(AF_INET, SOCK_DGRAM, 0);  
    assert(sock != INVALID_SOCKET);
```

```
BOOL t = true;  
result = setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (char*)&t, sizeof(BOOL));
```

```
sockaddr_in send_addr, recv_addr;  
send_addr.sin_family = AF_INET;  
send_addr.sin_port = htons(3303);  
send_addr.sin_addr.s_addr = inet_addr("192.168.0.255");
```

```
char *tekst = "Ala ma kota";  
int s = sizeof(sockaddr_in);  
result = sendto(sock, tekst, strlen(tekst) + 1, 0,  
                (sockaddr*)&send_addr, sizeof(sockaddr_in));
```

```
recv_addr.sin_family = AF_INET;  
recv_addr.sin_port = htons(3303);  
recv_addr.sin_addr.s_addr = INADDR_ANY;
```

```
result = bind(sock, (sockaddr*)&recv_addr, sizeof(sockaddr_in));  
char buffer[100];  
result = recvfrom(sock, buffer, strlen(buffer)+1, 0,  
                 (sockaddr*)&recv_addr, &s);
```

Broadcast wysyłanie (C#)

```
private void BroadcastSender()
{
    UdpClient udpClient = new UdpClient();
    Byte[] sendBytes = Encoding.Unicode.GetBytes("Message");
    IPEndPoint RemoteIpEndPoint = new IPEndPoint(IPAddress.Parse("255.255.255.255"), 2222);
    udpClient.Send(sendBytes, sendBytes.Length, RemoteIpEndPoint);
    udpClient.Close();
}
```

Broadcast odbieranie (C#)

```
private void BroadcastListener()
{
    UdpClient udpClient = new UdpClient();

    udpClient.ExclusiveAddressUse = false;
    IPEndPoint localEp = new IPEndPoint(IPAddress.Any, 2222);
    udpClient.Client.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

    udpClient.Client.Bind(localEp);

    IPEndPoint RemoteIpEndPoint = new IPEndPoint(IPAddress.Any, 0);
    string returnData = String.Empty;
    do
    {
        Byte[] receiveBytes = udpClient.Receive(ref RemoteIpEndPoint);
        returnData = Encoding.ASCII.GetString(receiveBytes);
        Console.WriteLine(returnData);

    } while (returnData != String.Empty);
    udpClient.Close();
}
```

Broadcast wysyłanie (JAVA)

```
import java.io.*;
import java.net.*;
import java.nio.charset.Charset;

public class UDP_Broadcast_Sender {

    public static void sendData(int port, String mesg) {
        try {
            DatagramSocket s = new DatagramSocket(port);
            s.setBroadcast(true);

            // Note that we don't have to join the multicast group if we are only
            // sending data and not receiving
            // Fill the buffer with some data
            byte buf[] = mesg.getBytes(Charset.forName("UTF-8"));
            // Create a DatagramPacket
            //DatagramPacket pack= new DatagramPacket(buf, buf.length);
            DatagramPacket pack = new DatagramPacket(buf, buf.length,
                InetAddress.getByName("255.255.255.255"), port);
            // Do a send. Note that send takes a byte for the ttl and not an int.
            s.send(pack);
            // And when we have finished sending data close the socket
            s.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            MainFrame.add2output(e.getMessage());
        }
    }
}
```

Grupa rozgłoszeniowa

Def

Grupa rozgłoszeniowa to zespół komputerów reprezentowanych jednym numerem IP

Rozgłaszanie grupowe jest opcją dodatkową oprogramowania IP

- brak obsługi rozgłaszania grupowego w implementacji stosu IP

Zasady rozgłaszania grupowego

- Każda stacja może należeć do jednej lub więcej grup rozgłoszeniowych
- Pakiet wysłany do grupy rozgłoszeniowej trafia do wszystkich komputerów niezależnie od przynależności do sieci fizycznej
- Przynależność do grupy rozgłoszeniowej jest dynamiczna
- Do obsługi rozgłaszania grupowego mogą być dedykowane specjalne komputery (multicasting routers)

Rozgłaszanie jest z osobna konfigurowane dla każdego interfejsu fizycznego w komputerze.



Adresy rozgłoszeniowe

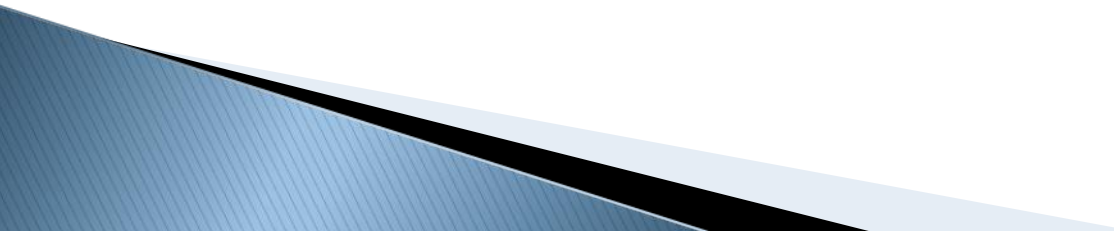
224.0.0.0 - 239.255.255.255 - Klasa D

224.0.0.0 - nie przypisany

224.0.0.1 - adres specjalny (wszystkie komputery)

Pozostałe numery - Assigned Numbers RFC 790

Zasady obsługi pakietów rozsyłania grupowego

- Umożliwić wysyłanie pakietów z adresem grupy D
 - Czas życia pakietów domyślnie musi być równy 1
 - Stos IP odrzuca te datagramy, które adresowane są do obcych grup
- 

Zasady obsługi pakietów rozsyłania grupowego

Przynależność do grupy jest dynamiczna

- JoinHostGroup
- LeaveHostGroup

Pakiet wysyłany do grupy musi być blokowany w lokalnej petli IP

Nie generuje się pakietów ICMP w odpowiedzi na pakiety rozgłoszeniowe

Pakiety skierowane pod adres 224.0.0.1 nie są transmitowane poza sieć lokalną

Internet Group Management Protocol (IGMP)

Pozyskiwanie i uaktualnianie informacji o przynależności do grup rozgłoszeniowych

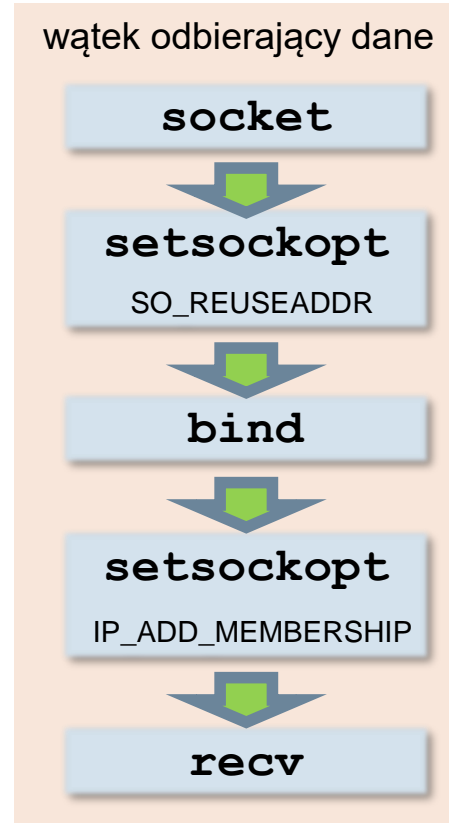
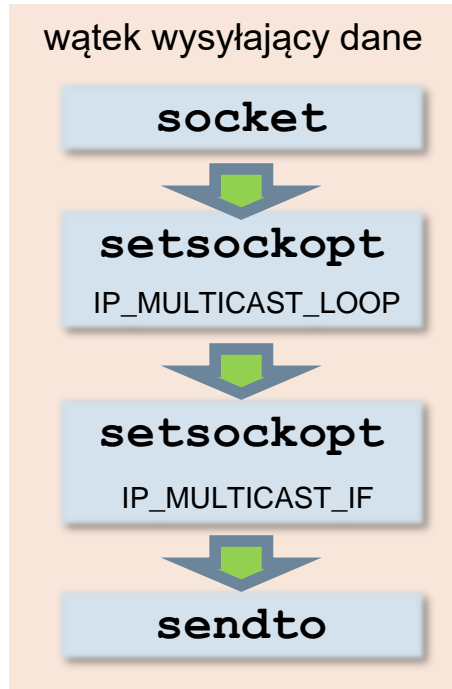
Komputery wykorzystują komunikaty IGMP do powiadamiania routerów w swojej sieci o chęci przyłączenia się do lub odejścia z określonej grupy multicastowej

Działa w warstwie sieciowej podobnie, jak ICMP.

Działa zarówno na poszczególnych komputerach, jak i na routerach.



Grupy multicastowe



Zmiana konfiguracji gniazda

socket ► **setsockopt** ► sendto

```
int setsockopt(  
    __in SOCKET s,  
    __in int level,  
    __in int optname,  
    __in const char *optval,  
    __in int optlen );
```

Funkcja zmienia konfigurację gniazda.

s – deskryptor identyfikujący gniazdo,
level – poziom, na jakim dokonujemy zmian konfiguracji.
optname – opcja gniazda, która jest zmieniana. Wartość opcji musi być zgodna z wybranym poziomem
optval – wskaźnik do bufora zawierający wartości dla zmienianych opcji
optlen – długość (w bajtach) bufora wskazywanego przez *optval*

Zmiana konfiguracji gniazda

socket ► **setsockopt** ► sendto

```
int setsockopt(  
    __in SOCKET s,  
    __in int level,  
    __in int optname,  
    __in const char *optval,  
    __in int optlen );
```

Wynik: **jeśli brak błędu 0**, lub:
SOCKET_ERROR, kod błędu z *WSAGetLastError* (Windows),

Jeśli **setsockopt** zostanie wywołana przed **bind**, zmiany nie zostaną sprawdzone, aż do momentu pojawienia się **bind**, a **setsockopt** zwróci zawsze sukces, w przeciwieństwie do **bind**.

Zmiana konfiguracji gniazda

socket ► **setsockopt** ► sendto

Są dwie grupy opcji:

Boolean: dla włączania lub wyłączania cech lub zachowań,

Wartości: dla wprowadzania dodatkowych wartości

Poziomy opcji:

SOL_SOCKET

IPPROTO_TCP

NSPROTO_IPX

IPX (ang. *Internetwork Packet Exchange*) to protokół warstwy sieciowej (trzeciej warstwy modelu OSI) będący częścią stosu IPX/SPX opracowanego przez firmę Novell na potrzeby środowiska sieciowego NetWare.

Zmiana konfiguracji gniazda

socket ► **setsockopt** ► sendto

SOL_SOCKET :

SO_BROADCAST (BOOL): włącza tryb transmisji broadcastowej,

SO_DONTLINGER (BOOL): proces zamykania gniazda nie czeka na wysłanie niewysłanych danych,

SO_REUSEADDR (BOOL) – umożliwia ponowne bindowanie zbindowanego gniazda;
umożliwia wielu instancjom aplikacji odbieranie kopii datagramów rozsyłania grupowego,

```
int reuse=1;  
setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, (char *)&reuse, sizeof(reuse));
```

Zmiana konfiguracji gniazda

socket ► **setsockopt** ► sendto

IPPROTO_TCP:

IP_MULTICAST_LOOP (CHAR): Wyłączenie pętli zwrotnej, aby nie otrzymywać własnych datagramów,

```
char valch = 0;
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_LOOP, (char *)&valch, sizeof(valch));
```

IP_MULTICAST_IF (CHAR): Skonfigurowanie lokalnego interfejsu dla wychodzących datagramów rozsyłania grupowego. Podany adres IP musi być przypisany do lokalnego interfejsu, który będzie obsługiwał rozsyłanie grupowe,

```
struct in_addr localInterface;
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_IF,
           (char *)&localInterface, sizeof(localInterface));
```

IP_MULTICAST_TTL: (CHAR): Skonfigurowanie czasu życia pakietów w rozsyłaniu grupowym

IP_ADD_MEMBERSHIP: Dołącz lokalny interfejs do grupy rozsyłania grupowego

IP_DROP_MEMBERSHIP: Odłącz

```
struct ip_mreq group;
group.imr_multiaddr.s_addr = inet_addr("225.1.1.1");
setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char *)&group, sizeof(group));
```

Multicast – wysyłanie

```
struct in_addr      localInterface;
struct sockaddr_in  groupSock;
SOCKET              sd;
int                 datalen;
char                databuf[1024];

int initSocket()
{
    // Utworzenie gniazda datagramu, do którego datagram ma być wysłany.
    sd = socket(AF_INET, SOCK_DGRAM, 0);

    //Inicjowanie struktury grupy sockaddr z adresem grupy 225.1.1.1 i portem 5555
    memset((char *) &groupSock, 0, sizeof(groupSock));
    groupSock.sin_family = AF_INET;
    groupSock.sin_addr.s_addr = inet_addr("225.1.1.1");
    groupSock.sin_port = htons(5555);

    //Wyłączenie pętli zwrotnej, aby nie otrzymywać własnych datagramów.
    char loopch=0;
    if (setsockopt(sd, IPPROTO_IP, IP_MULTICAST_LOOP,
        (char *)&loopch, sizeof(loopch)) < 0) {
        _tprintf(_T("Fatal Error: setting IP_MULTICAST_LOOP\n"));
        closesocket(sd);return 1;
    }
}
```

Multicast – wysyłanie

```
//Poznaj swój adres IP
char myname[32] = {0};
gethostname( myname, 32 );
hostent *he;
he = gethostbyname( myname );
printf("Host name : %s\n", he->h_name );
printf("Host addr : %s\n", inet_ntoa( *( in_addr *)he->h_addr) );

//Skonfigurowanie lokalnego interfejsu dla wychodzących datagramów rozsyłania grupowego.
//Podany adres IP musi być przypisany do lokalnego interfejsu, który obsługuje rozsyłanie grupowe.
memcpy(&localInterface.s_addr, *he->h_addr_list, 4);
if (setsockopt(sd, IPPROTO_IP, IP_MULTICAST_IF,
               (char *)&localInterface,
               sizeof(localInterface)) < 0) {
    _tprintf(_T("Fatal Error: local interface configuration\n"));
    closesocket(sd);return 1;
}
}
```

Multicast – wysyłanie

```
int sendData()  
{  
/*  
 * Wysyłanie komunikatu do grupy rozsyłania grupowego podanej przez  
 * strukturę groupSock sockaddr.  
 */  
datalen = 50;  
sprintf_s(databuf, datalen, "To ja!! Twój listonosz multicastowy");  
if (sendto(sd, databuf, datalen, 0, (struct sockaddr*)&groupSock,  
sizeof(groupSock)) < 0)  
{  
_tprintf(_T("Fatal Error: sending datagram message\n"));  
return 1;  
}  
_tprintf(_T("message datagram sending: OK\n"));  
return 0;  
}
```

Multicast – wysyłanie (C#)

```
private void MulticastSender()
{
    IPAddress multicastaddress = IPAddress.Parse("239.0.0.222");
    UdpClient udpclient = new UdpClient();
    udpclient.JoinMulticastGroup(multicastaddress);
    IPEndPoint remoteep = new IPEndPoint(multicastaddress, 2222);
    Byte[] buffer = null;
    buffer = Encoding.Unicode.GetBytes("Message");
    udpclient.Send(buffer, buffer.Length, remoteep);
}
```

Multicast – wysyłanie (JAVA)

```
import java.io.IOException;
import java.net.*;
import java.nio.charset.Charset;

public class UDP_Multicast_Sender {

    public static void main() {
        String mesg = "message";
        int ttl = 1;
        try {
            // Create the socket but we don't bind it as we are only going to send data
            MulticastSocket s = new MulticastSocket();
            // Note that we don't have to join the multicast group if we are only
            // sending data and not receiving
            // Fill the buffer with some data
            byte buf[] = mesg.getBytes(Charset.forName("UTF-8"));
            // Create a DatagramPacket
            DatagramPacket pack = new DatagramPacket(buf, buf.length,
                InetAddress.getByName("239.0.0.222"), 2222);
            // Do a send. Note that send takes a byte for the ttl and not an int.
            s.send(pack, (byte)ttl);
            // And when we have finished sending data close the socket
            s.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            MainFrame.add2output(e.getMessage());
        }
    }
}
```

Multicast – odbieranie

```
struct sockaddr_in    localSock;
struct ip_mreq        group;
int                  sd;
int                  datalen;
char                 databuf[1024];

int initSocket()
{
    //Utworzenie gniazda, z którego datagram ma być odebrany.
    sd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sd == INVALID_SOCKET)
    {
        _tprintf(_T("Fatal Error: while opening socket\n"));
        return 1;
    }

    // Włącz SO_REUSEADDR, aby umożliwić wielu instancjom tej aplikacji
    {
        int reuse=1;
        if (setsockopt(sd, SOL_SOCKET, SO_REUSEADDR,
                       (char *)&reuse, sizeof(reuse)) < 0) {
            _tprintf(_T("Fatal Error: setting SO_REUSEADDR\n"));
            closesocket(sd);
            return 1;
        }
        _tprintf(_T("setting SO_REUSEADDR: OK\n"));
    }
}
```


Multicast – odbieranie

```
//Powiąż odpowiedni numer portu z adresem IP podanym jako INADDR_ANY.
memset((char *) &localSock, 0, sizeof(localSock));
localSock.sin_family = AF_INET;
localSock.sin_port = htons(5555);
localSock.sin_addr.s_addr = INADDR_ANY;

if (bind(sd, (struct sockaddr*)&localSock, sizeof(localSock))) {
    _tprintf(_T("Fatal Error: while binding socket\n"));
    closesocket(sd); return 1;
}

// Poznaj swój adres IP
char myname[32] = {0};
gethostname( myname, 32 );
hostent *he;
he = gethostbyname( myname );
printf("Host name : %s\n", he->h_name );
printf("Host addr : %s\n", inet_ntoa( *( in_addr *)he->h_addr) );

/* Dołącz do grupy rozsyłania grupowego 225.1.1.1 w lokalnym interfejsie
 * Zauważ, że opcja IP_ADD_MEMBERSHIP musi być wywołana
 * dla każdego interfejsu lokalnego, przez który datagramy
 * rozsyłania grupowego mają być odbierane.
 */
group.imr_multiaddr.s_addr = inet_addr("225.1.1.1");
memcpy(&group.imr_interface.s_addr, *he->h_addr_list, 4);
if (setsockopt(sd, IPPROTO_IP, IP_ADD_MEMBERSHIP,
               (char *)&group, sizeof(group)) < 0) {
    _tprintf(_T("Fatal Error: adding IP to the multicasting group\n"));
    closesocket(sd); return 1;
}
_tprintf(_T("adding IP to the multicasting group: OK\n"));
return 0;
```

}

Multicast – odbieranie

```
int readData()
{
    /*
     * Odczyt z gniazda.
     */
    _tprintf(_T("reading datagram mesage: ..... (waiting)\n"));
    datalen = sizeof(databuf);
    if (recv(sd, databuf, datalen, 0) < 0) {
        _tprintf(_T("Fatal Error: reading datagram mesage\n"));
        closesocket(sd);
        return 1;
    }
    //_tprintf(_T("read mesage: %s"),databuf);
    printf("OK! read mesageis:      %s\n",databuf);
    return 0;
}
```

Multicast – odbieranie (C#)

```
private void MulticastListener()
{
    UdpClient client = new UdpClient();

    client.ExclusiveAddressUse = false;
    IPEndPoint localEp = new IPEndPoint(IPAddress.Any, 2222);
    client.Client.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

    client.Client.Bind(localEp);

    IPAddress multicastaddress = IPAddress.Parse("239.0.0.222");
    client.JoinMulticastGroup(multicastaddress);
    while (true)
    {
        Byte[] data = client.Receive(ref localEp);
        string strData = Encoding.Unicode.GetString(data);
        Console.WriteLine(strData);
    }
}
```

```
struct timeval tv;  
  
tv.tv_sec = 5000; /* 5 Secs Timeout */  
  
setsockopt(sockid, SOL_SOCKET, SO_RCVTIMEO, (char *)&tv, sizeof(struct timeval));
```

```
var udpClient = new UdpClient();  
  
udpClient.Client.SendTimeout = 5000;  
udpClient.Client.ReceiveTimeout = 5000;
```

Multicast – odbieranie (JAVA)

```
import java.io.*;
import java.net.*;
import sun.net.*;

public class UDP_Multicast_Receiver {

    public void main() {
        MulticastSocket s = new MulticastSocket(2222);
        s.setReuseAddress(true);
        s.setSoTimeout(1000);
        s.joinGroup(InetAddress.getByName("239.0.0.222"));
        byte buf[] = new byte[1024];
        DatagramPacket pack = new DatagramPacket(buf, buf.length);
        while(isReceiving)
        {
            s.receive(pack);
            System.out.println("Received " + pack.getLength() +
                               " from " + pack.getAddress().toString() + ":" + pack.getPort());
        }
        Stop();
    }

    public void Stop() {
        s.leaveGroup(InetAddress.getByName("239.0.0.222"));
        s.close();
    }
}
```