



Politechnika Łódzka

Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki

Programowanie sieciowe

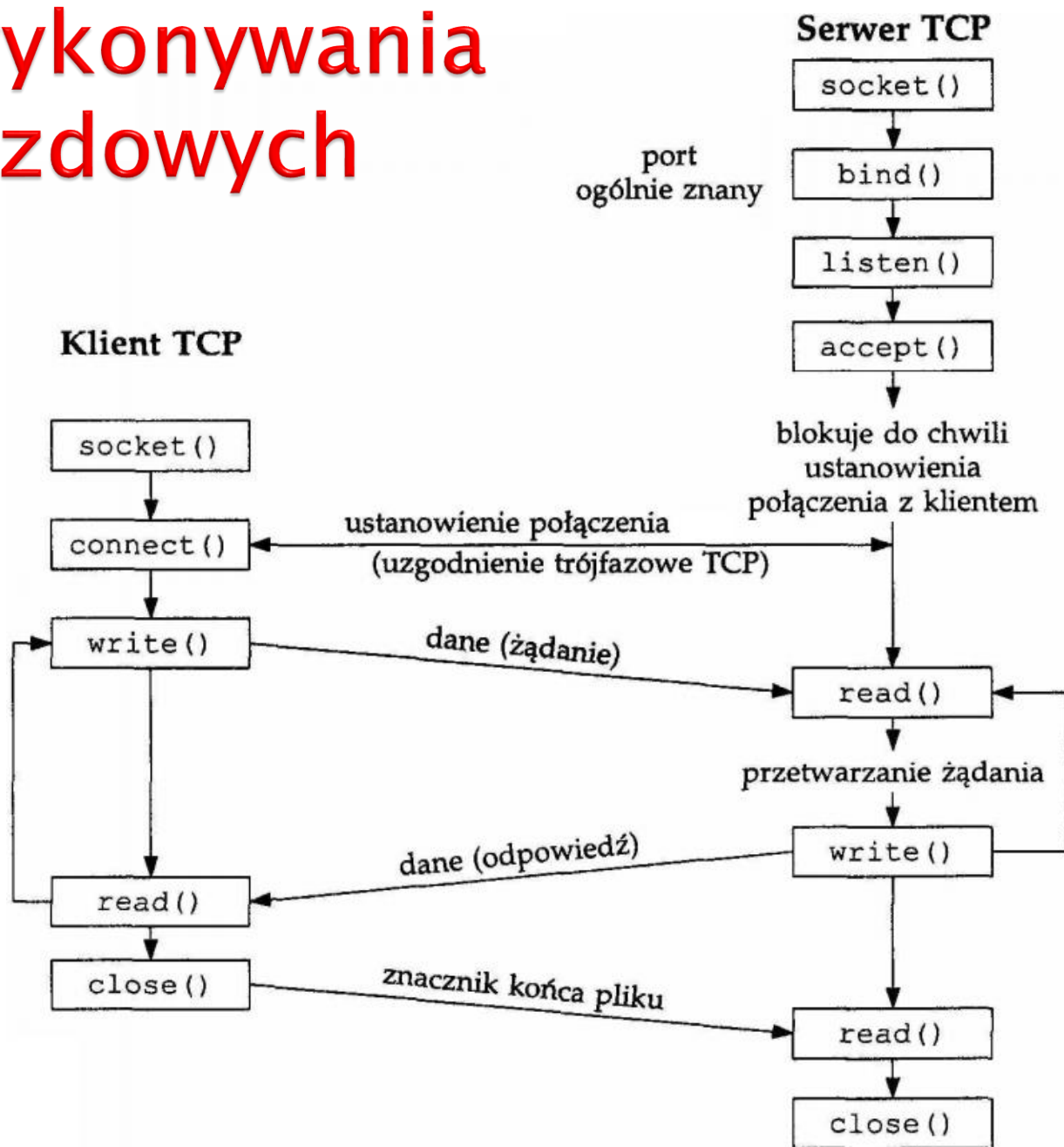
Wykład 2
Komunikacja TCP typu Klient–Serwer

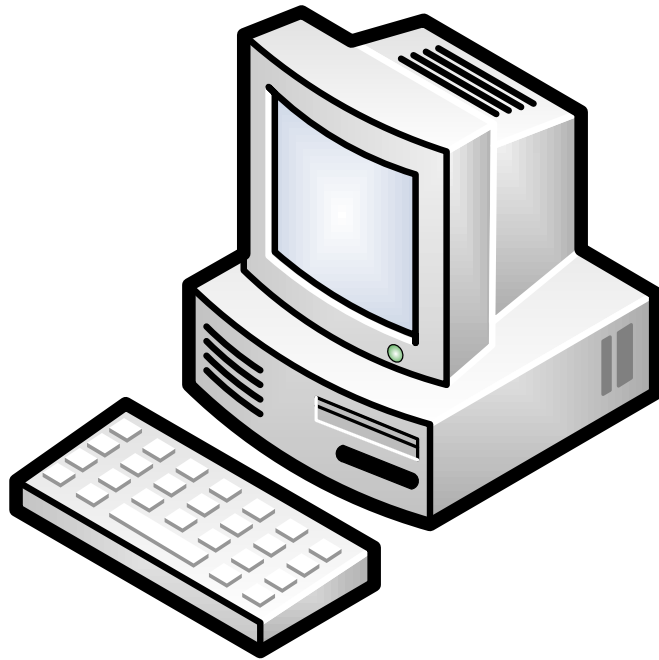


Instytut Informatyki Stosowanej
Politechniki Łódzkiej

Opracował: dr hab. inż. Radosław Wajman

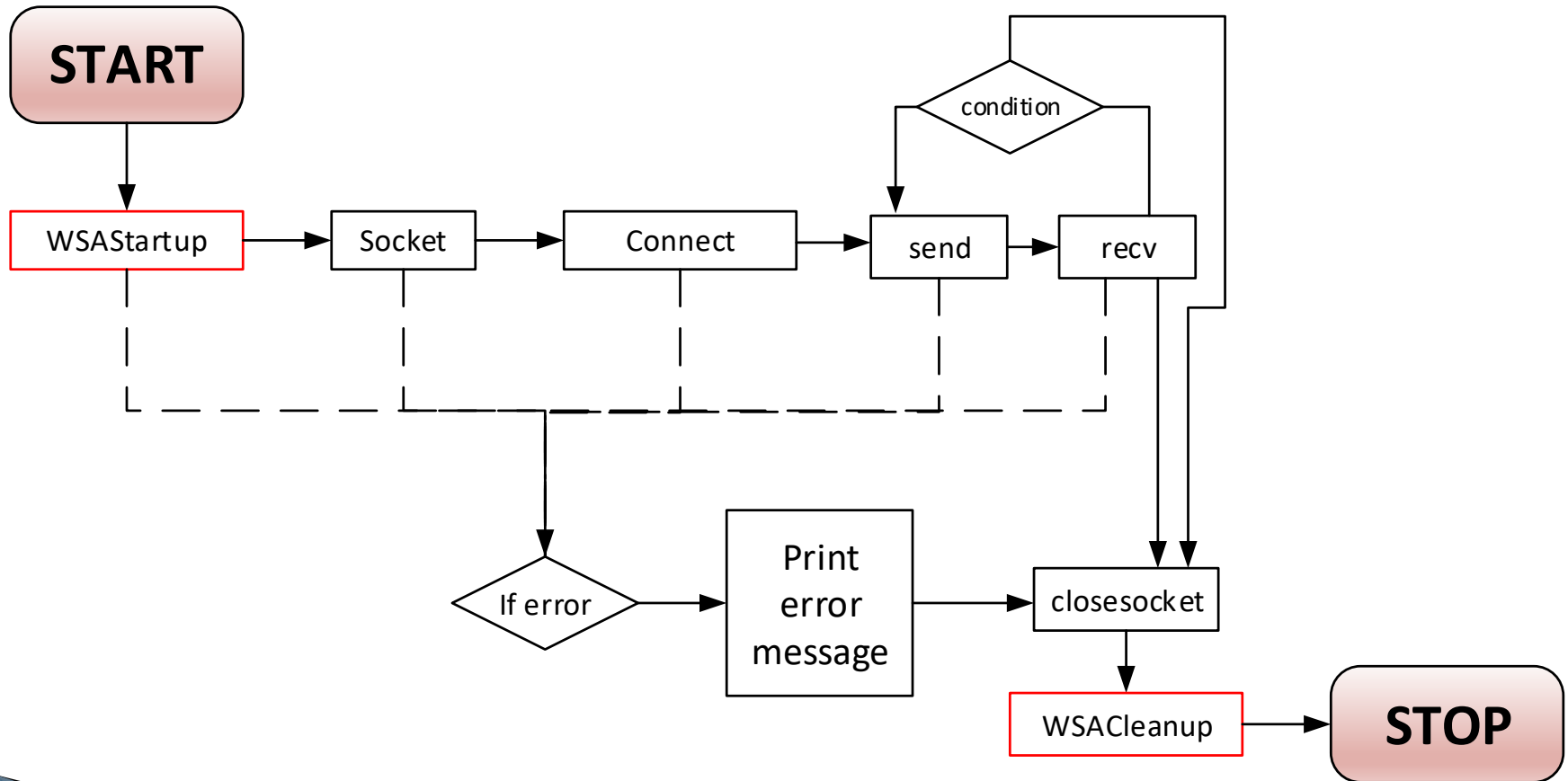
Kolejność wykonywania funkcji gniazdowych klienta TCP





Klient
TCP

Szkielet programu sieciowego – klient echo



Inicjalizacja WinSocket w VS

- Przed wywołaniem dowolnej funkcji Winsock należy załadować poprawną wersję biblioteki Winsock. Funkcją, która to realizuje jest WSAStartup:

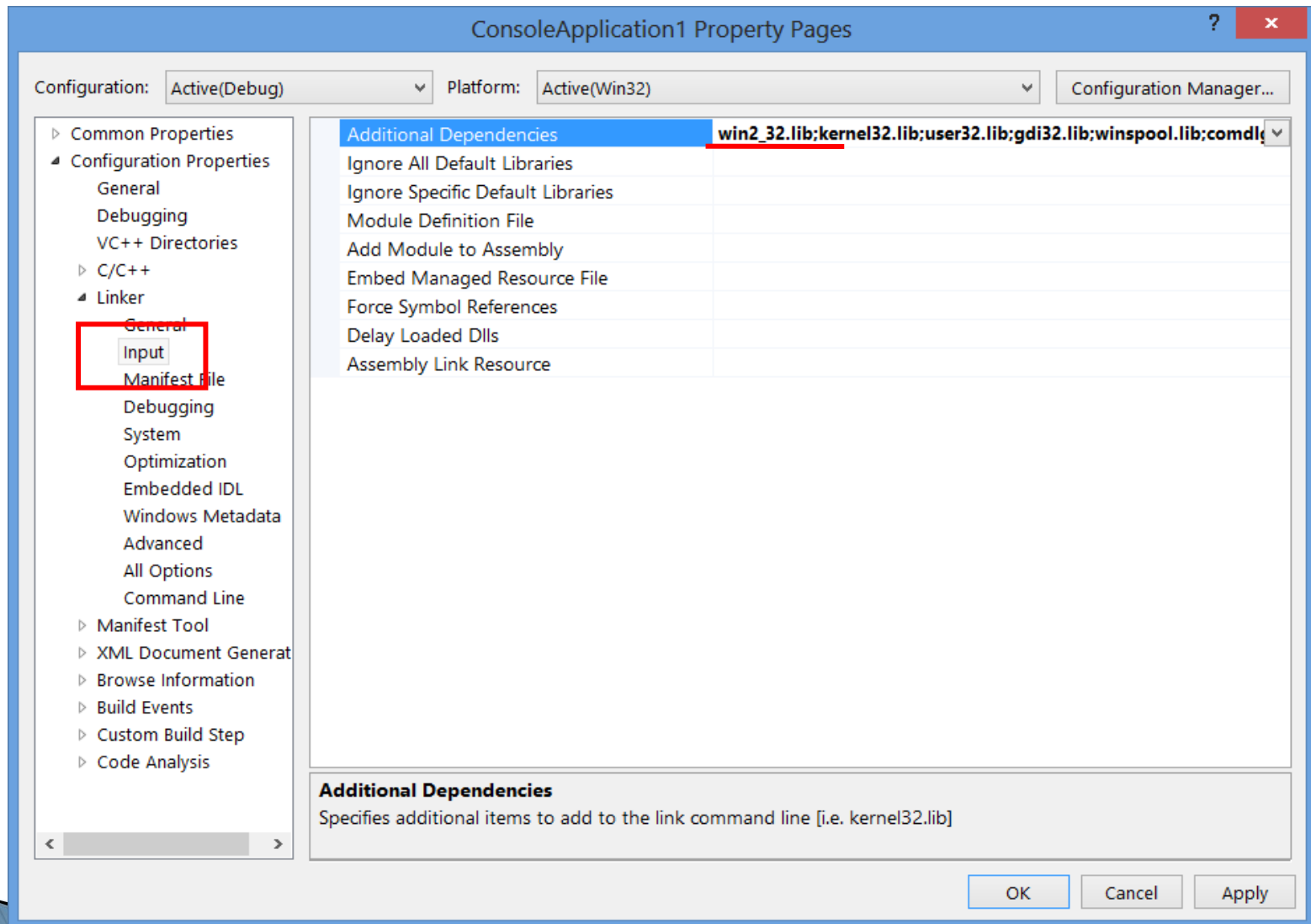
```
C/C++  
WSADATA wsaData;  
  
int result = WSAStartup( MAKEWORD( 2, 2 ), & wsaData );  
if( result != NO_ERROR )  
    printf( "Initialization error.\n" );
```

- Pierwszy parametr określa wersję biblioteki Winsock, którą chcemy załadować (np.. 2.2). Starszy bajt określa numer podwersji, młodszy określa główny numer wersji.
- Drugim parametrem (typ LPWSADATA) jest wskaźnik do struktury WSADATA. Struktura ta zawiera informacje o załadowanej wersji Winsock.
- Po zakończeniu korzystania z biblioteki Winsock należy wywołać funkcję
- WSACleanup(), która usunie bibliotekę z pamięci i zwolni zasoby.

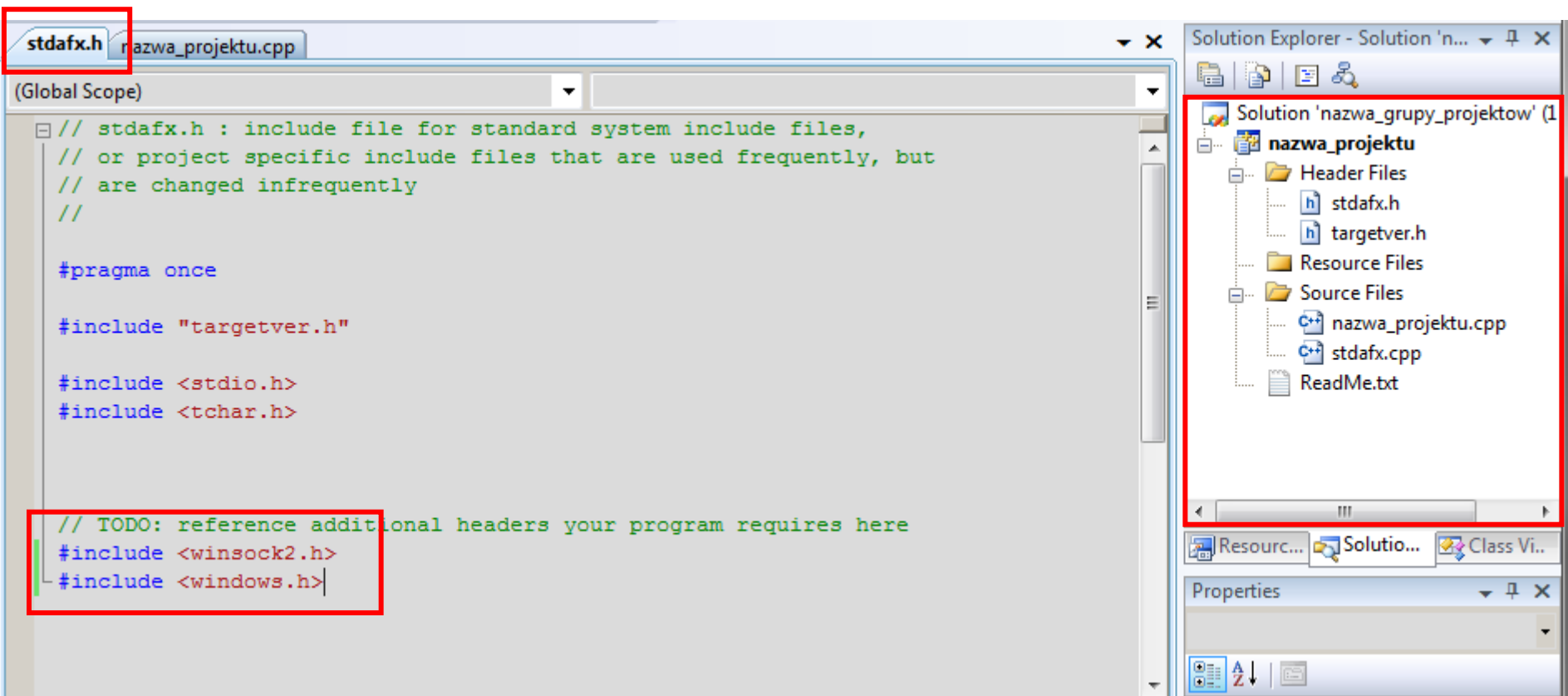
Inicjalizacja WinSocket w VS

```
WSADATA wsaData;  
int nCode;  
char errdesc[100];  
  
if ((nCode = WSStartup(MAKEWORD(1, 1), &wsaData)) != 0)  
{  
    sprintf(errdesc,  
        "Blad podczas inicjalizacji biblioteki WinSock. Blad %d", nCode);  
    exit(1);  
}  
printf("WinSock: %s [%s]\n", wsaData.szDescription, wsaData.szSystemStatus);  
printf("MaxSockets: %d\n", wsaData.iMaxSockets);
```

Dodanie biblioteki WinSockets w VS



Modyfikacja głównego pliku nagłówkowego



Klient: Tworzenie gniazda (C/C++)

socket ► connect ► write ► read ► 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 (TCP),
 - **SOCK_DGRAM** – gniazdo datagramowe (UDP),
 - **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: Tworzenie gniazda (C#)

`socket` ► `connect` ► `write` ► `read` ► `close`

```
public class Socket : IDisposable
```

Klasa w C# odpowiedzialna za utworzenie obiektu gniazda.

Konstruktor:

`Socket(AddressFamily, SocketType, ProtocolType)`

Pozwala utworzyć instancję obiektu definiując:

- ▶ `AddressFamily` – określa schemat adresowania,
 - `InternetNetwork` – wspiera adresację w ramach protokołu IPv4,
- ▶ `SocketType` – określa typ gniazda,
 - `Dgram` – wspiera gniazda datagramowe z użyciem protokołu UDP,
 - `Stream` – wspiera gniazda strumieniowe z użyciem protokołu TCP,
 - `Raw` – wspiera dostęp do bezprotokołowej transmisji danych,
- ▶ `ProtocolType` – przypisuje protokół sieciowy do gniazda,
 - `Tcp`
 - `Udp`
 - `IP`

Klient: Tworzenie gniazda

`socket` ► `connect` ► `write` ► `read` ► `close`

C/C++

```
SOCKET sock_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);  
SOCKET sock_fd = socket(AF_INET, SOCK_STREAM, 0);  
int sock_fd = socket(AF_INET, SOCK_STREAM, 0);  
SOCKET sock_fd = socket(AF_INET, SOCK_DGRAM, 0);
```

C#

```
new Socket(AddressFamily.InterNetwork,  
            SocketType.Stream, ProtocolType.Tcp );  
new Socket(AddressFamily.InterNetwork,  
            SocketType.Dgram, ProtocolType.Udp );
```

JAVA

```
new Socket("127.0.0.1", 1111);
```

Klient: Nawiązywanie połączenia (C/C++)

socket ► connect ► write ► read ► close

```
int connect(int sock, sockaddr *rmt, int rmtlen);
```

Funkcja nawiązuje połączenie ze zdalnym hostem, określonym w *rmt*. Wykonuje tzw. **otwarcie aktywne**.

- ▶ *sock* – uchwyt gniazda (zwrócony przez **socket**)
- ▶ *rmt* – wskaźnik na tzw. strukturę gniazdową **sockaddr** przechowującą adres zdalnego hosta oraz informację o protokole,
- ▶ *rmtlen* – długość, w bajtach, struktury **sockaddr** dla danego protokołu.
- ▶ **Wynik:** 0, lub:
 - **SOCKET_ERROR**, kod błędu z *WSAGetLastError* (Windows),
 - -1, kod błędu z *errno* (Unix)
- ▶ **Blocking:** Funkcja **connect** próbuje się połączyć ze zdalnym hostem przez określony czas. W przypadku porażki zwraca **SOCKET_ERROR**
- ▶ **Nonblocking:** Wtedy wynik = **SOCKET_ERROR** a kod błędu = **WSAEWOULDBLOCK/EWOULDBLOCK**.

Klient: Struktura gniazdowa

socket ► connect ► write ► read ► close

```
typedef struct sockaddr {  
    u_short sa_family;  
    CHAR sa_data[14];  
} SOCKADDR;  
  
typedef struct sockaddr_in {  
    short sin_family;  
    unsigned short sin_port;  
    IN_ADDR sin_addr;  
    CHAR sin_zero[8];  
} SOCKADDR_IN, *PSOCKADDR_IN;
```

```
typedef struct in_addr {  
    union {  
        struct {  
            UCHAR s_b1, s_b2, s_b3, s_b4;  
        } S_un_b;  
        struct {  
            USHORT s_w1, s_w2;  
        } S_un_w;  
        ULONG S_addr;  
    } S_un;  
} IN_ADDR, *PIN_ADDR, *LPIN_ADDR;
```

```
sockaddr_in service;  
service.sin_family = AF_INET;  
service.sin_addr.s_addr =  
    inet_addr("127.0.0.1");  
service.sin_port = htons(3370);
```

Klient: Nawiązywanie połączenia (C/C++)

socket ► **connect** ► write ► read ► close

Aby otrzymać adres IP odpowiadający znanej nazwie hosta, należy odpytać serwer DNS stosując funkcję **gethostbyname**:

```
hostent* h = gethostbyname("google.pl");  
if (h == NULL)  
{  
    printf(„blad”); exit(1);  
}  
service.sin_addr = *(struct in_addr*)h->h_addr_list[0];
```

Klient: Nawiązywanie połączenia (C#)

socket ► **connect** ► write ► read ► close

```
public void Connect(IPAddress address, int port);
```

Metoda obiektu `System.Net.Socket.Socket`

Parametry

- *address*: `System.Net.IPAddress`
Adres IP zdalnego hosta.
- *port*: `int`
Numer portu zdalnego hosta.

```
public static void Connect1(string host, int port)
{
    IPAddress[] IPs = Dns.GetHostAddresses(host);

    Socket s = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream,
        ProtocolType.Tcp);

    Console.WriteLine("Establishing Connection to {0}",
        host);
    s.Connect(IPs[0], port);
    Console.WriteLine("Connection established");
}
```


Klient: Nawiązywanie połączenia (C#)

socket ► **connect** ► write ► read ► close

```
public void Connect(EndPoint remoteEP);
```

Metoda obiektu `System.Net.Socket.TcpClient`

Parametry

- *remoteEP*: `System.Net.EndPoint`
Punkt końcowy zdalnego hosta.

`EndPoint(IPAddress, Int32)`

```
TcpClient tcpClient = new TcpClient ();  
IPAddress ipAddress = Dns.GetHostEntry ("www.contoso.com").AddressList[0];  
EndPoint ipEndPoint = new EndPoint (ipAddress, 11004);  
  
tcpClient.Connect (ipEndPoint);
```

Klient: Nawiązywanie połączenia (C#)

socket ► **connect** ► write ► read ► close

```
public void Connect (.....) ;
```

Wyjątek	Warunek
ArgumentNullException	<i>address</i> jest null.
ArgumentOutOfRangeException	Błędny numer portu
SocketException	Niepowodzenie dostępu do gniazda.
ObjectDisposedException	Gniazdo zostało zamknięte.
NotSupportedException	W przypadku próby dostępu do gniazda i do protokołów innych niż InterNetwork lub InterNetworkV6 .
ArgumentException	Długość <i>address</i> jest zero.
InvalidOperationException	Gniazdo jest słuchające – listener.

Klient: Zapisywanie danych do gniazda (C/C++)

socket ► connect ► **write** ► read ► close

lub

```
int write(int sock, const char *buf, int buflen );  
int send(int sock, const char* buf, int len, int flags);
```



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,
- **Wynik:** ilość wysłanych bajtów lub:
 - **SOCKET_ERROR**, kod błędu z *WSAGetLastError* (Windows),
 - **-1**, kod błędu z *errno* (Unix)
- **Blocking:** **send** czeka, aż w buforze nadawczym będzie wystarczająco dużo miejsca na wysłanie *buflen* bajtów
- **Nonblocking:** **send** zapisuje do bufora tyle, ile może (nie mniej niż 1) i zwraca ilość zapisanych bajtów. W przypadku braku miejsca w buforze, **send** zwraca **SOCKET_ERROR** a kod błędu = **WSAEWOULDBLOCK/EOULDBLOCK**.

Klient: Zapisywanie danych do gniazda (C#)

socket ► connect ► **write** ► read ► close

```
public int Send(byte[] buffer, SocketFlags socketFlags)
```

Metoda klasy `System.Net.Socket.Socket`
Zapisuje dane do bufora.

Parametry:

buffer: `System.Byte[]`

Tablica bajtów, które zawierają dane do wysłania.

socketFlags: `System.Net.Sockets.SocketFlags`

Bitowa kombinacja flag gniazda.



► Wynik: **ilość wysłanych bajtów**

Wyjątek	Warunek
<u>ArgumentNullException</u>	<i>buffer</i> jest null.
<u>SocketException</u>	Niepowodzenie dostępu do gniazda.
<u>ObjectDisposedException</u>	Gniazdo zostało zamknięte.

Klient: Wczytywanie danych z gniazda (C/C++)

socket ► connect ► write ► **read** ► close

lub

```
int read(int sock, char *buf, int buflen);  
int recv(int sock, char *buf, int buflen, int flags);
```

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,
- **Wynik:** $1 \leq \text{ilość odebranych bajtów} \leq \text{buflen}$, lub:
 - **0** – gdy połączenie zostało zerwane lub zdalnie zamknięte,
 - **SOCKET_ERROR**, kod błędu z *WSAGetLastError* (Windows),
 - **-1**, kod błędu z *errno* (Unix)
- **Blocking:** **recv** czeka, aż w buforze odbiorczym będzie minimum (domyślnie 1) bajtów do pobrania
- **Nonblocking:** **recv** wczytuje tyle danych, ile zostało odebrano (nie mniej niż 1) i zwraca ilość wczytanych bajtów.
W przypadku braku danych w buforze, **recv** zwraca **SOCKET_ERROR** a kod błędu = **WSAEWOULDBLOCK/EOULDBLOCK**.

Klient: Wczytywanie danych z gniazda (C#)

socket ► connect ► write ► **read** ► close

```
public int Receive(byte[] buffer, SocketFlags socketFlags)
```

Metoda klasy `System.Net.Socket.Socket`
Odczytuje dane z bufora gniazda.

Parametry:

buffer: [`System.Byte\[\]`](#)

Tablica bajtów, do której trafią odebrane dane.

socketFlags: [`System.Net.Sockets.SocketFlags`](#)

Bitowa kombinacja flag gniazda.

Odbieranie
danych!

► Wynik: ilość odebranych bajtów

Wyjątek	Warunek
<code>ArgumentNullException</code>	<i>buffer</i> jest null.
<code>SocketException</code>	Niepowodzenie dostępu do gniazda.
<code>ObjectDisposedException</code>	Gniazdo zostało zamknięte.
<code>SecurityException</code>	Brak uprawnień przy wywołaniu metody.

Klient: Transmisja danych z TcpClient (C#)

socket ► connect ► **NetworkStream** ► close

```
public int GetStream()
```

Metoda klasy `System.Net.Socket.TcpClient`

Zwraca obiekt `NetworkStream` do wysyłania i dobierania danych.

► Wynik: **NetworkStream**

Wyjątek	Warunek
<u>InvalidOperationException</u>	Obiekt <u>TcpClient</u> nie jest podłączony do zdalnego hosta.
<u>ObjectDisposedException</u>	Obiekt <u>TcpClient</u> został zamknięty.

Klient: Transmisja danych z TcpClient (C#)

socket ► connect ► **NetworkStream** ► close

```
TcpClient tcpClient = new TcpClient ();

// Uses the GetStream public method to return the NetworkStream.
NetworkStream netStream = tcpClient.GetStream ();

if (netStream.CanWrite)
{
    Byte[] sendBytes = Encoding.UTF8.GetBytes ("Is anybody there?");
    netStream.Write (sendBytes, 0, sendBytes.Length);
}

if (netStream.CanRead)
{
    // Reads NetworkStream into a byte buffer.
    byte[] bytes = new byte[tcpClient.ReceiveBufferSize];

    // Read can return anything from 0 to numBytesToRead.
    // This method blocks until at least one byte is read.
    netStream.Read (bytes, 0, (int)tcpClient.ReceiveBufferSize);

    // Returns the data received from the host to the console.
    string returndata = Encoding.UTF8.GetString (bytes);

    Console.WriteLine ("This is what the host returned to you: " + returndata);
}
```

Klient: Zamykanie połączenia (C/C++)

socket ► connect ► write ► read ► **close**

```
lub 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: Zamykanie połączenia (C#)

socket ► connect ► write ► read ► **close**

```
public int Close()
```

Metoda klasy `System.Net.Socket.Socket`

Zamyka gniazdo wraz z przydzielonymi zasobami oraz kończy aktywne połączenie ze zdalnym hostem.

Dla protokołu połączeniowego TCP warto uprzednio uruchomić funkcję **Shutdown**. Zapewnia ona wysłanie i odebranie wszystkich danych, które aktualnie są transmitowane.

Klient TCP: Implementacja w C++

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

    result = WSStartup(MAKEWORD(2, 0), &data);
    assert(result == 0);

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

    sockaddr_in service;
    service.sin_family = AF_INET;
    service.sin_port = htons(3301);
    service.sin_addr.s_addr = inet_addr("127.0.0.1");
    result = connect(sock, (sockaddr*)&service,
                    sizeof(sockaddr_in));
    assert(result != SOCKET_ERROR);

    char str[100];
    for(int i = 0; i < 3; i++) {
        if (!read_line(sock, str))
            break;
        printf("%d: %s", i, str);
    }
    closesocket(sock);
}
```

```
bool read_line(SOCKET sock, char* line)
{
    while(true)
    {
        int result = recv(sock, line, 1, 0);
        if (result == 0 || result ==
            SOCKET_ERROR)
            return false;
        if (*line++ == '\n')
            break;
    }
    *line = '\x0';
    return true;
}
```

Protokół serwera

```
Data 11/10/2010\r\n
Godzina 17:53:41\r\n
Jestes klientem #1\r\n
```

Klient TCP: A tak to będzie wyglądało w C#

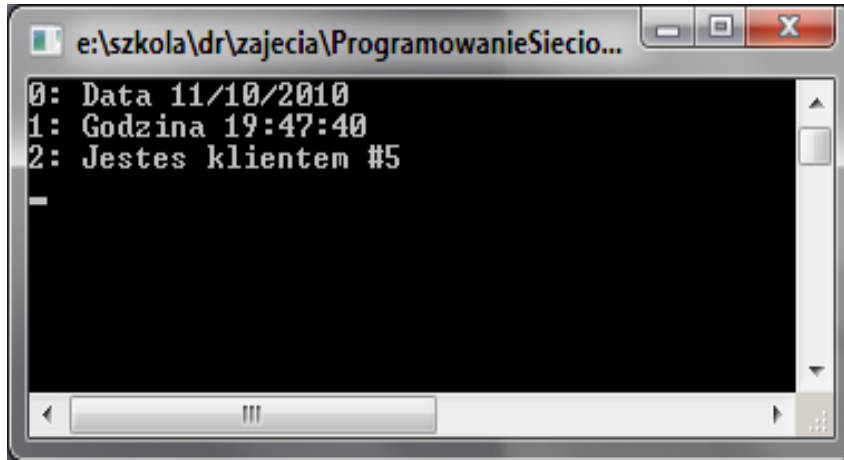
```
static void Main()
{
    Socket s = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Unspecified);
    s.Connect(new IPEndPoint(IPAddress.Parse("127.0.0.1"),
        3301));

    byte[] buffer = new byte[1024];
    int result = s.Receive(buffer);
    String time = Encoding.ASCII.GetString(buffer, 0,
        result);
    Console.WriteLine(time);
}
```

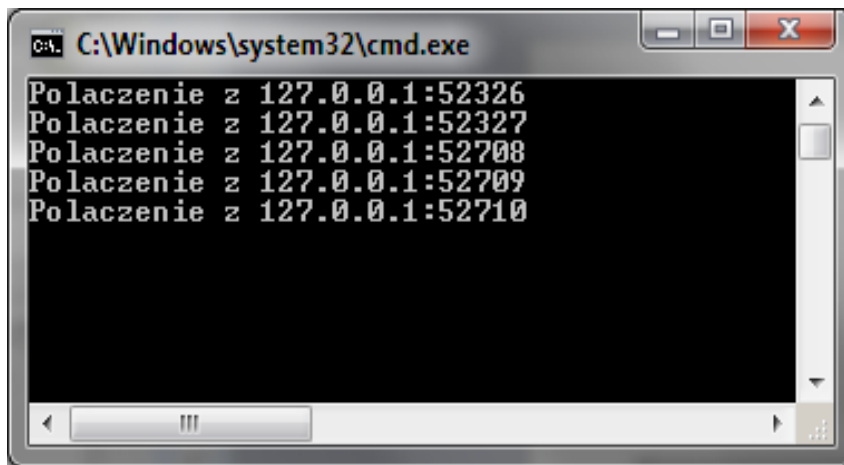
Klient TCP: A tak to będzie wyglądało w JAVA

```
3 import java.io.*;
4 import java.net.*;
5 import java.nio.charset.Charset;
6
7 public class TCP_Client {
8
9     Socket clientSocket;
10    public static void main() throws Exception
11    {
12        clientSocket = new Socket("127.0.0.1", 3301);
13        DataInputStream in = new DataInputStream(clientSocket.getInputStream());
14        byte[] bytes = new byte[1024];
15        int bytesRead = in.read(bytes);
16        modifiedMsg = new String(bytes, 0, bytesRead);
17        System.out.println("FROM SERVER: " + modifiedMsg);
18        clientSocket.close();
19    }
20 }
```

Klient i serwer TCP: Testowanie



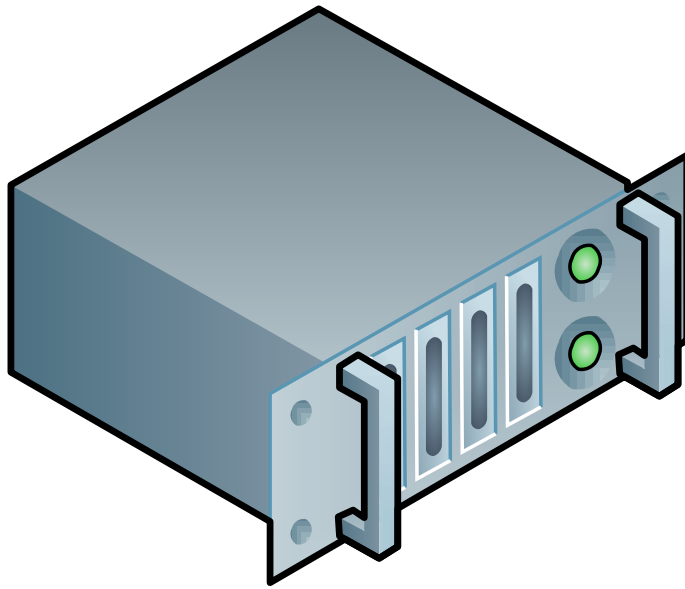
```
e:\szkola\dr\zajecia\ProgramowanieSiecio...
0: Data 11/10/2010
1: Godzina 19:47:40
2: Jestes klientem #5
```



```
C:\Windows\system32\cmd.exe
Polaczenie z 127.0.0.1:52326
Polaczenie z 127.0.0.1:52327
Polaczenie z 127.0.0.1:52708
Polaczenie z 127.0.0.1:52709
Polaczenie z 127.0.0.1:52710
```

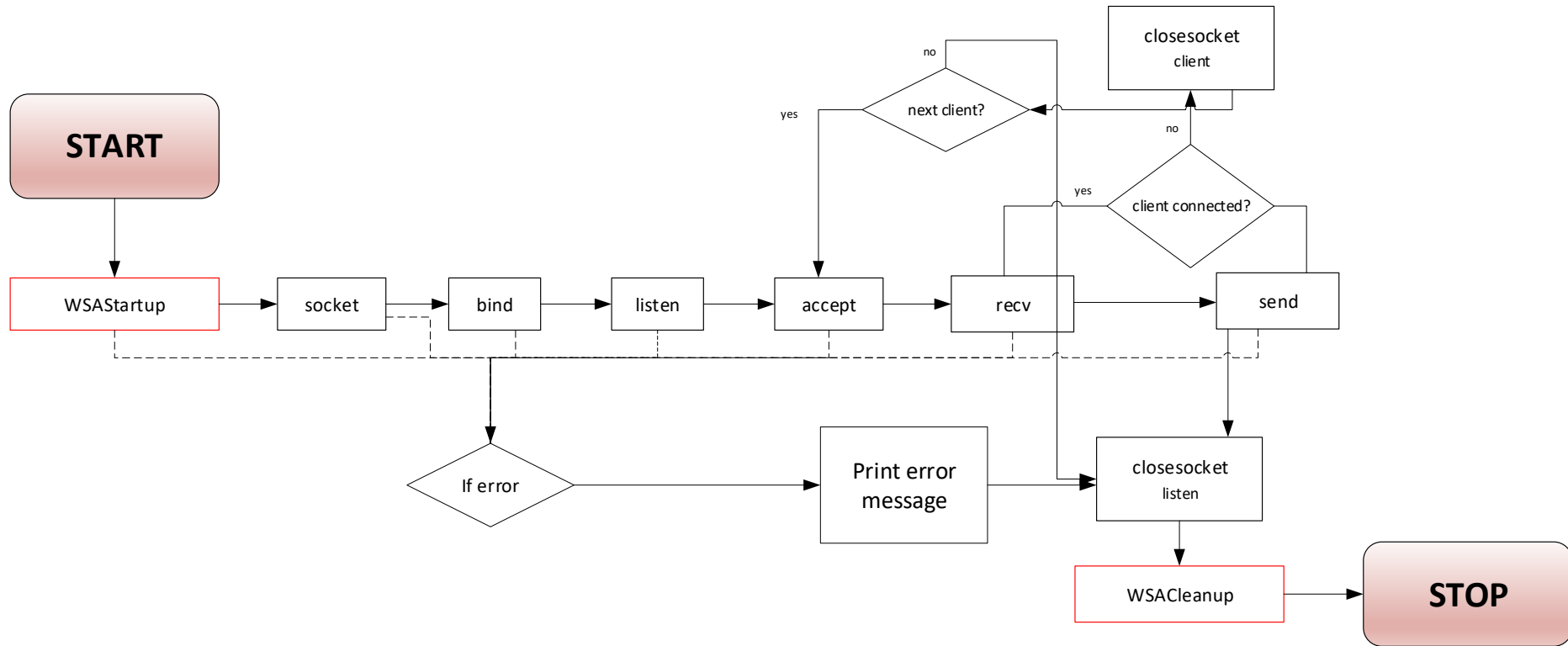
Protokół serwera

```
Data 11/10/2010\r\n
Godzina 17:53:41\r\n
Jestes klientem #1\r\n
```

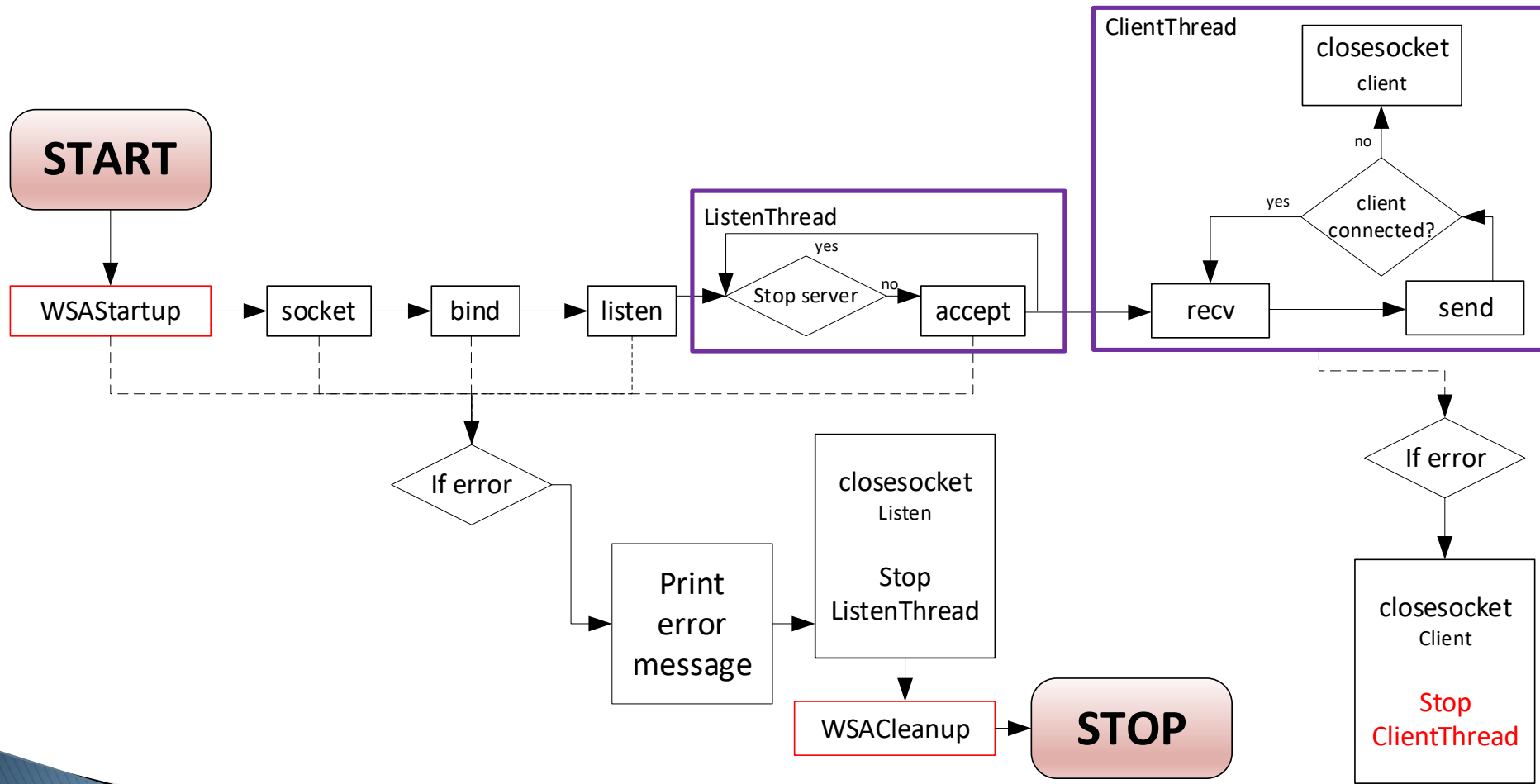



Server
TCP

Szkielet programu sieciowego – serwer echo



Szkielet programu sieciowego – serwer echo wielokliencki



Serwer: Tworzenie gniazda

socket ► bind ► listen ► accept ► read ► write ► close

Serwer wykorzystuje funkcję/klasę **socket** w taki sam sposób, jak klient – do tworzenia gniazda.

To samo tyczy się funkcji **transmisji danych** oraz **zamykania gniazda**.

Serwer: Konfiguracja gniazda (C/C++)

socket ► **bind** ► listen ► accept ► read ► write ► close

```
int bind(int sock, struct sockaddr *loc, int loclen);
```

Funkcja dowiązuje gniazdo do lokalnego adresu, określonego w *loc*. Wykonuje tzw. **otwarcie bierne**.

- ▶ *sock* – uchwyt gniazda (zwrócony przez **socket**)
- ▶ *loc* – wskaźnik na strukturę **sockaddr** przechowującą informacje o protokole oraz lokalny adres na jaki będą łączyć się klienci.
- ▶ *loclen* – długość, w bajtach, struktury **sockaddr** dla danego protokołu.
- ▶ **Wynik:** 0, lub:
 - **SOCKET_ERROR**, kod błędu z *WSAGetLastError* (Windows),
 - -1, kod błędu z *errno* (Unix)

Najczęstszy powód niepowodzenia funkcji **bind** to niezamknięcie gniazda nasłuchowego. Zdarza się to przy nagłym zamknięciu aplikacji serwera.

Serwer: Konfiguracja gniazda (C/C++)

socket ► **bind** ► listen ► accept ► read ► write ►
close

Inicjalizacja struktury gniazdowej

```
sockaddr_in service;  
service.sin_family = AF_INET;  
service.sin_addr.s_addr = INADDR_ANY;  
service.sin_port = htons(3370);
```

a dla przypomnienia, jak to było w przypadku klienta

```
sockaddr_in service;  
service.sin_family = AF_INET;  
service.sin_addr.s_addr = inet_addr("127.0.0.1");  
service.sin_port = htons(3370);
```

Serwer: Konfiguracja gniazda (C/C++)

socket ► **bind** ► listen ► accept ► read ► write ► close

```
sockaddr_in service;  
service.sin_family = AF_INET;  
service.sin_addr.s_addr = INADDR_ANY;  
service.sin_port = htons(3370);
```

```
#define INADDR_ANY      (ULONG)0x00000000  
#define INADDR_LOOPBACK 0x7f000001  
#define INADDR_BROADCAST (ULONG)0xffffffff
```

Adres IP w postaci tekstowej:

```
.s_addr = inet_addr("127.0.0.1")
```

Jeżeli serwer posiada trzy interfejsy:

192.168.1.1; 10.1.0.21; 212.191.78.134

to może nasłuchiwać na wszystkich
[INADDR_ANY] lub tylko na wybranym
[inet_addr("10.1.0.21")]

#define AF_UNSPEC	0	// unspecified
#define AF_UNIX	1	// local to host (pipes, portals)
#define AF_INET	2	// internetwork: UDP, TCP, etc.
#define AF_IMPLINK	3	// arpanet imp addresses
#define AF_PUP	4	// pup protocols: e.g. BSP
#define AF_CHAOS	5	// mit CHAOS protocols
#define AF_NS	6	// XEROX NS protocols
#define AF_IPX	AF_NS	// IPX protocols: IPX, SPX, etc.
#define AF_ISO	7	// ISO protocols
#define AF_OSI	AF_ISO	// OSI is ISO
#define AF_ECMA	8	// european computer manufacturers
#define AF_DATAKIT	9	// datakit protocols
#define AF_CCITT	10	// CCITT protocols, X.25 etc
#define AF_SNA	11	// IBM SNA
#define AF_DECnet	12	// DECnet
#define AF_DLI	13	// Direct data link interface
#define AF_LAT	14	// LAT
#define AF_HYLINK	15	// NSC Hyperchannel
#define AF_APPLETALK	16	// AppleTalk
#define AF_NETBIOS	17	// NetBios-style addresses
#define AF_VOICEVIEW	18	// VoiceView
#define AF_FIREFOX	19	// Protocols from Firefox
#define AF_UNKNOWN1	20	// Somebody is using this!
#define AF_BAN	21	// Banyan
#define AF_ATM	22	// Native ATM Services
#define AF_INET6	23	// Internetwork Version 6
#define AF_CLUSTER	24	// Microsoft Wolfpack
#define AF_12844	25	// IEEE 1284.4 WG AF
#define AF_IRDA	26	// IrDA
#define AF_NETDES	28	// Network Designers OSI & gateway

Serwer: Konfiguracja gniazda (C#)

socket ► **bind** ► listen ► accept ► read ► write ► close

```
public void Bind(EndPoint localEP)
```

Metoda obiektu `System.Net.Socket.Socket`
Dowiązuje gniazdo do punktu końcowego.

Parametry

- `localEP`: [System.Net.EndPoint](#)
Punkt końcowy dowiązywany do gniazda.

Wyjątek	Warunek
<u>ArgumentNullException</u>	<i>localEP</i> jest null.
<u>SocketException</u>	Błąd przy próbie dostępu do gniazda.
<u>ObjectDisposedException</u>	Gniazdo zostało zamknięte.
<u>SecurityException</u>	Metoda wywołująca stojąca wyżej na stosie wywołań nie posiada uprawnień do wykonywania tej operacji.

Serwer: Nasłuchiwanie (C/C++)

socket ► bind ► **listen** ► accept ► read ► write ► close

```
int listen(int sock, int backlog);
```

Funkcja uruchamia tryb nasłuchu dla zadanego gniazda.

- ▶ *sock* – uchwyt gniazda (zwrócony przez **socket**)
- ▶ *backlog* – ilość połączeń oczekujących na odebranie funkcją **accept**,
- ▶ *loclen* – długość, w bajtach, struktury **sockaddr** dla danego protokołu.
- ▶ **Wynik:** 0, lub:
 - **SOCKET_ERROR**, kod błędu z *WSAGetLastError* (Windows),
 - -1, kod błędu z *errno* (Unix)

Serwer: Nasłuchiwanie (C#)

socket ► bind ► **listen** ► accept ► read ► write ► close

```
public void Listen(int backlog)
```

Metoda obiektu `System.Net.Socket.Socket`
Zamienia gniazdo w gniazdo słuchacza

Parametry

- `backlog: int`
Długość kolejki połączeń oczekujących

Wyjątek	Warunek
SocketException	Błąd przy próbie dostępu do gniazda.
ObjectDisposedException	Gniazdo zostało zamknięte.

Serwer: Przyjmowanie połączenia (C/C++)

socket ► bind ► listen ► **accept** ► read ► write ► close

```
int accept(int sock, sockaddr *loc, int &len);
```

Funkcja uruchamia tryb nasłuchu dla zadanego gniazda.
Wykonuje tzw. **otwarcie bierne**.

- *sock* – uchwyt gniazda (zwrócony przez **socket**)
- *loc* – wskaźnik do struktury zawierającej informacje o zdalnym punkcie końcowym (adres IP + port),
- **Wynik:** **uchwyt gniazda połączenia przychodzącego + *loc***,
lub:
 - **INVALID_SOCKET**, kod błędu z *WSAGetLastError* (Windows),
 - **-1**, kod błędu z *errno* (Unix)

Serwer: Przyjmowanie połączenia (C#)

socket ► bind ► listen ► **accept** ► read ► write ► close

```
public Socket Accept()
```

Metoda obiektu `System.Net.Socket.Socket`

Zwraca nowe gniazdo dla nowo nawiązanego połączenia.

Wyjątek	Warunek
SocketException	Błąd przy próbie dostępu do gniazda.
ObjectDisposedException	Gniazdo zostało zamknięte.
InvalidOperationException	Gniazdo akceptujące nie jest gniazdem słuchającym. Należy wpierw uruchomić Bind i Listen .

Klient/Serwer:

Operacje blokujące i nieblokujące

Tryb blokujący (blocking): operacje gniazd wykonywane są synchronicznie.

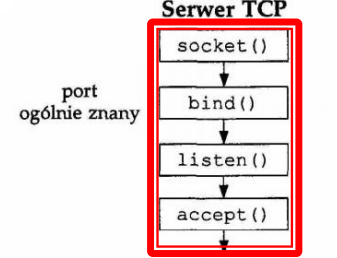
Funkcje kończą się po całkowitym wykonaniu operacji lub w przypadku błędu (np. `accept`, `recv`).

Tryb nieblokujący (nonblocking): operacje gniazdowe wykonywane są asynchronicznie.

Funkcje kończą się po zleceniu podsystemowi gniazd danej operacji (np. `connect`, `send`) lub w przypadku błędu.

Np. w przypadku wywołania metody `Accept` dla nieblokującego gniazda i braku oczekującego połączenia, natychmiast zostanie wyrzucony `SocketException`.

Serwer TCP: Implementacja w C++



```
int main(int argc, char* argv[])
{
```

```
    WSADATA data;
    int result, counter = 0;
    sockaddr_in service, remote;
```

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

```
    SOCKET listen_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    assert(listen_socket != INVALID_SOCKET);
```

```
    service.sin_family = AF_INET;
    service.sin_port = htons(3301);
    service.sin_addr.s_addr = INADDR_ANY;
    result = bind(listen_socket, (sockaddr*)&service, sizeof(sockaddr_in));
    assert(result != SOCKET_ERROR);
```

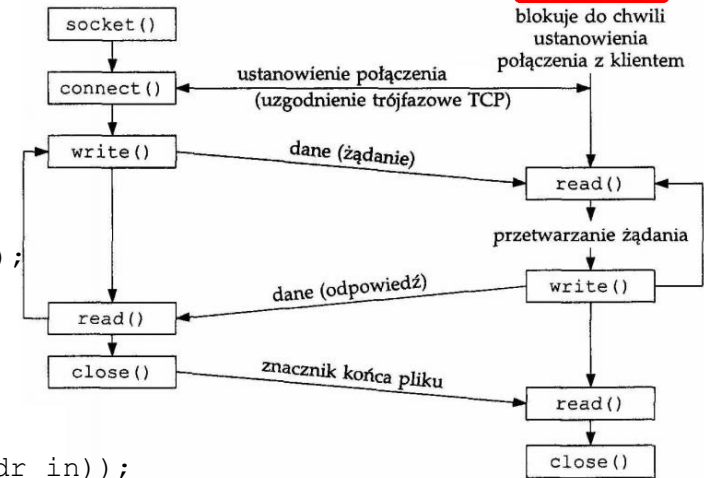
```
    result = listen(listen_socket, 5);
    assert(result != SOCKET_ERROR);
```

Tutaj główna pętla programu serwera

```
    closesocket(listen_socket);
    return 0;
```

```
}
```

Klient TCP



Protokół serwera

```
Data 11/10/2010\r\n
Godzina 17:53:41\r\n
Jestes klientem #1\r\n
```

Serwer TCP: Implementacja w C++

główna pętla programu serwera

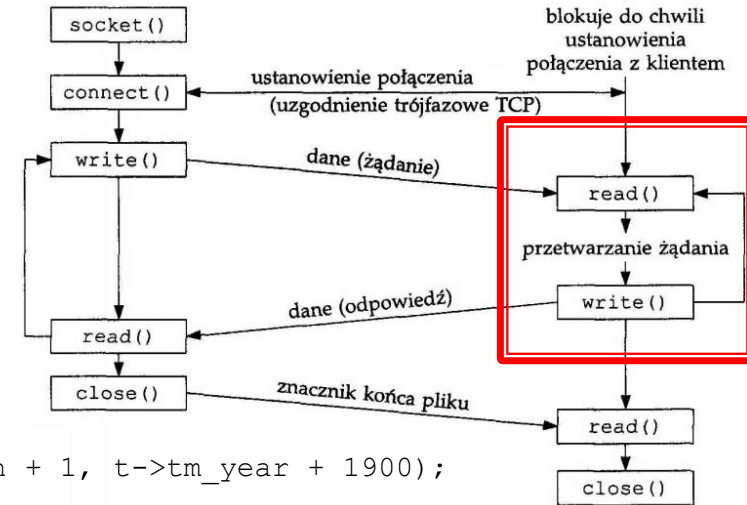
```
while(true)
{
    int size = sizeof(sockaddr_in);
    SOCKET client = accept(listen_socket,
        (sockaddr*)&remote, &size);
    printf("Polaczenie z %s:%d\n",
        inet_ntoa(remote.sin_addr),
        ntohs(remote.sin_port));
    assert(client != INVALID_SOCKET);
    char str[100];
    time_t curr_time;
    time(&curr_time);
    tm *t = gmtime(&curr_time);

    sprintf(str, "Data %02d/%02d/%04d\r\n", t->tm_mday, t->tm_mon + 1, t->tm_year + 1900);
    send(client, str, strlen(str), 0);

    sprintf(str, "Godzina %02d:%02d:%02d\r\n", t->tm_hour, t->tm_min, t->tm_sec);
    send(client, str, strlen(str), 0);

    counter++;
    sprintf(str, "Jestes klientem #%d\r\n", counter);
    send(client, str, strlen(str), 0);
    closesocket(client);
}
```

Klient TCP



Protokół serwera

```
Data 11/10/2010\r\n
Godzina 17:53:41\r\n
Jestes klientem #1\r\n
```


Serwer TCP: A tak to będzie wyglądało w C#

Klient TCP

```
static void Main()
{
    Socket s = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Unspecified);
    s.Bind(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 3301));
    s.Listen(5);

    int counter = 0;
    while (true)
    {
        Socket cli = s.Accept();
        Console.WriteLine("Polaczenie z {0}",
            cli.RemoteEndPoint.ToString());

        DateTime now = DateTime.Now;
        StringBuilder sb = new StringBuilder();
        sb.AppendLine(string.Format("Data: {0:00}/{1:00}/{2:0000}",
            now.Day, now.Month, now.Year));
        sb.AppendLine(string.Format("Czas: {0:00}:{1:00}:{2:00}",
            now.Hour, now.Minute, now.Second));
        sb.AppendLine(string.Format("Jestes klientem #{0}",
            counter));

        byte[] bufor = Encoding.ASCII.GetBytes(sb.ToString());
        cli.Send(bufor);
        cli.Close();
    }
}
```

```
static void Main()
{
    Socket s = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Unspecified);
    s.Connect(new IPEndPoint(IPAddress.Parse("127.0.0.1"),
        3301));

    byte[] buffer = new byte[1024];
    int result = s.Receive(buffer);
    String time = Encoding.ASCII.GetString(buffer, 0,
        result);
    Console.WriteLine(time);
}
```

Serwer TCP: A tak to będzie wyglądało w C#

Klient TCP

```
TcpListener server = new TcpListener(IPAddress.Parse("127.0.0.1"), 3301);
server.Start();

while (true)
{
    TcpClient cli = server.AcceptTcpClient();
    Console.WriteLine("Połączenie z {0}",
        cli.Client.RemoteEndPoint.ToString());

    NetworkStream stream = cli.GetStream();

    StringBuilder sb = new StringBuilder();

    //          ....

    byte[] bufor = System.Text.Encoding.ASCII.GetBytes(sb.ToString());
    stream.Write(bufor, 0, bufor.Length);

    cli.Close();
}
```

```
static void Main()
{
    Socket s = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Unspecified);
    s.Connect(new IPEndPoint(IPAddress.Parse("127.0.0.1"),
        3301));

    byte[] buffer = new byte[1024];
    int result = s.Receive(buffer);
    String time = Encoding.ASCII.GetString(buffer, 0,
        result);
    Console.WriteLine(time);
}
```

```
int i = stream.Read(bufor, 0, bufor.Length);
```

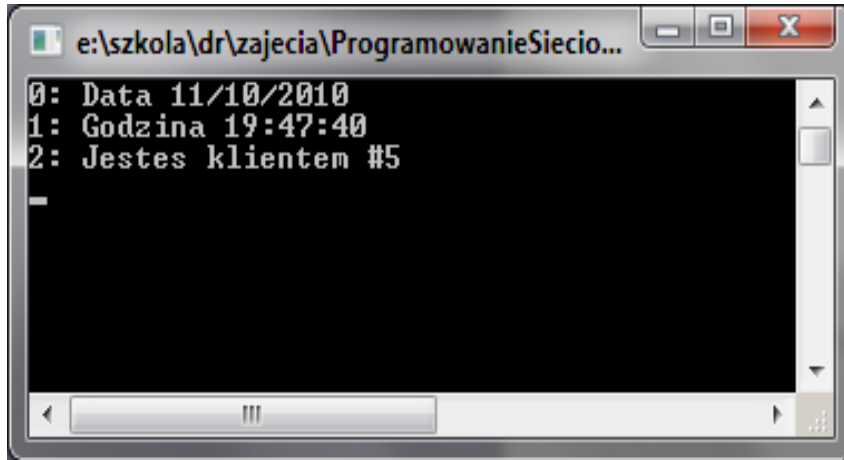
Serwer TCP: A tak to będzie wyglądało w JAVA

Klient TCP

```
2 import java.net.*;
3 import java.text.SimpleDateFormat;
4 import java.util.Calendar;
5
6 class TCPServer
7 {
8     public static void main(String argv[]) throws Exception
9     {
10         String clientSentence = "";
11         ServerSocket welcomeSocket = new ServerSocket(3301);
12         int counter = 0;
13
14         while(true)
15         {
16             Socket connectionSocket = welcomeSocket.accept();
17
18             Calendar cal = Calendar.getInstance();
19             SimpleDateFormat data_df = new SimpleDateFormat("yyyy.MM.dd");
20             clientSentence.concat("Data: " + data_df.format(cal.getTime()));
21             SimpleDateFormat time_df = new SimpleDateFormat("HH:mm:ss");
22             clientSentence.concat("Godzina: " + time_df.format(cal.getTime()));
23             counter++;
24             clientSentence.concat("Jestes klientem #" + counter);
25
26             byte buf[] = clientSentence.getBytes(Charset.forName("UTF-8"));
27             DataOutputStream out = new DataOutputStream(connectionSocket.getOutputStream());
28             out.write(buf);
29             connectionSocket.close();
30         }
31     }
32 }
```

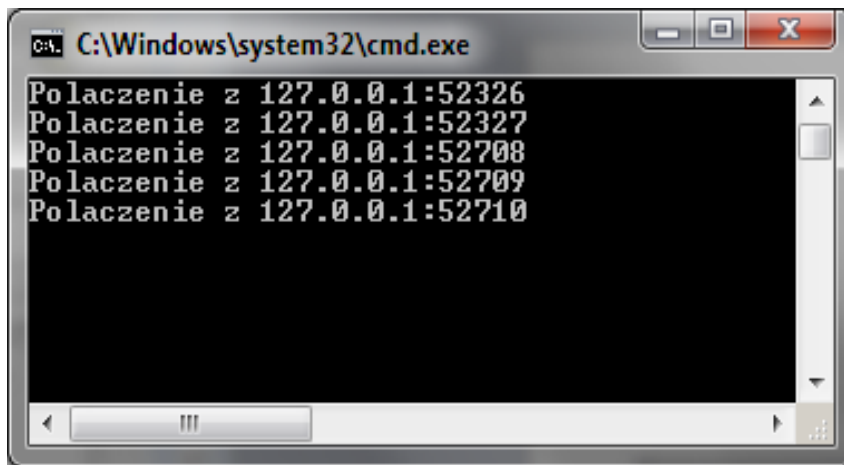
```
3 import java.io.*;
4 import java.net.*;
5 import java.nio.charset.Charset;
6
7 public class TCP_Client {
8
9     Socket clientSocket;
10     public static void main() throws Exception
11     {
12         clientSocket = new Socket("127.0.0.1", 3301);
13         DataInputStream in = new DataInputStream(clientSocket.getInputStream());
14         byte[] bytes = new byte[1024];
15         int bytesRead = in.read(bytes);
16         modifiedMsg = new String(bytes, 0, bytesRead);
17         System.out.println("FROM SERVER: " + modifiedMsg);
18         clientSocket.close();
19     }
20 }
```

Klient i serwer TCP: Testowanie



A screenshot of a text editor window with the title bar "e:\szkola\dr\zajecia\ProgramowanieSiecio...". The window contains the following text:

```
0: Data 11/10/2010
1: Godzina 19:47:40
2: Jestes klientem #5
-
```



A screenshot of a Windows command prompt window with the title bar "C:\Windows\system32\cmd.exe". The window shows five lines of connection attempts:

```
Polaczenie z 127.0.0.1:52326
Polaczenie z 127.0.0.1:52327
Polaczenie z 127.0.0.1:52708
Polaczenie z 127.0.0.1:52709
Polaczenie z 127.0.0.1:52710
```

1. Uruchomić serwer testowy (opisany w dalszej części wykładu).
2. Program *telnet* w celu przetestowania serwera.
3. Uruchomić klienta testowego.

Protokół serwera

```
Data 11/10/2010\r\n
Godzina 17:53:41\r\n
Jestes klientem #1\r\n
```