Politechnika Łódzka
Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki

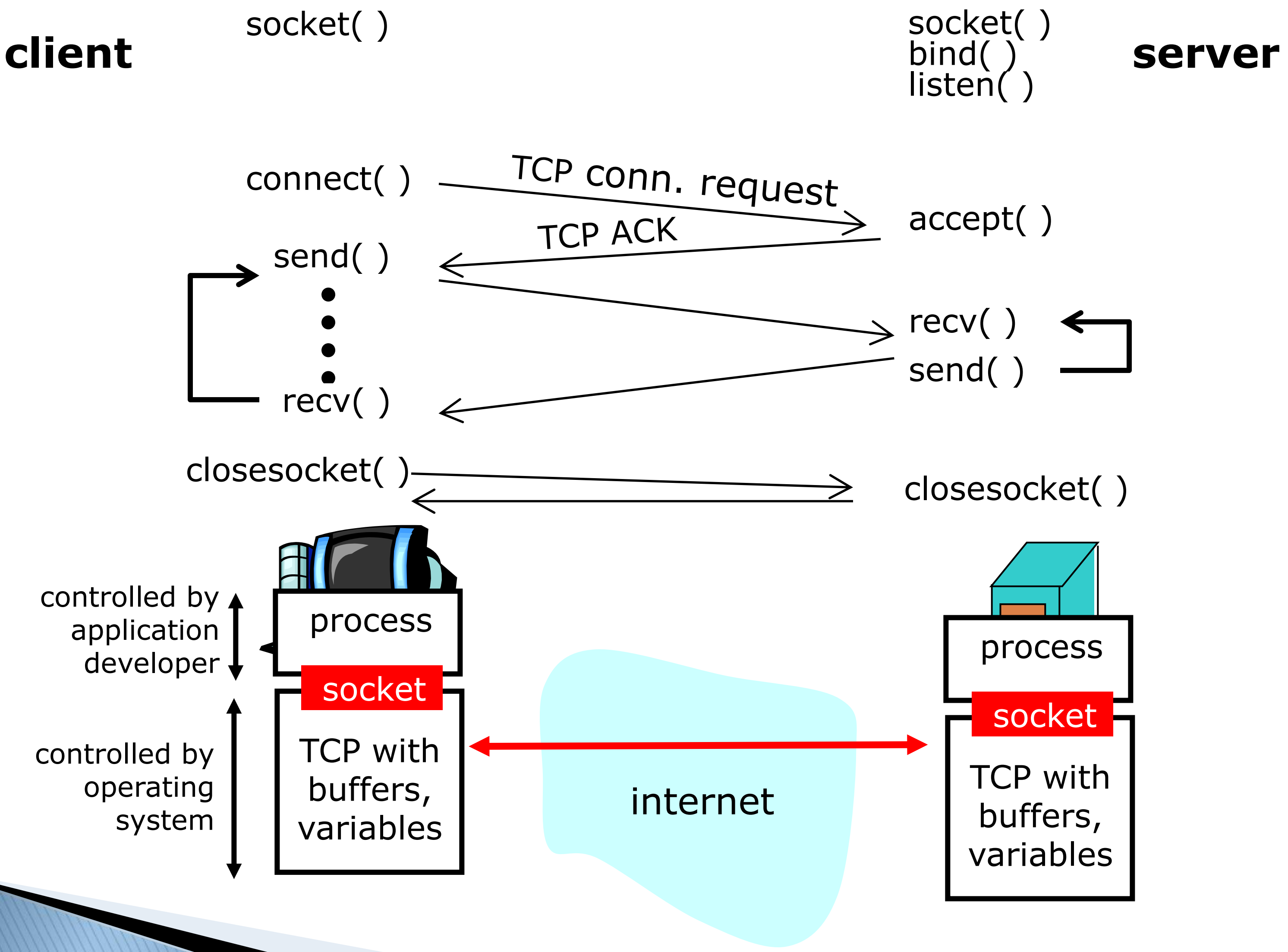
Network programming

Lecture 4
TCP server

Instytut Informatyki Stosowanej
Politechniki Łódzkiej

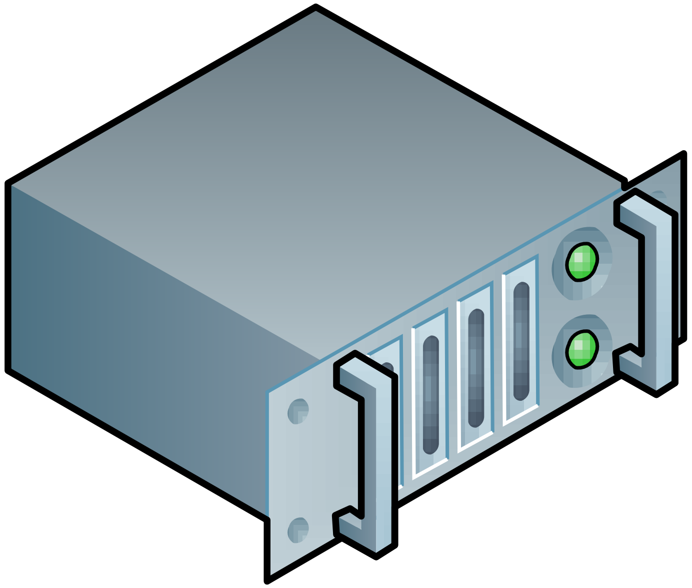
Dr Radosław Wajman

Client+server: connection-oriented



TCP

Server



Server: building socket

socket ► bind ► listen ► accept ► recv ► send ► closesocket

Server calls the **`socket`** function in the same way as client to create the socket.

Similar matter is for **`recv`**, **`send`** and **`closesocket`**.

Server: socket configuration [1]

socket ► **bind** ► listen ► accept ► recv ► send ► closesocket

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

The function binds the socket to the lockal IP adress(es) given in `loc`.

- *sock* – socket handle (returned by **socket**)
- *loc* – pointer at the **sockaddr** structure with the protocol, IP adresses (which will be used to connect by clients) and port number.
- *loclen* – length, in bytes, of the **sockaddr** structure for the given protocol.
- **Result**: 0, or:
  - SOCKET_ERROR, error code from *WSAGetLastError* (Windows),
  - -1, error code from *errno* (Unix)

The most common cause of error calling **bind**
is the non-closed listen socket.
It happens when the previous instance of the application is unexpectedly terminated..

Server: socket configuration [2]

socket ► **bind** ► listen ► accept ► recv ► send ► closesocket

The socket structure initialization

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

remeber the client case ....

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

Server: socket configuration [3]

socket ► **bind** ► listen ► accept ► recv ► send ► closesocket

```
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
```

IP address in the text format:

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

If the server consists of three interfaces:
192.168.1.1; 10.1.0.21; 212.191.78.134
than it can use them all for listening
[**INADDR_ANY**]
or use only one
[**inet_addr**("10.1.0.21”)]

Server: Listening

socket ► bind ► **listen** ► accept ► recv ► send ► closesocket

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

The function starts listening mode for given socket.

- *sock* – socket handle (returned by **socket**)
- *backlog* – number of connections waiting for accepting by the **accept** function,
- *loclen* – lentgth, in bytes, of the **sockaddr** structure for given protocol.
- **Result**: 0, or:
  - SOCKET_ERROR, error code from *WSAGetLastError* (Windows),
  - –1, error code from *errno* (Unix)

Serwer: Przyjmowanie połączenia

`socket ► bind ► listen ► accept ► recv ► send ► closesocket`

```
int accept(int sock, sockaddr *rmt, int backlog);
```

The function processes the listening mode on given socket. Processes the so called **passive openning**.

- *sock* - socket handle (returned by **`socket`**)
- *rmt* - pointer at the **`sockaddr`** structure with the remote address of the connecting entity,
- **Result**: socket handle for the incomming connection + rmt, or:
  - **INVALID_SOCKET**, error code from *WSAGetLastError* (Windows),
  - **-1**, error code from *errno* (Unix)

Client/Server: Blocking and non-blocking modes

Blocking mode: the socket functions are called synchronously and do not return until they can complete their action or in case of error (i.e. **`accept`**, **`recv`**).

Non-blocking mode: the socket funkctions are called asynchronously and do return right after delegating to the system performing the given action (i.e. **`connect`**, **`send`**) or in case of error.

Setting the non-blocking mode for the socket

```
int s = socket(AF_INET, SOCK_STREAM, 0);
u_long iMode = 1;
ioctlsocket(s, FIONBIO, &iMode);
```

TCP Client: C implementation

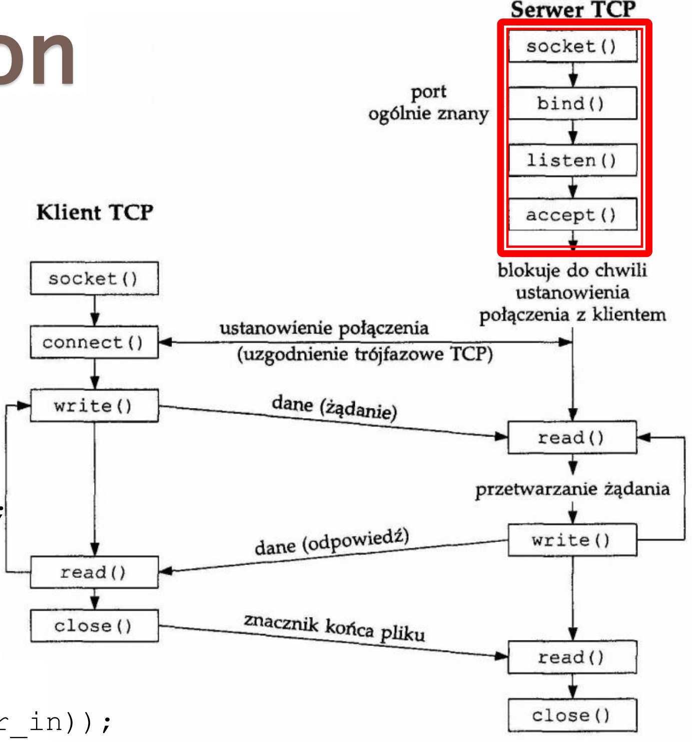
```
int main(int argc, char* argv[])
{
    WSAData data;
    int result;

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

    SOCKET sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    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;
}
```

Server message

```
Date 11/10/2010\r\n
Time 17:53:41\r\n
Client no. #1\r\n
```

TCP Server: C implementation

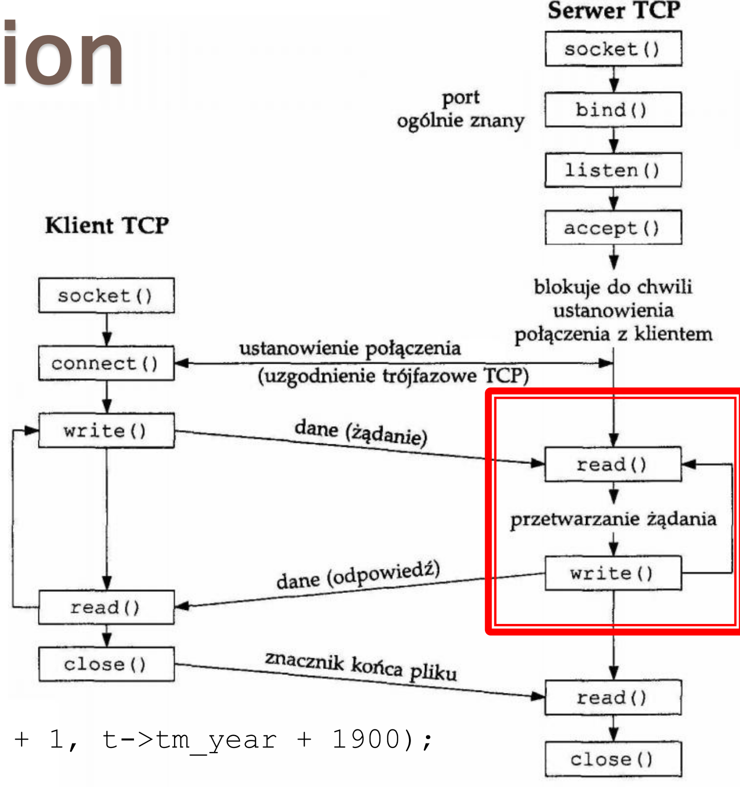


```
int main(int argc, char* argv[])
{
    WSAData data;
    int result, counter = 0;
    sockaddr_in service, remote;

    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;
}
```

TCP Server: C implementation

main server loop (client service)

```
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);
}
```



Server message

```
Date 11/10/2010\r\n
Time 17:53:41\r\n
Client no. #1\r\n
```

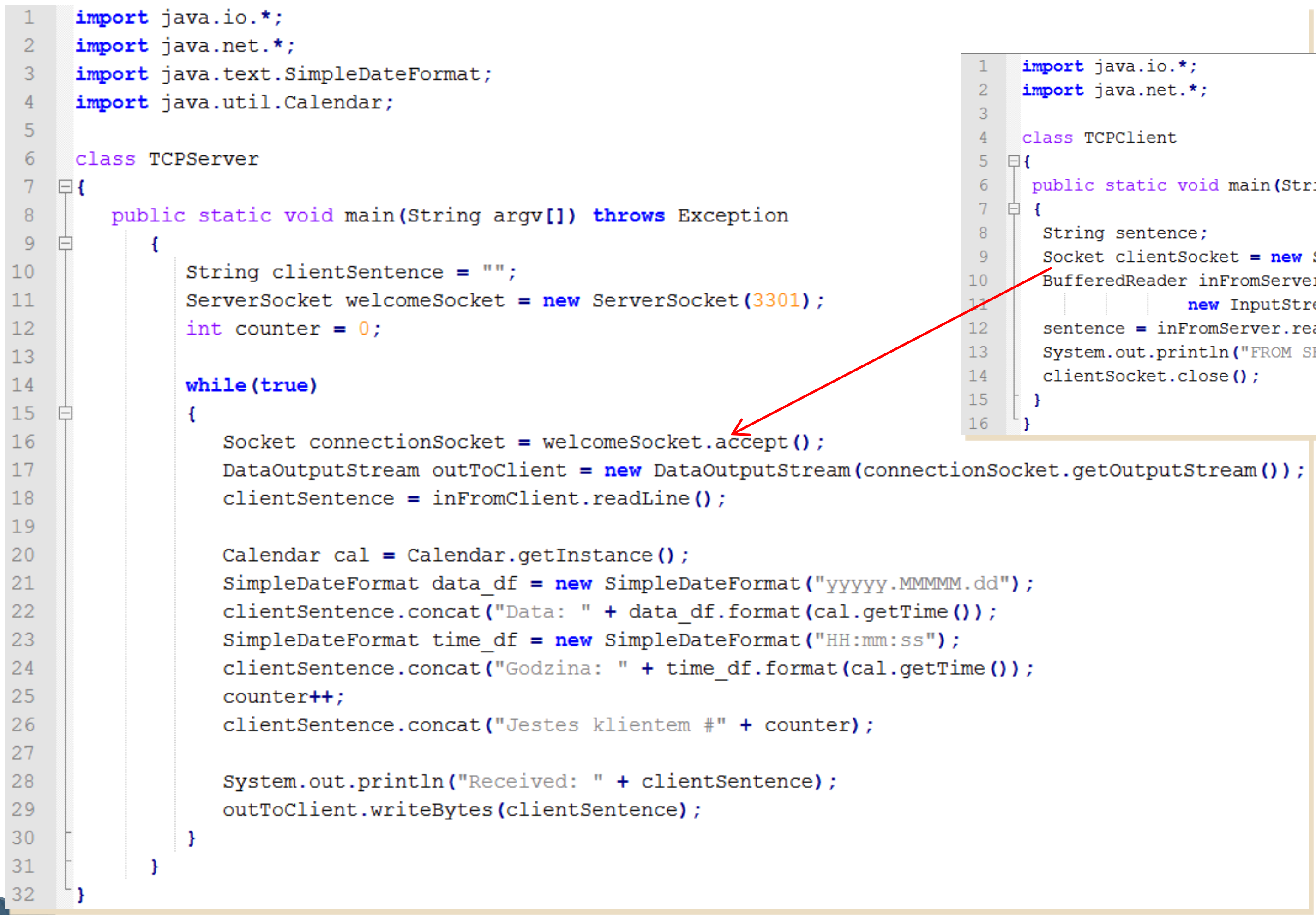
TCP server: C# implementation

```
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();
    }
}
```

TCP Client

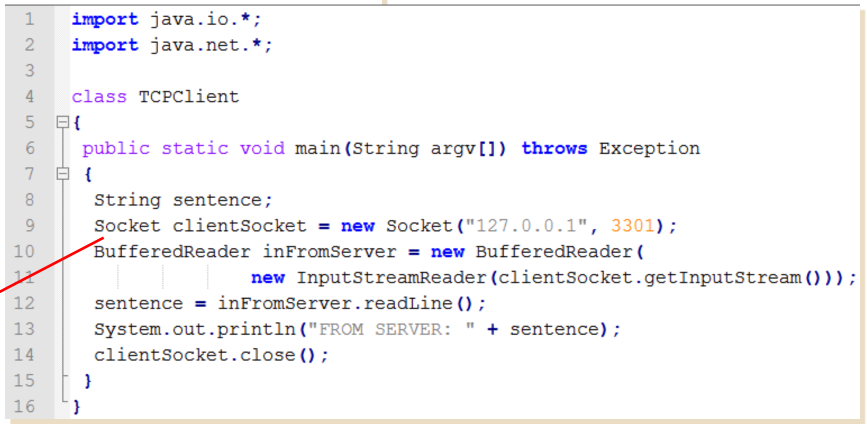
```
static void Main()
{
    Socket s = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Unspecified);
    s.Connect(new IPEndPoint(IPAddress.Parse("127.0.0.1"),
        3301));

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

TCP Server: JAVA implementation



```java
import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
import java.util.Calendar;

class TCPServer
{
    public static void main(String argv[]) throws Exception
      {
         String clientSentence = "";
         ServerSocket welcomeSocket = new ServerSocket(3301);
         int counter = 0;

         while(true)
         {
            Socket connectionSocket = welcomeSocket.accept();
            DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());
            clientSentence = inFromClient.readLine();

            Calendar cal = Calendar.getInstance();
            SimpleDateFormat data_df = new SimpleDateFormat("yyyyy.MMMMM.dd");
            clientSentence.concat("Data: " + data_df.format(cal.getTime());
            SimpleDateFormat time_df = new SimpleDateFormat("HH:mm:ss");
            clientSentence.concat("Godzina: " + time_df.format(cal.getTime());
            counter++;
            clientSentence.concat("Jestes klientem #" + counter);

            System.out.println("Received: " + clientSentence);
            outToClient.writeBytes(clientSentence);
         }
      }
}
```

TCP Client



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

class TCPClient
{
 public static void main(String argv[]) throws Exception
 {
  String sentence;
  Socket clientSocket = new Socket("127.0.0.1", 3301);
  BufferedReader inFromServer = new BufferedReader(
              new InputStreamReader(clientSocket.getInputStream()));
  sentence = inFromServer.readLine();
  System.out.println("FROM SERVER: " + sentence);
  clientSocket.close();
 }
}
```