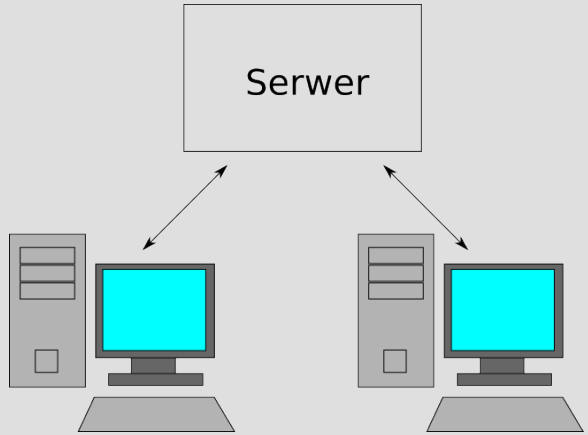
Networking

Adam Krechowicz

Client–Server architecture



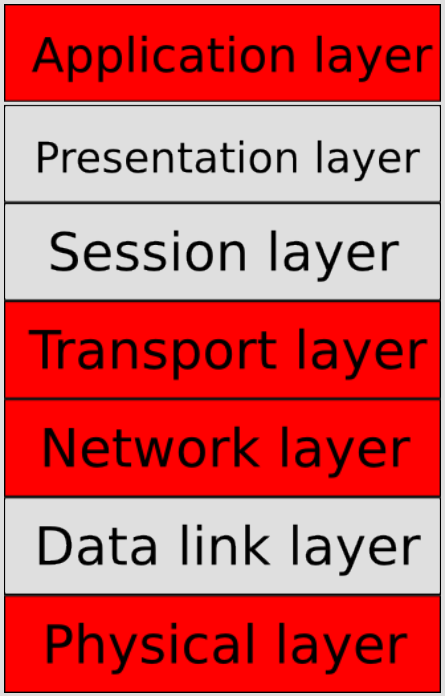
Advantages

- Many people can work on the same data
- All of them has access to actual data
- Updates are easy

Drawbacks

- Requires central server
- Server might become overloaded or may fail to work properly

Open System Interconnect



IP – Internet Protocol

- IPv4 – 192.168.0.1
- IPv6 – 2001:0db8:0000:0000:0000:0000:1428:57ab

IPv4

- 32–bit addresses
- Connections are identified by ports

Ports

- Many connections may coexists on one system
- Identified by unique 16–bits numbers
- 65535 available connections
- Unified port numbers
  - 80 – HTTP
  - 443 – HTTPS
  - 25 – SMTP
  - 143 – IMAP
- For custom systems high number should be choosed (e. g. 65000 and above)
- First 1024 ports are reserved

Nazwy

- *URI* – Uniform Resource Identifier
  - *URL* – Uniform Resource Locator
  - *URN* – Uniform Resource Name

Struktura

schema://user:password@address/path?query#part

```
1  try {
2    InetAddress addr = InetAddress.getByName("www.wp.pl");
3    System.out.println(Arrays.toString(addr.getAddress()));
4    System.out.println(addr.getHostAddress());
5    System.out.println(addr.getHostName());
6  } catch (UnknownHostException ex) {
7    Logger.getLogger(DNS.class.getName()).log(Level.SEVERE, null, ex);
8  }
```

Result

```
[-44, 77, 98, 9]
212.77.98.9
www.wp.pl
```

Local addresses

- localhost
- 127.0.0.1

```
1  try {
2    URL url = new URL("http://www.google.pl");
3    BufferedReader br = new BufferedReader(new InputStreamReader(url.openStream↩
       ()));
4    String buf;
5    while ((buf = br.readLine()) != null){
6      System.out.println(buf);
7    }
8    br.close();
9  } catch (MalformedURLException ex) {
10   Logger.getLogger(Urls.class.getName()).log(Level.SEVERE, null, ex);
11 } catch (IOException ex) {
12   Logger.getLogger(Urls.class.getName()).log(Level.SEVERE, null, ex);
13 }
```

URL

- getHost()
- getFile()
- getProtocol()
- getPort()
- getDefaultPort()

URLConnection

- Abstract class that represents connection with some protocol
- Allows to handle connection
- Concrete class is created based on the appropriate schema (e. g. "http")

```
1  try {
2    URL url = new URL("http://www.google.pl");
3    URLConnection con = url.openConnection();
4    Map<String, List<String>> fields = con.getHeaderFields();
5    for (String key: fields.keySet()){
6      System.out.println(key);
7      List<String> values = fields.get(key);
8      for (String value: values){
9        System.out.println("\t"+value);
10     }
11   }
12 } catch (MalformedURLException ex) {
13   Logger.getLogger(Urls.class.getName()).log(Level.SEVERE, null, ex);
14 } catch (IOException ex) {
15   Logger.getLogger(Urls.class.getName()).log(Level.SEVERE, null, ex);
16 }
```

Rezultat

```
1  null
2    HTTP\/1.1 200 OK
3  Date
4    Wed, 04 Apr 2018 10:19:15 GMT
5  Accept-Ranges
6    none
7  X-Frame-Options
8    SAMEORIGIN
9  Cache-Control
10   private, max-age=0
11 Vary
12   Accept-Encoding
13 Set-Cookie
14   ...
15 Content-Type
16   text\/html; charset=ISO-8859-2
```

HttpUrlConnection

```
1 HttpURLConnection huc = (HttpURLConnection)con;
2 System.out.println(huc.getRequestMethod());
3 System.out.println(huc.getResponseCode());
4 System.out.println(huc.getResponseMessage());
```

Result

```
GET
200
OK
```

Transfer Control Protocol

- Protocol for reliable connections
  - Connection is established in reliable way (SYN->ACK,SYN->ACK)
  - All transfers needs to be acknowledged (DANE->ACK)
  - Connection is closed in reliable way (FIN->ACK,FIN->ACK)
- All packages will be delivered
- None of packages will be delivered twice
- Packages are delivered in order in what they was sent
- Data can be group into bigger packages before sending
- Connection can be one-way or two-way
- Two way connection can be closed only on one-way

TCP uasage

- when none of package can be lost
- when order of packages is important
- for example turn-based games (chess, checkers, etc.)

User Datagram Protocol

- connect less protocol
  - can send data to multiple receivers
  - there is no guarantee that data will be received
- receiver may received packages in wrong order
- when resending, receiver may receive duplicates

UDP usage

- when none every package needs to be delivered
- when we have own implementation of package control delivery
- when there is a lot of packages to send
- when we want to send to many clients in the same time
- for example real–time games (ping-pong, shooters)
- sending audio and video

Implementation – Fundamentals

- We need at least two programs
- May be run on different computers or on the same
- One program for server
- Second for client
- Server waits for connections from clients
- Client establishes connection to server

TCP – Fundamentals

- Communication is handled my socket
- Data are wrapped into stream
- Programs are blocked until receiving information
- Usually server handle many clients in different threads

TCP Server

```
1  Socket s = null;
2  try {
3    ServerSocket server = new ServerSocket(65500);
4    s = server.accept();
5    InputStream is = s.getInputStream();
6    byte[] buffer = new byte[128];
7    int i = is.read(buffer);
8    System.out.println(new String(buffer));
9    s.close();
10   server.close();
11 } catch (IOException ex) {
12   Logger.getLogger(TCPServer.class.getName()).log(Level.SEVERE, null, ex);
13 }
```

TCP Client

```
1  try {
2    Socket s = new Socket("localhost", 65500);
3    OutputStream os = s.getOutputStream();
4    os.write("Hello World".getBytes());
5    s.close();
6  } catch (IOException ex) {
7    Logger.getLogger(TCPClient.class.getName()).log(Level.SEVERE, null, ex);
8  }
```

Order of execution

1. Server – ServerSocket server = new ServerSocket(65500);
2. Client – Socket s = new Socket(łocalhost", 65500);
3. Server – s = server.accept();
4. Client – OutputStream os = s.getOutputStream();
5. Server – InputStream is = s.getInputStream();
6. Client – os.write("Hello World".getBytes());
7. Server – int i = is.read(buffer);
8. Client – s.close();
9. Server – s.close();

Stream wrapping

Sending

```
1 ObjectOutputStream oos = new ObjectOutputStream(os);
2 oos.writeInt(5);
3 oos.writeFloat(3.14f);
```

Receiving

```
1 ObjectInputStream ois = new ObjectInputStream(is);
2 System.out.println(ois.readInt());
3 System.out.println(ois.readFloat());
```

Sending objects

```
1  package com.wyklad.networking;
2
3  import java.io.Serializable;
4
5  public class Obiekt implements Serializable {
6      int i;
7
8      public Obiekt(int i){
9          this.i = i;
10     }
11
12     public String toString(){
13         return ""+i;
14     }
15
16 }
```

Sending objects

Sending

```
1  ObjectOutputStream oos = new ObjectOutputStream(os);
2  Obiekt o = new Obiekt(9876);
3  oos.writeObject(o);
```

Receiving

```
1  ObjectInputStream ois = new ObjectInputStream(is);
2  Obiekt o = (Obiekt)ois.readObject();
3  System.out.println(o);
```

UDP – Fundamentals

- Data are send in a form of datagram
- Datagram wraps the array of bytes
- Datagram contains the information about receiver
- Each datagram may be directed to different receivers
- Datagram socket do not need to be assigned to concrete receiver

UDP Server

```
1  try {
2    DatagramSocket ds = new DatagramSocket(65000);
3    byte[] buf = new byte[100];
4    DatagramPacket dp = new DatagramPacket(buf, 100);
5    ds.receive(dp);
6    System.out.println(new String(buf));
7    ds.close();
8  } catch (SocketException ex) {
9    Logger.getLogger(UDPClient.class.getName()).log(Level.SEVERE, null, ex);
10 } catch (IOException ex) {
11   Logger.getLogger(UDPClient.class.getName()).log(Level.SEVERE, null, ex);
12 }
```

UDP Client

```
1  try {
2    DatagramSocket ds = new DatagramSocket();
3    String s = "Hello";
4    byte[] buf = s.getBytes();
5    DatagramPacket dp = new DatagramPacket(buf,s.length(), InetAddress.↩
       getLocalHost(), 65000);
6    ds.send(dp);
7    ds.close();
8  } catch (SocketException ex) {
9    Logger.getLogger(UDPClient.class.getName()).log(Level.SEVERE, null, ex);
10 }
```

Wrapping datagram with stream

Sending

```
1  ByteArrayOutputStream bos = new ByteArrayOutputStream(100);
2  ObjectOutputStream oss = new ObjectOutputStream(bos);
3  oos.writeInt(7);
4  oss.writeFloat(3.5f);
5  Datagram Packet dp = new DatagramPacket(bos.toByteArray(), bos.size(), ↩
       InetAddress.getLocalHost(), 65000);
6  ds.send(dp);
```

Wrapping datagram with stream

Receiving

```
1  byte[] buf = new byte[100];
2  DatagramPacket dp = new DatagramPacket(buf, 100);
3  ds.receive(dp);
4  ByteArrayInputStream bis = new ByteArrayInputStream(buf);
5  ObjectInputStream ois = new ObjectInputStream(bis);
6  System.out.println(ois.readInt());
7  System.out.println(ois.readFloat());
```

Sending object

```
1  package com.wyklad.networking;
2
3  import java.io.Serializable;
4
5  public class Obiekt implements Serializable {
6      int i;
7
8      public Obiekt(int i){
9          this.i = i;
10     }
11
12     public String toString(){
13         return ""+i;
14     }
15
16 }
```

Sending object

Sending

```
1  ByteArrayOutputStream bos = new ByteArrayOutputStream(100);
2  ObjectOutputStream oss = new ObjectOutputStream(bos);
3  oss.writeObject(new Obiekt(7));
4  dp = new DatagramPacket(bos.toByteArray(), bos.size(), InetAddress.getLocalHost↩
       (), 65000);
5  ds.send(dp);
```

Receiving

```
1  byte [] buf = new byte[100];
2  DatagramPacket dp = new DatagramPacket(buf, 100);
3  ds.receive(dp);
4  ByteArrayInputStream bis = new ByteArrayInputStream(buf);
5  ObjectInputStream ois = new ObjectInputStream(bis);
6  System.out.println((Obiekt)ois.readObject());
```

Data was not received

Sending

```
1  Socket s = new Socket("localhost", 65500);
2  OutputStream os = s.getOutputStream();
3  ObjectOutputStream oos = new ObjectOutputStream(os);
4  oos.writeFloat(3.14f);
5  s.close();
```

Receiving

```
1  ServerSocket server = new ServerSocket(65500);
2  s = server.accept();
3  InputStream is = s.getInputStream();
4  ObjectInputStream ois = new ObjectInputStream(is);
5  System.out.println(ois.readFloat());
6  s.close();
7  server.close();
```

Data was not received

- Server does not received any data
- There might be java.io.EOFException
- Socket was closed before data was sent
- Too little to send data
- Stream needs to be flushed

Flushing stream

```
1  Socket s = new Socket("localhost", 65500);
2  OutputStream os = s.getOutputStream();
3  ObjectOutputStream oos = new ObjectOutputStream(os);
4  oos.writeFloat(3.14f);
5  oos.flush();
6  s.close();
```

Server can not be run

- During program running there is java.net.BindException
- Address already in use
- Occurs when we run server for second time
- Previous connection was not closed properly
- Socket still waiting for some late data
- We can force to reuse address

Reusing addresses

```
1  ServerSocket server = new ServerSocket();
2  server.setReuseAddress(true);
3  server.bind(new InetSocketAddress(65000));
```

Blocking

- Server and clients are blocked
- Operations on sockets are blocking
- We need to ensure that we execute complementary operations on both sides
- If one end is sending the other should be receiving in the same time
- If both are receiving they will be blocked
- The order in which we are opening input and output streams is very important

Blocked streams

Server

```
1  ServerSocket server = new ServerSocket(65500);
2  s = server.accept();
3  ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
4  ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
```

Client

```
1  Socket s = new Socket("localhost", 65500);
2  ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
3  ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
```

Avoiding of stream blocking

- If both ends will first open input streams they will be blocked
- We should open input stream on the one side
- and output stream on the second
- Alternatively we can open output streams on both ends
- When creating input stream the socket is waiting for the header
- Second end should deliver header by opening output stream

THE END!

Additional reading:

- https://docs.oracle.com/javase/8/docs/technotes/guides/net/overview/
- https://docs.oracle.com/javase/tutorial/networking/index.html
- https://www.javatpoint.com/java-networking