Software Construction
Basic IO: Sockets

Dhammika Elkaduwe

Department of Computer Engineering
Faculty of Engineering

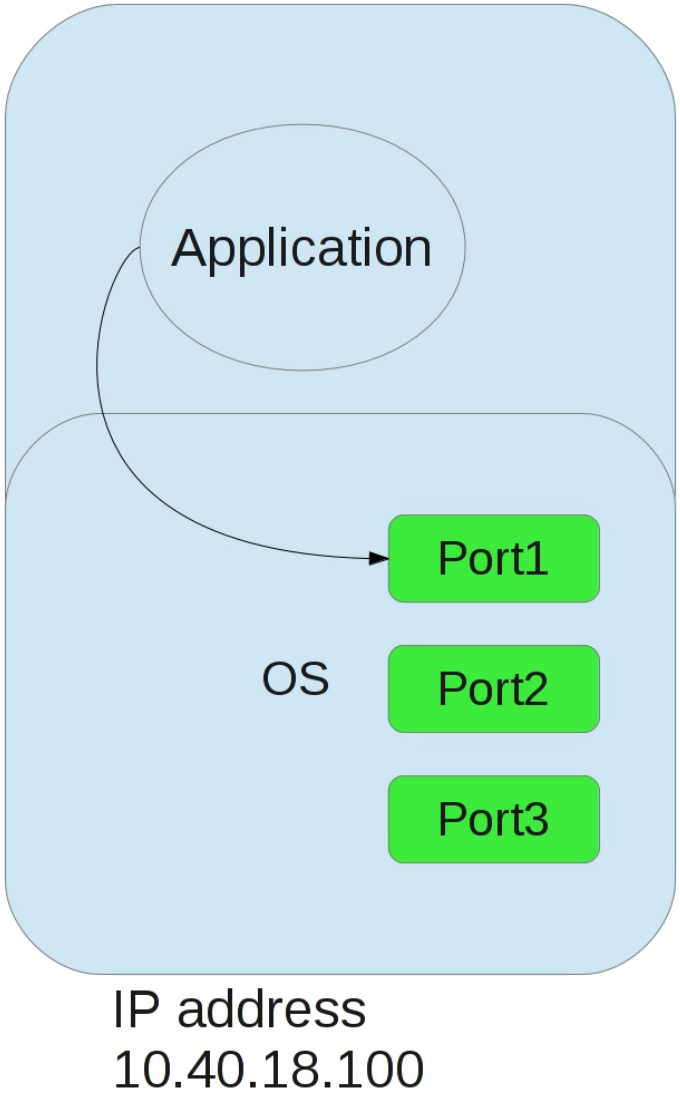
University of Peradeniya

ILO: what to look for

Look at the following concepts using an example:

- Use of sockets (in Java)
- OOP concepts
- Threads Vs. Code (how to use threads)
- IO handling
- Data exchanges

Sockets: basics



- Servers have ports to which one can establish connections
  - Clients make connection requests, server accepts
- Some ports are used for specific purposes (example: HTTP:80, DHCP: 67, SMTP:25 etc)and some are for general purpose communications
- Applications, via the OS can set-up connections via ports

Java TCP sockets

Java provides two types of sockets: server sockets and clients

```
import java.net.Socket;
// For clients
// Which provides
Sockets(String host, int port);
// for 2way communications
InputStream getInOutStream(); // for input
OutputStream getOutPutStream();// for output
```

```
import java.net.ServerSocket;
// For servers
// Which provides
serverSocket = new ServerSocket(socket);
Socket socket = serverSocket.accept();
```

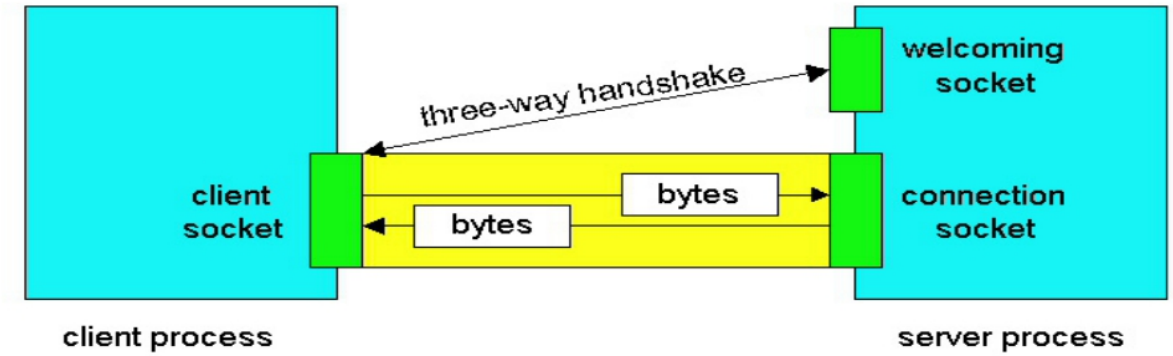
Java TCP sockets

Java provides two types of sockets: server sockets and clients

```
import java.net.Socket;
// For clients
// Which provides
Sockets(String host, int port);
// for 2way communications
InputStream getInOutStream(); // for input
OutputStream getOutPutStream();// for output
```

```
import java.net.ServerSocket;
// For servers
// Which provides
serverSocket = new ServerSocket(socket);
Socket socket = serverSocket.accept();
```

Server side

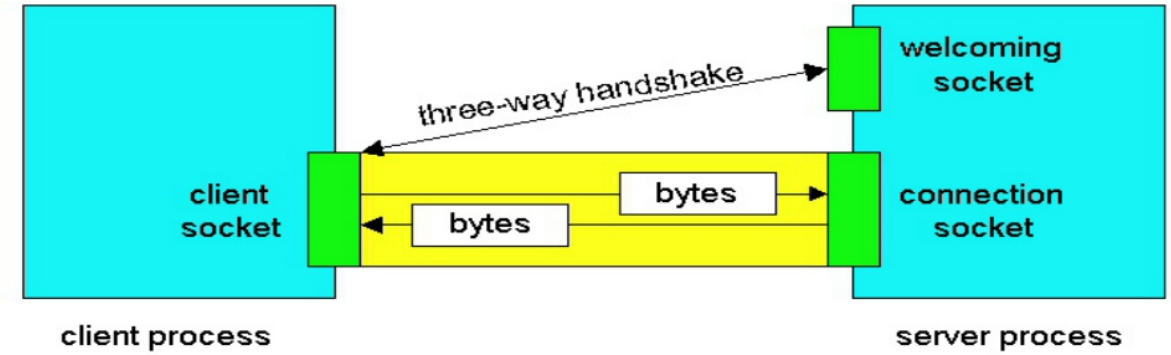


Server:

- Creates a socket (the welcoming socket)
- Waits for connections (using the *accept()* method)
- Once a request is received, opens a different socket for communications (returned by the accept method)

```
// create the welcome socket
serverSocket = new ServerSocket(socket);
Socket socket = serverSocket.accept();
// connection established.
// send and receive data via the new socket
```

Server side ...



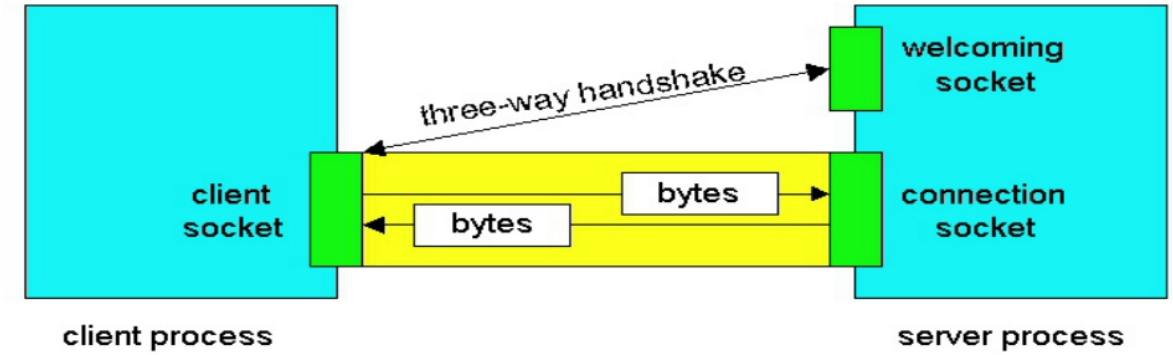
Server:

- A socket has two streams for input and output
- These streams can be used for receiving and sending
- These streams can be wrapped in buffered readers to improve performance.

```
InputStramReader inStream = new
    InputStreamReader(socket.getInputStream);
BufferedReader in = new BufferedReader(inStream);
```

see Server1.java

Connecting to the server



To connect as a client:

- You can write your own client code or
- use an existing tool like Linux tool *nc*

```
$
$ nc localhost 1250
```

see Server1.java

Question

Can more than one concurrent clients connect to this server?

```
while(true) {
  Socket socket = serverSocket.accept();
  handle(socket);
}

private void handle(Socket socket) throws IOException {

  ....
  for(line = in.readLine();
  line != null && !line.equals("quit");
  line = in.readLine()) {
    ...
  }
}
```

see Server1.java

Question

Can more than one concurrent clients connect to this server?

NO. Server socket is free, but the thread is doing something else.

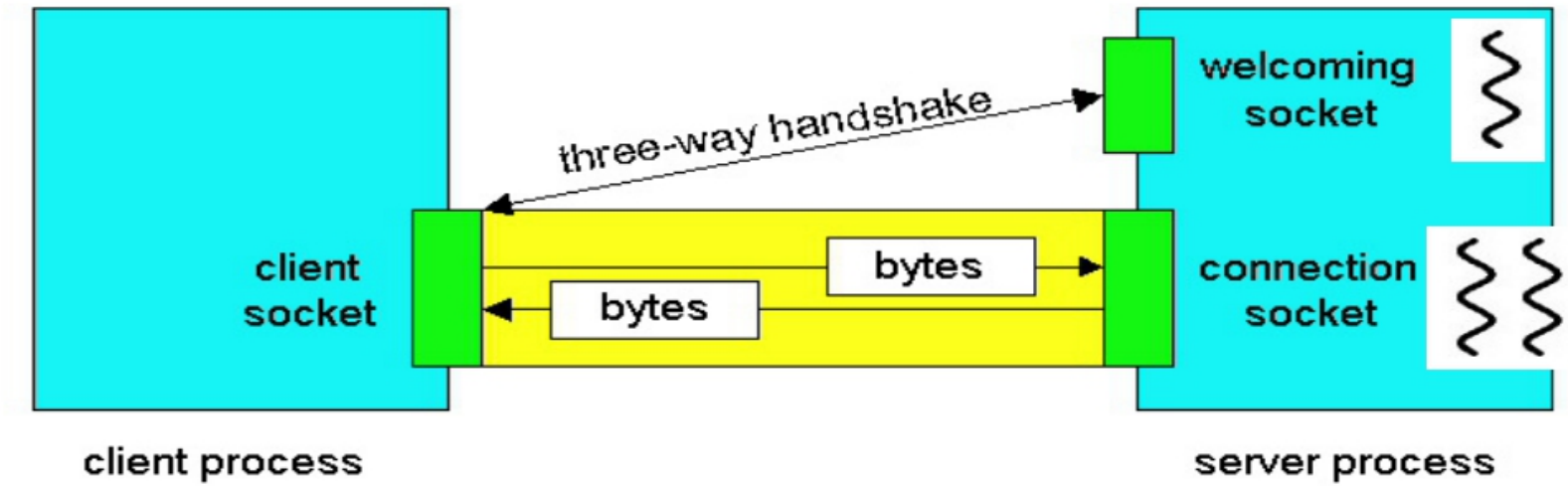
```
while(true) {
  Socket socket = serverSocket.accept();
  handle(socket);
}

private void handle(Socket socket) throws IOException {

  ....
  for(line = in.readLine();
  line != null && !line.equals("quit");
  line = in.readLine()) {
    ...
  }
}
```

see Server1.java

Task 1

- We cannot have concurrent connections since there is only one thread (two sockets)
- **Task 1** Make the server multi-threaded
  - Main thread handles the server part
  - Once a connection is established hand it over to another thread.
- Should we *extend* thread or **implement** runnable?

Task 1: Basic idea



Mo

of the problem:

- Single welcome socket (and associated state) common to all the communication sockets
- Welcome socket and communication sockets are different entities (communicating via interface)