2025/10/25 23:20 1/4 TCP - Socket communication

### **Socket communication**

The client sends requests to the server over a TCP socket connection, and the server responds to these requests. Here are the basic steps involved in integrating software systems or components using TCP socket communication:

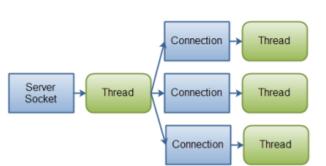
- 1. **Select a protocol**: TCP/IP is a common protocol for socket communication, but other protocols like UDP can also be used depending on the requirements.
- 2. **Determine the message format**: Decide on the format of the messages that will be exchanged between the client and server. This could be a simple text-based format or a more complex binary format.
- 3. **Define the communication interface**: Define the functions or APIs that will be used for communication between the client and server.
- 4. **Create the server**: Write the code for the server that listens for incoming client connections and handles incoming requests.
- 5. **Create the client**: Write the code for the client that connects to the server and sends requests.
- 6. **Handle errors**: Implement error handling mechanisms to ensure that communication errors are handled gracefully and do not cause the system to crash or become unstable.
- 7. **Test and iterate**: Test the system thoroughly and make any necessary changes or improvements to ensure that it is functioning correctly.

#### Features:

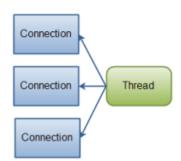
- Socket ::= IP address + (TCP/UPD) port number. A Socket is a combination of ip address and port number.
- TCP Sockets provides 'real-time' data transfer
  - binary data transfer but can be normal text or ISON, XML as well
  - no direct method sharing (can be implemented by hand)
  - TCP and UDP connections are possible. UDP is min 3 times quicker but one-way communication
- Persistent or On-Demand communication channel
  - because of connection time-loss usually persistent channels are better, but periodically 'ping' messages should be sent. (in order to avoid connection closing). In case of any problems reconnection is possible
  - in case of UDP channels an extra TCP channel is available for synchronizing in online games
- Results in the fastest possible transmission:
  - $\circ$  Where the number of transactions per second up to  $\sim$  50 transactions, there should have been applied. (20ms / sec transfer)

### Blocking and non-blocking TCP sockets in Java

Traditional Multi threaded socket

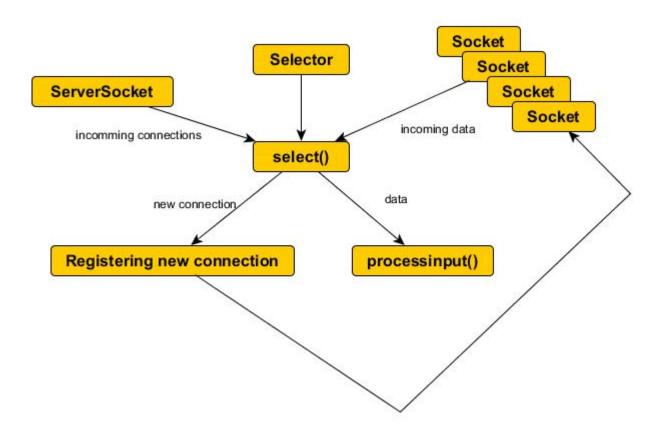


### Non blocking 1 thread socket



### Reading:

- http://tutorials.jenkov.com/java-nio/nio-vs-io.html
- http://www.javaworld.com/article/2073344/core-java/use-select-for-high-speed-networking.html



# **Non-blocking loop**

```
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
serverSocketChannel.socket().bind(new InetSocketAddress(9999));
serverSocketChannel.configureBlocking(false); // ez a sor jelzi a
blokkolásmentes működést

while(true){
    SocketChannel socketChannel = serverSocketChannel.accept();
```

2025/10/25 23:20 3/4 TCP - Socket communication

```
if(socketChannel != null){
    // the connection is accepted
}
```

# Java examples

Java example for Blocking and Non-Blocking Socket

## **Exercises**

### Exercise 1.

Create a simplified FTP (file transport) client and **blocking** server where the client can send or download text files from the server:

#### **General use-cases**

- 1. ) Client connects to the server and sends a 'file listing' message
- 2. ) Server sends back the list of the downloadable files
- Client lists the files and asks the user what action they want to take? Upload or download? ('u' or 'd')
- 4. ) In both cases users must give the full file name with extension
- 5. ) The client sends the selected file to the server (upload) or downloads the selected file from the server to a specific directory.

#### Server viewpoint

- 1. ) After connecting, it reads the files from the /store subdirectory and sends the file names to the client after receiving the listing message.
- 2. ) We are waiting for the client's 'u' or 'd' operation
- 3. ) We get a filename from the client and if the action is 'd' (download), we read the file content and return its contents
- 4. ) If the operation is 'u' (upload), we open a new file with the specified name and wait for the data to be written to the file.

### **Client viewpoint**

- 1. ) The client connects and waits for the list of files coming back and writes it to the console
- 2. ) We ask for the "u" or "d" key
- 3. ) Then we'll ask for the file-name as well.
- 4. ) The client reads the files from the /files folder, or creates the downloaded file here
- 5. ) If you press "d", it creates /files/ and writes data from the server

6. ) If you press "u", /files/ is sent to the server

**Exercise 2.** Modify the **blocking** UDP code so that you can transfer a burned-in name and existing text or image file larger than 2 kbytes and verify that it was successfully sent.

**Exercise 3.** Try out the non-blocking starter Java code and examine/debug the code.

From:

https://edu.iit.uni-miskolc.hu/ - Institute of Information Science - University of Miskolc

Permanent link:

https://edu.iit.uni-miskolc.hu/tanszek:oktatas:iss\_t:integration\_based\_on\_tcp\_ip\_sockets?rev=1678089229

Last update: 2023/03/06 07:53

