

TCP - 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:

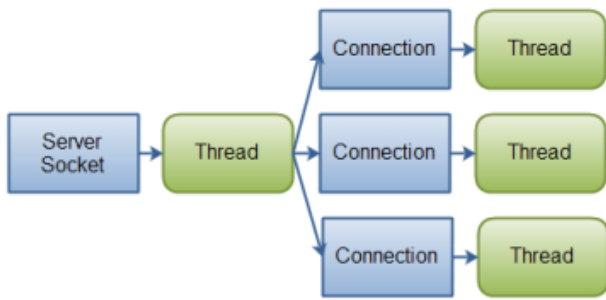
1. **Select a protocol:** TCP/IP is a common protocol for socket communication, but other protocols like UDP can also be used depending on the requirements.
2. **Determine the message format:** Decide on the format of the messages that will be exchanged between the client and server. This could be a 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:

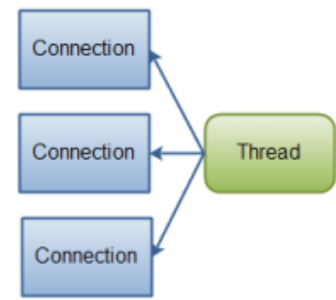
- Socket ::= IP address + (TCP/UDP) port number. A Socket is a combination of ip address and port number.
- TCP Sockets provides 'real-time' data transfer
 - binary data transfer but can be normal text or JSON, XML as well
 - no direct method sharing (can be implemented by hand)
 - TCP and UDP connections 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:
 - Where the number of transactions per second up to ~ 50 transactions, there should have been applied. (20ms / sec transfer)

Blocking and non-blocking TCP sockets in Java

Traditional Multi threaded socket

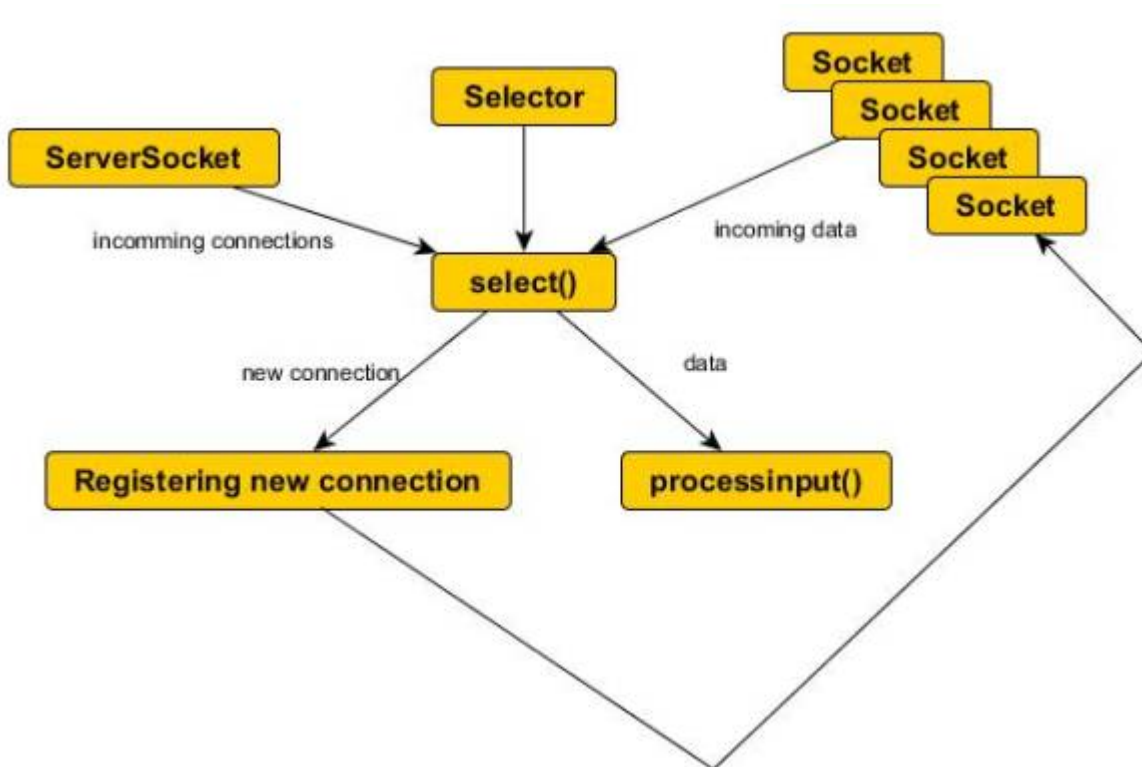


Non blocking 1 thread socket



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();
```

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

Java examples

[Java example for Blocking and Non-Blocking Socket](#)

[HTTP server](#)

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=1683572022

Last update: **2023/05/08 18:53**

