Tristan BRAU
Léo PHAV

Java pour les réseaux - TP 4

2 - Architecture Transformation

2.1 - Single-threaded Limitations Analysis

- Where does serverSocket.accept() block?

In the code, the method blocks at line 25: *socket = serverSocket.accept();*
When the program execution reaches this line, it pauses entirely. It hands control over to the Operating System and waits until a client initiates a TCP connection (via the 3-way handshake). Until a connection is established, no code below this line is executed.

- How does your handleClient() method prevent new connections?

In our specific code, there is no separate function named *handleClient()*, but the logic inside the *try* block (Lines 24-44) acts as this handler.

This logic prevents new connections because it runs on the **Main Thread**:

1. The code enters the inner *while(true)* loop (Lines 32-43) to talk to the connected client.
2. As long as the code is looping here, the program cannot return to the top of the outer loop (Line 22).
3. Since it cannot return to Line 23 (*serverSocket.accept()*), it is physically impossible for the server to accept a second connection.

- What happens when multiple clients try to connect simultaneously?

If "Client A" is currently chatting with the server (inside the inner loop) and "Client B" tries to connect:

1. The Operating System (kernel) creates the TCP connection for Client B and puts it into a "backlog" queue.
2. The Java program is busy with Client A and does not know Client B is waiting.
3. The server will only accept Client B after Client A sends "OK BYE" and disconnects.

- How is CPU utilization during I/O waits?

During I/O waits, the CPU utilization is near 0% for this process because *readLine()* is a blocking I/O operation. When the code reaches this line, the thread enters a *WAITING* or *BLOCKED* state. Therefore, the OS will give CPU time to other processes on the computer until data arrives.

Tristan BRAU
Léo PHAV

2.2 - Multithreading Strategy Design

4 - Step-by-Step Exercises

```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP4/MultithreadedTCPServer
Multithreaded Server started on port 8080
Active threads: 1
[Wed Nov 26 15:52:10 CET 2025] Client 1 connected
[Wed Nov 26 15:52:51 CET 2025] Client 1 sent: Bonjour
[Wed Nov 26 15:52:56 CET 2025] Client 1 sent: 12334
[Wed Nov 26 15:53:00 CET 2025] Client 1 sent: quit
[Wed Nov 26 15:53:00 CET 2025] Client 1 requested disconnect
[15:53:00] [Client 1] Connection closed.
```

```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP4/TCP_Client localhost 8080
Server : Welcome! You are client #1
You: Bonjour
Server : MSG Received: Bonjour
You: 12334
Server : MSG Received: 12334
You: quit
Server : MSG Received: quit
```

4.1.3 - Success Verification

- Single client connects and communicates normally

The screenshots show a complete, successful session. The client connected, exchanged multiple messages ("Bonjour", "12334"), and disconnected without the program crashing or freezing.

- Server shows connection message with correct client ID

In the top terminal (Server), the log clearly shows: *[Wed Nov 26 ...] Client 1 connected*
This confirms the server correctly assigned ID #1 to the incoming connection.

- Client receives welcome message with assigned ID

In the bottom terminal (Client), the first message received from the server is:
Server : Welcome! You are client #1
This matches the requirement to notify the client of their ID upon connection.

- Echo functionality works as expected

The conversation history in the client terminal proves the echo logic:

➢ Input: *You: Bonjour* → Response: *Server : MSG Received: Bonjour*
➢ Input: *You: 12334* → Response: *Server : MSG Received: 12334*

- "quit" command disconnects client cleanly

After typing quit on the client side, the server acknowledged it (*MSG Received: quit*) and the client program terminated.
The server detected the command and logged:
Client 1 requested disconnect [...][Client 1] Connection closed.
This confirms the disconnection was handled gracefully on both ends.

4.2 - Exercise 2: Concurrent Client Testing

```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP4/MultithreadedTCPServer
Multithreaded Server started on port 8080
Active threads: 1
[Wed Nov 26 15:54:38 CET 2025] Client 1 connected
Active threads: 2
[Wed Nov 26 15:55:20 CET 2025] Client 2 connected
[Wed Nov 26 15:55:26 CET 2025] Client 2 sent: 3232
[Wed Nov 26 15:55:32 CET 2025] Client 1 sent: Bonjour
[Wed Nov 26 15:56:29 CET 2025] Client 2 sent: quit
[Wed Nov 26 15:56:29 CET 2025] Client 2 requested disconnect
[15:56:29] [Client 2] Connection closed.
[Wed Nov 26 15:56:35 CET 2025] Client 1 sent: mmer
```

```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP4/TCP_Client localhost 8080
Server : Welcome! You are client #1
You: Bonjour
Server : MSG Received: Bonjour
You: mmer
Server : MSG Received: mmer
You:
```

```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP4/TCP_Client localhost 8080
Server : Welcome! You are client #2
You: 3232
Server : MSG Received: 3232
You: quit
Server : MSG Received: quit
```

4.2.2 - Expected Results

1. Simultaneous Connections & Sequential IDs

In the top terminal, we see *Active threads: 1* for Client 1, followed by *Active threads: 2* when Client 2 connects. The IDs are correctly assigned (#1 and #2).

2. Non-Blocking Communication

- 15:55:26: Client 2 sends "*3232*".
- 15:55:32: Client 1 sends "*Bonjour*".

Client 1 was able to send a message **while** Client 2 was also connected. They did not have to wait for each other.

3. Data Isolation

- Client 1 only sees "*Bonjour*" and "*mmer*" but does not see "*3232*".
- Client 2 only sees "*3232*".

This confirms that each thread is successfully keeping the data separate.

4. Interleaved Server Logs

The top terminal shows a perfect mix of events: *Client 2 connected* → *Client 2 sent* → *Client 1 sent* → *Client 2 quit*.

4.3 - Exercise 3: Thread Safety Validation

4.3.2 - Verification Criteria

When we launch 10 processes in the background of a single terminal window, they are all forced to share the exact same *Standard Input* (stdin) stream and when typing a character, the OS does not know which of the 10 waiting *Scanner* objects should receive that character.
To fix this, we tried to remove the *Scanner* function and coded the messages in scripts.
For the following questions, we came back to the script of the previous part.

Tristan BRAU
Léo PHAV

```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP4/MultithreadedTCPServer
Multithreaded Server started on port 8080
Active threads: 1
Active threads: 2
Active threads: 3
Active threads: 4
Active threads: 5
Active threads: 6
Active threads: 7
Active threads: 8
Active threads: 9
Active threads: 10
[Wed Nov 26 16:27:33 CET 2025] Client 2 connected
[Wed Nov 26 16:27:33 CET 2025] Client 9 connected
[Wed Nov 26 16:27:33 CET 2025] Client 10 connected
[Wed Nov 26 16:27:33 CET 2025] Client 4 connected
[Wed Nov 26 16:27:33 CET 2025] Client 3 connected
[Wed Nov 26 16:27:33 CET 2025] Client 5 connected
[Wed Nov 26 16:27:33 CET 2025] Client 6 connected
[Wed Nov 26 16:27:33 CET 2025] Client 7 connected
[Wed Nov 26 16:27:33 CET 2025] Client 1 connected
[Wed Nov 26 16:27:33 CET 2025] Client 8 connected
[Wed Nov 26 16:27:33 CET 2025] Client 2 sent: Hello
[Wed Nov 26 16:27:33 CET 2025] Client 1 sent: Hello
[Wed Nov 26 16:27:33 CET 2025] Client 9 sent: Hello
[Wed Nov 26 16:27:33 CET 2025] Client 4 sent: Hello
[Wed Nov 26 16:27:33 CET 2025] Client 5 sent: Hello
[Wed Nov 26 16:27:33 CET 2025] Client 6 sent: Hello
[Wed Nov 26 16:27:33 CET 2025] Client 3 sent: Hello
[Wed Nov 26 16:27:33 CET 2025] Client 7 sent: Hello
[Wed Nov 26 16:27:33 CET 2025] Client 8 sent: Hello
[Wed Nov 26 16:27:33 CET 2025] Client 10 sent: Hello
[16:27:33] [Client 7] Connection closed.
[16:27:33] [Client 1] Connection closed.
[16:27:33] [Client 5] Connection closed.
[16:27:33] [Client 2] Connection closed.
[16:27:33] [Client 10] Connection closed.
[16:27:33] [Client 8] Connection closed.
[16:27:33] [Client 9] Connection closed.
[16:27:33] [Client 3] Connection closed.
[16:27:33] [Client 4] Connection closed.
[16:27:33] [Client 6] Connection closed.
```

Tristan BRAU
Léo PHAV

```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ for i in {1..10}; do
java TP4/TCP_Client localhost 8080 &
done
wait
[5] 189042
[6] 189043
[7] 189044
[8] 189045
[9] 189046
[10] 189047
[11] 189048
[12] 189049
[13] 189050
[14] 189052
Server: Welcome! You are client #6
Server: Welcome! You are client #8
Server: Welcome! You are client #10
Server: Welcome! You are client #7
Server: Welcome! You are client #9
Server: Welcome! You are client #1
Server: Welcome! You are client #5
Server: Welcome! You are client #4
Server: Welcome! You are client #2
Server: Welcome! You are client #3
Server: MSG Received: Hello
Server: MSG Received: Hello
Server: MSG Received: Hello
Server: MSG Received: Hello
Server: MSG Received: Hello
Server: MSG Received: Hello
Server: MSG Received: Hello
Server: MSG Received: Hello
Server: MSG Received: Hello
Server: MSG Received: Hello
[6]    done       java TP4/TCP_Client localhost 8080
[7]    done       java TP4/TCP_Client localhost 8080
[13]    done       java TP4/TCP_Client localhost 8080
[14]    done       java TP4/TCP_Client localhost 8080
[8]    done       java TP4/TCP_Client localhost 8080
[11]    done       java TP4/TCP_Client localhost 8080
[10]    done       java TP4/TCP_Client localhost 8080
[12]    done       java TP4/TCP_Client localhost 8080
[5]    done       java TP4/TCP_Client localhost 8080
[9]    done       java TP4/TCP_Client localhost 8080
```