Socket Programming - Math Service

You are required to implement a math server and a math client. The client should allow the user to continuously issue requests until the user inputs “Quit”.

* A user will input some math expression, such as “2 + 3”, “2 \*3 ”, “4/5”, etc., in the client. The client will send the math expression to the server.
* The server will evaluate the result of the received math expression, and send back the result back to the requesting client. The result should be sent back as a string. For example, if the server receives “2 + 3”, it will send back “5”. If the server receives “2 \*3 ”, it will send back “6”.
* The server should always be running, waiting to serve more clients, before it is manually terminated.

On the server side, you need some evaluation function to evaluate a math expression. You can implement your own, or you can use some existing library functions/methods. In either case, your server should support valid math expressions that involve the four basic operators: “+”, “-”, “\*”, and “/”. The math expressions can have arbitrary number of spaces/tabs within them, and they should still be regarded as valid.

# 1. Requirements:

1. You are required to implement the **UDP** version. The UDP server should be able to serve multiple clients at the same time by default, without multi-threading.
2. You are required to implement the **TCP non-persistent** version. The TCP non-persistent server should be able to serve multiple clients at the same time by default, without multi-threading.
3. You are required to implement the **TCP persistent version without multi-threading**. The TCP persistent server should be able to serve multiple clients, not necessarily at the same time. That is, the server should be able to serve one client, and have several other clients pending while it is serving the first client. When the first client is done (quit gracefully), the server can pick up the next pending client and serve it; after this client is done, the server can again pick up the next pending client and serve it.
4. You are required to submit a **project report**. In your project report, you should include comprehensive tests to demonstrate that all your programs meet the requirements. For example, for the UDP version, your testing should show that your server really can serve multiple clients at the same time. You can include a screenshot with multiple clients concurrently running and getting the service. Also, show that when a client gracefully quits (by issuing “Quit”), other clients can still get the service without a problem, and new clients can come to get the service without a problem. The project report will have a large weight in terms of grading.
5. You are highly recommended to implement the **TCP persistent version with multi-threading**. The multi-threaded TCP persistent server can serve multiple clients at the same time.

# 2. Sample Pseudo Code for the Four Versions

## 2.1 UDP Version:

**UDPMathServer:**

Create a UDP socket

While (true) {

Receive a string from a client through the socket

Evaluate the result of the math expression in the string

Send back the result as a string to client using the socket

}

**UDPMathClient:**

Create a UDP socket

Get an input string from the standard input

While (the input string is not quit) {

Send the input string to the server using the socket

Receive the result from the server

Print the result as a string

Get another input string from the standard input

}

Close the socket

## 2.2 TCP Non-Persistent Version:

**TCPMathServerNonPersistent:**

Create a listening TCP socket

While (true) {

Wait for the connection from a client

Use the established TCP connection socket to receive a string from the client

Evaluate the result of the math expression in the string

Send back the result as a string to client using the connection socket

Close the TCP connection socket

}

**TCPMathClientNonPersistent:**

Get an input string from the standard input

While (the input string is not QUIT) {

Connect to the server

Use the newly created TCP connection socket to send the input string to the server

Receive the result from the server through the TCP connection socket

Print the result as a string

Close the TCP connection socket

Get another input string from the standard input

}

Remarks: The Server will use a new TCP connection for each new request. Thus, if the client has multiple requests, the client and the server need to use multiple TCP connections.

## 2.3 TCP Persistent Version without Multi-threading

**TCPMathServerPersistent:**

Create a listening TCP socket

While (true) {

Wait for the connection from a client

While (true) {

Use the established TCP connection socket to receive a string from the client

If (the received request is quit) {

Close the connection

Break

} else {

Evaluate the result of the math expression in the string

Send back the result as a string to client using the connection socket

}

}

}

**TCPMathClientPersistent:**

Connect to the server

While (true) {

Get an input string from the standard input

Send the string to the server

If (the input string is “QUIT”) {

Break

} Else {

Receive the result from the server

Print the result as a string

}

}

Close the connection socket

Remarks: The Server will use the same TCP connection for multiple requests from the same client. Thus, each client only needs to connect to the TCP server once. Since the server is single-threaded, when it is serving one client, it will be stuck in the inner while loop; thus, the server cannot serve any other client at the same time. The server can only serve multiple clients one by one.

## 2.4 TCP Persistent Version with Multi-threading

**TCPMathServerPersistentMultithread:**

Define a thread that will use a TCP connection socket to serve a client

Behavior of the thread: it will receive a math expression as a string from the client, calculate the result, and send the result back to the client; the thread should exit after it finishes serving all the requests of a client

Create a listening TCP socket

While (true) {

Wait for the connection from a client

Create a new thread that will use the newly created TCP connection socket to serve the client

Start the new thread.

}

**TCPMathClientPersistentMultithread**:

This can be the same as TCPMathClientPersistent in Section 2.3.

Remarks: The server is multi-threaded, and serving for a new client will always be done by a new thread; the server code can loop to the next iteration to serve a new client when existing threads are serving existing clients. Thus, the server can serve all clients at the same time.

# 3. Sample Code on Multi-threading

You can implement the project in any programming language you like. Examples on how to implement multi-threading can be found in the “socket programming projects” folder of Canvas Files:

Java: *MultiThreadDemo.java*, *MultiThreadDemo2.java, and MultiThreadDemo3.java*

Python: *multithread.py*, *multithread2.py, and multithread3.py*

# 4. Math Expression Evaluation

If you are using Python, you can find built-in functions to evaluate a math expression.

If you are using Java, you should implement your own evaluation function. Your evaluation function should be able to accept a math expression that consists of two positive integer numbers and one operator, such as +, -, \*, and /, with arbitrary an arbitrary number of spaces/tabs. For example, “ 2 \* 3” should be a valid expression; “245 / 3 ” should also be a valid expression.

# 5. Submission:

7 files are required:

1. UDPMathServer.java/UDPMathServer.py
2. UDPMathClient.java/UDPMathClient.py
3. TCPMathServerNonPersistent.java/TCPMathServerNonPersistent.py
4. TCPMathClientNonPersistent.java/TCPMathClientNonPersistent.py
5. TCPMathServerPersistent.java/TCPMathServerPersistent.py
6. TCPMathClientPersistent.java/TCPMathClientPersistent.py
7. Project Report

If you implement the TCP persistent version with multi-threading, you should submit one additional file:

* TCPMathServerPersistentMultithreading.java/ TCPMathServerPersistentMultithreading.py

The code in TCPMathClientPersistent should be able to work with the multi-threaded server without any modification. You should also include tests for the multi-threaded server in your project report, if you implement it.

**Related Resources:**

Python 2 multithreading:

<http://www.techbeamers.com/python-multithreading-concepts/>

<https://www.tutorialspoint.com/python/python_multithreading.htm>

Python 3 multithreading:

<https://www.tutorialspoint.com/python3/python_multithreading.htm>

Java multithreading:

<https://www.tutorialspoint.com/java/java_multithreading.htm>

<https://www.geeksforgeeks.org/multithreading-in-java/>

C multithreading:

<https://www.geeksforgeeks.org/multithreading-c-2/>