Thread Performance Project Report

Tanvir Irfan Chowdhury
Department of Computer Science
The University of Texas at San Antonio
tanvirirfan.chowdhury@utsa.edu

November 4, 2014

Abstract

This article is a technical report for the Thread Performance Project in CS-5523 : Operating System. In this article different aspect of the project is discussed.

1 Project Status

The project status is **"Complete"**.

All the requirement for the project is implemented in the code that is submitted. There is a Server class and there is also a CLient class. The code is implemented in such a way that it support one Server and multiple clients which can request services from the server. When the server is started it start listening in a specific port, say, port number 10000. When a client request to connect, server accept the connection and start reading the requested service string (in a way the server and client agrees) and then deliver the service string to one of the Thread in the ThreadPool which were started at the beginning when the server started. Then that thread parse the request string and handed over to another Thread for processing the request. After processing the request the thread reply to the client using the socket.

Here is the instruction how to run the program:

Run the "Server":

You need to run the server first. Otherwise client will print "'Server Off-line!' message. And please run the server only once. Otherwise, you will have bind exceptions.

```
cd␣YOUR_LOATION\Project\SocketProgramming\bin
java␣edu.tanvirirfan.chowdhury.server.MagicMathServer
␣
```

Run the "Client"::

You can run multiple clients to check. All the clients will send exactly 1000 requests.

```
cd␣YOUR_LOATION\Project\SocketProgramming\bin
java␣edu.tanvirirfan.chowdhury.client.MagicMathClient␣"localhost"␣10000
␣
```

2 Protocols Between Server and Clients

In this project, I used a simple protocol for the communication between server and client. As I implemented the project using programming language Java, I can create the whole request as a string and then send it to the server using the OutputStream. For the three arithmetic operations- ***magicAdd***, ***magicSub*** and ***magicMul*** the request string is same. Thus there are two possible type of request strings-

- OPERATION TYPE ***HASH*** FIRST DATA ***HASH*** SECOND DATA ***HASH*** ITERATION
- OPERATION TYPE ***HASH*** NUMBER OF INTEGER ***HASH*** ALL INTEGER SEPERATED BY SPACE

3 Synchronization

In this project synchronization was an essential part for successful implementation. In Java, Synchronization can be implemented into three level-

- Class level synchronization
- Method level synchronization
- Block level synchronization

In my project, I used all the three types of synchronization. I printed the result in the Client and when doing so, I used class level synchronization. I used a Queue for all the requests and the enQueue and deQueue functions of that queue used the method level synchronization. And finally when the requests were being processed, block level synchronization was used.

4 Challenges

Here are some of the challenges I faced while doing the implementation of the this project.

DeadLock because of "Socket Reset" by different thread:

Challenge:
Specification says that I need to use 5 THREADs in each CLIENT. What I was doing, I was using only one socket for the all Threads. This force my application to have DEAD LOCK. Here is why- I was starting all the THREADs in the constructor of the CLIENT. After they got the CPU slot, they request to the SERVER for one specific service that request was done successfully. As I was creating one socket before every connection, all the Thread could request their first request. But then the were waiting for the response from the SERVER. In the mean time as the socket is created again, server could not send response to the proper client. Which makes the "DEAD LOCK" situation. This problem wasted a HUGE AMOUNT of my time.

Solution:
I had to debug the program for a long time. It was a tedious work.

Request was discarded by Server:

Challenge:
I was doing the parsing of the request in the main thread of Server, which take a lot of time in case of "magicSort" operation and because of this lots of other operations were discarded. Later, I use dedicated thread for parsing the input. They are also from the THREAD POOL (NOT a new thread for each request)

Solution:
Used dedicated thread from the Thread-Pool to parse the inputs. Now main thread of server can focus on the accepting the requests.

Overflow in Input:

Challenge:
In magicAdd, magicSub, magicMul operations, range of the operand is from Float.MINVALUE to Float.MAXVALUE they can lead to Overflow.

Solution:
Used simple mathematics to figure out that range was not working. I made my program based on the constants which can be defined and changed any time because they are located in one separate file.

5 Comments and Suggestions

I liked this project very much as it gives me a chance to engage myself in socket programming after a long time. Mostly, the project was fun to work in. I tried to implement the project much more in a "Industry standard" codding by handling many of the usual exceptions. But if there were more points on this part I could have improve my project more. One thing is that the time given for this project was more than enough. Besides doing the usual work of implementing the project, I also used version control system named "Perforce" to keep track of my work. As there were more time given, I was able to experiment on the version control system.

Acknoledgements

I searched in google for the idea of Thread-Pool and looked up some example code and used one Queue class. It was because I was lazy to write code for myself. It was a simple class with only two functions enQueue and deQueue.