Multi-Threaded Chat Service

Hugo David Franco Ávila A01654856

Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Querétaro

Popoloca Mz 1 Lt 47, Col. Carlos Zapata Vela, Iztacalco, CDMX, 08040

Abstract

The project is a Java Application that uses threads to connect multiple clients with a single server. The clients connect via sockets, and each message is processed by a thread and sent to everyone connected. One thread is used for the connection and another to send and receive messages. Concurrency was used because you need more than one thread to be able to connect and do messaging for one user, let alone any number of them. Because of how Java provides a wealth of classes it was the language selected.

Keywords: Java, Concurrency, Threads, Sockets, Network, Swing, Process

Multi-Threaded Chat Service

Concurrency and parallelism are no longer a novelty, it is a necessity for the modern developer. Computers are able to use multiple CPU cores to execute tasks, as opposed to what was the norm just 15 years ago[1], when most of the processors where single-core/single-threaded. This has made imperative for software developers to use tools that allow them to take advantage of this hardware; and the reason why concurrency and parallelism where added to a wealth of programming languages[2].

Process

For us to truly understand how concurrency works we must provide a definition for a process. A process is an instance of a program running on a computer[3]. It can be a command, a script or any binary executable. It has a some properties, the most important of which is the Process-ID (PID), with this the CPU is able to identify exactly in which state it was the last time the process was executed and what where the values of its variables.

Process vs Thread

Threads and processes have been a topic of discussion in the past, with many people not knowing that they are different entities. A thread is the part of the process that executes, this means a program can have many threads but it is only one process. This is important when making assumptions and claims about concurrency and parallelism, because you are normally not doing different things at ones, you split a large process into several threads which are then able to complete faster by having resources allocated to them.

Concurrency

Concurrency means that multiple computations are running at the same time[4]. It is present in a plethora of applications: from a network with multiple computers, multiple applications running on a single machine, a web site that must handle multiple users, etc. One of the biggest drivers of Concurrency, is that recent CPU releases often make an emphasis on a higher core count instead of faster clock speeds, such is the case of AMD's Ryzen [5] line of processors.

Models of Concurrency

- Shared memory. Concurrent modules interact by reading and writing shared object in memory.
- Message passing. In this model, the modules interact by sending messages to each other through a communication channel.

Concurrency vs Parallelism

Concurrency and parallelism are quite similar, the main difference being that even though context switches almost instantaneously, threads running concurrently do not run at "exactly the same time", there is a brief pause between the execution of one and another; whereas in parallelism two or more cores are processing data at the same time, something that can be inferenced from the figure below (figure 1).

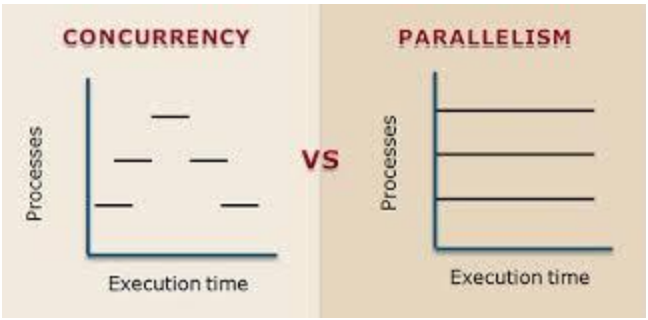

Figure 1. Concurrency and parallelism side by side

Sockets

Sockets are a mechanism of inter-process communication (IPC). They allow two or more processes to share data. In the context of the Chat Service application they allow a channel to be formed between the host and the clients, through which messages are sent. There are four types of Sockets: stream, datagram, raw and sequenced packet. The ones used in the app are stream because you need to have a connection and delivery of the packets is guaranteed through use of the Transmission Control Protocol[6].

Description of the problem

In this project my objective is to build an application that uses threads to handle multiple user requests, and that a user never waits for other users to be able to finish its task.

Solution

The first thing I had to decide on was which language I was going to use, I needed something that had great built-in support for multi-threading, which is why I settled on Java to do it. Then I needed to define the classes that I was implementing, and the following Class Diagram was built (figure 2). How a Server and a ClientThread interact is explained well by the diagram, but the interaction between the ServerThread and the Client is a bit more complex. First it is important to understand how are the processes communicating, they are not your regular Java classes that one class instantiates another and that's it, you have the socket component through which most of the communication is made.

With this in mind, we need to ask ourselves the question, how is the client able to read the messages that are sent to him if he is already busy enough sending messages himself? The answer is the ServerThread class. What this class does, is that messages get queued up in a Linked List, and it is constantly synchronizing itself with the server so no messages are lost, after the complete

list of messages is received it prints them out on the console so the user can see what other people in the chat are saying.

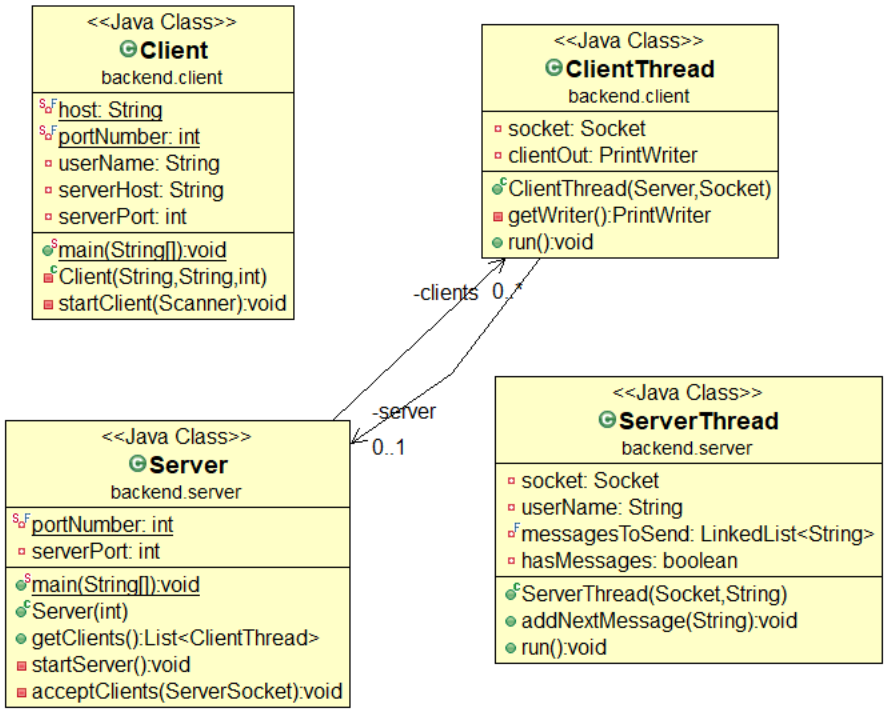

Figure 2. Class diagram of the solution, with only backend classes shown

When running the program in the Eclipse console I get the following output:

```
server starts port = 0.0.0.0/0.0.0.0:4444
accepts : /127.0.0.1:62183
accepts : /127.0.0.1:62190
```

Figure 3. Server output after running the project

```
Please input username:
Hugo
Welcome :Hugo
Local Port :62183
Server = localhost/127.0.0.1:4444:4444
```

Figure 4. Client output after running the project

```
Please input username:
Benji
Welcome :Benji
Local Port :62190
Server = localhost/127.0.0.1:4444:4444
```

Figure 5. Client output after running the project

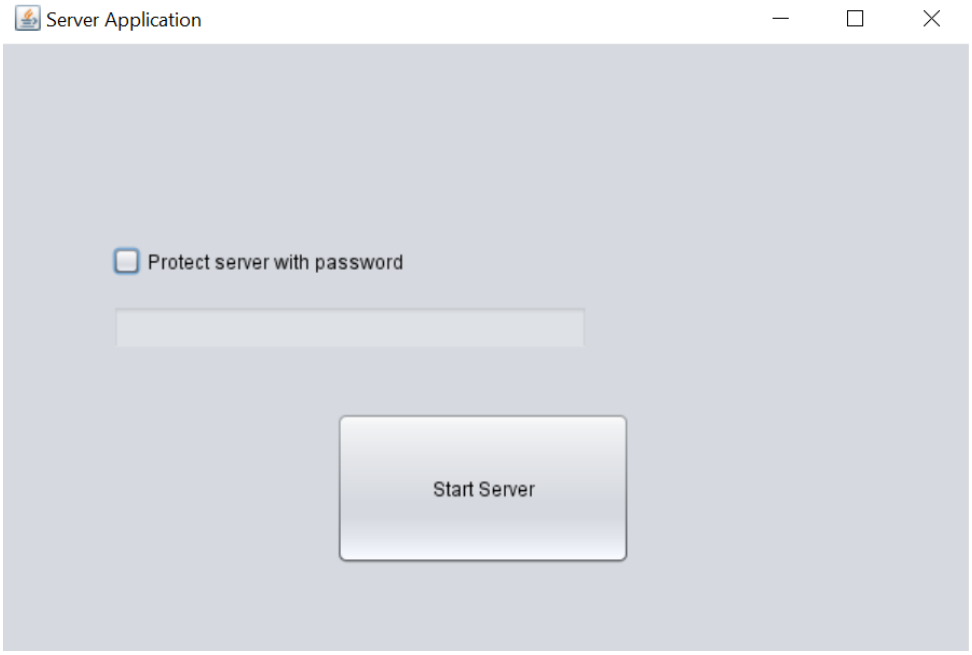

Figure 6. Server GUI

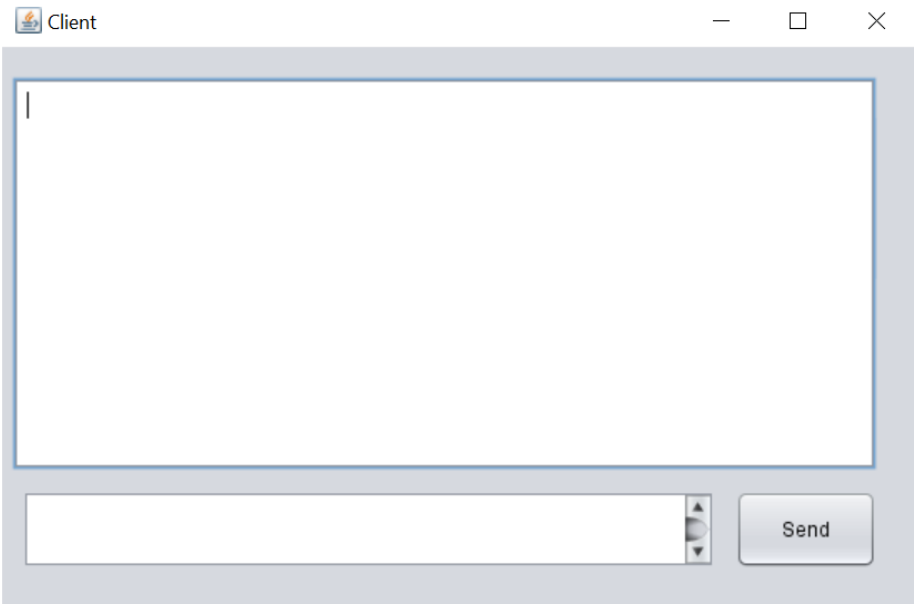

Figure 7. Client GUI

Project Setup

The steps to download and run the project are as follow:

1. Download Java 8 SDK from https://www.oracle.com/mx/java/technologies/javase/javase-jdk8-downloads.html
2. Download NetBeans (recommended) or any other IDE like Eclipse and JetBrains
3. Clone the repository from https://github.com/hugoFranco21/ChatService.git, to do this git needs to be installed in your computer (Check it out at: https://git-scm.com/downloads) After git is installed, run this command in CMD or Powershell (or your OS terminal): git clone https://github.com/hugoFranco21/ChatService.git,
4. After the project is cloned, open it in your IDE of choice, right click the ServerGUI class and click “Run as Java Application”
5. Do the same for the number of clients you want to create
6. Done

License

The project was released under the GNU General Public License v3.0, this means anyone can copy it, distribute it, and modify it at will, however I am not liable for any harm or wrongdoing that could be done with the software, and I also do not offer a warranty for it.

Conclusion

The need for applications that can fully utilize the power of modern CPU's is rising, and it is up to us the developers to fully embrace the challenge. Concurrency is more widespread in day to day applications than many people realize; with the growing complexity of applications new and innovative ways of developing are going to be used, and parallelizing the tasks performed on an application means extra performance while spending the same on hardware.

Even though there are multiple languages and frameworks supporting concurrency, it is still necessary to keep in mind unexpected results can happen if not everything is synchronized, and that a bad implementation can cripple performance on even the most advanced synchronized data structure.

While performance increases are seen across the board, it is important to remember single-threaded workloads are still important and probably the most common type of processes still, some tasks offer diminishing returns when trying to parallelize them, so it is important to test the application first to see if there is any tangible improvement, that can save the developer a lot of time that could be better used elsewhere.

References

[1]Editor. (May 23, 2018). *Intel Processors Over The Years.* Busines News Daily. Retrieved from https://www.businessnewsdaily.com/10817-slideshow-intel-processors-over-the-years.html#:~:text=company's%20product%20history.-,2005%3A%20Pentium%20D,the%20Pentium%20D%20800%20series.

[2]Ghezi, C. (August 14, 2003). *Concurrency in Programming Languages: A Survey.* Science Direct. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/0167819185900055

[3]Rouse, M. (September 2005). *Process Definition.* Tech Target. Retrieved from https://whatis.techtarget.com/definition/process

[4]Academia. (2014). *Concurrency.* Massachusetts Institute of Technology. Retrieved from https://web.mit.edu/6.005/www/fa14/classes/17-concurrency/

[5]Castle, C. (2020). *AMD Ryzen 5800x Review.* Rock, paper, shotgun. Retrieved from https://www.rockpapershotgun.com/2020/11/09/amd-ryzen-7-5800x-review/

[6]Anonymous. (n.d.). *What is a socket?* Tutorials Point. Retrieved from https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm