Concurrent Socket Server Assignment

Sara Almohamed, Jess Jackson

Class: CNT4504 - Computer Networks

Professor's Name: Professor John Kelly

**Introduction:**

The purpose of the Concurrent Socket Server project was to implement a concurrent multi-threaded server and a multi-threaded client in Java to examine the effects of concurrency on the efficiency of processing client requests. By creating these programs, we aimed to understand the benefits and trade-offs of using concurrent servers compared to iterative servers in a client-server configuration.

In this report, we will present the goals of the project, the design of the client and server programs, the testing and data collection procedures, data analysis results, and the conclusions drawn from the experiment. The report will also include lessons learned throughout the development process.

**Client-Server Setup and Configuration:**

The design of the client and server programs was based on the Java Socket programming paradigm. The server program utilized Java's server socket to listen for incoming client connections. Upon receiving a client request, the server made a new thread to handle the request concurrently, enabling multiple client requests to be processed simultaneously.

The client program was also multi-threaded, capable of creating numerous client sessions. It prompted the user for the network address and port of the server and allowed the user to select one of the six available operations to request. Additionally, the client could specify the number of client requests to generate for each operation.

The design decisions made involved choosing a multi-threaded approach for both the client and server, as this would allow efficient handling of multiple client requests concurrently, thereby reducing the turn-around time for individual clients.

**Testing and Data Collection:**

The concurrent server was tested using the provided course server, where the client and server programs were connected to the same network address and port. The client was used to generate multiple client requests, and the turn-around time for each request was measured.

Data collected for each operation included the time required for the client request to travel to the server, be processed by the server, and return to the client. For accurate analysis, each operation was tested multiple times with different numbers of client requests, ranging from 1 to 100.

The collected data was organized into charts and graphs for each operation, illustrating the turn-around time for individual clients and the average turn-around time for each number of client requests.

A screenshot of a graph

Description automatically generated

**Data Analysis:**

Based on the data collected, the following questions were analyzed:

1. Effect of increasing the number of clients on Turn-around Time for individual clients:

As the number of clients increases in the concurrent server, the Turn-around Time for individual clients also rises, though the impact remains relatively small. The server handles multiple requests simultaneously, leading to a slight increase in processing time for each client. However, the concurrent approach still demonstrates efficiency in handling multiple clients concurrently.

2. Effect of increasing the number of clients on Average Turn-around Time:

With more clients, the Average Turn-around Time gradually increases in the concurrent server. This is due to longer waiting times as clients compete for server resources. Despite the increase, the concurrent server maintains its efficiency in handling multiple clients effectively.

3. Comparison with the iterative server:

The concurrent server shows significantly lower average turn-around time compared to the iterative server, especially with a larger number of clients. The concurrent server's ability to handle multiple client requests concurrently leads to better performance.

4. Use of an iterative server:

An iterative server is suitable for low client loads and scenarios where concurrency is not a major concern. It can be simpler to implement and manage for smaller-scale applications.

5. Use of a concurrent server:

A concurrent server is preferable when handling a large number of clients simultaneously is essential for optimal performance. It efficiently utilizes system resources and minimizes waiting times, making it ideal for high client concurrency scenarios.

**Conclusion:**

Upon analyzing the data, it becomes evident that the concurrent server surpasses the iterative server significantly in terms of average turn-around time, especially when dealing with a substantial number of clients. Thanks to its powerful thread management, the concurrent server efficiently handles multiple client requests concurrently, resulting in noteworthy reductions in waiting times and average turn-around time. Conversely, the iterative server's sequential approach hampers its performance as the number of clients increases, leading to linearly increasing turn-around times.

However, the advantages of the concurrent server extend beyond average turn-around time reduction. Its inherent ability to enhance system responsiveness and scalability proves particularly advantageous in scenarios with unpredictable or high client concurrency. The concurrent server remains a good choice when managing a large number of clients concurrently, leveraging its multitasking capabilities to ensure efficient processing.

In conclusion, the data analysis supports the preference for a concurrent server over an iterative one. Its prowess in handling multiple client requests concurrently makes it an ideal solution for optimizing client-server systems' efficiency and responsiveness. By adopting a concurrent server, organizations can significantly enhance their system performance, catering to diverse client demands with ease and efficiency.

**Lessons Learned:**

Throughout the concurrent server assignment, we had the opportunity to explore various aspects of concurrent programming and client-server configurations. The project provided invaluable hands-on experience, leading to several key takeaways.

1. Implementing the Client and Server programs:

One of the most significant takeaways was gaining proficiency in developing multi-threaded client and server programs. Understanding how to manage threads effectively enabled us to handle multiple client connections concurrently, enhancing the overall performance and responsiveness of the server. This experience bettered our knowledge of concurrent programming concepts and techniques.

2. Executing Linux commands:

Another essential learning point was the integration of Linux commands within our Java program. By executing system commands, we were able to retrieve valuable information from the server, facilitating data collection for analysis. This skill proved to be particularly valuable in real-world scenarios where accessing system information is crucial for monitoring server health and performance.

3. Data collection:

We realized the importance of accurate data collection throughout the project. Properly collecting and organizing data allowed us to conduct meaningful analysis and draw valid conclusions. The emphasis on precise data collection emphasized the significance of reliable evidence in evaluating the performance of different server configurations.

4. Problem-solving:

During the development phase, we encountered various challenges related to thread synchronization and concurrent programming. By troubleshooting and resolving these issues, we practiced our problem-solving skills and gained a deeper understanding of concurrent systems. This hands-on experience proved instrumental in overcoming complexities and optimizing the performance of the concurrent server.

In conclusion, the concurrent server assignment offered practical insights into the world of concurrent servers and their impact on client request processing efficiency. Through the development of multi-threaded client and server programs, we acquired valuable skills in managing concurrent connections and ensuring efficient resource utilization. The integration of Linux commands allowed us to gather critical server information, enhancing our ability to monitor and analyze system performance. Overall, the lessons from this project will be great in our future pursuits within the field of computer networks. The hands-on experience, combined with a deeper understanding of concurrent systems, positions us to approach future projects with confidence and effectiveness.