Hellowold-ice deployment Documentation

Team Members

- Juan Jose López López
- Sara Cardona Vásquez

Task Distribution

Juan Jose:

- Implemented client-side methods like `registerHost`.
- Constructed the overall project structure.
- Made minor adjustments to the quality tests.

Sara:

- Developed server-side methods such as `mTox`, `mBC`.
- Implemented the ThreadPool.

General Description

The project, HelloWorld-Ice, is a client-server communication system based on Ice. The main goal of this system is to enable the client to interact with the server via a command-line interface. Additionally, the project incorporates quality attribute tests for the communication system, assessing aspects such as throughput, response time, missing event rate, and unprocessed event rate.

Usage

1. **Initial Configuration:** SDK 11 Java, Middleware Ice tools downloaded, Gradle 6.6.
2. **Middleware:** ICE 3.7.9.
3. **Compilation:** Start the server first, followed by the client.

Class Descriptions

Class `Client`

The `Client` class provides functionalities for interaction with the server using Ice. Through an interactive menu, users can send messages, test throughput and response time, and evaluate the missing event and unprocessed event rates.

Class `Server`

Within the `Server`, the `PrinterI` class serves as an implementation of the `Demo.Printer` interface. It is responsible for processing messages received from the client. The class also includes operations for calculating prime factors, listing ports and network interfaces, and executing specific commands.

`Callback.ice`

This class implements the `Demo` interface, processing messages sent from the client. Functions include calculating the prime factors of numbers, listing ports and network interfaces, and executing commands.

Enhancements and Features

ThreadPool

The inclusion of a ThreadPool significantly boosts the project's performance. This design choice was rooted in the aim to enhance server scalability. A ThreadPool manages a pool of worker threads. These threads can efficiently execute tasks in parallel, allowing the system to handle a larger volume of requests simultaneously without compromising on performance.

Asynchronous Functions (AMD)

To enhance the server's capacity to process multiple concurrent requests while maintaining optimal response times, we integrated the Asynchronous Method Dispatch (AMD) mechanism.

Distinction between AMD and Callback:

While both the AMD and Callback are asynchronous techniques, they differ in their core functionalities:

- **AMD (Asynchronous Method Dispatch):** This technique allows the server to process client requests without blocking the main thread. Once a request is received, it is dispatched to another thread for processing, enabling the server to handle other incoming requests.
- **Callback:** Callbacks are functions passed as arguments to other functions, intended to be executed after the main function completes. They can be used to notify when an asynchronous operation has been finished.

Correction on Measurements

To ensure accurate measurement and avoid potential inaccuracies, the team refined the measurement techniques by...

Timeout Test

During our testing phase, an experiment was conducted to induce a timeout. We executed 20 clients, each sending an exceedingly large number to the prime decomposition method. Interestingly, the server did not timeout. Evidence of this observation is documented in the `client.log` file.

References

- "Asynchronous Method Dispatch (AMD) in Java." ZeroC. Link.