RMI-LE-PETERSON

semestrální project

Roman Stepa

2023

**Starting program**

Runs on Java 17.

**Fisrt step**: to launch use command: <java Node.java>. It will automatically start node with public ip eth0. Will use local ip otherwise.

**Second step**: type <help> command for introduction.

**Third step:** type <join <remote node ip> <remote port>> between nodes. When you will join all nodes then type <info> on each node.

**Example:**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

**Fourth step:** type <bye>(orderly destroys node) or <leave>(only leaving from network) on leader node then use command <info> on each working node to check new leader .

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

**Fifth step:** type <write <String>> on any node to store shared variable. Then type <read > on any node to show shared variable.

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

**Introduction**

This is a simple implementation of a distributed system using Java RMI. The system is a ring of nodes that can communicate with each other. The nodes can join and leave the ring. The nodes can also elect a leader using Peterson's algorithm. The leader is responsible for a shared variable that can be read and written by the nodes. The system is implemented in a single Java file that can be compiled into jar file. The jar file can be run on multiple machines.

**Java RMI**

Java RMI (Remote Method Invocation) is a Java API that performs remote method invocation, the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java classes and distributed garbage-collection. The RMI allows an object running in one Java virtual machine (VM) to invoke methods on an object running in another Java VM. This project uses RMI to implement a distributed system. The nodes are implemented as Java objects that can be accessed remotely. The nodes can communicate with each other by invoking methods on each other. The nodes can also export themselves as remote objects that can be accessed by other nodes.

**Leader Election**

Leader election is the process of designating a single process as the organizer of some task distributed among several computers (nodes). Before the task is begun, all network nodes are unaware which node will serve as the "leader," or coordinator, of the task. After a leader election algorithm has been run, however, each node throughout the network recognizes a particular, unique node as the task leader. This project uses Peterson's algorithm to elect a leader. Peterson's algorithm is a leader election algorithm for a ring topology. The algorithm is based on the idea of a token that is passed around the ring. The token is passed around the ring until it reaches the node with the highest priority. The node with the highest priority becomes the leader. The algorithm is implemented as a method that is invoked by the nodes. The method returns the leader node.

**Peterson**

Peterson's algorithm is a leader election algorithm for a ring topology. The algorithm is based on the idea of a token that is passed around the ring. The token is passed around the ring until it reaches the node with the highest priority. The node with the highest priority becomes the leader.

**Shared Variable**

A shared variable is a variable that can be accessed by multiple processes. The shared variable is implemented as a Java object that can be accessed remotely. The shared variable can be read and written by the nodes. This algorithm uses leader election to determine which node is responsible for the shared variable. Other nodes can read and write the shared variable by invoking methods on the leader node.

**Implementation**

This implementation is based on separating layers of functionality into defirent chapters. Each chapter is a section in the orginal file. The chapters are

**Node**

This chapter implements the node functionality. The node is implemented as a Java object that can be accessed remotely. The node can communicate with other nodes by invoking methods on them. The node can also export itself as a remote object that can be accessed by other nodes.

**Attributes:**

1. **Registry –** RMI registry. Each node has its own.
2. **Ring –** list of all knows nodes in ring sorted in correct order with this node at the end.
3. **Active –** Peterson algorithm state.
4. **Leader –** if not in election proccess stores current leader. During collections stores best known candidate.
5. **Queue –** message queue. Used only in Peterson algorithm.
6. **Shared –** used to store shared variable. Nodes ask leader for it.
7. **Logger –** stores file to log in.

**Methods:**

**1.Main -** Creates node on public ip if possible and starts REPL.

**2.Construct -** Constructs node and exports it.

**3.Connect -** Connects to remote node by host and port.

**4.Export -** Exports node. Used to create remote object of itself.

**5.Endpoint -** Extracts endpoint string from remote object. It is unique identifier of remote object.

**6.Check -** Does nothing. Used to check if node is alive.

**Output**

This chapter describes the output functionality. The output is implemented as a set of methods that can be invoked by the nodes. The output methods can print messages in different colors.

**Methods:**

**1.Notify -** Prints message in green color.

**2.Error -** Prints message in red color.

**3.Print -** Prints message in default color.

**4.Logger -** Prints message in log file with timestamp.

**REPL**

This chapter describes the REPL functionality. The REPL is implemented as a method that is invoked by main. The REPL is a loop that reads commands from the user and executes them. The REPL can execute commands that are implemented as methods in the Node class.

**Commands:**

**1.Help -** Prints out available commands.

**2.Info -** Prints out information about current node

**3.Bye –** Leaves with destroying node

**4.Leave -** Leaves ring without destroying node

**5.Join -** Joins node to ring.

**6.Read -** Reads string data from shared variable

**7.Write -**Writes string data to shared variable

**Ring Stability**

This chapter describes the ring stability functionality.

The ring stability is managed by storing ordered ring with public interfaces of all known nodes. The ring is automatically fixed on node join and leave. Method for getting next node can skip dead nodes.

**Methods:**

**1.SetNext -** Ask remote node to add given node to ring. Ring for this node based on ring of remote node on success.

**2.DelNext -**Call remote node to delete given node from ring.

**3.CheckRing -** Check if ring is broken. If it is broken try to fix it.

**4.Next -** Get next node. If node is dead skip it.

**Elections**

Nodes uses queue for sending election messages. Nodes can be in two states: active and passive. Active node sends messages to next node. Passive node waits for messages. When node receives message it can become active or passive. If node is active it sends message to next node. If node is passive it sends message to next node.

**Methods:**

**1.GetLeader -**Remote getter for leader.

**2.Send -** Add message to remote message queue i.e send message to remote node.

**3.Election -**Starts election. It starts a thread for each node in ring. Each thread thread tarts peterson algorithm on different remote node. When one of the threads finishes election is finished.

**4.Peterson -** Peterson's algorithm is a leader election algorithm for a ring topology. The algorithm is based on the idea of a token that is passed around the ring. The token is passed around the ring until it reaches the node with the highest priority. The node with the highest priority becomes the leader.

**5.Better -** Compares three nodes and returns true if first node is better than other two.

**Shared Variable**

This chapter implements the shared variable functionality. The shared variable is implemented as a Java object that can be accessed remotely. The shared variable can be read and written by the nodes.

**Methods:**

**1.Load shared -**Remote getter for shared variable.

**2.Store shared -** Remote setter for shared variable.