UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ENGENHARIA DE COMPUTAÇÃO
SISTEMAS DISTRIBUIDOS - SDCO8A

EDUARDO YUJI YOSHIDA YAMADA

PROJETO FINAL DE SISTEMAS DISTRIBUIDOS

APUCARANA
2025

EDUARDO YUJI YOSHIDA YAMADA

PROJETO FINAL DE SISTEMAS DISTRIBUIDOS

Relatório Técnico do Trabalho Disciplinar apresentado como requisito parcial à obtenção de nota na disciplina de Sistemas Distribuídos do Curso Superior de Engenharia de Computação da Universidade Tecnológica Federal do Paraná

Orientador: Prof: Lucio Agostinho Rocha

APUCARANA
2025

SUMÁRIO

1 INTRODUÇÃO

Os sistemas distribuídos são amplamente utilizados para garantir disponibilidade, escalabilidade e eficiência em diversas aplicações modernas, como *e-commerce*, redes sociais e leilões online. No contexto deste trabalho, foi desenvolvido um sistema distribuído que permite a participação de usuários em um leilão remoto, utilizando tecnologias como Java RMI e controle de concorrência via *Two-Phase Locking* (2PL).

A comunicação entre os componentes do sistema é realizada por meio de Invocação de Métodos Remotos (RMI), um *middleware* que permite a interação entre processos distribuídos. Para garantir a consistência dos dados e evitar condições de corrida durante o acesso concorrente, o protocolo *Two-Phase Locking* (2PL) foi implementado. Além disso, um *Web Service* foi desenvolvido para expor a aplicação na web, possibilitando o acesso remoto ao leilão por meio de um servidor dedicado.

O presente trabalho detalha a implementação do sistema, abordando seus principais componentes, arquitetura, desafios enfrentados e soluções adotadas para garantir um funcionamento eficiente e seguro.

2 OBJETIVO

O objetivo deste projeto é desenvolver um sistema distribuído para gerenciar leilões online, permitindo que usuários participem e acompanhem o andamento dos lances de forma segura e eficiente. Para isso, serão utilizadas tecnologias como Java RMI para comunicação entre processos e *Two-Phase Locking* (2PL) para garantir a integridade dos dados. Além disso, um *Web Service* será implementado para permitir o acesso remoto à aplicação através de um servidor.

O sistema deve garantir que os participantes possam realizar lances de forma concorrente, evitando conflitos e garantindo a consistência dos dados. Outro objetivo é avaliar o desempenho e a eficácia das tecnologias aplicadas na implementação de um sistema distribuído, analisando possíveis melhorias e desafios enfrentados.

3 DESENVOLVIMENTO

O desenvolvimento deste projeto envolveu a implementação de um sistema distribuído de leilão online, utilizando Java RMI para comunicação entre cliente e servidor, *Two-Phase Locking* (2PL) para controle de concorrência e um *Web Service* para permitir acesso remoto.

Nesta seção, são detalhados os aspectos técnicos da implementação, abordando a estrutura do sistema, as principais tecnologias utilizadas e os desafios enfrentados durante o processo de desenvolvimento.

3.1 ARQUITETURA DO SISTEMA

O sistema de leilão distribuído segue uma arquitetura baseada em Invocação de Métodos Remotos (RMI), permitindo a comunicação entre um servidor e múltiplos clientes. Sua estrutura é composta pelos seguintes componentes:

- Servidor de Leilão: Gerencia os lances, armazena o maior lance e controla a concorrência.
- Cliente de Leilão: Interface interativa para os usuários enviarem lances e consultarem o status do leilão.
- Registro RMI: Mecanismo que permite que os clientes localizem e utilizem os métodos remotos do servidor.

O servidor implementa um mecanismo de controle de concorrência baseado em *Two-Phase Locking* (2PL), garantindo que múltiplos clientes possam interagir com segurança, evitando condições de corrida e inconsistências nos lances.

3.2 IMPLEMENTAÇÃO

3.2.1 SERVIDOR DO LEILÃO

A classe *AuctionServer* é responsável por gerenciar os lances e garantir a integridade dos dados. Sua implementação segue os seguintes passos:

1. O servidor inicia um Registro RMI e disponibiliza um serviço remoto acessível pelos clientes.
2. Os clientes enviam lances por meio do método remoto *placeBid*, que verifica se o valor do lance é maior que o atual e, se for, atualiza os dados.
3. Para garantir a consistência dos lances, foi implementado um mecanismo de bloqueio (*CustomLock*), baseado no *Two-Phase Locking* (2PL), impedindo que múltiplos clientes atualizem o maior lance simultaneamente.

4. O servidor também implementa o método *closeAuction*, permitindo encerrar o leilão e exibir o vencedor.

```
private String highestBidder; // Armazena o nome do maior
   lance
private double highestBid;    // Armazena o valor do maior
   lance
private boolean isClosed;     // Indica se o leilao esta
   encerrado
private final CustomLock lock = new CustomLock(); // Tranca
   personalizada
```

Listing 3.1 – Atributos do AuctionServer

```
public String placeBid(String user, double amount) throws
   RemoteException {
    try {
        lock.lock(); // Adquire a tranca para evitar
           condicoes de corrida
        if (isClosed) {
            return "Leilao encerrado!";
        }
        if (amount > highestBid) {
            highestBid = amount;
            highestBidder = user;
            return "Lance de " + user + " aceito: R$" +
               amount;
        } else {
            return "Lance muito baixo! O maior lance eh: R$"
               + highestBid;
        }
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt(); // Restaura o
           estado de interrupcao
        return "Erro ao processar o lance.";
    } finally {
        lock.unlock(); // Libera a tranca
    }
}
```

Listing 3.2 – Método placeBid

```
public void closeAuction() {
    try {
        lock.lock(); // Adquire a tranca
        isClosed = true;
        System.out.println("Leilao encerrado! Vencedor: " +
           highestBidder + " com R$" + highestBid);
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt(); // Restaura o
           estado de interrupcao
    } finally {
        lock.unlock(); // Libera a tranca
    }
}
```

Listing 3.3 – Método closeAuction

3.2.2 CLIENTE

A classe *AuctionClient* é responsável por interagir com o usuário e enviar requisições ao servidor. Seu funcionamento segue os seguintes passos:

1. O cliente se conecta ao Registro RMI e localiza o serviço remoto.
2. Ele fornece uma interface interativa para que o usuário possa enviar lances e consultar o maior lance.
3. A comunicação ocorre por meio dos métodos remotos *placeBid* e *getStatus*, garantindo que o usuário receba atualizações em tempo real sobre o estado do leilão.

Os métodos remotos *getBid* e *getStatus* estão localizados na classe Auction:

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Auction extends Remote {
    String placeBid(String user, double amount) throws
       RemoteException;
    String getStatus() throws RemoteException;
}
```

Listing 3.4 – Classe Auction

3.2.3 CONTROLE DE CONCORRÊNCIA

Para evitar problemas de concorrência, foi implementada uma tranca personalizada (*CustomLock*), garantindo que apenas um cliente por vez possa atualizar o maior lance. Essa tranca segue a lógica do protocolo Two-Phase Locking (2PL), onde:

- O cliente que deseja atualizar o lance deve adquirir a tranca antes de realizar qualquer modificação.
- Após atualizar o lance, a tranca é liberada, permitindo que outros clientes realizem novas operações.
- Caso um cliente tente acessar o recurso enquanto ele está bloqueado, ele entra em um estado de espera até que a tranca seja liberada.

Esse mecanismo evita condições de corrida e inconsistências nos dados do leilão, garantindo uma execução segura e confiável.

```
public class CustomLock {
private boolean isLocked = false;
private Thread lockingThread = null;

public synchronized void lock() throws InterruptedException {
    while (isLocked) {
        wait(); // Espera ate que a tranca seja liberada
    }
    isLocked = true;
    lockingThread = Thread.currentThread();
}

public synchronized void unlock() {
    if (Thread.currentThread() != lockingThread) {
        throw new IllegalMonitorStateException("A␣thread␣que␣
           esta␣tentando␣liberar␣a␣tranca␣nao␣eh␣a␣que␣a␣
           correta.");
    }
    isLocked = false;
    lockingThread = null;
    notify(); // Notifica uma thread em espera
}
}
```

Listing 3.5 – Classe CustomLock

4 RESULTADOS

Ao iniciar o servidor, temos a seguinte interface no servidor.

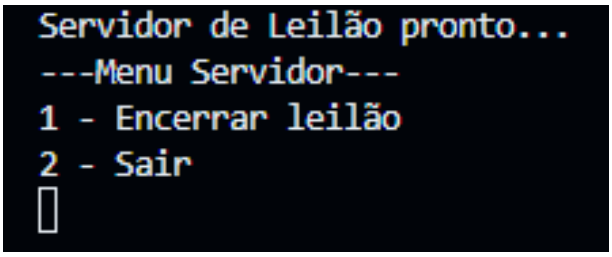

Figura 1 – Interface Servidor

Já para o usuário temos outro modelo, onde o usuário inicialmente escreve seu nome e após é possível navegar através das opções.

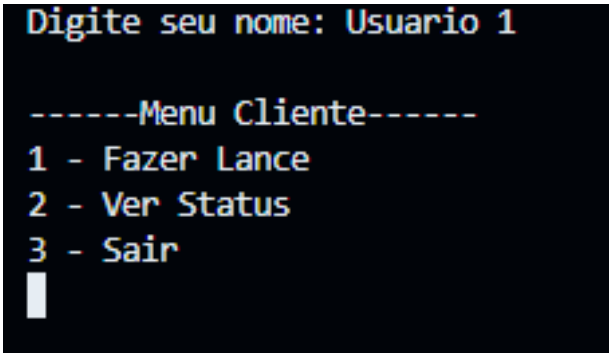

Figura 2 – Interface Cliente

Seguindo agora para um exemplo prático, temos 3 usuários, sendo eles, respectivamente, o Usuario 1, Usuario 2 e o Usuario 3. Inicialmente o Usuario 1 faz uma verificação de status e após isso faz um lance de 25 unidades.

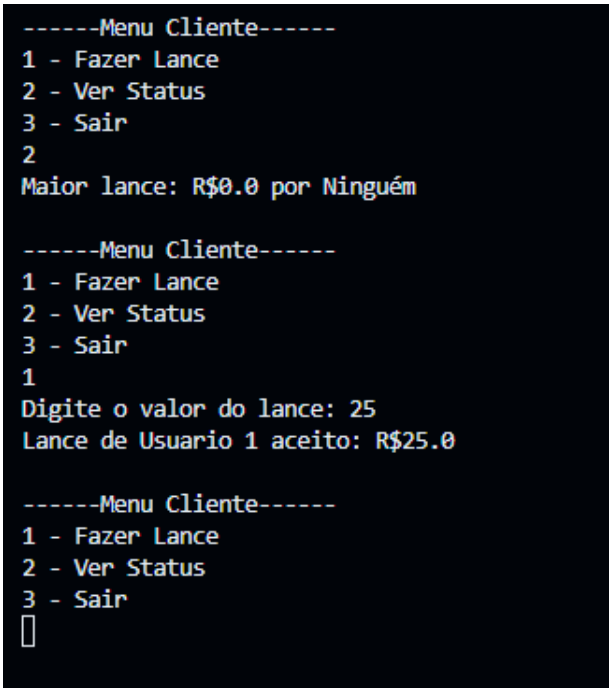

Figura 3 – Lance Usuario 1

Após isso o Usuario 2 verifica os status e faz um novo lance de 40 unidades.

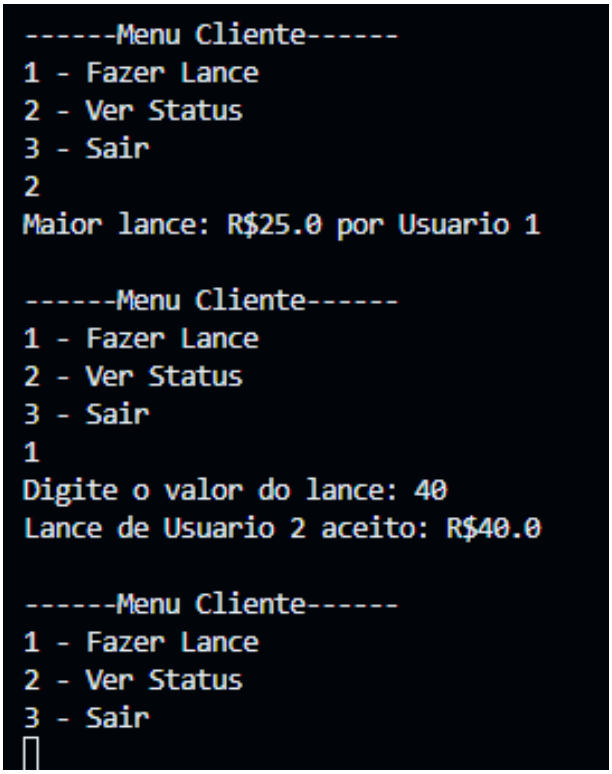

Figura 4 – Lance Usuario 2

Em seguida o Usuario 3 faz um lance de 30 unidades sem ter verificado os status dos lances. E após isso ele aumenta o lance para 100 unidades.

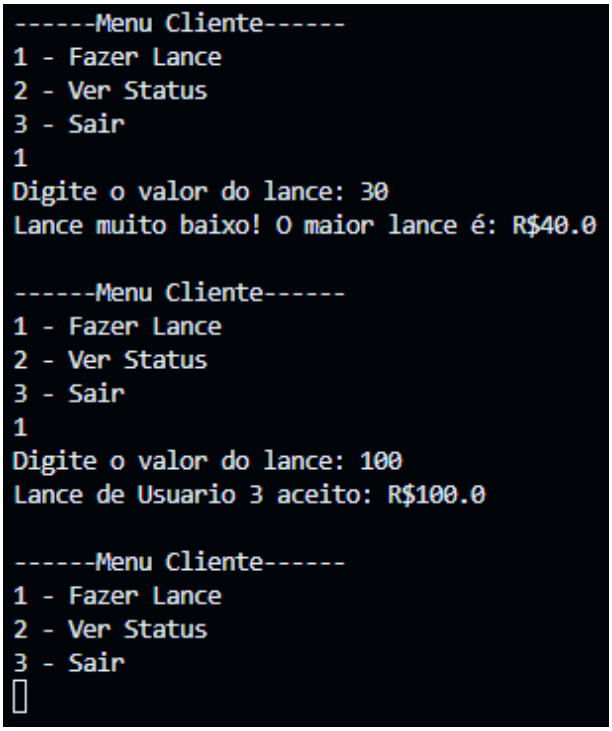

Figura 5 – Lance Usuario 3

Após o lance do Usuario 3, o servidor encerra o Leilão e encerra o servidor também.

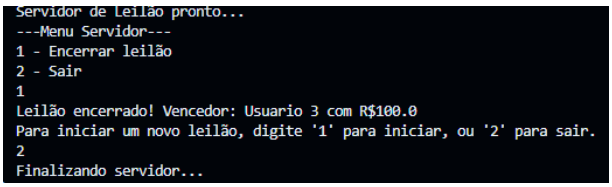

Figura 6 – Encerrar Leilão

O servidor Web, através do Payara, não foi implementado nesse projeto. Os principais motivos foram erros de compilação e problemas de conflitos com a versão Java RMI e as bibliotecas utilizadas.

O código desenvolvido também não apresenta prevenções de possíveis erros, como, por exemplo, o usuário digitar um caractere especial ou um número muito grande. Isso acarreta em uma mensagem de erro do compilador e a finalização do cliente.

5 CÓDIGO

Arquivo Auction.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Auction extends Remote {
    String placeBid(String user, double amount) throws
       RemoteException;
    String getStatus() throws RemoteException;
}
```

Listing 5.1 – Auction.java

Arquivo AuctionClient.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;

public class AuctionClient {
    public static void main(String[] args) {
        try {
            Registry registry = LocateRegistry.getRegistry("
               localhost", 1099);
            Auction auction = (Auction) registry.lookup("
               AuctionService");

            Scanner scanner = new Scanner(System.in);
            System.out.print("Digite␣seu␣nome:␣");
            String user = scanner.nextLine();

            while (true) {
                System.out.print("\n------Menu␣Cliente------\
                   n");
                System.out.println("1␣-␣Fazer␣Lance");
                System.out.println("2␣-␣Ver␣Status");
                System.out.println("3␣-␣Sair");
                int choice = scanner.nextInt();

                switch (choice) {
```

```
                    case 1:
                        System.out.print("Digite o valor do
                            lance: ");
                        double bid = scanner.nextDouble();
                        System.out.println(auction.placeBid(
                            user, bid));
                        break;
                    case 2:
                        System.out.println(auction.getStatus
                            ());
                        break;
                    case 3:
                        System.out.println("Saindo...");
                        scanner.close();
                        return;
                    default:
                        System.out.println("Op   o inv lida
                            !");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Listing 5.2 – AuctionClient.java

Arquivo AuctionServer.java

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.Scanner;

public class AuctionServer extends UnicastRemoteObject
   implements Auction {

    private String highestBidder; // Armazena o nome do maior
        lance
```

```
    private double highestBid;     // Armazena o valor do
       maior lance
    private boolean isClosed;      // Indica se o leil o
       est  encerrado
    private final CustomLock lock = new CustomLock(); //
       Tranca personalizada

    // Construtor do servidor
    protected AuctionServer() throws RemoteException {
        super();
        this.highestBid = 0; // Inicializa o maior lance como
            0
        this.highestBidder = "Ninguem"; // Inicializa o maior
            lance como "Ninguem"
        this.isClosed = false; // Inicializa o leilao como
           aberto
    }

    @Override
    public String placeBid(String user, double amount) throws
        RemoteException {
        try {
            lock.lock(); // Adquire a tranca para evitar
               condicoes de corrida
            if (isClosed) {
                return "Leilao encerrado!";
            }
            if (amount > highestBid) {
                highestBid = amount;
                highestBidder = user;
                return "Lance de " + user + " aceito: R$" +
                   amount;
            } else {
                return "Lance muito baixo! O maior lance eh:
                   R$" + highestBid;
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt(); // Restaura o
                estado de interrupcao
```

```
            return "Erro ao processar o lance.";
        } finally {
            lock.unlock(); // Libera a tranca
        }
    }

    @Override
    public String getStatus() throws RemoteException {
        return "Maior lance: R$" + highestBid + " por " +
           highestBidder;
    }

    // Metodo para encerrar o leil o
    public void closeAuction() {
        try {
            lock.lock(); // Adquire a tranca
            isClosed = true;
            System.out.println("Leilao encerrado! Vencedor: "
                + highestBidder + " com R$" + highestBid);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt(); // Restaura o
                estado de interrupcao
        } finally {
            lock.unlock(); // Libera a tranca
        }
    }

    public static void main(String[] args) {
        try {
            AuctionServer server = new AuctionServer();
            Registry registry = LocateRegistry.createRegistry
               (1099);
            registry.rebind("AuctionService", server);
            System.out.println("Servidor de Leilao pronto..."
               );

            Scanner scanner = new Scanner(System.in);
            while (true) {
                System.out.println("---Menu Servidor---");
```

```
                System.out.println("1 - Encerrar leilao");
                System.out.println("2 - Sair");
                String input = scanner.nextLine();
                if (input.equalsIgnoreCase("1")) {
                    server.closeAuction();
                    System.out.println("Para iniciar um novo
                        leilao, digite '1' para iniciar, ou
                        '2' para sair.");
                    input = scanner.nextLine();
                    if (input.equalsIgnoreCase("1")) {
                        server = new AuctionServer();
                        registry.rebind("AuctionService",
                            server);
                        System.out.println("Novo leilao
                            iniciado!");
                    } else if (input.equalsIgnoreCase("2")) {
                        System.out.println("Finalizando
                            servidor...");
                        UnicastRemoteObject.unexportObject(
                            server, true); // Desregistra o
                            servidor RMI
                        System.exit(0); // Encerra o programa
                    }
                } else if (input.equalsIgnoreCase("2")) {
                    System.out.println("Finalizando servidor
                        ...");
                    UnicastRemoteObject.unexportObject(server
                        , true); // Desregistra o servidor RMI
                    System.exit(0); // Encerra o programa
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Listing 5.3 – AuctionServer.java

Arquivo CustomLock.java

```
public class CustomLock {
    private boolean isLocked = false;
    private Thread lockingThread = null;

    public synchronized void lock() throws
       InterruptedException {
        while (isLocked) {
            wait(); // Espera ate que a tranca seja liberada
        }
        isLocked = true;
        lockingThread = Thread.currentThread();
    }

    public synchronized void unlock() {
        if (Thread.currentThread() != lockingThread) {
            throw new IllegalMonitorStateException("A␣thread␣
               que␣esta␣tentando␣liberar␣a␣tranca␣nao␣eh␣a␣
               que␣a␣adquiriu.");
        }
        isLocked = false;
        lockingThread = null;
        notify(); // Notifica uma thread em espera
    }
}
```

Listing 5.4 – CustomLock.java

6 CONCLUSÃO

O sistema distribuído desenvolvido utiliza Java RMI para comunicação remota e um mecanismo de 2PL para garantir consistência nos lances. A implementação seguiu uma estrutura modular, permitindo escalabilidade e manutenção eficientes. Os testes demonstraram que o controle de concorrência é eficaz na prevenção de conflitos, garantindo que o maior lance seja sempre registrado corretamente. No entanto, o sistema apresenta limitações em termos de escalabilidade para um grande número de usuários simultâneos.

REFERÊNCIAS

DEVMEDIA. **Introdução ao Java RMI**. 2025. https://www.devmedia.com.br/introducao-ao-java-rmi/2012. Acesso em: 9 fev. 2025. Citado na página 19.

DEVMEDIA. **Uma introdução ao RMI em Java**. 2025. https://www.devmedia.com.br/uma-introducao-ao-rmi-em-java/28681. Acesso em: 9 fev. 2025. Citado na página 19.

ROCHA, Lucio A. **Controle de Concorrência - Propriedades, Estados e Protocolos**. 2025. Slides de aula. Disponibilizado pelo professor Lucio Agostinho Rocha na disciplina Sistemas Distribuídos. Citado na página 19.

ROCHA, Lucio A. **Invocação de métodos remotos**. 2025. Slides de aula. Disponibilizado pelo professor Lucio Agostinho Rocha na disciplina Sistemas Distribuídos. Citado na página 19.

(DEVMEDIA, 2025a) (DEVMEDIA, 2025b) (ROCHA, 2025b) (ROCHA, 2025a)