

Instituto Tecnológico de Aeronáutica
Engenharia Aeroespacial e Mecânica



CE-288
Programação Distribuída

Professor: Dr. Celso Massaki Hirata

**Relatório da Realização do Lab02 “Exclusão Mútua
pelo Algoritmo de Ricart e Agrawala”**

Lucas Kriesel Sperotto

21 de Novembro de 2011

1 – Objetivo:

Este trabalho tem como objetivo descrever as atividades desenvolvidas na elaboração do Lab02 para a disciplina de Programação Distribuída, abordando os passos para a implementação da aplicação em JAVA do algoritmo de Ricart & Agrawala (1981) para a obtenção da exclusão mútua por relógios lógicos, elaboração das interfaces gráficas e interações com o usuário. Por fim relatar testes e funcionamento.

2 – Introdução:

Na elaboração deste trabalho, algumas classes e a interface gráfica do projeto Lab01 foram reutilizadas e adaptadas para a elaboração do Lab02.

Este relatório está dividido em nove tópicos fundamentais: Objetivo; Introdução; Descrição da arquitetura distribuída; Descrição detalhada do projeto; Manual do Usuário; Manual de Instalação; Teste de Funcionamento; Conclusão e Referencias.

Na descrição da arquitetura distribuída será comentada a topologia e conexões entre os processos. No tópico descrição detalhada do projeto será abordado às características de implementação do cenário e do algoritmo, bem como descrito classes e métodos e seus comportamentos. Os tópicos manual do usuário e manual de instalação fornecem respectivamente informações para a utilização e instalação do software. No tópico teste de funcionamento está descrito os testes realizados e a prova do funcionamento. Também temos um arquivo anexado com os executáveis, fontes, projetos e o “javadoc” gerado para as classes.

Ainda na introdução gostaria de comentar algumas informações importantes como a exposição do algoritmo de Ricart & Agrawala (1981) e a interpretação do seu funcionamento.

Este algoritmo é uma versão melhorada do algoritmo de Lamport (1978) para a obtenção de exclusão mútua através de relógios lógicos. O algoritmo de Ricart & Agrawala possui algumas melhoras sobre o algoritmo de Lamport, como exemplo um menor tráfego de mensagens na rede (Lamport: $3(n-1)$ mensagens; Ricart & Agrawala: $2(n-1)$ mensagens – sendo n o número de processos), além de um tamanho reduzido das mensagens (menos informação em cada mensagem).

O funcionamento do algoritmo é bastante simples. Quando um processo P_i deseja exclusão, ele envia uma mensagem de REQUEST para os processos P_j (broadcast). Nessa mensagem está anexado seu relógio lógico e seu id. Quando P_j recebe um REQUEST, ele pode imediatamente responder com um REPLY ou adiar a resposta. P_j adia a resposta se possui um pedido de exclusão e seu tempo de relógio for menor que o tempo do pedido recebido, no caso do tempo ser igual ele compara o Id do processo. P_i somente entrará na zona de exclusão quando tiver recebido um REPLY para cada REQUEST enviado. Ao sair da zona de exclusão ele envia todos os REPLY adiados.

3 – Descrição da Arquitetura Distribuída:

Na figura 1 pode-se ver o detalhe das conexões entre os processos. Para os clientes tem-se uma topologia completamente conectada. Já o processo “Trem” possui conexão bidirecional com todos os clientes.

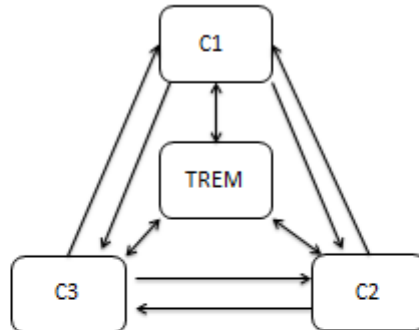


Figura 1 - Detalhe da Arquitetura Distribuída.

Todas as comunicações entre os processos é feita utilizando DatagramaSocket com confirmação de recebimento de mensagem. Este critério de confirmação se fez necessário pois o algoritmo não permite perda de mensagens. Como base para a implementação das conexões utilizou-se exemplos do livro do Deitel (2005).

4 – Descrição detalhada do Projeto:

Dada à abstração do cenário e do algoritmo, pode-se ver na figura 2 o diagrama de classes para o aplicativo “Trem”.

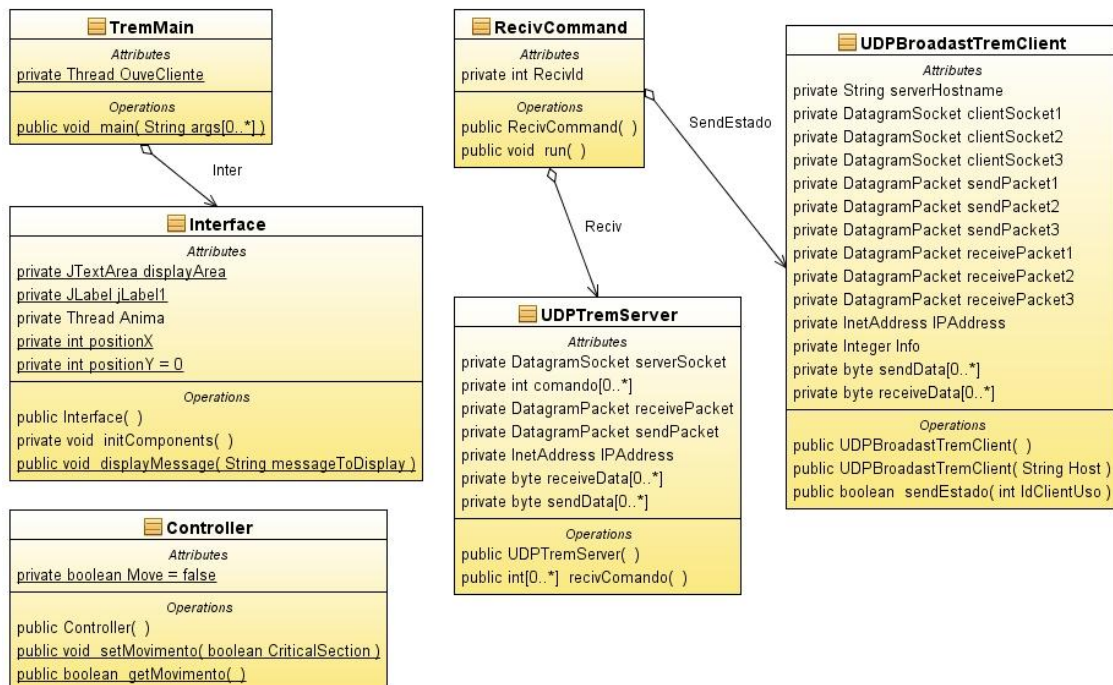


Figura 2 - Diagrama de Classes do Aplicativo Trem.

No aplicativo “Trem”, a classe “Interface” descreve a interface com o usuário, e internamente tem-se uma Thread responsável pela animação da “String” (nomeada Anima). A classe “Controller” é apenas um repositório para variáveis estáticas acessadas por métodos “synchronized”, são informações que necessitam ser vistas por mais de uma thread/classe.

A classe “RecvCommand” é uma thread que instancia a classe “UDPTremServer” (responsável por receber as solicitações do cliente) e fica ouvindo a porta de comunicação para receber comandos dos clientes. A classe “UDPTremServer” é responsável pela conexão com o cliente, apenas uma porta recebe comandos de todos os clientes.

Na classe “TremMain” (classe principal) temos a instanciação da interface e da Threads do tipo “RecvCommand” (para ouvir todos os clientes). E por fim a classe “UDPBroadcastTremClient” é responsável por avisar os clientes sobre quem está com a exclusão. Isto foi feito para cumprir com o requisito de que os botões “iniciar_controle” dos clientes sem exclusão fossem desabilitados.

Na figura 3 pode ser visto o diagrama de classes para os aplicativos “Cliente”, tem-se uma total simetria de código, quanto à simetria de dados, uma única variável deve ser inicializada com valores diferentes, o Id de cada Cliente.

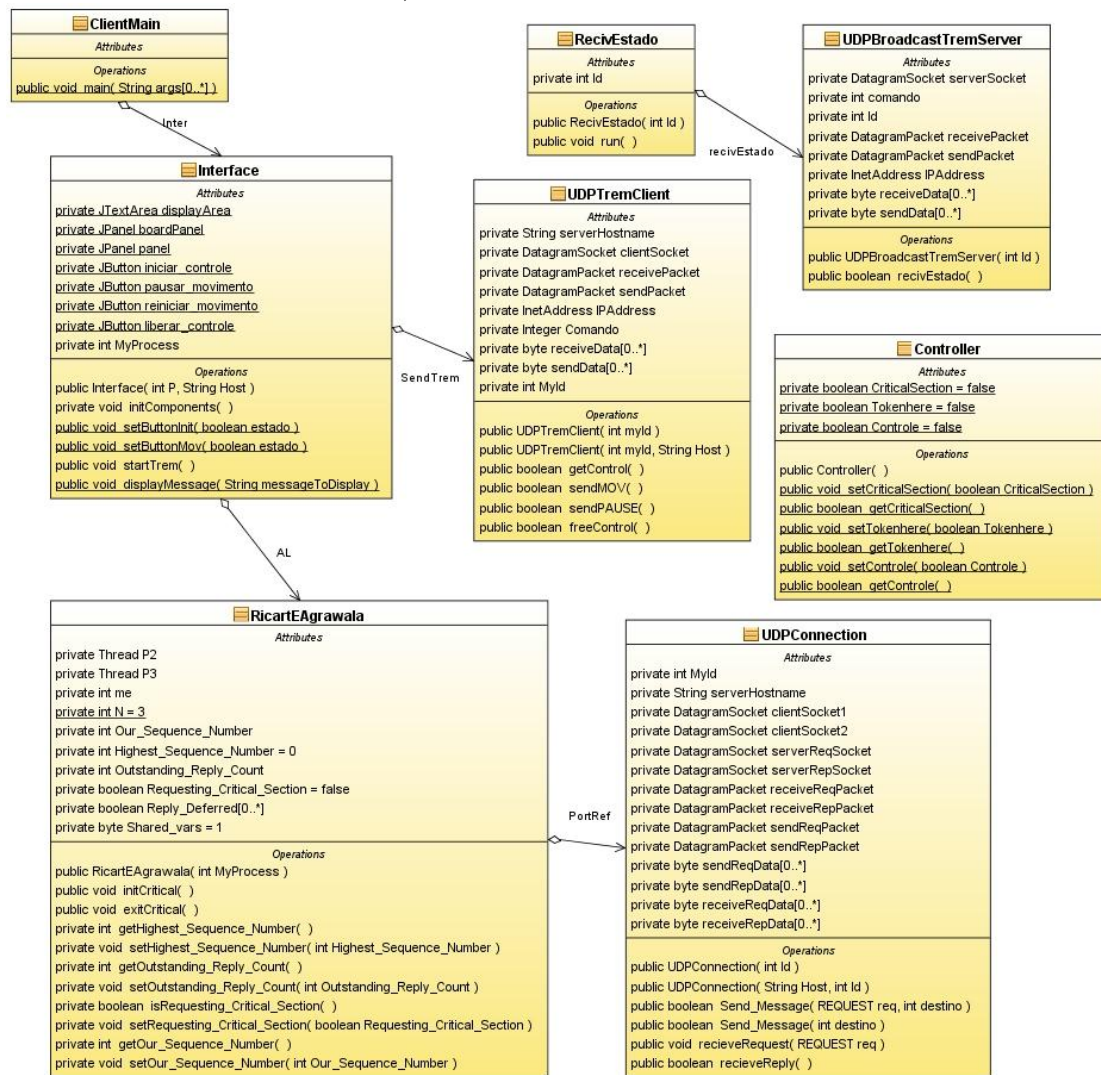


Figura 2 - Diagrama de Classes do Aplicativo Cliente.

A classe “Interface” é a classe onde esta definida a interface com o usuário. A classe “ClientMain” é a classe principal, onde é instanciada a interface e iniciada a Thread “RecvEstado”.

A classe “RecvEstado” é uma Thread que instancia a classe “UDPBroadcastTremServer” que por sua vez recebe as mensagens vindas do processo Trem. Essas mensagens são informações sobre quem esta usando o serviço. Já a classe

“UDPTremClient” é responsável por enviar os comando ao Trem, quando o cliente consegue a exclusão mútua.

A classe “Controller” tem a mesma função descrita para o processo Trem. A classe “UDPAnelConnection” é responsável pelas conexões entre os clientes. Possuindo métodos para o envio e o recebimento de mensagens.

A classe “RicartEAgrawala” é onde está a implementação do algoritmo para obtenção da exclusão. Essa classe é composta por dois threads, uma para ouvir mensagens de “reply”, e outra para ouvir mensagens de “request”, também possui um método que deve ser chamado quando se requer exclusão mútua e outro para a liberação da exclusão.

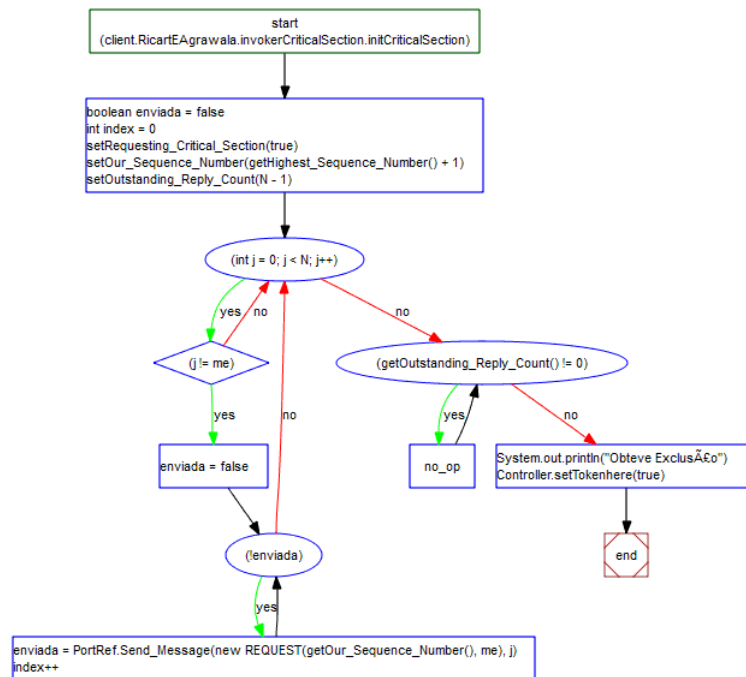


Figura 3 – Diagrama de Controle de Fluxo do método que requisita exclusão.

Na figura 3 tem-se o diagrama de fluxo para o método que requisita a exclusão, pode-se ver claramente com o diagrama o comportamento do algoritmo. Quando se deseja exclusão, primeiramente inicializa-se algumas variáveis e se envia uma mensagem de request para cada outro cliente. Assim que as mensagens forem enviadas, o processo aguarda o recebimento de todas as mensagens de reply para informar a obtenção da exclusão.

Para facilitar a implementação, foi criado um método para liberar a exclusão. Esse método é chamado quando o botão “liberar” é pressionado. Na figura 4 pode-se ver o gráfico de fluxo para esse método.

Quando o processo libera a exclusão, as correspondentes variáveis são alteradas e as mensagens de reply adiadas são enviadas.

Os código fontes se encontram na pasta fontes no arquivo anexado.

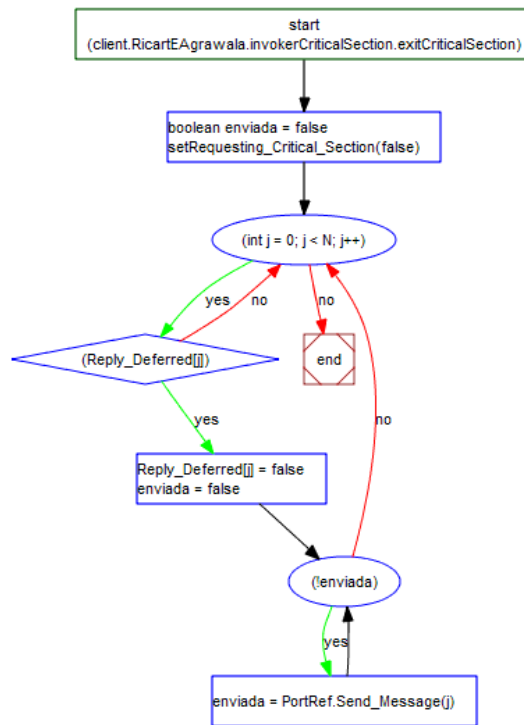


Figura 4 – Diagrama de Controle de Fluxo do método que libera exclusão.

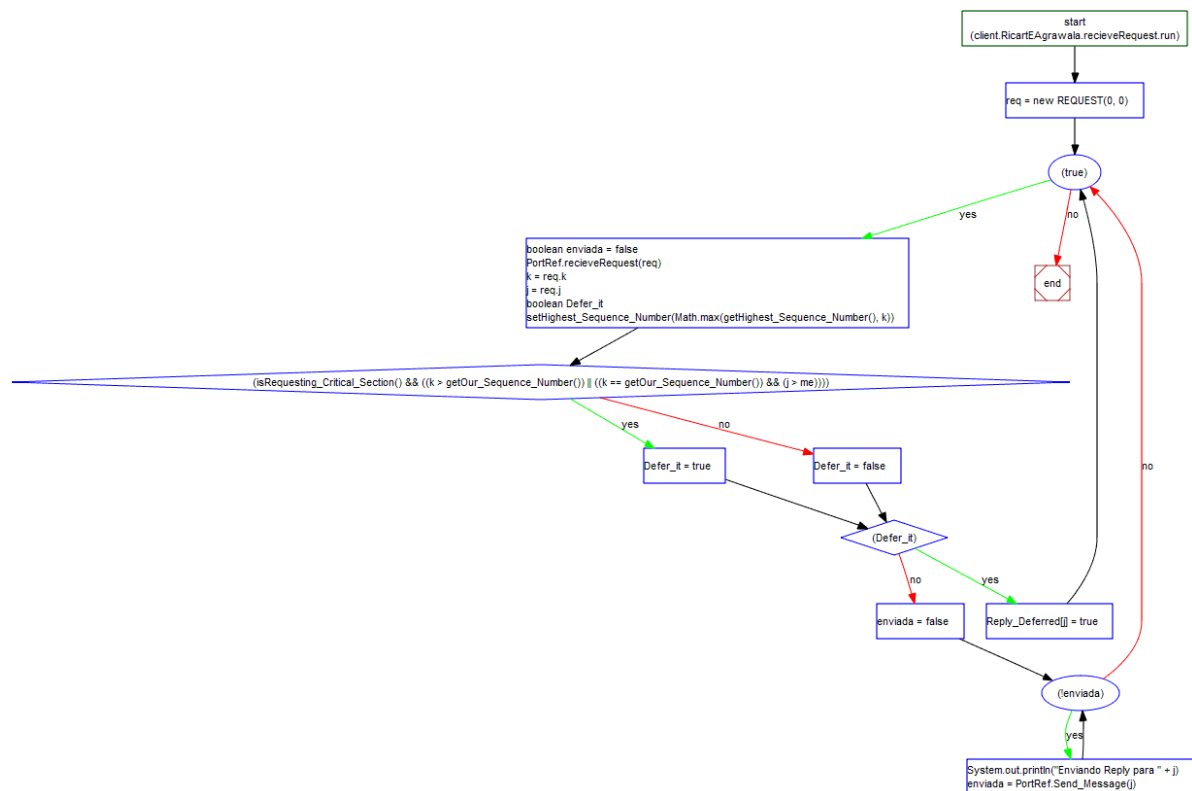


Figura 5 – Diagrama de Controle de Fluxo do Processo que recebe mensagem de request.

Na figura 5, pode-se ver o diagrama de fluxo do processo que recebe uma mensagem de request. Esse processo decide se envia uma mensagem de reply, ou se adia o envio. Com o auxílio do diagrama, pode-se ver que para cada mensagem recebida, se o processo não possui pedido de request, ele envia a mensagem de reply. Caso possua um pedido de request o processo compara o tempo lógico do pedido e/ou o Id (no caso do tempo se igual), e adia a mensagem, guardando em um array para que quando liberada a exclusão esses reply adiados possam ser enviados.

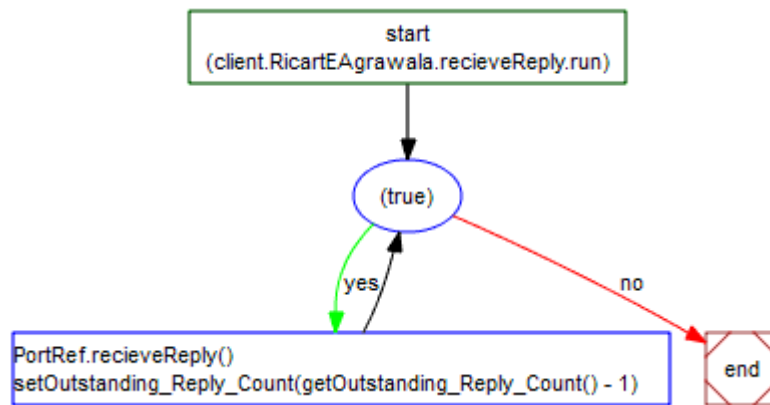


Figura 6 – Diagrama de Controle de Fluxo do Processo que recebe mensagem de reply.

Por ultimo na figura 6 tem-se o diagrama de controle do processo que recebe uma mensagem de reply. Esse processo decrementa a variável que guarda a informação de quantas mensagens de request foi enviado, quando essa variável assumir valor igual à zero, o cliente pode obter exclusão.

Esses diagramas apresentados facilitam a visualização da implementação e garantem a verificação da conformidade com requisitos.

5 – Manual do Usuário:

Uma interface adicional foi criada de modo que o usuário possa entrar com o Id do cliente. Na figura 7 pode-se ver essa interface.

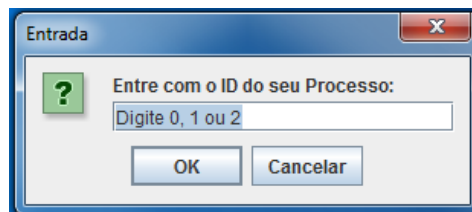


Figura 7 - Detalhe da Interface do Processo Trem

As interfaces gráficas foram criadas de acordo com o especificado nas instruções do Lab02. Sendo assim para a interface do processo Trem, tem-se um “JFrame” com uma “String” que ao receber o comando de um dos clientes se movimenta circularmente na tela.

Para um melhor entendimento do funcionamento por parte do usuário, foi adicionado uma “JTextArea” para mostrar ao usuário mensagens que descrevem os eventos. Por exemplo, quais comandos foram recebidos, quem requisitou o serviço etc... (detalhe dessa interface pode ser visto na figura 8).

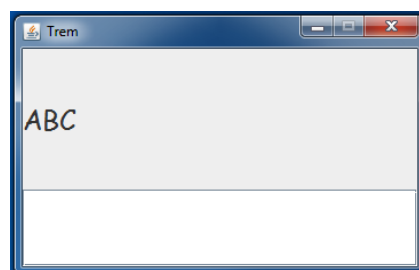


Figura 8 - Detalhe da Interface do Processo Trem

As mensagens exibidas na “JTextArea” são as seguintes:

- “Recebeu de C(nº) comando: STOP”; Quando recebe o comando para parar a String de algum cliente.
- “Recebeu de C(nº) comando: MOV”; Quando recebe o comando para movimentar a String de algum cliente.
- “C(nº) Requisitou Serviço”; Quando recebe a requisição de serviço de algum cliente.
- “C(nº) Liberou Serviço”; Quando o cliente libera o serviço.

A interface dos Clientes (detalhe na figura 9) possui quatro botões com suas respectivas funcionalidades:

- **Iniciar Controle:** Este botão requisita a exclusão mútua, suas ações são as seguintes: inicia o processo de envio de mensagens REQUEST, imprime na área de texto que esta requisitando a exclusão mútua, estabelece uma conexão e envia para o processo Trem o comando de que ele requisitou o serviço, e por fim habilita o botão Liberar Controle e se desabilita.
- **Liberar Controle:** Este botão fica disponível apenas quando o cliente está com o controle do processo Trem. Quando pressionado ele escreve na tela a mensagem de que esta liberando o controle, ele envia para o processo Trem o comando para liberar o serviço, e inicia o processo de enviar as mensagens de REPLY pendentes. E desabilita a si próprio e os botões de controle da “String”.
- **Pausar Movimento:** Este botão se torna habilitado quando o cliente tem o controle da “String” e ela esta em movimento. Sua principal ação é escrever em tela que ira parar o movimento e envia o comando “PAUSE” para o processo Trem.
- **Reiniciar Movimento:** Este botão somente esta disponível quando o cliente possui controle sobre a String e a String encontra-se parada no processo Trem, esse botão envia um comando “MOV” para o processo Trem.

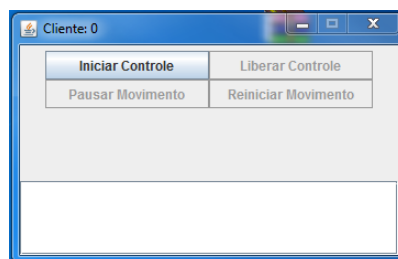


Figura 9 - Detalhe da Interface Cliente

Neste aplicativo, tombem-se inseriu uma área de texto para informar o usuário sobre quem esta controlando o serviço e outras informações relevantes. As mensagens que a “JTextArea” exibe são as seguintes:

- “Cliente C0 Requisitou Serviço. Aguarde Liberação...”; “Cliente C1 Requisitou Serviço. Aguarde Liberação...”; “Cliente C2 Requisitou Serviço. Aguarde Liberação...”; Quando o respectivo cliente está com o controle do processo Trem.
- “Serviço Liberado...”; Quando o processo trem está disponível.
- “Requisitando Exclusão Mútua...”; Quando o cliente deseja a Exclusão Mútua.
- “Pausando movimento...”; Quando o cliente deseja pausar o movimento.
- “Iniciando Movimento...”; Quando o cliente deseja iniciar o movimento.
- “Liberando Controle...”; Quando o cliente libera o controle do processo Trem.

6 – Manual de Instalação:

Este trabalho foi codificado no ambiente de desenvolvimento integrado NetBeans 7.0.1, utilizando a JDK 7.1. Dessa forma no arquivo anexado temos um arquivo “Projetos.rar” onde se encontram os dois projetos do NetBeans (caso deseje abrir os projetos).

Também temos dois arquivos “.jar” (Client.jar e Trem.jar), que podem ser executados com os comandos: “**java -jar Trem.jar**” e “**java -jar Client.jar**”, como mostrado na figura 10.

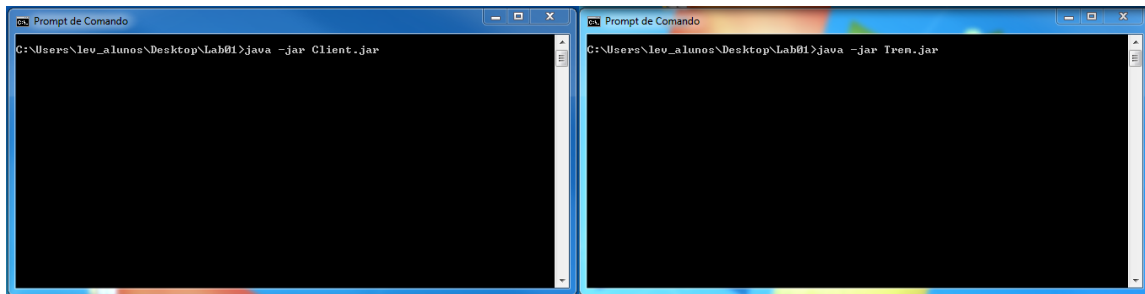


Figura 10 - Detalhe Comando de Execução

Para a execução, devem-se abrir quatro “prompts” de comando (um para cada execução), executar três clientes e um trem. Quando se executar um cliente irá abrir uma janela (detalhe figura 11) requisitando que se insira o “Id” do cliente (0, 1 ou 2 não importando a ordem). Confirmando no botão “OK”, o programa dará início a execução do anel e o usuário poderá requisitar o controle da “String”.

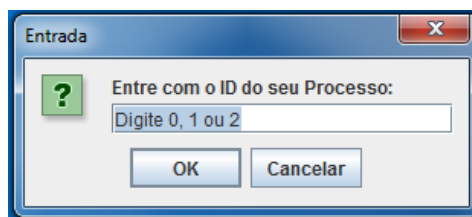


Figura 11 - Detalhe da Caixa de Dialogo do Cliente

7 – Testes e Funcionamento:

Tendo executado as instâncias do aplicativo como descrito no tópico anterior, os clientes irão exibir suas telas e mostrar no prompt mensagens de que esta aguardando reply e request. Dado que o funcionamento do algoritmo e do software já foi discutido, será apresentado um caso de execução para a obtenção de exclusão para o cliente C0. E em seguida outro caso de teste para o pedido de exclusão por todos os processos, de forma a verificar se todos conseguem a exclusão sem conflito e na ordem do pedido.

Como comentado, tendo iniciado os aplicativos, e o Cliente C0 requisita o controle da string, na figura 12 pode-se ver os passos para o cliente C0. Primeiro ele envia mensagens de request para os dois vizinhos, recebendo os reply correspondentes ele obtém a exclusão e informa o processo Trem que possui o controle.

Na figura 13, o Cliente C1 recebe mensagem de request e em seguida devolve um reply, pois o mesmo não possui pedido de exclusão. Para o cliente C2, é análogo a C1, ele recebe um request e já retorna um reply.

```

Microsoft Windows [versão 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\lev_alunos>java -jar Client.jar
Ouvindo REPLY
Ouvindo REQUEST
Ouvindo porta de comunicação: 1000
Tentando se conectar com /127.0.0.1> via porta UDP: 5001
REQUEST Enviado para 1!!!!
Recebeu Retorno de REQUEST!!!!
Tentando se conectar com /127.0.0.1> via porta UDP: 5002
Recebeu REPLY de: 49465
Criou pacote para envio da confirmação
REQUEST Enviado para 2!!!!
Recebeu Retorno de REQUEST!!!!
Enviou confirmação de REPLY
Ouvindo REPLY
Recebeu REPLY de: 49467
Criou pacote para envio da confirmação
Enviou confirmação de REPLY
Ouvindo REPLY
Obteve Exclusão
Enviando Comando CONTROL
Comando CONTROL Enviado, Aguardando Retorno!!!!
Recebeu Retorno!!!!
Recebeu Pacote com o Estado:
Criou pacote para envio da confirmação
Enviou confirmação

```

Figura 12 - Detalhe da Caixa de Dialogo do Cliente

```

Microsoft Windows [versão 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\lev_alunos>java -jar Client.jar
Ouvindo REPLY
Ouvindo REQUEST
Ouvindo porta de comunicação: 1001
Recebeu REQUEST de: 49464
Criou pacote para envio da confirmação
Enviou confirmação de REQUEST
Enviando Reply para 6
Tentando se conectar com /127.0.0.1> via porta UDP: 2000
REPLY Enviado para 0!!!!
Recebeu Retorno de REPLY !!!!
Ouvindo REQUEST
Recebeu Pacote com o Estado:
Criou pacote para envio da confirmação
Enviou confirmação

```

Figura 13 - Detalhe da Caixa de Dialogo do Cliente

Na figura 14 o processo Trem recebe mensagem com o comando de que o cliente obteve o controle da string.

```

Microsoft Windows [versão 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\lev_alunos>java -jar Trem.jar
Ouvindo porta de comunicação: 4000
Recebeu Pacote com o Comando: CONTROL
Criou pacote para envio da confirmação
Enviou confirmação
Preparando envio do Broadcast.
Broadcast Enviado, Aguardando Retorno!!!!

```

Figura 14 - Detalhe da Caixa de Dialogo do Cliente

Pode-se ver nas figuras adiante o que acontece com as interfaces para a execução descrita anteriormente. Na figura 15, temos o estado inicial para o cliente C0, na figura 16 o cliente requisitou exclusão e obteve a exclusão. Na figura 17 como o estado da interface dos demais clientes quando C0 obteve exclusão.

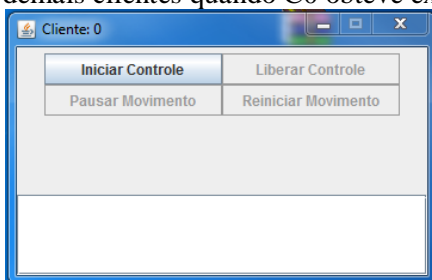


Figura 3 - Estado Inicial do Cliente.

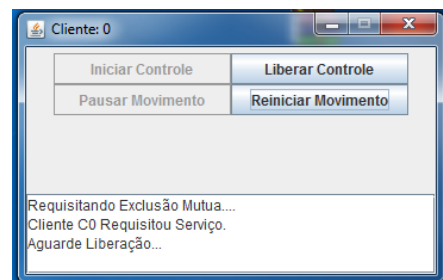


Figura 2 - Iniciando Controle.

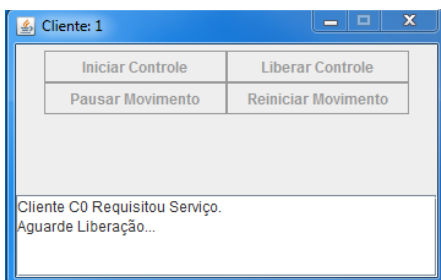


Figura 5 - Cliente Bloqueado.

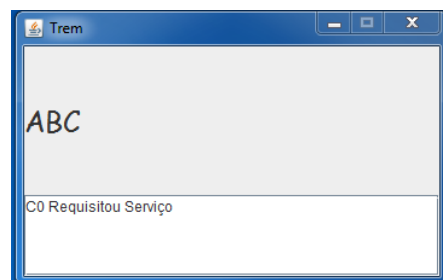


Figura 4 - Requisição no Trem.

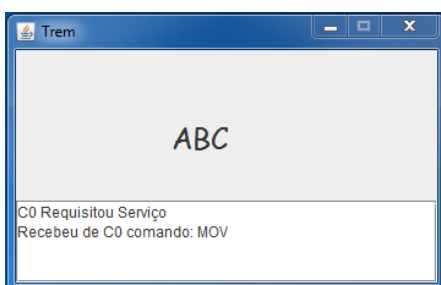


Figura 6 - Recebimento de Comando MOV.

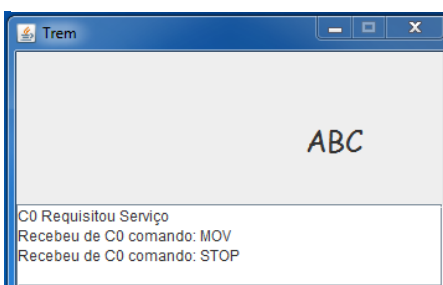


Figura 20 - Recebimento de Comando STOP.

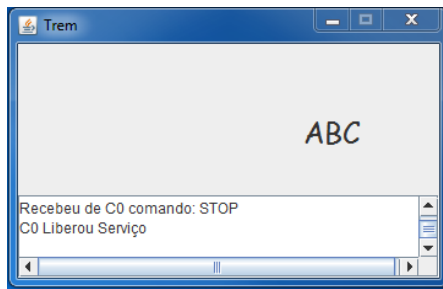


Figura 21 - Liberação de Serviço.

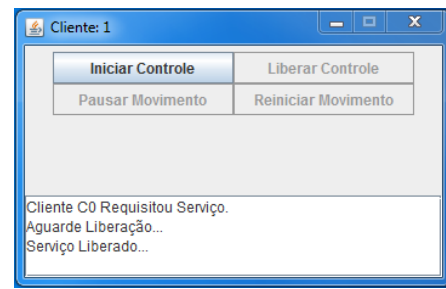


Figura 22 - Liberação de Serviço.

Nas figuras 18 a 21 tem-se os estados do processo Trem para quando o usuário com a exclusão pede para movimentar a String, parar a String e libera o controle. Na figura 22 tem-se o estado dos demais clientes quando o controle é liberado.

No segundo teste, foi inserido um tempo para o envio das mensagens de request, de forma a possibilitar que o pedido de exclusão fosse feito por todos os clientes. A seguir será descrito as trocas de mensagens para provar o correto funcionamento.

Tendo como ordem de pedido de exclusão C1, C0 e C2, C1 deverá obter a exclusão, e quando liberado C0 deve obter em seguida e finalmente quando liberado C2 deve obter a exclusão.

```

C:\> Prompt de Comando - java -jar Client.jar

Enviou confirmação
Tentando se conectar com /127.0.0.1> via porta UDP: 5000
REQUEST Enviado para 0!!!!
Recebido Retorno de REQUEST!!!!
Tentando se conectar com /127.0.0.1> via porta UDP: 5002
Recebeu REPLY de: 49398
Criou pacote para envio da confirmação
REQUEST Enviado para 2!!!!
Recebido Retorno de REQUEST!!!!
Enviou confirmação de REPLY
Ouvindo REPLY
Recebeu REQUEST de: 49401
Criou pacote para envio da confirmação
Enviou confirmação de REQUEST
Ouvindo REQUEST
Recebeu REQUEST de: 49404
Criou pacote para envio da confirmação
Enviou confirmação de REQUEST
Ouvindo REQUEST
Recebeu REPLY de: 49405
Criou pacote para envio da confirmação
Enviou confirmação de REPLY
Ouvindo REPLY
Obteve Exclusão
Enviando Comando CONTROL.
Comando CONTROL Enviado, Aguardando Retorno!!!!
Recebido Retorno!!!!
Recebeu Pacote com o Estado: 8
Criou pacote para envio da confirmação
Enviou confirmação

```

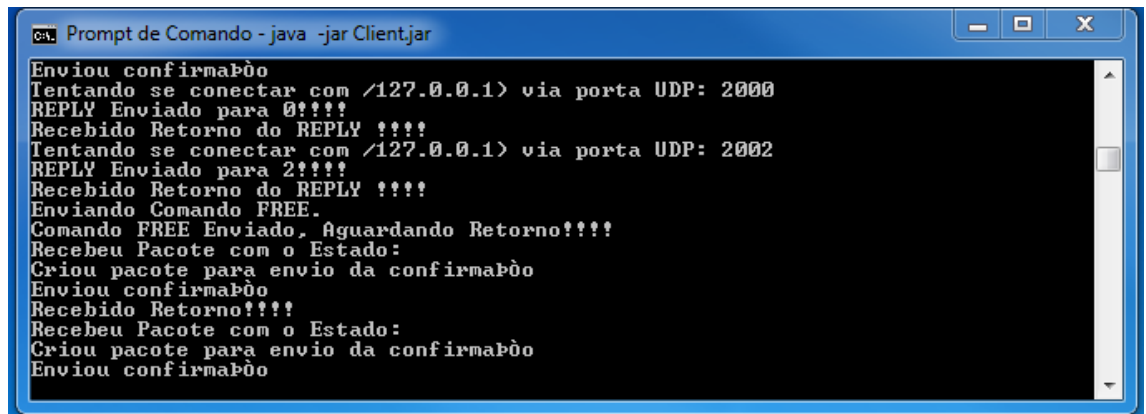
Figura 23 – Mensagens do Processo C1 obtendo exclusão

Na figura 23, tem-se as etapas do Cliente C1 para obter a exclusão, ele envia duas mensagens de request, onde uma é respondida imediatamente, e a outra, logo depois recebe as duas mensagens de request dos outros processos e o ultimo reply, o que lhe garante a exclusão.

Os clientes C2 recebe o pedido de request de C1, e imediatamente responde com reply, em seguida envia seus pedidos de request, porém não recebe nenhum reply, quando o pedido de request de C0 chega, ele imediatamente envia o reply correspondente.

Para o cliente C0, a diferença é que ele decide adiar o envio de reply para C2, e fica aguardando a chegada do reply de C1.

Na figura 24 pode-se ver o cliente C1 liberando a exclusão, e enviando os reply adiados para C2 e C0. Com essas mensagens C0 obtém a exclusão e C2 aguarda um reply de C0.



```
Prompt de Comando - java -jar Client.jar
Enviou confirmação
Tentando se conectar com /127.0.0.1> via porta UDP: 2000
REPLY Enviado para 0!!!!
Recebido Retorno do REPLY !!!!
Tentando se conectar com /127.0.0.1> via porta UDP: 2002
REPLY Enviado para 2!!!!
Recebido Retorno do REPLY !!!!
Enviando Comando FREE.
Comando FREE Enviado, Aguardando Retorno!!!!
Recebeu Pacote com o Estado:
Criou pacote para envio da confirmação
Enviou confirmação
Recebido Retorno!!!!
Recebeu Pacote com o Estado:
Criou pacote para envio da confirmação
Enviou confirmação
```

Figura 24 – Liberação da exclusão por C1

Por fim quando C0 libera a exclusão, ele envia o reply adiado para C2 de forma que C2 obtém a exclusão. Dada a sequencia de mensagens comprova-se o correto funcionamento do algoritmo, onde as decisões não ocasionaram conflito ou mesmo deadlock.

8 – Conclusão:

Esta atividade proporcionou um melhor entendimento do algoritmo para obtenção de exclusão. A implementação e abstração dos processos necessários ao algoritmo, bem como a tomada de decisões para adiar o envio de reply sem que cause deadlock, mostraram uma visão mais ampla da elaboração e estratégias distribuídas. A rotulação de tempo também é uma ferramenta muito útil para manter uma ordenação global nos processos, bastando poucas mensagens.

9 - Referências:

DEITEL, H. M. ;DEITEL, P. J. Java: Como Programar. 6ª Ed.Trad. Edson Furmankiewicz. São Paulo: Pearson Pretince Hall, 2005.

LAMPOR, L. **Time, Clocks and the ordering of events in a distributed system.** Comm. ACM 21, 7 p. 558-565, 1978.

RICART, G; AGRAWALA, A. **An Optimal Algorithm for Mutual Exclusion in Computer Networks.** Comm. ACM 24, 1 p. 9-17, 1981.