Instituto Tecnológico de Aeronáutica

Engenharia Aeroespacial e Mecânica

CE-288
Programação Distribuída

Professor: Dr. Celso Massaki Hirata

Relatório da Realização do Lab01 “Exclusão Mútua pelo Algoritmo Ping-Pong”

Lucas Kriesel Sperotto

22 de Setembro de 2011

Objetivo:

Este trabalho tem como objetivo descrever as atividades desenvolvidas na elaboração do Lab01, abordando os passos para a implementação da aplicação JAVA, bem como as interfaces e interações com o usuário, a conclusão e a implementação do algoritmo "ping-pong" de Misra (1983) para a obtenção da exclusão mútua com regeneração de "token".

Introdução:

Este relatório esta dividido em quatro tópicos fundamentais: Introdução, Procedimentos, Conclusão e Referencias. Sendo que os procedimentos estão divididos em quatro subgrupos: o primeiro com a descrição dos passos para-se executar o aplicativo, o segundo a descrição das interfaces gráficas e suas funcionalidades, o terceiro com a descrição da implementação e por ultimo a explicação sobre o comportamento na execução e os testes realizados. Também temos um arquivo anexado com os executáveis, fontes, projetos e o "javadoc" gerado para as classes.

Ainda na introdução gostaria de expor algumas informações importantes como a exposição do algoritmo de Misra (1983) e a interpretação do se funcionamento:

```
t = PortRef.recivPingPong();//recebe ping ou pong
if (t > 0) { //se for maior é ping
    pinghere = true;
    nbping = t;
    if (ponghere) {// se eles se encontraram
        nbping = (nbping % 4) + 1;
        nbpong = (nbpong % 4) - 1;
    } else {
        if (token != nbping) {//é diferente do ultimo
            token = nbping;    //     que passou...
        } else {//se são o pong se perdeu
            nbping = (nbping % 4) + 1;
            nbpong = -(nbping);
            ponghere = true;//regenerou
        }
    }
} else {//se não é pong
    ponghere = true;
    nbpong = t;
    if (pinghere) {//eles se encontraram
        nbping = (nbping % 4) + 1;
        nbpong = (nbpong % 4) - 1;
    } else {
        if (token != nbpong) {//é diferente do ultimo
            token = nbpong;   //     que passou...
        } else {//se são o ping se perdeu
            nbpong = (nbpong % 4) + 1;
            nbping = -(nbpong);
            pinghere = true;//regenerou
        }
    }
}
if (pinghere) {//envia o ping
    token = nbping;
    PortRef.sendPingPong(nbping);
    pinghere = false;
}
if (ponghere) {//envia o pong
    token = nbpong;
    PortRef.sendPingPong(nbpong);
    ponghere = false;
}
```

O algoritmo de Misra (1983) propõe uma modificação do algoritmo de "Token-Ring", visando através de dois tokens (ping e pong) reestabelecer a perda de algum deles na comunicação.

No quadro ao lado, podemos ver o algoritmo "ping-pong" completo (de acordo com minha interpretação). Inicialmente temos as variáveis "nbping" e "nbpong" iniciadas com 1 e -1 respectivamente, a variável "token" com 0, "ponghere" e "pinghere" com "false". A variável "token" guarda o valor do ultimo token (ping ou pong) que passou pelo cliente, de forma reconhecer se algum deles se perdeu na comunicação.

Quando uma mensagem chega, o primeiro "if" compara para saber se é o "ping" ou o "pong", no caso de ser o

“ping”, ele seta “true” na variável “pinghere” (para indicar que o ping esta presente). No segundo instante ele verifica se o “pong” também esta presente (sinal de que eles se encontraram) e incrementam os valores de “nbping” e “nbpong”, esse tratamento no incremento é citado por Misra (1983) para que os valores das variáveis não venham a crescer demasiadamente e explodam a capacidade de um inteiro (já os tokens não irão se encontrar mais de n vezes).

Quando o “pong” não se encontra ele verifica se o “ping” que chegou é diferente do ultimo “token” que passou pelo processo (comparando com a variável token), no caso de ser igual é sinal que o “pong” se perdeu no caminho e necessita ser recuperado. Logo depois de recuperado a variável seta que o pong se encontra ali e o algoritmo faz o envio do ping e depois do pong. Notamos que o algoritmo regenera a perda de um dos tokens após uma volta completa

Para o caso do recebimento do pong é análogo.

Procedimento:

Instalação e Execução:

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 1.

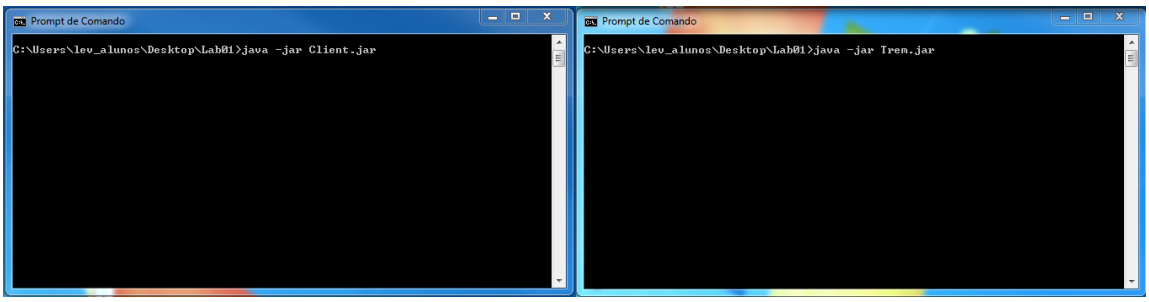

Figura 1 - Detalhe Comando de Execução

Para a execução, devem-se abrir quatro “prompts” de comando, executar três clientes e um trem. Quando se executa um cliente irá abrir uma janela (detalhe figura 2) requisitando que se insira o “Id” do cliente (0, 1 ou 2). Para a correta inicialização do anel, deve-se iniciar o cliente com “Id” 0 por ultimo (ele que envia o primeiro “ping-pong”). Confirmando no botão “OK”, o programa dará inicio a execução do anel e o usuário poderá requisitar o controle da “String”.

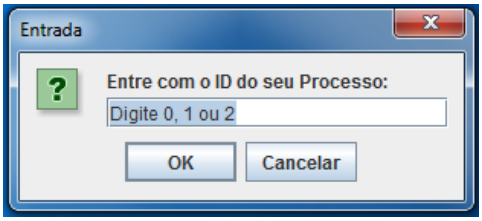

Figura 2 - Detalhe da Caixa de Dialogo do Cliente

Interface e Interação com Usuário:

Foram criadas as interfaces gráficas de acordo com o especificado nas instruções do Lab1. Sendo assim para a interface do processo Trem, temos um "JFrame" com uma "String" que ao receber o comando de um dos clientes se movimenta circularmente na tela.

Adicionei uma "JTextArea" para mostrar ao usuário o que esta acontecendo. Por exemplo, quais comandos foram recebidos, quem requisitou o serviço etc... (detalhe dessa interface pode ser visto na figura 3).

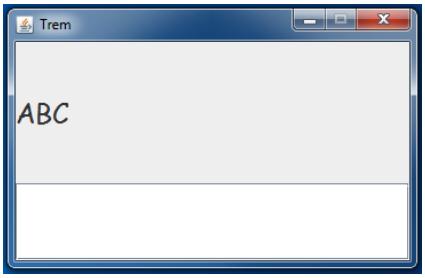

Figura 3 - Detalhe da Interface do Processo Trem

As mensagens exibidas 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 4) possuem quatro botões com suas respectivas funcionalidades:

- **Iniciar Controle:** Este botão requisita a exclusão mútua, suas ações são as seguintes: avisa a thread de comunicação para segurar o "ping" e o "pong" (seta que esta na seção critica) 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, avisa a thread de comunicação que pode voltar

a enviar o “ping” e o “pong” 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.

Temos também uma área de texto para informar o usuário sobre quem esta controlando o serviço e outras informações relevantes. As mensagens que a TextArea 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 Mutua....”; 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.

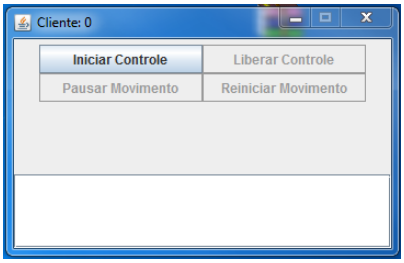

Figura 4 - Detalhe da Interface Cliente

Arquitetura e Codificação:

Na figura 5 podemos ver o detalhe das conexões entre os processos, temos um anel unidirecional ligando os três clientes. Já o processo “Trem” possui conexão bidirecional com todos os clientes. Confesso que tive duvidas quanto a essa arquitetura, já que o processo Trem estando ligado a todos poderia controlar a exclusão, e mesmo essa conexão poderia estar exercendo alguns controles nos clientes que fogem ao objetivo do trabalho.

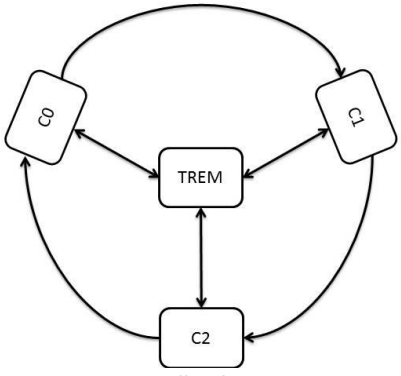

Figura 5 - Detalhe da Arquitetura.

Todas as comunicações entre os processos foi utilizando DatagramaSocket. Utilizei como referência para a implementação o livro do Deitel (2005). Dada minha abstração do problema elaborei para o cliente o diagrama de classes da figura 6, e para o Trem o diagrama da figura 7.

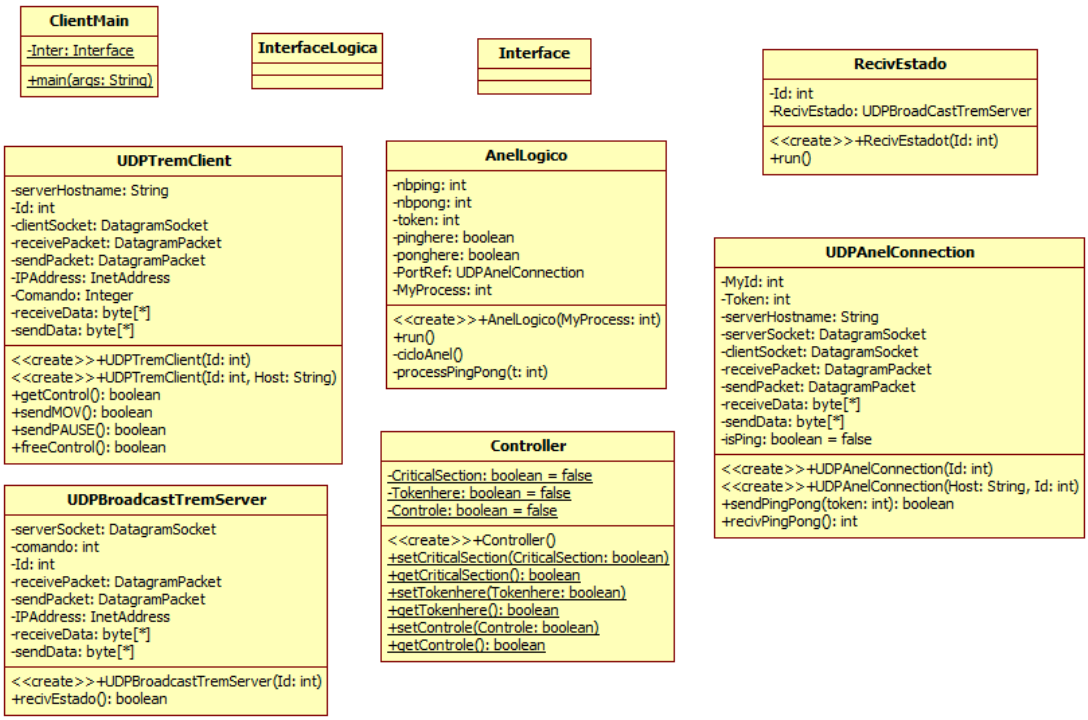

Figura 6 - Diagrama de Classes do Aplicativo Cliente.

A classe “Interface” é a classe onde esta definida a interface com o usuário. A classe “InterfaeLogica” é uma classe privada dentro da classe interface que trata da logica associada a interface (botões, ações etc...). A classe “ClientMain” é a classe principal, onde é instanciada a interface e iniciada a Thread “RecivEstado”.

A classe “RecivEstado” 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” é apenas um repositório para variáveis estáticas acessadas por métodos “syncronized”, são informações que necessitam ser vistas por mais de uma thread (exemplo “Interface” e “AnelLogico”).

A classe “AnelLogico” é onde o algoritmo de Misra (1983) esta implementado. Essa classe é uma thread que instancia a classe “UDPAnelConnection” (classe responsável pelas conexões do anel) e recebe o ping ou pong, processa através do algoritmo e envia para o próximo cliente. As comunicações no anel lógico não possuem nenhuma segurança de que um pacote foi entregue, inicialmente eu implementei uma confirmação para o recebimento do pacote. Mas tornava a regeneração do token desnecessária já que eu garantia as comunicações.

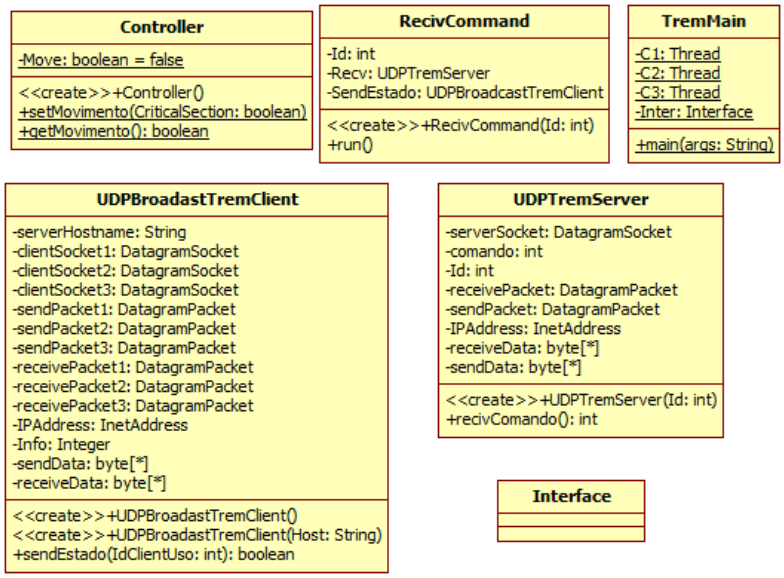

Figura 7 - Diagrama de Classes do Aplicativo Trem.

No aplicativo "Trem", a classe Interface descreve a interface com o usuário, e dentro dela tem uma Thread responsável pela animação da "String". A classe "Controller" tem a mesma função que no cliente.

A classe "RecivCommand" 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.

Na classe "TremMain" (classe principal) temos a instanciação da interface e de três Threads do tipo "RecivCommand" (uma para ouvir cada cliente como solicitado no Lab01). 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.

Testes e Funcionamento:

Tendo executado as instâncias do aplicativo como descrito no primeiro tópico, os clientes irão exibir suas telas e iniciar as conexões do anel lógico, exibindo no "prompt" de comando as mensagens da figura 8.

Nessa figura as mensagens são impressas muito rapidamente dado à velocidade de envio e recebimento das informações. Mas pode-se observar o envio do Ping e do Pong em momentos distintos.

Cada vez que o cliente recebe uma mensagem (ping ou pong) a função de processamento é chamada e o algoritmo processa a perda de uma das mensagens e regenera o "token" perdido.

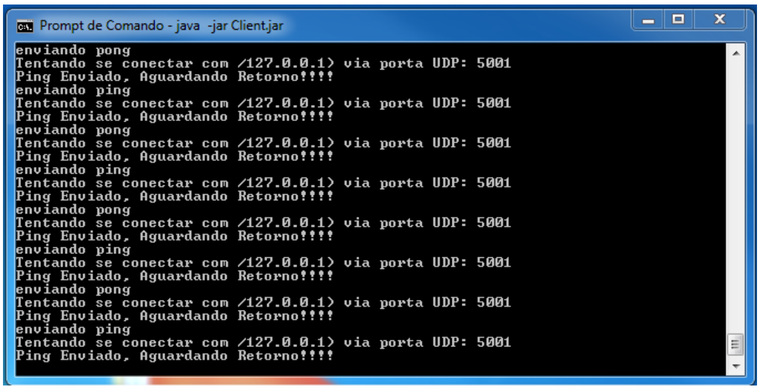

Figura 9 - Saída no prompt de Execução do Cliente.

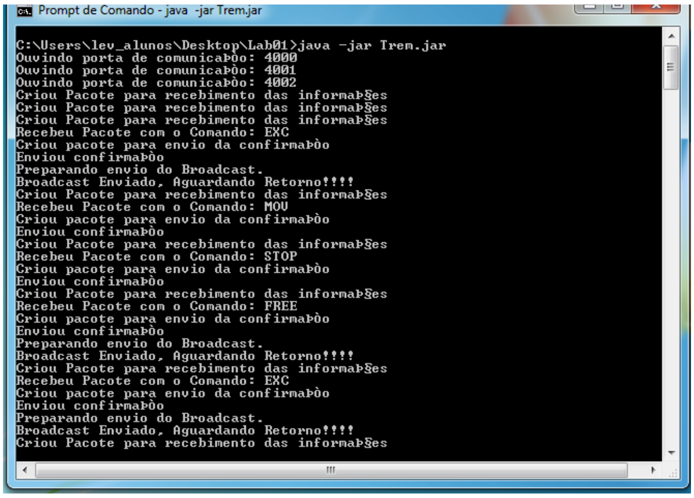

Figura 8 - Saída no prompt de execução do Trem.

Já no processo trem podemos ver no prompt as ações que estão acontecendo em cada uma das threads. E cada etapa das comunicações, como mostrado na figura 8. Na sequencia de figuras que seguem temos um exemplo da execução dos aplicativos.

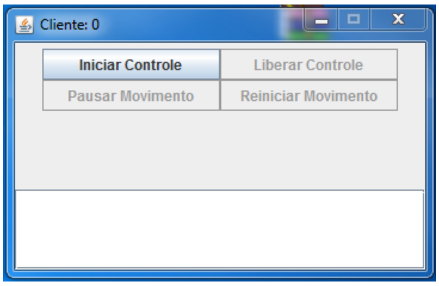

Figura 11 - Estado Inicial do Cliente.

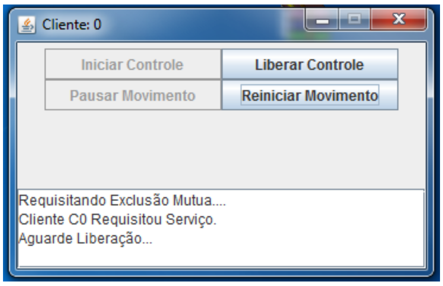

Figura 10 - Iniciando Controle.

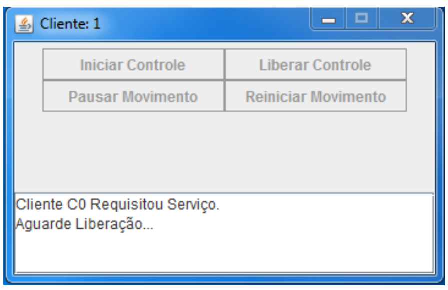

Figura 13 - Cliente Bloqueado.

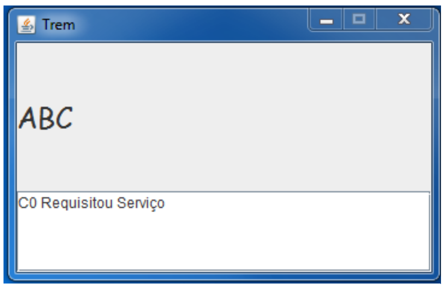

Figura 12 - Requisição no Trem.

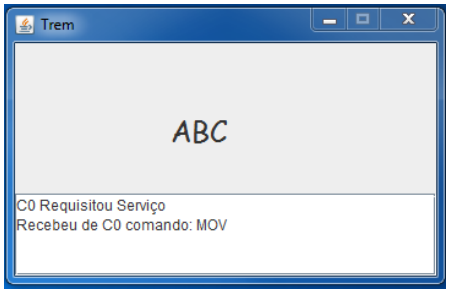

Figura 15 - Recebimento de Comando MOV.

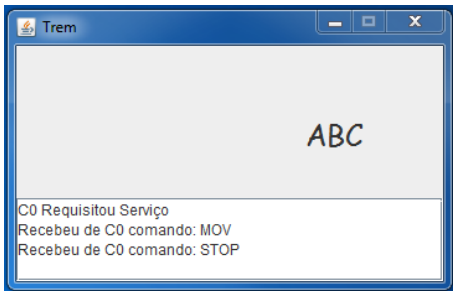

Figura 14 - Recebimento de Comando STOP.

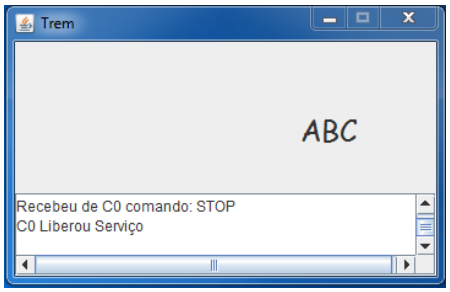

Figura 17 - Liberação de Serviço.

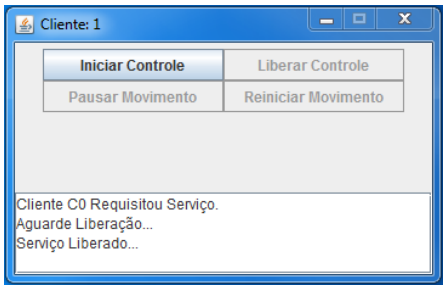

Figura 16 - Liberação de Serviço.

Temos na figura 11, mostramos o estado inicial de um cliente, e na figura 10 quando o usuário clica no botão “Iniciar Controle”, o cliente envia uma mensagem para o processo Trem (figura 12) que envia um retorno com o estado da String e um broadcast para os demais clientes sobre quem requisitou o serviço (figura 13).

Nas figuras 14,15 e 17 temos 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 16 temos o estado dos demais clientes quando o controle é liberado.

O teste efetuado para testar a regeneração de “token”, foi iniciar o anel apenas com o “ping” ou o “ping”. Dessa forma se observou que em uma volta completa do anel o “token” ausente se tornou presente (foi regenerado).

Conclusão:

Gostei da atividade, e posso dizer que aprendi ferramentas Java que ainda não tinha conhecimento. Outro ponto importante é que ver o algoritmo é muito diferente de implementa-lo, para a implementação você deve ter um entendimento correto do funcionamento do mesmo, e claro aprende mesmo é implementando, fazendo “teste de mesa”, verificando possíveis erros.

Tive um pouco de dificuldades para fazer funcionar o algoritmo, por isso primeiro implementei por “token-ring”, que possui um funcionamento bem simples. Tive de procurar e ler o artigo do autor do algoritmo para buscar uma melhor explicação do seu funcionamento.

Realmente gostei de ver o algoritmo em funcionamento. Acredito que a complexidade em implementar esse algoritmo agregou bastante conhecimento.

Referências:

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

LE LANN, G. **Distributed Systems: Towards a formal approach**. IFIP Congress, Toronto, 1977.

MSRA, J. **Detecting Termination of Distributed Computations Using Markers**. In Proceedings of the ACM Symposium on Principles of Distributed Computing, pages 290–294, 1983.