**ROTEIRO VÍDEO**

**Disciplina:** Sistemas Distribuídos

**Professor:** Aurélio Faustino Hoppe

**Equipe:** Alan Felipe Jantz, Caroline Belli Regalin e Matheus Manhke

**Algoritmo de Eleição – Anel Lógico**

* Em geral sistemas distribuídos requerem um coordenador, ao falhar o coordenador, o grupo de processos precisa eleger um novo coordenador.
* O algoritmo de anel serve para eleger um novo líder se os processos tiverem em um anel.
* Cada processo deve conhecer seu vizinho a direita e à esquerda e deve ter um identificador único atribuído antes do processo de eleição.
* Quando algum processo faz uma requisição ao coordenador e nota que ele não está mais ativo, esse algoritmo busca eleger o processo de maior identificador para ser o coordenador e fazer com que todos os membros do anel reconheçam o novo líder.

**Especificação**

* a cada 30 segundos um novo processo deve ser criado.
* a cada 25 segundos um processo deve fazer uma requisição para o coordenador.
* a cada 100 segundos o coordenador fica inativo.
* a cada 80 segundos um processo da lista de processos fica inativo.
* dois processos não podem ter o mesmo ID.
* dois processos de eleição não podem acontecer simultaneamente;

**Solução: Interfaces, Classes e Métodos implementados**

As interfaces definidas na implementação do trabalho são, IRing e IProcedure, representando a assinatura dos métodos do anel logico e do processo.

A classe **Procedure**, representa um processo, e ela possui dois atributos: Indentifier, que é o identificador único do processo e Manager, identificando se esse processo é o coordenador ou não.

Os principais métodos:

SendRequest()

Percorre toda a lista de processos ativos, buscando o coordenador para fazer uma requisição, caso a lista de processos não tenha um coordenador, ou seja, o coordenador está inativo, começa uma nova eleição para definir um coordenador.

BeginElection(long indetifier)

Inicia uma nova eleição, buscando o processo com o maior identificador e atualizando ele como o novo coordenador.

ReceiveRequest(long identifier)

Simula o recebimento de uma requisição pelo processo.

UpdateManager(IProcedure newManager)

Atualiza o novo coordenador e atualiza o atributo Manager dos outros processos, para false, garantindo que não vai haver outro processo como coordenador.

Na classe **Ring**, utilizamos threads para executar as especificações do trabslho, temos a propriedade \_activeProcedures, que é uma lista com todas os processos ativos.

Ainda temos as constantes ADD, REQUEST, INACTIVATE\_MANAGER e INACTIVATE\_PROCEDURE, contendo o intervalo de tempo especificado no trabalho para a execução de cada thread.

Métodos:

CreateProcedures()

Através de uma Thread, cria um processo do tipo Procedure, para obter o atributo Identifier, busca um número randômico entre 1000 e 9999, caso o número sorteado já exista na lista de processos ativos, segue incrementando 10 ao identificador até conseguir achar um número que ainda não foi usado, garantindo assim o identificador único para cada processo, e dorme por 30 segundos, para criar o próximo processo, assim executa infinitamente.

ExecuteRequest()

A cada 25 segundos, seleciona um processo aleatório da lista de processos ativos e envia uma requisição ao coordenador.

InactivateManager()

A cada 100 segundos, procura o coordenador na lista de processos e o inativa.

InactivateProcedure()

A cada 80 segundos, sorteia um processo aleatório na lista de processos ativos e o inativa.

Utilizamos o lock em todos os principais métodos que utilizam as threads, ele obtém o bloqueio de exclusão mútua para um determinado objeto, executa um bloco de instruções e, em seguida, libera o bloqueio. Embora um bloqueio seja mantido, o thread que mantém o bloqueio pode adquiri-lo novamente e liberá-lo. Qualquer outro thread é impedido de adquirir o bloqueio e aguarda até que ele seja liberado. Garantindo o requisito da especificação do trabalho em que dois processos de eleição não podem ocorrer simultaneamente.