**Notas de Aula:**

**Prova Final – cap III ao cap VI**

**Tópicos de Estudo:**

•**Processos x Threads**

Processo: abstração de um programa em execução e cuida do gerenciamento de recursos.

Thread: tarefas de um mesmo processo que executam concorrentemente e compartilham recursos do mesmo processo, é uma entidade própria. Alem disso, um unico processo pode ter diversas threads, ou seja, pode executar diversas atividades simultaneamente (ou concorrentemente).

**• O que é transparência?**

Transparência é uma característica do sistema distribuído que permite ocultar dos usuários os detalhes da distribuição do sistema.

São 4 tipos:

Acesso – Ocultar as diferenças na forma de acesso ao sistema.

Localização – Ocultar o lugar de instalação de um recurso.

Replicação – Ocultar o fato de um recurso estar replicado.

Persistência – Ocultar o fato de um recurso estar armazenado em meio volátil ou permanente.

**•O que é condição de corrida? quando ela pode dar problema?**

Condição que ocorre quando o resultado de uma computação depende do sincronismo das instruções de múltiplas threads que estejam executando a computação. O erro depende do sincronismo das duas threads e pode se manifestar muito raramente.

**• O que é exclusão mútua e como conseguir ela com semáforo e monitor ?**

Exclusão mútua é o ato de assegurar que duas ou mais threads não se encontrem executando, simultaneamente, as respectivas regiões críticas.

Sincronização condicional= um processo atrasa sua execução até que uma condição torne-se verdadeira

Semáforo controla o número de threads que podem acessar um recurso ou executar uma ação simultaneamente.

Monitores são primitivas de sincronização de nível mais alto que os semáforos. São basicamente constituídos por uma estrutura que encapsula os dados compartilhados pelas diversas threads.

•**Defina o que é espera ocupada, como ela é usada e qual é o principal problema dela.**

Reposta: Fato de desperdiçar ciclos do processador.

Principal problema: Pode fazer com que um outro thread que tenha que ter acesso não entre na sessão critica.

•**o que é atomicidade?**

Duas operações A e B são atômicas uma em relação à outra se, do ponto de vista de uma thread executando A, quando outra thread executa B, ou toda a operação de B foi executada e nenhuma parte de A foi executada.

ou

Definição Operação Atômica: Uma operação é atômica se é atômica em relação a todas as outras que acessam o mesmo estado (inclusive ela mesma). (Um thread nunca pode enxergar outra thread no meio da execução de sua região critica.) Exclusão mutua

• **Diferença entre Thread.interrupt() e Thread.stop()**

Seta um flag dentro do thread de interrupção se o isInterrupted é chamado depois o flag muda de valor Thread.Stop força a thread a parar sua execução A diferença é que o primeiro interrompe a thread e o segundo para a thread

ou

Thread.interrupt() Thread.stop() interrompe bruscamente a execução da Thread.

•**Vantagens e desvantagens dos sistemas paralelos e distribuídos.**

Resposta:

**Vantagens Distribuídos:** Escalabilidade, modularidade, confiabilidade e baixo custo

Os diversos nodos (máquinas hospedeiras de processos) podem ser de tipos completamente diferentes, cada um com sua própria estrutura e arquitetura e com seu próprio sistema operacional, permitindo dessa forma mais independência e não ficar restrito a uma máquina, arquitetura ou sistema operacional

**Desvantagem do Distribuído:** Utilização de Troca de mensagens pela rede o que demanda controle e esforço para os processos estarem sincronizados

Ausência de Memória Compartilhada Ausência de Relógio Global

**Vantagens Paralelos:** Eficiência na atualização de dados compartilhados

Eficiência na comunicação e distribuição de alterações nos dados compartilhados.

Utilização de Memória compartilhada

**Desvantagem Paralelos:** Caso a máquina hospedeira do processo para de funcionar, o processamento é interrompido

• **Vantagens e desvantagens de mensagens síncronas e assíncronas em aplicações distribuídas.**

Resposta: Sincronismo (necessita de relógio global, não se pode obter sincronismo em sistemas dispersos)

Assincronismo (não existe tempo de transmissão de mensagens, impossibilita diferenciar atrasos de falhas)

\*TIME-STAMP. Todas as mensagens são marcadas.

**•Defina Deadlock e de um exemplo de onde ocorre.**

Deadlock: Condição de bloqueio em que duas ou mais entidades esperam pela liberação de recursos que estão de posse das demais, em uma cadeia circular.

Exemplo: Jantar dos Filosofos. Acontece o deadlock se todos os filosofos adquirem simultaneamente o garfo da direita e assim todos ficam bloqueados esperando a liberação do garfo da esquerda que esta de posse do filosofo da esquerda, ou seja, ninguém ira mais sair dessa situação de bloqueio. Isso pode ser evitado fazendo com que um dos filosofos adquira um garfo na ordem inversa ou fazendo com que os filosofos peguem os 2 garfos simultaneamente(ATOMICAMENTE).

**•Definir e justificar qual dos métodos estudados você usaria para implementar um sistema de comunicação de fichas de pacientes em um hospital.**

Partindo do principio que a rede em que irão trafegar estas informações não é confiável, utilizaria o protocolo TCP, pois este é um protocolo confiável e orientado a conexão. Isto significa que, para trocar mensagens, os dois extremos da comunicação devem estabelecer uma conexão entre eles; isto garante que todas as mensagens serão entregues, e na mesma ordem em que foram enviadas.

Outra diferença em relação ao protocolo UDP é que os dados das mensagens aparecem como um fluxo de bytes contínuo, sem pacotes de tamanho fixo.

**• Por que as operações P () e V () do semáforo precisam ser atômicas**

Duas operações atômicas P () (obtém acesso a um recurso) e V () (libera acesso). Se não for atômico P () permitirá a thread entrar na seção crítica e decrementará o contador e V () ira incrementá-lo novamente permitindo outro thread acessar a cessão critica.

• **Como fazer exclusão mutua e sincronização condicional com monitores**

Apenas um thread pode estar executando qualquer método de entrada em determinado momento. Método wait() faz uma thread bloquear na fila da variável de condição. Método notify() retira uma das threads da fila. wait() e notify() devem ser executados por threads ativas no monitor.

•**Com o algoritmo dos jantares dos filosofos, indicar o porquê ocorre deadlock**

Condição de bloqueio em que duas ou mais entidades esperam pela liberação de recursos que estão de posse das demais, em uma cadeia circular. A solução mostrada apresenta um sério problema: suponha que todos os filósofos adquirem simultaneamente a posse do garfo à sua esquerda e bloqueiam esperando pelo garfo à direita (que está de posse do filósofo à direita!). Esta condição de espera circular é conhecida como deadlock e é um dos principais perigos da programação concorrente quando diversas threads competem pelo acesso a múltiplos recursos. Neste exemplo, o deadlock é causado pela simetria da solução apresentada: todos os filósofos adquirem os recursos na mesma ordem (garfo direito após o garfo esquerdo). Isto pode ser evitado fazendo com que um deles adquira os garfos em ordem inversa (direita/esquerda); a implementação é deixada como exercício. Outra estratégia possível é fazer com que um filósofo sempre adquira ambos os garfos simultaneamente. Uma estratégia frequentemente adotada para evitar deadlocks é exigir que locks para todos os recursos compartilhados sejam obtidos obedecendo sempre uma determinada ordem. Deadlocks serão tratados com profundidade em aulas futuras.

**Resuminho**

**Syncronized:** Esse modificador faz com que um objeto que está sendo acessado só possa ser acessado novamente quando a tarefa que está sendo realizada sobre ele seja concluida.

Processos assíncronos se comunicam pela da troca de mensagens através da comunicação entre os processos

**Exclusão Mútua** Evitam que um recurso em comum seja utilizado simultaneamente, como variáveis globais, por pedaços de código chamados seções críticas.

Exemplos destes recursos são flags, contadores e queues(filas), usados para comunicação entre códigos que são executados concorrentes, como uma aplicação e seus tratadores de interrupção tcps clássicos permitem deadlocks, no qual um processo recebe um semáforo, outro processo recebe o outro e ambos permanecem no estado de bloqueado esperando que o outro semáforo seja liberado.

**Efeitos colaterais:** Starvation no qual um processo nunca consegue obter recursos necessários para sua execução e inversão de prioridade no qual um thread de maior prioridade espera por uma de menor prioridade.

Um **semáforo** é uma estrutura de dados que consiste em um número inteiro e em uma fila que armazena descritores de tarefas.

O conceito de semáforos consiste na colocação de proteções em torno do código que acessa esta estrutura para oferecer acesso limitado aos dados.

Em geral, a estrutura de dados é uma fila, funcionando em regime de primeiro a entrar, primeiro a sair.

O semáforo tem uma variável especial protegida (ou tipo abstrato de dados) que tem como função o controle de acesso a recursos compartilhados (por exemplo, um espaço de armazenamento) num ambiente multitarefa.

Há um problema em semáforos, o deadlock.

Este ocorre quando um processo altera o valor do semáforo e entra na região crítica, mas ocorre uma falha na saída da região porque ele foi danificado ou morto.

Sincronização: Objetivo é manter a consistência dos dados. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**•Monitor Contador**

class However {

int contador;

Public However(contador) {

this.contador = contador;

}

syncronized P() {

while (this.contador <= 0)

this.wait();

this.contador--;

}

syncronized V() {

this.contador++; this.notify();

}

}

•**Qual a diferenca entre UDP e TCP**

UDP User Datagram Protocol |

Características: - Simples e eficiente. –

Não fornece garantias de entrega e ordenação de mensagens.

- Não utiliza conexões.

- Trabalha com datagramas (pacotes de tamanho conhecido).

- Soquete Java: DatagramSocket.

TCP Transmission Control Protocol |

características: \_ Orientado a conexão.

\_ Oferece garantia de entrega e ordenação.

\_ Trabalha com fluxos de bytes (sem início e fim definidos).

\_ Mais confiável e menos eficiente.

\_ Soquetes Java: Socket e ServerSocket

**•Como que um cliente usa o RMI e como é implementada.**

Rafael: Forma de invocar metodos de objetos remotos como se estes estivessem na mesma maquina. O Objetivo do RMI é tornar os principios de programação orientada a objetos disponiveis para o desenvolvimento de sistemas distribuidos. RMI pode ser encarada tambem como uma forma de alto nivel para enviar e receber mensagens.

Os passos para implementar uma RMI são:

1-Definir a interface remota

2-Implementar o serviço

3-Desenvolver o servidor

4-Desenvolver o cliente

5-Iniciar o RMI registry e o servidor

6-Rodar o cliente \_\_\_

• **Lista de todas as diferenças entre uma chamada de método local e uma invocação de método remoto.**

A latência de uma chamada local é bastante menor que de uma chamada remota

Numa chamada local, o método é executado apenas uma vez. As chamadas remotas têm um tempo de resposta maior do que as chamadas locais passagem de parâmetro em chamadas locais: tipo primitivo=VALOR; outro tipo(objeto)=referência passagem de parâmetro em chamadas remotas: tipo primitivo=VALOR; tipo não remoto(serializado)= referencia; tipo remoto(stub é serializado)=referência \*\*\*\*\* (verificar resposta)

As chamadas remotas são mais vulneráveis e possuem falhas mais complexas Invocação remota transparência de localização

As invocações de métodos remotos são iguais as invocações aos métodos locais, porém as diferenças entre os objetos remotos e locais devem estar expressas nas suas interfaces

•**Dê vantagens e desvantagens do uso de RMI sobre sockets TCP para o desenvolvimento de aplicações distribuídas.**

Mudanças feitas no objeto do lado do servidor não são refletidas do lado do cliente.

Faz com que os objetos continuem em execução contínua a espera de chamadas de clientes remotos. Confiabilidade Ordenação de entrega das mensagens TCP é necessário abrir e fechar socketes no RMI apenas chamada ao método remoto

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

• **Papel da INTERFACE e como exatamente é implementada.**

O papel da interface é ser um contrato, contém apenas a declaração dos métodos. Possui apenas o cabeçalho (assinatura) dos métodos sem nenhuma implementação. Deve ser pública, estender Java.rmi.remote. e os métodos lançarem remote.exception.

**• Principal característica de MESSAGERIA e vantagem.**

Comunicação realizada de forma assíncrona (o remetente da mensagem não fica bloqueado e não terá necessidade de ficar aguardando a resposta), as mensagens podem ser mantidas no servidor sendo entregues assim que o servidor estiver disponível. Suporte à comunicação síncrona e assíncrona, suporte a transações no envio e recebimento de mensagens, mensagens armazenadas em fila, grau de tolerância é muito alto.

**• Como os monitores garantem exclusão mutua e sincronização.**

Sincronização condicional: cada variável x possui uma fila de threads bloqueadas e duas operações, podendo ser executadas por threads dentro do monitor x.wait(): thread executora é bloqueada e colocada na fila de x. x.notify(): se a fila não estiver vazia, uma das threads é escolhida e colocada em estado de pronta para processar.

• **O que é RMI REGISTRY e objetivos**

Base de dados que converte nomes lógicos para o endereço do objeto remoto, servidor registra e cliente consulta. Objetivos RMI: Fornecer transparência de localização, de modo que o cliente não precise conhecer a localização do servidor.

• V**antagem e desvantagem do RM**I –

Vantagens RMI. 1.) Transparência de localização

2.) Modelagem fácil e natural de sistemas distribuídos

3.) sintaxe da chamada de método local = sintaxe da chamada de método remoto

4.) Acesso a objetos remotos

5.) Protocolo cliente servidor

6.) API de alto nível

7.) Transparente

Desvantagens de RMI. 1.) Não há garantia da entrega dos pacotes

2.) Não há ordenação no envio dos pacotes

3.) Objeto remoto e servidor de nomes (Registry) devem estar na mesma máquina

4.) Comunicação síncrona, o cliente não pode fazer outra tarefa enquanto espera a resposta do servidor

5.) limitação para usuários de browser proxy server 2-

**•O que é serialização e como pode ser feita em RMI e messageria**

Serialização: conversão do objeto em bytes, sendo seus parâmetros passados como valor. -- \*\*duvidas nessa \*\*

Messageria: suponho que a serialização em messageria seria o conceito de tópicos, quando as mensagens são entregues numa fila e os assinantes recebem a cópia das mensagens, mesmo não estando disponíveis no momento. Para que seja possível serializar um objeto é necessário que a classe do objeto a serializar implemente a interface java.io.Serializable.

•**Situação hipotética: você está numa empresa com um servidor critico novo e um não tão crítico, velho, que para de vez em quando quase sempre. Melhor jeito de implementar um meio de comunicação entre eles (UDP, TCP, RMI, MESSAGERIA)?**

A comunicação a ser implementação nesta situação, seria por MESSAGERIA, porque o cliente e o servidor necessariamente não precisam estar disponíveis ao mesmo tempo (comunicação assíncrona), ou seja, com isso o cliente pode executando outras operações.

•**Ponto a ponto publisher/subscribe(o que é como funciona)**

Comunicação ponto a ponto: permite que um programa envie mensagens para uma fila específica que será lida por outro. A resposta de publisher/subscribe é o conceito de tópico, onde refere-se ao conceito de assinatura. Cada assinatura recebe uma cópia da mensagem, o que faz com que a mesma possa ser distribuída por todos os clientes interessados, distribuída através de broadcast. Implementa um tipo de comunicação publisher/subscriber. Wikipedia (publisher/subscriber): Este modelo suporta a publicação de mensagens para um determinado tópico de mensagens (message topic). O(s) "assinante(s)" (subscriber) podem registrar interesse em receber ("em assinar") mensagens de um tópico. Neste modelo, nem o "publicador" (publisher) ou o "assinante" sabem um do outro. Wikipedia (ponto-a-ponto): No modelo ponto a ponto, ou por filas, um "produtor" (producer) envia mensagens para uma fila e um "consumidor" (consumer) as lê. Neste caso, o produtor conhece o destino da mensagem e a envia diretamente para a fila do consumidor

•**Passagem de parâmetros (quais são os tipos)**

Tipos primitivos são passados por valor. - Objetos não remotos são passados por valor utilizando serialização.

- Objetos remotos são passados por referência (o stub correspondente é serializado e enviado).

**•Papel da INTERFACE e como exatamente é implementada**.

O papel da interface é ser um contrato, é esconder a implementação de um determinado serviço. Desta forma a implementação do serviço pode ser alterada inúmeras vezes sem preocupar com as aplicações clientes, contanto que mantenha a compatibilidade do serviço com a interface. O primeiro passo na criação de um serviço RMI é criar uma descrição desse Serviço de modo que os clientes possam saber como invocá-lo. Esta descrição baseia-se em um importante princípio: a definição do comportamento (em Java, a entidade para esta definição chama-se interface) e sua implementação devem ser feitas separadamente. Na interface temos apenas o cabeçalho dos métodos sem nenhum tipo de implementação, estabelecendo um contrato; já na classe que define (implementa) o serviço temos os métodos com suas implementações, obedecendo ao contrato estabelecido pela interface. No caso do RMI, deve-se criar uma interface descrevendo os métodos que podem ser chamados remotamente e esta interface deverá obedecer as seguintes restrições: INTERFACE REMOTA · Descreve os métodos para a chamada remota; · Descreve o comportamento do serviço; · Deve ser pública; · Obrigatoriamente deve estender Java.rmi.Remote.

Obrigatoriamente todos os métodos devem lançar Java.rmi.RemoteException INTERFACE tem que extender a interface REMOTE e todos os METODOS da interface do serviço devem lançar obrigatoriamente a exceção REMOTEEXCEPTION.

• **Principal característica de MESSAGERIA e vantagem.**

Conceito JMS (Java Message Service) é a API padrão do Java que permite a construção de componentes que criem, enviem, recebam e leiam mensagens, fornecendo uma forma de comunicação confiável, desacoplada e assíncrona. Características Comunicação realizada de forma assíncrona (ou seja o remetente da mensagem não terá necessidade de ficar aguardando a resposta daquela mensagem). Comunicação assíncrona (o remetente da mensagem não fica bloqueado esperando a resposta). Caso o destinatário esteja off-line, as mensagens podem ser mantidas no servidor de aplicação, garantindo a entrega assim que o destinatário estiver on-line. -Suporte à comunicação síncrona e assíncrona; -Suporte a transações no envio e recebimento de mensagens; - Mensagens armazenadas em fila; - Grau de tolerância é muito alto; · Fila (queue): é uma fila no sentido usual, que contém uma cópia de cada mensagem, que será consumida pelo primeiro cliente que a retirar da fila. · Implementa um tipo de comunicação ponto a ponto. Tópico (topic): trabalha com o conceito de assinatura. Cada assinatura recebe uma cópia da mensagem, o que faz com que a mesma possa ser distribuída por todos os clientes interessados. Implementa um tipo de comunicação publisher/subscriber. Desacoplamento - Comunicação Assincrona, permite que o servidor não necessita realizar a conexão com o cliente. Corpo - Transmite os dados, corpo da mensagem, cabeçalho da msg (String) Propriedade e Atributos - toda msg têm uma propriedade específica, como também um atributo. Fila - Comunicação ponto a ponto, onde há vários consumidores na fila Tópico - assinatura a msg são distribuídas via broadcast.

• **como os monitores garantem exclusão mutua e sincronização.**

Os métodos sincronizados implementam os métodos de entrada e fornecem exclusão mútua para acesso aos dados controlados pelo monitor. Apenas uma thread pode estar executando qualquer método de entrada em determinado momento.

Exclusão Mútua - Ato de assegurar que duas ou mais threads não se encontrem executando, simultaneamente, as respectivas regiões críticas.

Sincronização condicional - A thread deve aguardar até que uma condição se torne verdadeira para que ela possa prosseguir na região crítica. Sincronização –

Métodos de entrada são implementados com a palavra reservada “synchronized” que permite definir um bloco sincronizado da seguinte forma: synchronized (obj) f...g. É comum utilizar a referência this como objeto de sincronização. Método de entrada synchronized (obj). Variáveis de condição para sincronização condicional.

**• O que é RMI REGISTRY e objetivos Conceito**

- Base de dados que converte o nome lógico para o endereço do objeto remoto.

- Servidor – Registro o serviço ao RMI Registry.

- Cliente – Consulta o RMI Registry para encontrar um serviço.

Objetivos Fornecer transparência de localização, de modo que o cliente não precise conhecer a localização do servidor.

**•O que era e como implementar uma interface remota.**

Definição - Forma de invocar métodos de objetos remotos como se estes estivessem na mesma máquina.

O Objetivo do RMI é tornar os princípios de programação orientada a objetos disponíveis para o desenvolvimento de sistemas distribuídos. RMI pode ser encarada também como uma forma de alto nivel para enviar e receber mensagens.

Os passos para implementar uma RMI são: 1-Definir a interface remota 2-Implementar o serviço 3-Desenvolver o serviço 4-Desenvolver o cliente 5-Iniciar o RMI registry e o servidor

6-Rodar o cliente

**•Você iria implementar uma aplicação que envia informações para seu cliente sobre atividades esportivas, pedia que você explicasse os métodos e protocolos que você usaria**. RESPOSTA 1(MARCINHO): Partindo do principio que a rede em que irão trafegar estas informações não é confiável, utilizaria o protocolo TCP, pois este é um protocolo confiável e orientado a conexão. Isto significa que, para trocar mensagens, os dois extremos da comunicação devem estabelecer uma conexão entre eles; isto garante que todas as mensagens serão entregues, e na mesma ordem em que foram enviadas. Outra diferença em relação ao protocolo UDP é que os dados das mensagens aparecem como um fluxo de bytes contínuo, sem pacotes de tamanho fixo.

RESPOSTA 2: **3) Deu um caso de uma ponte onde só pode passar pessoas em uma direção, não pode passar nos dois sentidos simultâneamente, pediu para explicar como implementaria essa aplicação**.

É o mesmo caso do Leitor-Escritor. Em que: • Várias pessoas podem ler simultaneamente • Não pode ter conflitos do tipo escrita-escrita • e não pode ter leitura-escrita Então a partir destes conceitos temos: • Várias pessoas podem passar por um mesmo lado simultâneamente. • Duas pessoas não podem passar por lados opostos ao mesmo tempo. A implementação seria utilizando um semáforo, para controle do acesso à ponte, que é um meio compartilhado. Cada lado da ponte seria demarcada por um nome. No primeiro ponto A e no segundo ponto B. Quando um carro que encontra-se no ponto A desejasse passar, ele iria chamar o método "iniciaPassagemPontoA", e quando o mesmo quisesse sair, iria chamar o método "finalizarPassagemPontoA". A mesma coisa ocorre quando um carro encontra-se no ponto B, que iria chamar os métodos de entrada e saída "iniciaPassagemPontoB" e "finalizarPassagemPontoB" subsequentemente. Seria adicionado dois semáforos binários, o semaforoControlePonte e o semaforoControleVariaveis, que fará o controle de acesso a ponte e o controle das variáveis. Também será adicionado as seguintes variáveis: numAcessoA, numAcessoB. Para que seja possível fazer a contagem da quantidade de pessoas que desejam acessar de cada lado, e em seguinda dar a possibilidade de fazer o balanceamento.

O Método iniciaPassagemPontoA ficaria assim:

public void iniciaPassagemPontoA() throws InterruptedException { semaforoControleVariaveis.acquire();

numAcessoA++;

if (numAcessoA== 1)

semaforoControlePonte.acquire();

semaforoControleVariaveis.release();

}

O Método iniciaPassagemPontoB ficaria assim:

public void iniciaPassagemPontoB() throws InterruptedException { semaforoControleVariaveis.acquire();

numAcessoB++;

if (numAcessoB== 1)

semaforoControlePonte.acquire();

semaforoControleVariaveis.release();

}

O Método finalizarPassagemPontoA ficaria assim:

public void finalizarPassagemPontoA () throws InterruptedException { semaforoControleVariaveis.acquire();

numAcessoA--;

if (numAcessoA== 0) s

emaforoControlePonte.release();

semaforoControleVariaveis.release();

}

O Método finalizarPassagemPontoB ficaria assim: public void finalizarPassagemPontoB() throws InterruptedException {

semaforoControleVariaveis.acquire();

numAcessoB--;

if (numAcessoB== 0)

semaforoControlePonte.release();

semaforoControleVariaveis.release();

}

**• Explique o deadlock no algoritmo do jantar dos filósofos com semáforo.**

Deadlock é a condição em que duas ou mais entidades esperam pela liberação de recursos que estão de posse das demais, em uma cadeia circular. No jantar dos filosofos acontece o deadlock se todos os filosofos adquirem simultaneamente o garfo da direita por exemplo e assim todos ficam bloqueados esperando a liberação do garfo da esquerda que esta de posse do filosofo da esquerda, ou seja, ninguem ira mais sair dessa situação de bloqueio. Isso pode ser evitado fazendo com que um dos filosofos adquira um garfo na ordem inversa ou fazendo com que os filosofos peguem os 2 garfos simultaneamente(ATOMICAMENTE).

**Notas de aula**

Notificação são perdidas se não tiver ninguém esperando

Usar notify all ..

1 variável de condição, pode-se ter vários threads bloqueadas por motivos distintos, então quando notifico, não tenho certeza de qual thread sairá, por isso, notifica a todos, que assim que uma delas se encaixar, um thread sairá.

No caso do **problema produtor/consumidor** – usou-se notify, as condições se contrariam/excluem

Não pode ter buffer vazio e cheio ao mesmo tempo

Então quando bloquear no ‘deposito’ é porque todas as threads bloquearam pelo mesmo motivo – buffer cheio

Quando bloquear no ‘consumir’ é porque todas as threads bloquearam pelo mesmo motivo – buffer vazio.

**Problema dos Filosofos**

Obrigue que as threads capturem recursos sempre na mesma ordem (deadLock)

Essa regra ataca a condição de espera circular. Quando se obedece essa regra você evita que o círculo se fecha.

Condição de posse e espera – é quando se precisa de mais de um recurso, ou se pega todos, ou nenhum.

**-Jantar dos filósofos com monitores**

Metodo test- tenta progredir um filosofo de hungry para eating – que é a única transição com restrição.