1. Introdução

2. Concorrêmcia

2.1 Exclusão Mútua e Locking Recursivo

Em Java, existe um lock associado a cada objecto.

Uma thread que adquiriu um lock, através de synchronized, pode em seguida invocar métodos ou código synchronized relativamente ao mesmo objecto sem ficar bloqueada.

O lock conta quantas vezes foi adquirido, bloqueando outras threads até ser libertado o mesmo número de vezes. Isto pode ser designado de locking recursivo, pois permite métodos synchronized serem recursivos.

2.3 Objectos e Actividades

Objectos passivos → objectos e threads são considerados conceitos independentes;

Objectos activos → existe uma unificação entre objecto e thread. A concorrência é criada implicitamente das seguintes formas:

Instanciação assíncrona → O objecto criado fica a executar um body (que pode ser implícito ou explícito) que fica em ciclo à espera de pedidos;

Invocação assíncrona → o cliente prossegue concorrentemente e o resultado é obtido mais tarde, de diferentes modos:

one way invocations → não devolvem resultados; se necessário o servidor envia o resultado através de outra invocação;

objectos futuros → podem ser devolvidos por invocações assíncronas. O futuro é usado pelo cliente para obter o resultado. Os futuros podem ser implícitos ou explícitos.

2.4 Monitores / Objectos Quase-Concorrentes

O conceito de monitor permite a obtenção de objectos quase-concorrentes: que suportam várias invocações em curso, ainda que só uma no máximo esteja não bloqueada.

Ao contrário dos objectos atómicos, as invocações não são serializadas: ainda que bloqueadas, podem já ter executado parcialmente, estando à espera de um evento externo a elas.

2.5 Objectos Concorrentes

Um serviço poderá disponibilizar operações que poderão ser demoradas a executar (por exemplo input/output), ainda que não dependam umas

das outras.

Nestes casos deverá ser implementado como um servidor concorrente, ou seja como um objecto que permita verdadeira concorrência entre invocações a ser processadas.

2.6 Deadlock – Bounded Buffer

Considerando um buffer com tamanho máximo 1 e inicialmente vazio e também a chegada em simultâneo de dois consumidores e dois produtores.

O primeiro consumidor executa e fica bloqueado na variável de condição dado que o buffer está vazio, libertando o lock.

O segundo consumidor adquire o lock e executa, ficando também bloqueado na variável de condição, libertando o lock.

O primeiro produtor adquire o lock, executa e coloca o buffer cheio. Notifica o segundo consumidor e termina com sucesso.

O segundo produtor adquire o lock e fica bloqueado na variável de condição dado que o buffer está cheio.

O segundo consumidor readquire o lock e notifica o primeiro consumidor. Executa colocando o buffer vazio e liberta o lock. Termina com sucesso.

O primeiro consumidor adquire o lock e executa ficando bloqueado na variável de condição porque o buffer está vazio. Liberta o lock.

Nesta situação acontece o deadlock porque o primeiro consumidor e o segundo produtor estão bloqueados na variável de condição à espera de serem notificados, o que nunca irá acontecer porque dependem um do outro para que essa notificação ocorra.

3. Heteronegeidade

3.1 Heterogeneidade de Hardware

Diferente hardware pode significar:

diferente representação de dados: necessidade de mecanismos de serialização com representação neutra;

diferentes requisitos de alinhamento em memória;

diferente código máquina que dificulta migração de código:

implementação em cada máquina;

uso de máquinas virtuais.

3.2 Heterogeneidade de Linguagens

Resolução da heterogeneidade de linguagens:

utilização de tipos abstractos de dados (e.g. sequencias);

especificação de mapeamentos para cada linguagem;

geração (automática) de filtros de conversão.

3.3 Serialização

Conversão de estruturas de dados para uma representação plana, sem utilização de apontadores, para transmissão ou armazenamento.

Existem formatos normalizados de serialização;

Dividem-se em:

formatos binários :

São compactos mas sobretudo permitem uma serialização muito eficiente em termos de tempo de computação;

Utilizam representações standard para números de vírgula flutuante (e.g. IEEE).

Resolucao da questão little/big endian:

formato neutro (como network order); e.g. XDR do ONC RPC;

anotando o formato do emissor: o receptor apenas converte se diferente (receiver makes it right), mais eficiente; e.g. CDR.

Alinhamento no stream de serialização:

com alinhamento; e.g. CDR do CORBA GIOP ou XDR;

remoção de padding entre elementos, levando a melhor uso de largura de banda (e.g. java.io.DataOutputStream).

formatos baseados em texto :

Convertem os dados para a sua representação textual.

São menos compactos do que os binários.

O problema principal é o poder de computação requerido:

conversões nativo-texto e texto-nativo;

parsing que poderá ser custoso;

3.4 Filtros Básicos

Um objecto stream oferece primitivas para serializar tipos básicos: int, float, strings, etc…

Exemplo: java.io.DataOutputStream/java.io.DataInputStream.

3.5 Composição de Filtros

Um filtro é uma operação usada para ler/escrever no stream um tipo de dados.

Para tipos compostos podem ser usados filtros que espelham a composição do tipo.

3.5 Estruturas Recursivas

Para transmitir estruturas recursivas, como listas ou árvores, é necessário processar apontadores;

Tal é feito enviando um booleano, seguido da estrutura apontada caso o apontador não seja nulo;

Podemos ter uma operação para processar apontadores.

3.6 Polimorfismo: informação sobre tipos

Se sabemos os tipos dos objectos enviados não necessitamos de o codificar no stream: mais eficiente;

Para permitir polimorfismo de inclusão necessitamos de codificar o tipo do objecto enviado (e.g. com uma string);

Tal permite o receptor instanciar um objecto apropriado.

4. Resolução Exames

4.1 Exame 1ª Chamada : 08-01-2004

1. Descreva o papel das IDL (*interface definition language)* na programação de sistemas distríbuidos.

As IDL, linguagens de definição de interfaces, são utilizadas normalmente para especificar os serviços de um sistema distribuído. O IDL descreve operações, seus respectivos parâmetros e resultado, e os tipos de dados relevantes. Uma IDL pode ser independente das linguagens que são utilizadas no código cliente e servidor. São neste caso definidos mapeamentos da IDL em linguagens de programação passivas de serem usadas.

2. Descreva as razões porque a interface RPC de um sistema de ficheiros distribuído não possui normalmente as operações de *open* e *close.*

As razões porque a interface RPC de um sistema de ficheiros distribuído não possui normalmente as operações de *open* e *close* são as seguintes:

Os sistemas de ficheiros distribuídos têm vantagens em serem implementados com servidores stateless (sem estado conversacional). Um servidor deste tipo tem as seguintes características:

Cada pedido vindo de um cliente contém toda a informação para ser entendido;

O servidor responde a um pedido independentemente dos pedidos

anteriores;

O estado de sessão é mantido no cliente.

As vantagens do servidor stateless são as seguintes:

Não sendo necessário manter estado conversacional, são necessários menos recursos, é obtida maior escalabilidade e maior tolerância a faltas.

Se um cliente falha o servidor não tem que se preocupar.

Se o servidor falha pode ser reiniciado sem problemas e os clientes podem continuar a enviar pedidos.

A operação *open* pode ser implementada no cliente com invocações locais a **read_file_from_to(fich, from, count, buffer)**, contendo este método toda a informação para o pedido ser satisfeito no servidor. E deste modo a operação close torna-se desnecessária para o cliente.

3. Explique em que consiste o problema da *falsa partilha* no contexto de um sistema DSM baseado em páginas.

O problema da falsa partilha consiste no seguinte: elementos que logicamente (em termos da aplicação) não são partilhados, ficam em unidade partilhada (e.g. página) do DSM.

4. Descreva as vantagens de ter um servidor *multi-threaded* face a um servidor que atende os pedidos sequencialmente, no caso em que a máquina servidora é extremamente rápida (o poder de processamento não é limitativo).

Um servidor sequencial tem as seguintes características:

Um único processo atende um cliente de cada vez.

Só é viável para pedidos com atendimento rápido.
De qualquer modo, não é escalável com o número de pedidos, mesmo

que o poder de processamento seja muito elevado devido à latência introduzida pela rede;

Muito vulnerável a clientes lentos ou maliciosos podendo originar Denial of Service.

Um servidor multi-threaded tem as seguintes vantagens em relação a um servidor sequencial:

Atende vários clientes ao mesmo tempo, podendo ser atribuída uma thread por pedido ou por cliente;

O tempo de atendimento dos pedidos não é relevante, podendo ser variável.

Diminui o tempo de resposta aos pedidos dos clientes tirando partido da elevada rapidez de processamento da máquina;

É escalável com o número de pedidos;

Mais robusto em relação a clientes lentos e maliciosos, provavelmente apenas no caso de uma rajada de pedidos num curto espaço de tempo é que poderia originar Denial of Service;

Poderá permitir estado partilhado em memória, caso seja adoptado o modelo de thread por cliente.

5. Explique em que consiste o problema dos "monitores aninhados no contexto de programação concorrente em orientação aos objectos. Ilustre com um exemplo de código.

A combinação de monitores pode dar origem a *deadlock* senão forem tomadas precauções.

Um *bounded buffer* implementado descuidadamente via semáforos:

```
public class Semaforo {
    protected int v;


    public Semaforo(int i) { v = i; }
```

```
    public synchronized void up() {
        ++v;
        notify();
    }



    public synchronized void down() throws InterruptedException {
        while (v == 0) wait();
        --v;
    }
}


  class Buffer {
    protected Semaforo full, empty;
    ...
    public synchronized void put(Object O) {
        empty.down();
        ...
        full.up();
    }
    public synchronized Object get() {
        full.down();
        ...
        empty.up();
    }
  }
```

O código anterior dá origem a *deadlock* pois o *lock* de um buffer nao é libertado ao ser feito um wait() num semáforo, impedindo outras threads de entrarem nesse objecto buffer.

8. Em CORBA podem ser utilizados *Object Adapters* com políticas diversas, de acordo com o modelo de associação entre referências e servants pretendido. Descreva a utilidade de associar vários objectos CORBA a um mesmo servant. Dê um exemplo.

A utilidade de associar vários objectos CORBA a um mesmo servant é a de que é possível distinguir os vários clientes de um serviço implementado num servant dando diferentes referências de objectos

Corba a estes, de acordo com o seu estatuto. Por exemplo: num serviço de sondagens para um votante é dado um objecto Poll e para um Administrador da sondagem um objecto PollAdmin, podendo isto ser feito a partir de uma factory de objectos CORBA.

Um cliente obtém um objecto que lhe permite votar e inquirir resultados; representa a sua "sessão". O criador de uma sondagem obtém um objecto de administração, que lhe permite encerrar a sondagem. Isto quer dizer que cada um dos clientes (administrador ou votante) vai ter diferentes operações disponíveis mas utiliza o mesmo servant.

Pelo facto dos objectos CORBA estarem relacionados e terem que cooperar é mais útil utilizar um único servant por sondagem do que instanciar um servant por objecto CORBA. A segunda opção iria conduzir a um consumo de memória desnecessário, dificultando escalabilidade.

4.1 Exame 2ª Chamada : 17-01-2004

1. Diga o que entende por transparência de localização, e descreva alguns modos de a obter.

A transparência de localização é não ser necessário saber a localização de um componente para o utilizar. Esta pode ser obtida usando um serviço de nomes. O cliente obtém o endereço do serviço desejado através do serviço de nomes, fornecendo apenas o nome do mesmo. O servidor do serviço desejado tem que se registar previamente no serviço de nomes.

2. Descreva a utilidade do uso de funções idempotentes na API de serviços distribuídos, como a interface RPC de sistemas de ficheiros distribuídos.

As operações idempotentes: têm o mesmo efeito se executadas uma ou mais vezes. Por isso podem ser utilizadas em sistemas de ficheiros distribuídos como por exemplo: escrever um conteúdo num dado bloco de um ficheiro. As operações idempotentes são mais robustas:

Podem ser usadas com menos garantias de fiabilidade.
Oferecem maior tolerância a faltas: se há uma falha a operação pode ser repetida sem problemas.

3. Relativamente aos modelos de despacho de invocações remotas de procedimentos num servidor, estabeleça uma comparação entre *thread por invocação* e *thread por cliente*.

Thread por invocação:

. Cada pedido, mesmo do cliente é atendido em thread separada
. Custo extra de criação de threads
. Permite dependências entre operações vindas de um cliente multi-thread
. Permite explorar mais paralelismo mas não é relevanta pois cada cliente é tipicamente sequencial, e a maior fonte de rendimento provém de servir clientes diferentes

Thread por Cliente:

. Permite manter facilmente estado partilhado em memória
. Requer cuidados com controlo de concorrencia no servidor
. Apropriado quando há dependências entre clientes.

4. Explique o motivo porque faz sentido a concepcção de um sistema de memória partilhada distribuída ser feita juntamente com a das primitivas de sincronização a oferecer.

O motivo é de que como é necessário fornecer ao programador primitivas de sincronização para programação em memória partilhada (caso da DSM) estas primitivas são integradas naturalmente como parte do mecanismo de DSM.
Daí advindo um maior ganho de desempenho dado que o DSM e as primitivas de sincronização estão mais ligadas entre si e a concepção torna-se mais fácil.

7. Descreva o que são operações oneway em CORBA e a razão porque estas não permitem a declaração de excepções.

As operações one way em CORBA não devolvem resultados; se necessário o servidor envia o resultado através de outra invocação. Por isso não podem também devolver ou lançar excepções.

9. Existem formatos binários para serialização de estruturas de dados. Compare o XDR (utilizado no ONC RPC) com o CDR

(utilizado no CORGBA GIOP) no que diz respeito à resolução da questão *little/big endian.*

O XDR do ONC RPC usa formato neutro (como network order) e o CDR anota o formato do emissor: o receptor apenas converte se diferente (receiver makes it right), mais eficiente.

8. Em CORBA podem ser utilizados *Object Adapters* com políticas diversas, de acordo com o modelo de associação entre referências e servants pretendido. Descreva a utilidade de usar vários POAS, todos com o mesmo conjunto de políticas, num dado servidor. Dê um exemplo.

A utilidade de utilizar vários POA's, todos com o mesmo conjunto de políticas, num dado servidor, é a de que é possível agrupar os objectos criados para um serviço composto por várias instâncias iguais e usar um POA para cada instância deste serviço com um default servant. Por exemplo: num serviço de sondagens pode ser usado um POA por sondagem. Nesta situação, a remoção de todos os objectos criados para a sondagem é feito fazendo destroy do POA, não sendo necessário destruir individualmente todos os objectos CORBA criados para essa sondagem. Além disso não é necessário estruturar espaço de nomes de OIDs com o nome da sondagem para identificar a que sondagem pertencem os votantes e admistradores. O rootPOA encaminha os clientes das sondagens para o POA responsável pela sondagem pretendida.