Teoria Programação Concorrente

Perguntas e Respostas [Concorrência Múltiplos Objetos]

1. **Explique a diferença entre controle de concorrência interno e externo.**
   - **Resposta**: O controle de concorrência interno é implementado pelo próprio objeto, encapsulando a lógica de concorrência e aliviando o cliente de se preocupar com isso. Já o controle externo é gerido pelo cliente, permitindo mais flexibilidade, mas exigindo que ele escreva o código de controle de concorrência ao acessar o objeto.

2. **O que são objetos quase-concorrentes e como eles funcionam?**
   - **Resposta**: Objetos quase-concorrentes suportam várias invocações em curso, mas no máximo uma não está suspensa. Eles permitem que invocações possam ser parcialmente executadas e bloqueadas aguardando um evento externo, facilitando a colaboração entre clientes sem esperas ativas. Exemplo: um buffer limitado usado por threads produtoras e consumidoras.

3. **Descreva o problema de deadlock com um exemplo envolvendo duas threads e dois objetos.**
   - **Resposta**: Deadlock ocorre quando duas threads tentam adquirir locks em dois objetos na ordem inversa. Por exemplo, se a thread T1 bloqueia o objeto A e tenta bloquear B, enquanto a thread T2 bloqueia B e tenta bloquear A, ambas ficarão esperando indefinidamente, pois cada uma possui um lock que a outra precisa.

4. **O que é a técnica de two-phase locking e como ela previne deadlocks?**
   - **Resposta**: Two-phase locking é uma técnica de controle de concorrência onde cada transação adquire todos os locks necessários antes de liberar qualquer um. Consiste em duas fases: aquisição de locks e liberação de locks. Essa técnica garante isolamento entre transações e previne deadlocks ao não permitir que locks sejam adquiridos após algum lock ter sido liberado.

5. **Qual é a vantagem do locking hierárquico em sistemas de objetos hierárquicos?**
    - **Resposta**: O locking hierárquico permite bloquear toda uma sub-árvore de um container sem fazer lock individualmente para cada objeto, o que é vantajoso em sistemas com muitos objetos e hierarquias de composição pouco profundas. Esse método melhora a eficiência e facilita a gestão de locks em estruturas complexas.

6. **Por que a ordem de aquisição de locks é importante para evitar deadlocks?**
    - **Resposta**: Adquirir locks em uma ordem total predeterminada (do menor para o maior) evita dependências cíclicas, que são a principal causa de deadlocks. Garantindo uma ordem consistente de aquisição, threads não ficarão presas esperando por locks que outra thread já possui.

7. **Como funcionam os locks de leitura e escrita e por que são úteis?**
    - **Resposta**: Locks de leitura permitem que várias threads leiam concorrentemente, enquanto locks de escrita garantem exclusividade. São úteis porque permitem maior concorrência em operações de leitura, sem comprometer a integridade dos dados durante operações de escrita. A tabela de compatibilidade de locks ajuda a gerenciar diferentes tipos de acesso de forma eficiente.

Perguntas e Respostas [Monitores]

1. **O que são monitores e qual é sua principal função em controle de concorrência?**
    - **Resposta**: Monitores são primitivas estruturadas de controle de concorrência que encapsulam o estado e as operações de um tipo de dado. Sua principal função é garantir a exclusão mútua, permitindo que apenas um processo esteja executando dentro do monitor em um dado momento.

2. **Como os monitores garantem a exclusão mútua?**
    - **Resposta**: A exclusão mútua em monitores é garantida implicitamente. Quando uma operação é invocada em um monitor, o mutex (mutual exclusion lock) é adquirido automaticamente no início e liberado no final da operação, sem necessidade de código explícito para locking e unlocking.

3. **O que são variáveis de condição e como são utilizadas em monitores?**
    - **Resposta**: Variáveis de condição permitem que processos se bloqueiem voluntariamente até que uma certa condição seja satisfeita. Elas são usadas em monitores para gerenciar a ordem de execução, permitindo a um processo esperar por uma condição antes de continuar a execução, utilizando operações como `wait` e `signal`.

4. **Explique as operações `wait` e `signal` em monitores clássicos.**
    - **Resposta**: A operação `wait` bloqueia o processo na variável de condição, libera o mutex associado e suspende o processo. A operação `signal` libera o primeiro processo bloqueado na variável de condição, se houver algum, permitindo que este retome a execução.

5. **Qual é a diferença entre monitores clássicos e monitores modernos em relação à ordem de execução após um `signal`?**
    - **Resposta**: Em monitores clássicos, após um `signal`, o processo que estava bloqueado na variável de condição continua a execução imediatamente. Em monitores modernos, o processo que fez o `signal` continua a execução, e depois pode retomar o processo acordado ou um terceiro processo que esteja aguardando para entrar.

6. **O que é um spurious wakeup e como ele é tratado em monitores modernos?**
    - **Resposta**: Um spurious wakeup ocorre quando um processo bloqueado em um `wait` é despertado sem que um `signal` tenha sido executado. Em monitores modernos, isso é tratado usando loops para esperar pela condição desejada, garantindo que o processo apenas continue quando a condição for realmente verdadeira.

7. **Descreva o problema de starvation em monitores e como ele pode ser evitado.**
    - **Resposta**: Starvation ocorre quando certos processos nunca conseguem acesso ao recurso compartilhado. Para evitá-lo, monitores podem ser implementados com diferentes variantes de controle de prioridade, como garantir que processos que esperam há mais tempo tenham prioridade ou usar turnos entre leitores e escritores.

8. **O que são as variantes `signalAll` e como elas diferem de `signal` em monitores modernos?**

   - **Resposta**: A variante `signalAll` acorda todos os processos bloqueados na variável de condição, enquanto `signal` acorda apenas um. `SignalAll` pode ser mais ineficiente, pois envolve acordar e avaliar múltiplos processos, mas garante que nenhum processo bloqueado permaneça indefinidamente se a condição mudar.

(turnos e variáveis de condição evita starvation)