Programação Paralela e Concorrente — Questões Teóricas

Lista de Exercícios: Deadlocks e Redes de Petri

1. Quais são as quatro condições necessárias para a ocorrência de um Deadlock? Explique-as. Essas quatro condições são suficientes para a ocorrência de Deadlocks? Por quê?

As quatro condições necessárias são:
• **Exclusão Mútua:** apenas um processo pode usar um recurso por vez.
• **Posse e Espera:** um processo mantém recursos enquanto aguarda outros.
• **Não Preempção:** recursos não podem ser forçados a liberar antes do término do processo.
• **Espera Circular:** existe um ciclo de processos esperando recursos uns dos outros.

Essas condições são **necessárias, mas não suficientes**, pois o deadlock só ocorre se todas elas estiverem presentes simultaneamente.

2. Quais são as principais estratégias para se lidar com um Deadlock? Explique-as.

As principais estratégias são:
• **Prevenção:** evita que as condições necessárias ocorram (atacando posse e espera ou espera circular).
• **Evitação:** utiliza algoritmos como o do banqueiro para verificar estados seguros antes da alocação.
• **Detecção e Recuperação:** permite o deadlock ocorrer, detecta-o e toma ações como abortar processos.
• **Ignorar:** usada em sistemas simples, assumindo que deadlocks são raros.

3. Explique como prevenir um deadlock atacando as condições de posse e espera e espera circular. Como elas se aplicam à solução errada do problema do jantar dos filósofos?

• **Posse e Espera:** pode ser evitada exigindo que um processo solicite todos os recursos de uma vez.
• **Espera Circular:** pode ser evitada impondo uma ordem hierárquica na requisição de recursos.

No jantar dos filósofos, a solução errada permite que cada filósofo pegue um garfo e espere pelo outro, criando uma espera circular. A prevenção define que todos peguem os garfos em ordem ou que apenas um número limitado possa comer simultaneamente.

4. Mostre que há um deadlock nesse sistema utilizando o algoritmo de detecção de deadlocks estudado.

Aplicando o algoritmo de detecção, verifica-se que nenhum processo pode prosseguir, pois todos aguardam recursos mantidos por outros. O vetor de recursos disponíveis não é suficiente para liberar nenhum processo. Assim, o sistema está em **deadlock**.

5. É possível ocorrer um Deadlock em um sistema com 2 processos e 3 recursos?

(a) Sim, se cada processo possuir um recurso e ambos aguardarem o terceiro, pode ocorrer deadlock.
(b) Generalizando, o deadlock pode ocorrer se o número de recursos for insuficiente para atender todos os processos de forma segura. A condição é: **R < P** (recursos menores que processos) pode gerar risco.

6. Proponha uma melhoria para o sistema que elimina a condição de posse-e-espera.

Uma melhoria é permitir que processos solicitem gradualmente recursos, desde que liberem apenas quando garantido o acesso ao conjunto completo necessário. Outra abordagem é usar prioridade para evitar fome e filas ordenadas de requisição.

7. Proponha uma solução que evite Deadlock no sistema de transferência de fundos.

Pode-se impor uma **ordem global de bloqueio de contas**, garantindo que todas as threads travem as contas em ordem crescente (por ID, por exemplo). Assim, elimina-se a espera circular sem liberar travas prematuramente.

8. Qual a diferença entre deadlock, livelock e starvation?

• **Deadlock:** processos bloqueados indefinidamente, nenhum progride.
• **Livelock:** processos mudam de estado continuamente tentando resolver o impasse, mas nenhum avança.
• **Starvation:** um processo nunca obtém acesso aos recursos por baixa prioridade ou agendamento injusto.

Lista de Exercícios: Exclusão Mútua e Sincronização

1. O que é uma condição de corrida e região crítica? Quais são as 4 condições para prover exclusão mútua?

• **Região crítica:** parte do código onde há acesso a recursos compartilhados.
• **Condição de corrida:** ocorre quando múltiplas threads acessam o mesmo dado sem sincronização, produzindo resultados imprevisíveis.

As 4 condições para exclusão mútua são:
1. Apenas um processo por vez dentro da região crítica.
2. Progresso garantido (não há inanição).
3. Espera limitada.
4. Sem suposição sobre a velocidade relativa dos processos.

2. Defina e diferencie semáforos, mutexes e monitores.

• **Semáforos:** variáveis inteiras controladas por operações P (wait) e V (signal).
• **Mutexes:** travas binárias usadas entre threads para exclusão mútua.
• **Monitores:** abstrações de alto nível que encapsulam variáveis e funções com controle de acesso

automático.

Semáforos e mutexes são de baixo nível e exigem cuidado manual, enquanto monitores automatizam a exclusão mútua.

3. Como escolher o próximo processo a ser executado em um semáforo genérico?

O próximo processo pode ser escolhido com base em política FIFO (primeiro a esperar, primeiro a ser atendido) ou por prioridade. FIFO evita fome, enquanto prioridades otimizam desempenho para tarefas críticas.

4. Como utilizar semáforos, monitores e barreiras para sincronizar threads?

• **Semáforos:** coordenam o acesso a recursos compartilhados com contadores.
• **Monitores:** permitem sincronização automática de métodos críticos.
• **Barreiras:** forçam threads a esperar todas alcançarem um ponto antes de continuar, garantindo sincronização por fase.

5. Padrões de Projeto Concorrente

• **Fork/Join:** divide tarefas em subtarefas paralelas e as une ao final.
• **Lock/Unlock:** controla acesso manualmente com travas.
• **Barreiras:** sincronizam múltiplas threads em pontos fixos.
• **Despachante-operário:** separa despacho e execução de tarefas.
• **Bloquear e Esperar:** threads dormem até uma condição ser satisfeita, evitando polling.

Fichamento Detalhado para Estudo

Deadlocks: Situações em que processos esperam recursos uns dos outros indefinidamente.
Condições: exclusão mútua, posse e espera, não preempção, espera circular.
Prevenção: eliminar posse e espera (solicitação total) ou espera circular (ordem de requisição).
Evitação: verificar estados seguros (algoritmo do banqueiro).
Detecção: análise de grafos de alocação.
Starvation: processo nunca é atendido; **Livelock:** atividade sem progresso.
Região Crítica: parte do código que acessa recursos compartilhados.
Semáforos: controle via operações P e V.
Mutex: trava binária de exclusão.
Monitores: abstração de sincronização automática.
Barreiras: sincronização coletiva.
Padrões concorrentes: Fork/Join, Lock/Unlock, Despachante-operário, Bloquear e Esperar.
Critérios de Exclusão Mútua: acesso único, progresso, espera limitada, independência de velocidade.
Resumo: a concorrência segura depende da sincronização entre processos e da prevenção de deadlocks e condições de corrida.