Programação Paralela e Concorrente - Questões Teóricas

1 - Região Crítica, Condição de Corrida e Exclusão Mútua

Uma **região crítica** é a parte do código onde ocorre o acesso a recursos compartilhados, como variáveis ou dispositivos, que não podem ser acessados simultaneamente por múltiplos processos ou threads sem controle.

A **condição de corrida** (race condition) ocorre quando dois ou mais processos tentam acessar ou modificar um mesmo recurso compartilhado ao mesmo tempo, levando a resultados imprevisíveis.

A **exclusão mútua** é o mecanismo utilizado para garantir que apenas um processo por vez entre em uma região crítica.

Esses conceitos estão relacionados porque a exclusão mútua é a técnica usada para evitar condições de corrida dentro de regiões críticas.

2 - Semáforos, Monitores e Mutexes

Semáforos são variáveis inteiras que controlam o acesso a recursos compartilhados através de operações atômicas de *wait* (P) e *signal* (V). São usados para sincronização entre processos.

Monitores são estruturas de alto nível que encapsulam variáveis compartilhadas, operações e mecanismos de sincronização. Permitem acesso controlado a recursos através de métodos sincronizados.

Mutexes (Mutual Exclusion Locks) são travas binárias que permitem que apenas uma thread por vez acesse um recurso. São semelhantes a semáforos binários, mas usados geralmente em contextos de threads.

Em termos de implementação, semáforos e mutexes são suportados pelo sistema operacional com instruções atômicas, enquanto monitores são implementados em nível de linguagem de programação.

3 - Padrões de Projeto Concorrente

a) Fork/Join: divide uma tarefa em subtarefas independentes (fork), que são executadas em paralelo, e depois sincroniza os resultados (join).

b) Travar e Destravar: padrão que usa mecanismos de travamento (lock) e destravamento (unlock) para garantir exclusão mútua em regiões críticas.

c) Bloquear e Esperar: uma thread que não pode prosseguir entra em estado de bloqueio até que uma condição seja satisfeita, evitando polling contínuo.

d) Despachante-operário: separa a geração de tarefas (despachante) da execução (operário), permitindo distribuição eficiente de trabalho entre threads.

e) Barreiras: sincronizam múltiplas threads, fazendo com que todas esperem umas pelas outras antes de prosseguir para a próxima etapa de execução.

4 - Condições para Deadlock

As quatro condições necessárias para um deadlock são:

1. Exclusão mútua: recursos só podem ser usados por um processo de cada vez.
2. Não preempção: um recurso só pode ser liberado voluntariamente pelo processo que o detém.
3. Posse e espera: um processo que possui recursos pode solicitar novos, ficando bloqueado se não conseguir.
4. Espera circular: há uma cadeia circular de processos, cada um esperando por um recurso mantido pelo próximo.

Para prevenir deadlocks, é possível atacar a **posse e espera** (exigindo que processos peçam todos os recursos de uma vez) ou a **espera circular** (impondo uma ordem hierárquica para requisição de recursos).

5 - Sistema com P1, P2, P3 e recursos RS1-RS4

Aplicando o algoritmo do banqueiro e verificando as matrizes de alocação e requisição, percebe-se que há processos que ainda podem ser atendidos e liberam recursos para os demais.

Assim, o sistema **não está em deadlock**, pois existe uma sequência segura de execução possível entre os processos.

6 - Algoritmo do Banqueiro (Processos A, B, C, D)

O processo C solicita 2 recursos, passando a usar 6 de 8 possíveis. Após atender C, os recursos disponíveis seriam 2.

Analisando o estado, ainda é possível liberar recursos de outros processos após suas finalizações, formando uma sequência segura.

Portanto, o sistema **não entrará em deadlock** após conceder a requisição de C.

7 - Rede de Petri do Jantar dos Filósofos

O problema do jantar dos filósofos pode ser modelado com três filósofos (P1, P2, P3) e três garfos (G1, G2, G3). Cada filósofo possui três estados: pensando, com fome e comendo. As transições representam pegar garfos e devolver garfos.

Para evitar deadlock, implementa-se uma regra: um filósofo só pode pegar ambos os garfos se ambos estiverem disponíveis. Outra solução é permitir que apenas dois filósofos tentem comer simultaneamente.

Essa modelagem garante que nunca ocorra deadlock, pois impede a espera circular.

Fichamento / Resumo para Estudo

• Região crítica: parte do código que acessa recursos compartilhados.
• Condição de corrida: execução concorrente sem controle, resultados imprevisíveis.
• Exclusão mútua: garante acesso único à região crítica.
• Semáforo: controle com operações P e V.

• Monitor: abstração de sincronização em alto nível.
• Mutex: trava binária entre threads.
• Fork/Join: divide e sincroniza tarefas.
• Travar/Destravar: controle manual de acesso.
• Bloquear/Esperar: sincronização condicional.
• Despachante-operário: separa tarefas e execução.
• Barreiras: sincronização coletiva.
• Deadlock: bloqueio mútuo entre processos; prevenido atacando posse/espera e espera circular.
• Algoritmo do banqueiro: evita alocação insegura de recursos.
• Jantar dos filósofos: ilustra sincronização e prevenção de deadlock com regras de acesso a recursos limitados.