Capítulo 06
Sincronização de Processos

1. O que você entende por ***Race Condition***? De forma geral, o que pode ser feito para evitar que ela aconteça?

É um problema que ocorre em sistemas de computação concorrente, onde múltiplas threads ou processos compartilham recursos ou dados em memória e a saída do programa se torna imprevisível devido à ordem de execução das threads. Para evitar isso, práticas como sincronização usando mutexes, semáforos, ou transações atômicas são empregadas, juntamente com a preferência por estruturas de dados imutáveis, limitando o compartilhamento de dados entre threads e realizando testes de concorrência e análise estática para garantir que o código funcione de forma segura em ambientes concorrentes.

2. No contexto de Sistemas Operacionais, o que é uma **Seção Crítica**? Que exigências devem ser implementadas para resolver o "Problema da Seção Crítica"? Cite brevemente estes requerimentos:

Uma seção crítica em sistemas operacionais é uma parte do código onde processos ou threads compartilham recursos e precisam garantir exclusão mútua, progresso, espera limitada, não-preempção e notificação para evitar conflitos e race conditions. Para resolver o "Problema da Seção Crítica", esses requisitos são essenciais. Mecanismos como mutexes, semáforos e monitores são usados para implementar essas exigências, assegurando que o acesso aos recursos compartilhados ocorra de maneira coordenada e segura, evitando impasses e garantindo a integridade dos dados em ambientes multiusuários ou multitarefa.

3. O que você entende por **Espera em Ação**? O que pode ser feito para evitá-la?

É uma situação na qual processos ou threads ficam bloqueados, aguardando indefinidamente uns aos outros para liberar recursos, resultando em uma paralisação do sistema. Para evitar o deadlock, podem ser aplicadas várias estratégias, como prevenção, detecção e recuperação, preempção de recursos, garantir a não existência de espera circular, limites de tempo e uso cauteloso de recursos compartilhados. A escolha da estratégia depende da aplicação específica, mas geralmente envolve uma combinação dessas abordagens para garantir que os sistemas possam continuar funcionando de maneira eficaz, mesmo quando ocorrem bloqueios.

4. O que são **Semáforos**? Explique sucintamente e indique quais os tipos de semáforos existentes:

Semáforos são mecanismos de sincronização usados na programação concorrente para coordenar o acesso a recursos compartilhados. Existem dois tipos principais de semáforos: os binários, que têm apenas dois valores (0 e 1) e são usados para garantir o acesso exclusivo a um recurso, e os semáforos gerais, que têm valores inteiros maiores que 1 e controlam o acesso a um número limitado de instâncias de um recurso. Além disso, existem variações e extensões, como semáforos de contagem reversa, semáforos com operações de espera e notificação, e semáforos nomeados. Os semáforos desempenham um papel crucial na prevenção de condições de corrida, garantindo a exclusão mútua e controlando o acesso a recursos críticos em ambientes concorrentes.

5. Cite 3 problemas clássicos que podem ser resolvidos utilizando de semáforos e explique sucintamente como eles são resolvidos.

Problema do Produtor-Consumidor, que envolve o compartilhamento de um buffer entre produtores e consumidores, o Problema dos Leitores-Escritores, que gerencia o acesso concorrente a recursos compartilhados para leitura e escrita, e o Problema dos Filósofos Famintos, que previne impasses ao coordenar o acesso de filósofos famintos aos garfos em uma mesa.

6. O que são **Monitores**? De forma geral, como eles podem ser implementados.

Monitores são uma abstração de alto nível na programação concorrente que combina dados e funções relacionadas em uma unidade única. Eles facilitam a coordenação do acesso concorrente a recursos compartilhados, oferecendo métodos para operar nesses recursos, variáveis de condição para aguardar ou notificar eventos relevantes e semáforos ou mutexes internos para garantir a exclusão mútua. Monitores simplificam o desenvolvimento de código concorrente, tornando-o mais legível e menos propenso a erros, ao mesmo tempo que abstraem detalhes de baixo nível envolvidos na sincronização de threads ou processos. A implementação de monitores pode variar de acordo com a linguagem de programação e o sistema operacional, mas o conceito subjacente permanece consistente.

7. Que tipo de solução pode ser feita em Java para prover a exclusão mútua de métodos de um programa? Explique.

Em Java, a exclusão mútua de métodos é provida através da palavra-chave synchronized, que pode ser usada tanto em blocos de código quanto na definição de métodos. Através de blocos sincronizados, você delimita seções críticas de código, garantindo que apenas um thread por vez execute o código dentro desse bloco, com base em um objeto monitor especificado. Da mesma forma, marcando um método como synchronized, você assegura que apenas um thread pode acessá-lo de cada vez, automaticamente associando um monitor à instância da classe que possui o método. Essa abordagem simplifica a programação concorrente em Java, evitando condições de corrida e garantindo que operações críticas sejam executadas de forma segura e ordenada, tornando-se uma prática comum na construção de programas concorrentes.

8. Faça uma rápida pesquisa na Internet e descreva uma possível solução (apenas uma) para evitar Deadlock no Problema do Jantar dos Filósofos.

Uma solução para evitar deadlock no Problema do Jantar dos Filósofos é a "Hierarquia de Garfos", que envolve os filósofos adquirindo garfos de acordo com sua numeração e respeitando uma ordem hierárquica. Por exemplo, filósofos com números ímpares pegam primeiro o garfo à direita e depois o garfo à esquerda, enquanto filósofos com números pares fazem o oposto. Essa abordagem impede a formação de loops de espera circular, que é a principal causa de deadlock. Embora essa solução seja eficaz na prevenção de deadlock, ela pode resultar em um aumento no tempo de espera em algumas situações, uma vez que os filósofos precisam adquirir garfos de acordo com uma ordem específica.