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?

Uma race condition ocorre quando vários processos concorrentes tentam acessar e manipular os mesmos recursos ou variáveis compartilhadas de forma não coordenada.
Para evitar race conditions, você pode usar mecanismos de sincronização, como semáforos, mutexes ou exclusão mútua. Isso garante que apenas um processo por vez tenha acesso ao recurso compartilhado, evitando assim conflitos.

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 é uma parte de um programa em que um processo acessa ou modifica recursos compartilhados.
Para resolver o "Problema da Seção Crítica", os seguintes requisitos devem ser atendidos:
Mutual Exclusion: Apenas um processo pode estar na seção crítica por vez.
Progress: Se um processo não está na seção crítica e outros desejam entrar, um deles deve entrar em breve.
Bounded Waiting: Deve haver um limite para o tempo que um processo espera para entrar na seção crítica.

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

A espera em ação ocorre quando um processo fica em um loop ocioso, aguardando por um recurso que está atualmente ocupado.
Para evitá-la, você pode usar mecanismos de sincronização, como semáforos ou mutexes, que permitem que um processo aguarde de maneira eficiente em vez de gastar recursos de CPU em um loop ocioso.

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

Semáforos são uma abstração de sincronização que podem ser usados para controlar o acesso a recursos compartilhados.
Existem dois tipos de semáforos:
Semáforos Binários: Eles têm apenas dois valores, 0 e 1, sendo usados para exclusão mútua.
Semáforos Gerais: Podem ter um valor maior que 1 e são usados para controlar o número de processos que podem acessar um recurso compartilhado.

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

Problema do Produtor-Consumidor: Usando semáforos, os produtores produzem dados e os consumidores os consomem, garantindo que os dados estejam disponíveis quando necessário.

Problema dos Leitores e Escritores: Semáforos podem ser usados para controlar o acesso a uma região de memória compartilhada por leitores e escritores.
Problema dos Filósofos Famintos: Semáforos podem ser usados para representar garfos (recursos compartilhados) e garantir que os filósofos não entrem em um deadlock enquanto tentam comer.

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

Monitores são uma abstração de programação que combina dados e procedimentos (métodos) em uma única unidade. Eles garantem a exclusão mútua usando mecanismos internos.
Monitores podem ser implementados usando linguagens de programação que os suportam nativamente, como Java com a palavra-chave synchronized.

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 pode ser alcançada usando a palavra-chave synchronized. Você pode marcar métodos ou blocos críticos com synchronized para garantir que apenas um thread execute esses métodos ou blocos por vez.

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

Uma possível solução para evitar deadlock no Problema do Jantar dos Filósofos é introduzir um "árbitro" ou um semáforo geral. O árbitro controla o acesso aos garfos e garante que apenas alguns filósofos possam pegar os garfos simultaneamente. Isso evita o impasse, pois os filósofos não podem pegar todos os garfos ao mesmo tempo, o que levaria ao bloqueio.