RESPOSTAS (1-15) – PROGRAMAÇÃO CONCORRENTE

1. Qual a vantagem fundamental que você obteria executando uma aplicação multithread em um sistema multiprocessador em vez de um sistema uniprocessador?

   R: Os múltiplos threads da aplicação que realizam tarefas paralelas poderíam executar verdadeiramente de maneira simultânea em processadores separados acelerando a execução da aplicação.

2. Por que os processos tradicionais são chamados de processos pesados?

   R: A distinção primária entre ‘pesos’ relativos de processos tradicionais e threads está no modo como os espaços de endereçamento são alocados. Quando um processo é criado, recebe um espaço de endereçamento alocado exclusivamente a ele. Quando um thread é criado ele compartilha o espaço de endereçamento do processo, portanto, threads são mais ‘leves’ do que processos.

3. Como um thread entra no estado morto?
   R: Um thread entra no estado morto quando conclui sua tarefa ou quando um outro thread otermina.

4. Quais as semelhanças entre os estados de espera, bloqueado e adormecido? Quais as diferenças?
   R: Esses estados são semelhantes porque quando os threads estão neles não podem usar um processador mesmo que esteja disponível. Um thread bloqueado não pode ser despachado, pois está esperando por uma operação de E/S requisitada. Nesse caso, o sistema operacional é responsável por eventualmente desbloquear o thread. Um thread em espera não pode ser despachado até receber um evento do hardware ou do software que não é iniciado pelo sistema operacional (por exemplo, utilização do teclado ou um sinal de um outro thread). Nesse caso o sistema operacional não pode controlar se, ou quando, um thread em espera eventualmente será acordado. Um thread adormecido não pode executar porque notificou explicitamente ao sistema que ele não deve executar até que seu intervalo de sono expire.

5. O que são primitivas de sincronização de threads?

   R: Primitivas de sincronização são mecanismos utilizados em programação concorrente para controlar o acesso a recursos compartilhados entre threads ou processos, a fim de garantir que esses recursos sejam acessados de maneira ordenada e segura.

6. Como pode-se evitar condições de corrida em um programa concorrente?

   R: Para evitar condições de corrida em um programa concorrente, é importante garantir que as threads acessem recursos compartilhados de maneira exclusiva. Isso pode ser feito usando primitivas de sincronização, como semáforos, mutexes ou monitores. Além disso, é importante garantir que as operações em recursos compartilhados sejam atomicas, ou seja, que sejam realizadas em uma única etapa, para evitar que outras threads interfiram no meio da operação.

7. Qual é a diferença entre semáforos e mutexes? Quando pode-se usar um ao invez do outro?

   R: Semáforos e mutexes são duas primitivas de sincronização diferentes usadas para controlar o acesso a recursos compartilhados em um ambiente concorrente. O semáforo é mais adequado para cenários em que um número limitado de threads pode acessar um recurso compartilhado simultaneamente. Já o mutex é mais apropriado para proteger recursos compartilhados em cenários em que apenas uma thread pode acessar o recurso de cada vez.

8. O que é um deadlock e como ele pode ocorrer em um sistema com múltiplos processos ou threads?

   R: Um deadlock é uma situação em que duas ou mais threads ou processos estão bloqueados, aguardando que o outro libere um recurso que eles precisam para continuar. Isso pode ocorrer quando as threads ou processos não liberam recursos adequadamente ou quando são alocados recursos insuficientes para executar as tarefas. Para evitar deadlocks, é importante usar primitivas de sincronização corretamente e garantir que haja recursos suficientes disponíveis.

9. Quais são as vantagens e desvantagens de usar monitores em vez de semáforos ou mutexes?
   R: Os monitores são uma primitiva de sincronização que permite que apenas uma thread acesse um recurso compartilhado por vez, em vez de permitir que várias threads acessem simultaneamente. Isso pode tornar o código mais fácil de ler e entender, pois a lógica de sincronização é encapsulada no monitor em vez de espalhada pelo código. No entanto, os monitores podem ser mais lentos do que outras primitivas de sincronização e podem levar a problemas de desempenho em ambientes de alta concorrência.

10. O que são construções concorrentes de alto nível e como elas podem simplificar a implementação de sistemas concorrentes?
    R: Construções concorrentes de alto nível são abstrações ou padrões de programação que permitem aos desenvolvedores escrever código concorrente de maneira mais simples e segura, sem a necessidade de se preocupar com detalhes de baixo nível, como sincronização de threads e gerenciamento de recursos compartilhados. Essas construções geralmente envolvem a criação de objetos ou entidades que representam unidades independentes de execução, como threads, processos ou tarefas, e fornecem mecanismos para coordenar e sincronizar a comunicação e o compartilhamento de dados entre essas unidades.

11. Quais são as principais construções concorrentes de alto nível disponíveis para os programadores, e como elas diferem em termos de recursos e limitações?

R: Algumas das principais construções concorrentes de alto nível são:

• Threads: Threads são a construção concorrente mais básica disponível em muitas linguagens de programação. Elas permitem que várias tarefas sejam executadas em paralelo dentro de um único processo.

- Recursos: threads podem compartilhar dados e recursos de forma fácil e eficiente. São relativamente fáceis de criar e gerenciar. Permitem o uso eficiente de CPUs com vários núcleos.
- Limitações: Sincronização manual é necessária para garantir a consistência dos dados compartilhados. Threads podem ser vulneráveis a problemas de corrida, bloqueio e deadlock.

• Processos: Processos são unidades de execução separadas que são independentes entre si e podem ser executadas em paralelo em sistemas operacionais multitarefa.

- Recursos: Cada processo tem seu próprio espaço de endereçamento, o que significa que os dados são isolados entre os processos.Processos podem ser executados em diferentes CPUs e até mesmo em diferentes máquinas em uma rede. Problemas de sincronização e bloqueio são menos comuns do que em threads.
- Limitações: Processos são mais pesados do que as threads, pois requerem a alocação de recursos adicionais, como espaço na memória e descritores de arquivo. A comunicação entre processos é mais lenta e menos eficiente do que a comunicação entre threads.

12. Como as construções concorrentes de alto nível podem ser utilizadas para aumentar a eficiência e o desempenho de sistemas computacionais?
    R: As construções concorrentes de alto nível podem aumentar a eficiência e o desempenho de sistemas computacionais dividindo tarefas em sub-tarefas menores executadas em paralelo, reduzindo o tempo de espera em operações de entrada/saída, acelerando cálculos intensivos e melhorando a responsividade e escalabilidade do sistema.

13. Como as construções concorrentes de alto nível podem ser aplicadas em diferentes contextos, como sistemas operacionais, jogos e bancos de dados?
    R: As construções concorrentes de alto nível podem ser aplicadas em diferentes contextos, como sistemas operacionais, jogos e bancos de dados, de maneiras específicas a cada contexto. Algumas das principais aplicações são:

    - Sistemas Operacionais: As construções concorrentes de alto nível podem ser usadas para gerenciar recursos do sistema, como processos e threads, permitindo que várias tarefas sejam executadas simultaneamente. Isso pode melhorar a eficiência e o desempenho do sistema.
    - Jogos: As construções concorrentes de alto nível podem ser usadas para gerenciar a lógica de jogo e a renderização de gráficos. Elas podem ser usadas para dividir a renderização em vários threads, permitindo que diferentes partes da cena sejam renderizadas

simultaneamente. Além disso, as construções concorrentes podem ser usadas para gerenciar eventos de entrada, permitindo que o jogo responda rapidamente às ações do jogador.

- Bancos de Dados: As construções concorrentes de alto nível podem ser usadas para melhorar o desempenho de bancos de dados, permitindo que várias consultas sejam executadas simultaneamente em diferentes threads. Isso pode reduzir o tempo de espera para consultas e melhorar a capacidade do banco de dados para lidar com cargas de trabalho pesadas.

14. Dê exemplos de alguns problemas clássicos de programação concorrente.

    R: Alguns exemplos de alguns problemas clássicos de programação concorrente são:

    - Problema dos produtores e consumidores ;
    - Problema dos fumantes;
    - Problema dos escritores/leitores;
    - Problema do jantar dos filósofos;
    - Problema do barbeiro sonolento.

15. Como poderíamos resolver o problema do buffer limitado?

    R: A solução do problema de buffer limitado envolve três aspectos de coordenação distintos e complementares:

    • A exclusão mútua no acesso ao buffer, para evitar condições de disputa entre produtores e/ou consumidores que poderiam corromper o conteúdo do buffer.

    • A suspensão dos produtores no caso do buffer estar cheio: os produtores devem esperar até que surjam vagas livres no buffer.

    • A suspensão dos consumidores no caso do buffer estar totalmente vazio: os consumidores devem esperar até que as vagas no buffer sejam preenchidas.