***Campus* Campina Grande**

Disciplina: Sistemas Operacionais

Data: 06 de Agosto de 2021

Professor: David Candeia Medeiros Maia

Aluno: Luiz Medeiros Neto

Respostas da prova sobre sincronização de processos e escalonamento.

1-

a) As condições de corrida ocorrem quando dada a alocação de processos e threads na memória, pode existir um recurso compartilhado entre elas, que pode ser tanto um dado, uma variável ou até mesmo um endereço de memória. Isto posto, a condição de corrida acontece quando dois ou mais threads ou processos tentam modificar e acessar essas variáveis compartilhadas, o que gera um problema com a gravação e alteração dos dados ou variáveis, visto que apenas a thread que mexeu por último nessa variável terá seu papel concretizado, enquanto os dados gravados pelas outras threads por exemplo, são perdidos ou podem gerar saídas inválidas.

b) A região crítica pode ser definida como a região do programa em que existe acesso à memória compartilhada, haja vista a existência de recursos nesse local que serão compartilhados entre processos ou threads.

c) Exclusão mútua é o nome dado ao método que tem por objetivo evitar que dois ou mais processos ou threads acessem a região crítica ao mesmo tempo, assim, evitando possíveis erros na apresentação e na computação de dados.

d) Recurso compartilhado como o próprio nome já o define, diz respeito aos recursos que são alocados por um dois ou mais processos ou threads. Esse recurso pode ser variáveis, dados de entrada, constantes, endereços de memória, etc. Nesse sentido, múltiplos processos e threads podem ler e escrever nesse recurso.

2-

Sim, mediante aos conceitos de CPU Bound e I/O Bound é possível identificar o tipo do processo não só pelo código fonte, como também por meio da verificação do tempo de execução do processo na CPU, através da utilização dos devidos instrumentos dispostos para a análise. Dessa maneira, infere-se que os processos I/O Bound passam a maior porção do tempo em estado de espera aguardando dados de I/O, diferentemente dos CPU Bound, que passam a maior parcela de seu tempo na CPU computando dados e efetuando cálculos matemáticos, por essa razão pode-se citar, também, que os CPU Bound são comuns nas aplicações científicas.

Portanto, ao observar o código fonte do processo e notar-se a presença de um grande número de fórmulas e cálculos matemáticos sendo feitos, grande quantidade de endereços sendo manipulados juntamente com variáveis por exemplo, e constatar-se que esse processo faz pouca utilização de comandos de entrada e saída, bem como passa grande parte do tempo na CPU; é possível caracterizá-lo como CPU Bound.

Convém lembrar que, ao ler o código fonte de um processo detectando que ele passa a maior parte do tempo em estado de espera, se baseia em leitura, gravação e apresentação de dados por exemplo; é plausível inferir que ele é do tipo I/O Bound.

3-

Dado o contexto de espera ociosa na execução de processos e uma vez dada prioridades a esses processos; o problema de inversão ocorre quando um processo que tenha maior prioridade pode ficar testando indefinidamente a variável turn (variável que indica o processo que terá a vez de execução) que no momento pertence a ele , fazendo com que o processo de mais baixa prioridade nunca seja alocado para liberar a variável.

É importante ressaltar que o problema de inversão não acontece com as threads de usuário, haja vista que a quando a thread de usuário é gerada, o programa aloca as threads apenas em espaço de usuário, o que implica que o kernel não saiba sobre elas.

4-

<u>Algoritmo First Come, First Served (FCFS):</u>

- **É alocado para sistemas em lote**, baseados numa fita, visto que o algoritmo **só cede a CPU a outro processo, se o processo que está em execução terminar ou for bloqueado.**
- Não é interessante para sistemas interativos, haja vista que a interação do processo com o usuário seria insuficiente ou mal alocada, além disso, uma vez alocado muitos processos CPU bound, outros processos do tipo I/O bound podem ficar bloqueados ou ociosos por muito tempo quando deveriam estar funcionando utilizando dispositivos I/O.
- Trabalha com uma **fila de processos**, o que implica que o a**tendimento das requisições deve ser por ordem de chegada**, ou seja, **conforme chegam novos processos, eles devem ser alocados no final da fila de atendimento.**

- **Objetivos:** manter a CPU ocupada, vazão, etc.
- É um algoritmo não preemptivo, ou seja, nesse algoritmo executa-se um determinado processo até que ele termine ou seja bloqueado.
    - Se o processo **terminar corretamente**, o algoritmo **cede a CPU normalmente** para o próximo processo da fila.
    - Se o processo é **bloqueado** devido a algum problema, **ele volta para o final da fila.**
- **Um exemplo** da aplicação do algoritmo pode ser **o atendimento de pacotes em roteadores.**

Algoritmo de Job mais curto primeiro:

- Não preemptivo;
- O algoritmo tem como objetivo fundamental **servir sistemas em lote**;
- Nesse algoritmo, o **tempo de execução** total de cada processo **deve ser conhecido;**
- **Os jobs devem estar disponíveis simultaneamente;**
- O algoritmo aloca os processos de forma que os processos com job mais curto devam ser executados primeiro.
    - Ex: Numa determinada fila, chegam 3 processos, P1, P2, P3, com tempo de execução de 3s, 6s, 3s respectivamente. Isto posto, o algoritmo irá alocar os processos numa sequência em que os com o menor tempo sejam executados primeiro. Assim, P1 e P3 serão executados antes de P2.
    - **O que ganha-se com isso?**
        - Menor tempo médio de espera para processamento, tendo em vista cada job da fita.
- **Objetivos:**
    - Aumentar o número jobs concluídos dado um determinado tempo(vazão);
    - Tempo de retorno(diminuir o tempo médio de espera para processamento do job, dado o intervalo do ingresso do processo e sua finalização).
    - manter a CPU ocupada o maior tempo possível.

Algoritmo Round-Robin:

- Preemptivo;
- Para a execução da preempção, é dado um tempo máximo de alocação de um processo da CPU que recebe o nome de “Quantum”;
- Um processo só fica alocado na CPU até o momento em que seu tempo de execução restante é esgotado ou se ele atinge o Quantum.
- Um processo pode ser alocado e desalocado várias vezes da CPU conforme sua demanda atual de tempo para finalização;
- Após o tempo de demanda do processo se esgotar, ele é finalizado;
- **Objetivos:**

    - Tempo de resposta ao usuário, atendendo suas expectativas;
    - Proporcionalidade no tempo de resposta conforme as requisições;
    - Justiça:
        - Dar a cada processo, uma fatia de execução justa(igual) na CPU.

5-

- **Job mais curto primeiro:**
    - Alocado para sistemas em lote;
    - Não Preemptivo, ou seja, o algoritmo só cede a CPU caso o processo seja concluído;
    - Todos os tempos de execução são conhecidos e os jobs estão disponíveis simultaneamente;
    - O algoritmo aloca os processos de forma que os processos com job mais curto devam ser executados primeiro.
    - Objetivos: utilização da CPU, vazão, tempo de retorno.

- **Próximo de menor tempo restante,** se assemelha ao de job mais curto primeiro**, entretanto:**
    - É preemptivo, ou seja, **o sistema pode interromper um processo e alocar outro**, caso o novo processo que chegue na fita tenha o seu tempo de execução menor que o tempo restante do processo que está em execução**;**
        - Se a interrupção ocorrer, o processo que estava sendo executado é suspenso e só volta a executar quando o processo de menor tempo for concluído, após a conclusão, o processo suspenso entra em execução novamente, além de que ele volta em seu tempo restante para finalização. O ciclo pode se repetir conforme outros processos.

.

6)

a) Circular(Round-Robin Simples):

- Vamos trocar m por unidade de tempo (termo mais genérico):
    - u.t = unidade de tempo;
- Considere um quantum hipotético de 2u.t a fins de demonstração simples;
- Considere os tempos para cada processo:
    - A = 10u.t
    - B = 6u.t
    - C = 2u.t
    - D = 4u.t

    - E = 8u.t
- Dado o quantum hipotético, a fila de processamento (diagrama de Gantt) será:
    - **A(2u.t) -> B(2u.t) -> C(2u.t) -> D(2u.t) -> E(2u.t) -> A(2u.t) -> B(2u.t) -> D(2u.t) -> E(2u.t) -> A(2u.t) -> B(2u.t) -> E(2u.t) -> A(2u.t) -> E(2u.t) -> A(2u.t)**

- A partir disso é possível ter os seguintes resultados:
    - O **tempo total de espera** de cada processo será:
        - A = 30 (última execução) - 0(tempo de chegada) - 10(tempo total de execução) = **20 u.t (tempo total de espera)**
        - B = 22 - 0 - 6 = **16 u.t**
        - C = 6 - 0 - 2 = **4 u.t**
        - D = 16 - 0 - 4 = **12 u.t**
        - E = 28 - 0 - 8 = **20 u.t**
    - O **tempo médio de espera** dos processos será:
        - (20 + 16 + 4 + 12 + 20) / 5 = **14.4 u.t**
    - O **tempo de turnaround** (tempo entre a chegada do processo e sua finalização) **de cada processo e tempo médio de turnaround** será:
        - A = 20 (tempo de espera) + 10 (tempo de execução) = **30 u.t**
        - B = 16 + 6 = **22 u.t**
        - C = 4 + 2 = **6 u.t**
        - D = 12 + 4 = **16 u.t**
        - E = 20 + 8 = **28 u.t**
        - O tempo médio de turnaround será:
            - (30 + 22 + 6 + 16 + 28) / 5 = **20.4 u.t**

b) Escalonamento por prioridade:

- Considere a não preempção;
- Vamos trocar m por unidade de tempo (termo mais genérico):
    - u.t = unidade de tempo;
- Considere as prioridades 3, 5, 2 ,1 ,4 para os processos A, B, C, D, E, respectivamente.
- A execução da fila conforme o algoritmo de escalonamento por prioridades será:
    - **B(6u.t = tempo de execução) -> E(8u.t) -> A(10u.t) -> C(2u.t) -> D(4u.t)**
    -
- A partir disso é possível ter os seguintes resultados:
    - O **tempo para finalização** de cada processo será:

        - A = 14u.t (tempo de espera) + 10u.t (tempo de execução) = **24 u.t (**tempo para finalizar**);**
        - B = 0u.t + 6u.t = **6u.t (**tempo para finalizar**);**
        - C = 24u.t + 2u.t = **26 u.t (**tempo para finalizar**);**
        - D = 26u.t + 4u.t = **30 u.t (**tempo para finalizar**);**
        - E = 6u.t + 8u.t = **14 u.t (**tempo para finalizar**);**
    - O **tempo médio de espera** será:
        - (14 + 0 + 24 + 26 + 6) / 5 = **14 u.t**
    - O **tempo médio de turnaround** será:
        - (24 + 6 + 26 + 30 + 14) / 5 = **20 u.t**

c) Primeiro a chegar, primeiro a ser servido(siga a ordem 10, 6, 2, 4, 8):

- Vamos trocar m por unidade de tempo (termo mais genérico):
    - u.t = unidade de tempo;
- Considere a chegada ao mesmo tempo dos processos e os seus tempos:
    - A (10 unidades de tempo)
    - B (6 unidades de tempo)
    - C (2 unidades de tempo)
    - D (4 unidades de tempo)
    - E (8 unidades de tempo)
- A execução da fila será a seguinte conforme o algoritmo FCFS e as unidades de tempo:
    - A = 0 **->** 10;
    - B = 10 **->** 16;
    - C = 16 -> 18;
    - D = 18 **->** 22;
    - E = 22 -> 30;
- A partir disso é possível ter os seguintes resultados:
    - O **tempo para finalização** de cada processo será:
        - A = 0u.t + 10u.t = **10u.t (**tempo para finalizar**);**
        - B = 10u.t + 6u.t = **16u.t (**tempo para finalizar**);**
        - C = 16u.t + 2u.t = **18u.t (**tempo para finalizar**);**
        - D = 18u.t + 4u.t = **22u.t (**tempo para finalizar**);**
        - E = 22u.t + 8u.t; = **30u.t (**tempo para finalizar**);**
    - O **tempo médio de espera** será:
        - (0 + 10 + 16 + 18 + 22) / 5 = **13.2 u.t**
    - O **tempo médio de turnaround** será:
        - (10 + 16 + 18 +22 + 30) / 5 = **19.2 u.t**