==========

Lista de Exercícios de Consolidação

O objetivo da lista é ajudar no estudo individual dos alunos. Soluções de questões específicas poderão ser discutidas em sala de aula, conforme interesse dos alunos.

==========

1. Qual o problema com a solução que desabilita as interrupções para implementar a exclusão mútua?

   Com as interrupções desabilitadas, a CPU não realiza o chaveamento entre processos, ou seja, caso haja algum problema na tarefa e a mesma fique presa dentro da região crítica, o sistema inteiro travará. Além disso, tal solução funciona apenas em ambientes monoprocessadores, uma vez que, numa máquina com mais de um processador, processos concorrentes podem ser executados simultaneamente em processadores diferentes, e desabilitar as interrupções em um processador não impede que outros processadores interfiram com as operações que o primeiro processador está realizando.

2. O que é espera ocupada (busy wait) e qual o seu problema? Em que situações esse tipo de solução pode ser utilizado?

   A espera ocupada é o teste contínuo de uma condição. Quando um processo deseja entrar em sua região crítica, o mesmo verifica se sua entrada é permitida (condição), caso não seja, o processo fica preso num laço de espera, até que sua entrada na região crítica seja enfim permitida. O maior problema desse mecanismo se dá no desperdício de tempo da CPU, uma vez que o processo fica preso no laço de espera testando continuamente uma condição. Por este motivo, a espera ocupada (busy wait) deve ser utilizada apenas quando há uma expectativa razoável de que tal espera será curta.

3. Em muitos sistemas operacionais existe uma chamada de sistema “yield()”, cujo efeito é passar o processo (ou thread) chamador para o final da fila de aptos (prontos). Suponha que um dado kernel escalone o processador utilizando fatias de tempo e suporte a chamada “yield()”. Suponha ainda que, em todas as aplicações, o tempo necessário para executar uma seção (região) crítica seja menor que a duração de uma fatia de tempo. Para este caso em particular, buscando resolver o problema da seção crítica, considere o seguinte mecanismo de sincronização:

   Imediatamente antes de entrar na seção crítica, a thread chama “yield()”;
   Ao sair da seção crítica, a thread não faz nada.

   Responda as questões abaixo, sempre justificando sua resposta.

   Responda as questões abaixo, sempre justificando sua resposta.

   (a) Esta solução oferece exclusão mútua?

   Resposta: Sim, pois há uma preempção para entrar na parte crítica, uma por vez, e o tempo necessário para executar uma região crítica é menor que a duração de uma fatia de tempo

   (b) Esta solução apresenta a possibilidade de postergação indefinida **(starvation)**?

Resposta: Não, porque após o yeld() o processo é jogado para o final da fila e em algum momento vai chegar a vez dele, e quando chegar, irá ser completamente executado

(c) Esta solução funciona para mais de duas threads/processos?

Resposta: Sim, pois possui exclusão mútua, então somente um processo irá ter acesso a região crítica por vez, e não há starvation

(d) Esta solução apresenta busy waiting?

Resposta: Não, porque em nenhum momento o processo fica em um loop de teste de alguma variável esperando ser acordado ou algo do tipo, ele é sempre chamado pelo SÓ quando ele vai entrar na região crítica.

4. Considerando a afirmação : “A técnica de busy waiting (spin lock) não deve ser empregada para sincronização em máquinas monoprocessadores, seu uso é indicado apenas em sistemas que executam sobre máquinas multiprocessadoras” Responda : Essa afirmação está correta ? Justifique a sua resposta.

   A afirmação está correta. Em uma máquina monoprocessadora, bloquear uma tarefa que espera por uma condição é lógico pois o único processador da máquina deve ser liberado para executar outras tarefas. Uma dessas tarefas irá alterar tal condição de forma a desbloquear a tarefa à espera dessa. Em uma máquina multiprocessadora, esta troca de contexto pode ser evitada permitindo a tarefa testar continuamente a condição enquanto outro processador executa a tarefa que irá alterar a condição que a primeira tarefa está esperando. Esse é o princípio do spin-lock, ou seja, enquanto uma tarefa está aguardando no laço de espera a condição para entrar na região crítica, outro processador executa outra tarefa que está dentro de tal região e ao sair da mesma, é alterado o estado da condição, liberando o acesso à região crítica da tarefa que está sendo executada no primeiro processador.

5. No projeto de um sistema operacional para um monoprocessador a equipe de desenvolvimento utilizou um escalonador não preemptivo e a técnica de espera ocupada (busy waiting) na implementação de suas primitivas de exclusão mútua (lock/unlock). Pergunta-se : a decisão do tipo de escalonador e da técnica de busy waiting foi correta ? Em que aspectos sua resposta seria modificada se esse sistema executasse em máquinas multiprocessadoras ? Justifique sua resposta.

   A decisão está correta. Sendo um sistema monoprocessador com escalonamento não-preemptivo, tem-se que uma tarefa em execução nunca perde a CPU para eventos externos, ou seja, só há troca de contexto (chaveamento entre tarefas) uma vez que esta finalizar ou for bloqueada (ou ceder a CPU por vontade própria). Dessa forma, o uso de trava se dá de maneira segura, pois não há a possibilidade de uma troca de contexto entre o teste da variável de trava e a sua atribuição, não havendo assim, condição de corrida envolvendo tal variável. Para o caso de um sistema multiprocessador, tarefas em processadores diferentes poderiam acessar a variável trava simultaneamente, gerando uma condição de corrida e consequentemente, um possível mau funcionamento. Portanto, outro mecanismo de exclusão mútua deveria ser utilizado.