Curso de Tecnologia em Sistemas de Computação
Disciplina de Sistemas Operacionais
Professores: Valmir C. Barbosa e Felipe M. G. França
Assistente: Alexandre H. L. Porto

Quarto Período
AP1 - Primeiro Semestre de 2007

Nome -
Assinatura -

Observações:

1. Prova sem consulta e sem uso de máquina de calcular.
2. Use caneta para preencher o seu nome e assinar nas folhas de questões e nas folhas de respostas.
3. Você pode usar lápis para responder as questões.
4. Ao final da prova devolva as folhas de questões e as de respostas.
5. Todas as respostas devem ser transcritas nas folhas de respostas. As respostas nas folhas de questões não serão corrigidas.

---

1. (1.0) Por que a multiprogramação é importante para as últimas gerações de computadores?

   Resp.: Com o passar das gerações, a velocidade de processamento dos computadores se tornou cada vez maior. A velocidade dos dispositivos físicos também aumentou, mas muito mais lentamente do que a velocidade de processamento. Com isso, o tempo de ociosidade do processador quando o processo em execução fazia operações de E/S ficou, com o passar das gerações, cada vez maior, pois cada processo era executado até terminar, sem interrupções. Além disso, existia uma grande demora para se obter os resultados dos processos. Para evitar esses problemas, o conceito de multiprogramação, que permite que mais de um processo esteja em execução no sistema através da divisão do tempo de processamento entre os processos, foi definido a partir da terceira geração.

2. (2.0) Descreva as seguintes hierarquias que podem ocorrer em um sistema operacional:

   (a) (1.0) Árvore de processos.

   Resp.: Como vimos na Aula 2, um processo pode criar, durante a sua execução, outros processos. Uma **árvore de processos** é criada quando um processo, chamado de **pai**, cria um outro processo, chamado de **filho**, que por sua vez pode criar outros processos. Nesta árvore, cada vértice representa um processo diferente, sendo que a raiz é um processo que foi criado pelo sistema operacional (na verdade todos os processos em execução no sistema formam uma única árvore), e uma aresta de um processo em um nível para um outro no próximo nível significa que este último foi criado pelo primeiro.

   (b) (1.0) Hierarquia de diretórios.

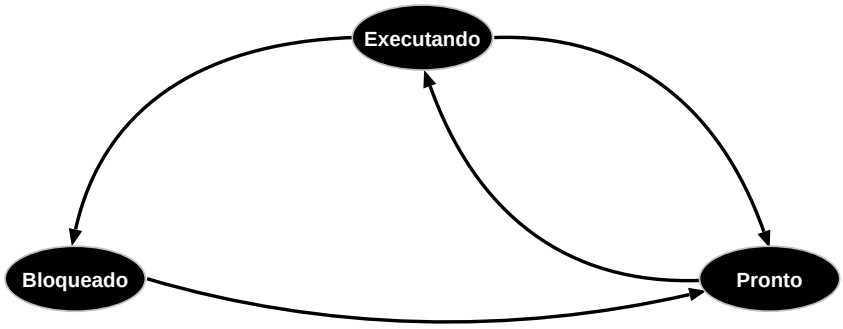
   Resp.: Quando estudamos o conceito de arquivos na Aula 2, vimos que um **diretório** é um arquivo especial que permite agrupar

arquivos relacionados. Um diretório é composto por diversas entradas, sendo que cada uma delas está associada a um arquivo do sistema que pode ser um outro diretório. Como este diretório também pode possuir entradas associadas a outros diretórios, uma hierarquia de diretórios pode então ser formada. Nesta hierarquia, que também define uma árvore, cada vértice representa um arquivo diferente, sendo que a raiz é o único diretório (chamado de **diretório raiz**) não associado a uma entrada de um outro diretório. Já uma aresta entre um diretório de um nível para um arquivo no próximo nível significa que o último está associado a uma entrada do primeiro.

3. (1.0) O conceito de máquina virtual permite que vários sistemas operacionais sejam executados na mesma máquina. Como isso é feito?

   Resp.: Como vimos na Aula 3, uma máquina virtual é uma cópia exata do hardware do computador, tão complexa de ser usada quanto ele. Para permitir que vários sistemas operacionais executem na mesma máquina, uma máquina virtual é definida para cada um dos sistemas, dando a cada sistema a ilusão de que ele está usando exclusivamente a máquina real. Agora, quando um dos sistemas operacionais acessar o "hardware" da sua máquina virtual, este acesso será interceptado por um programa, em execução na máquina real, responsável pelo gerenciamento das máquinas virtuais, e será mapeado para um acesso real ao hardware. Com isso, vários sistemas operacionais poderão executar simultaneamente na mesma máquina, sem interferirem uns com os outros.

4. (2.0) Na figura a seguir damos os estados de um processo e as possíveis transições entre estes estados. Pela figura, vemos que um processo não pode passar do estado pronto para o bloqueado e nem passar do estado bloqueado para o executando. Por que estas transições não existem na figura?



Resp.: Como vimos na Aula 4, quando um processo está no estado pronto, ele está esperando a sua vez de executar em um dos processadores da máquina. Logo, a transição do estado pronto para o bloqueado não tem sentido, pois um processo passa ao estado bloqueado quando está em execução e não pode continuar a executar, pois passou a esperar pela ocorrência de um evento externo. Já a transição do estado bloqueado para o executando não existe porque o processo que foi desbloqueado, apesar de agora está pronto para a execução, deverá esperar pela próxima vez que o escalonador for chamado pelo sistema operacional. Note que a transição não deve existir mesmo se o processo for mais prioritário do que os outros não bloqueados, pois neste caso o escalonador pode ser imediatamente chamado após o processo ser desbloqueado.

5. (2.0) Suponha que um processo A executa uma seqüência de operações, sendo que cada operação pode ser uma dentre as duas possíveis definida pela palavra de memória R. De tempos em tempos, o processo A verifica a palavra R para determinar se deve continuar executando a operação atual, ou se deve passar a executar a outra operação. Suponha que dois processos, B e C, controlem, através da palavra R, qual operação A deverá executar. Responda:

   (a) (1.0) Temos condições de corrida. Quais são elas?

   Resp.: A existência de condições de corrida está intimamente ligada ao tipo de computação que está sendo realizada pelos processos envolvidos. Por exemplo, quando discutimos o assunto na

Aula 5, havia uma condição de corrida entre os processos acessando a lista de arquivos a serem impressos porque queríamos garantir que todos os arquivos submetidos fossem de fato impressos. No caso da presente questão, a afirmação de que "temos condições de corrida" obriga-nos, antes de dar uma resposta, a fazer alguma hipótese sobre o funcionamento do processo A e sobre como B e C interferem nesse funcionamento. É possível fazer hipóteses variadas, mas ilustraremos a resposta para apenas uma: a hipótese de que A tem que ler todos os valores escritos por B ou C. Feita essa hipótese, a condição de corrida presente na situação descrita ocorre entre o par de processos B, C e o processo A. Especificamente, dependendo de como se intercalem as escritas de B ou C com as leituras de A, valores escritos poderão jamais ser lidos por A.

(b) (1.0) Quais são os semáforos necessários para evitar as condições de corrida? Justifique a sua resposta.

Resp.: Como vimos no item anterior, as condições de corrida dependem de uma hipótese sobre o funcionamento do processo A e de como B e C interferem neste funcionamento. A seguir vamos mostrar uma solução que resolve a condição de corrida para a hipótese ilustrada no item anterior. Neste caso, precisaremos de um semáforo binário *dado_ja_lido* para controlar a alteração da palavra R, de tal modo que o valor de R somente possa ser alterado após A ler este valor. O valor inicial deste semáforo deverá ser 0, pois como A já está em execução, R deverá ter o valor da operação inicial a ser executada. Para garantir que A possa ler o valor de R antes de B ou C alterá-lo novamente, B e C deverão executar a operação **P** sobre *dado_ja_lido* imediatamente antes de alterar o valor de R, e A deverá executar a operação **V** sobre *dado_ja_lido* imediatamente após ler o valor atual de R.

6. (2.0) Suponha que os processos A, B e C estão em execução no sistema, e suponha que usamos o algoritmo por *round robin*, com um quantum de 2 unidades de tempo, para escalonar os processos. Por quanto tempo cada um dos processos executou realmente no processador, se o esca-

lonador escolher, inicialmente, os processos na ordem A, B e C, e se A terminar a execução após 12 unidades de tempo, B após 15 unidades de tempo, C após 11 unidades de tempo? Justifique a sua resposta.

Resp.: Na tabela dada a seguir mostramos a execução dos processos de acordo com o algoritmo de escalonamento por *round robin*, supondo que a ordem inicial de execução é A, B e C, e que o quantum tem 2 unidades de tempo. Nesta tabela temos, na primeira linha, os instantes de tempo em que os processos foram escolhidos para executar no processador. Já na segunda linha desta tabela temos os processos executados em cada um dos tempos dados na primeira linha. Note que se um processo usar somente metade do seu quantum (1 unidade de tempo), o próximo processo segundo o algoritmo *round robin* será executado, para evitar que o processador fique ocioso.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	A	B	B	C	C	A	A	B	B	C	A	B	B	B

Como podemos ver pela tabela dada acima, cada um dos três processos usa somente metade de seu último quantum. O tempo real de execução de cada um deles será o número de vezes que cada um aparece na tabela, isto é, os tempos serão de, respectivamente, 5, 7 e 3 unidades de tempo.