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 2010

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. (1.5) Por que os programas de usuário não podem executar no modo supervisor? O núcleo do sistema operacional pode executar no modo usuário?

   Resp.: -Um programa de usuário não pode executar no modo supervisor porque, neste modo, ele teria acesso direto a todo o hardware do computador. Devido a isso, a integridade dos dispositivos físicos poderia ser comprometida, porque cada programa de usuário poderia acessar o mesmo dispositivo de um modo diferente (por exemplo, os dados armazenados em um disco poderiam ser comprometidos). Além disso, também poderíamos ter problemas de segurança e de consistência durante a execução dos programas, porque um programa poderia acessar e/ou alterar os dados usados exclusivamente por um outro programa. Finalmente, um programa poderia executar do seu início até o seu término, mesmo que o sistema operacional usasse a multiprogramação, porque ele poderia usar exclusivamente o processador.
   - O núcleo do sistema operacional não pode executar no modo usuário, porque ele precisaria ter acesso direto aos dispositivos físicos para, pelo menos, poder enviar comandos para as suas controladoras (o gerenciamento dos dispositivos pelo núcleo depende da estrutura na qual o sistema operacional foi baseado). Além disso, o sistema também precisaria gerenciar a memória alocada aos programas, para poder distribuí-la entre eles, e garantir que um programa não acesse a memória dos outros programas. Finalmente, o sistema precisaria gerenciar a alocação do processador entre os diversos programas em execução.

2. (2.5) Diga se as seguintes afirmativas são falsas ou verdadeiras. Para responder, escreva apenas F ou V para cada item em seu caderno de respostas.

   (a) (0.5) Cada processo do sistema operacional possui uma identificação (UID) única. Além disso, existe um espaço de endereçamento asssociado a cada processo com o seu código, os seus dados, e a sua pilha de execução.

   Resp.: F (Falsa, pois a identificação de um processo é denominada PID; UID se refere à identificação do usuário).

(b) (0.5) Em uma árvore de processos, quando dois processos estiverem ligados por uma aresta, isso significará que um dos processos dependerá de um resultado da execução do outro processo.

Resp.: F (Falsa, pois uma aresta ligando dois processos de uma árvore de processos indica que um deles criou o outro).

(c) (0.5) Normalmente, para podermos acessar o sistema de arquivos com o sistema operacional, precisaremos primeiramente montá-lo em algum diretório de um outro sistema de arquivos.

Resp.: F (Falsa, pois é exatamente o contrário, ou seja, para usarmos um sistema de arquivos que não contenha o sistema operacional, precisaremos montá-lo antes em algum diretório do sistema de arquivos com o sistema operacional).

(d) (0.5) Um *pipe* permite a troca de dados de modo transparente entre dois processos, porque conecta a saída de um dos processos à entrada do outro processo.

Resp.: V (Verdadeira).

(e) (0.5) As bibliotecas são, em geral, usadas para fazermos as chamadas ao sistema operacional, porque elas fornecem vários modos de usarmos uma mesma chamada, e porque elas facilitam o uso de uma chamada ao ocultarem os detalhes necessários à sua execução.

Resp.: V (Verdadeira).

3. (1.5) Por que a velocidade de execução de um processo pode ser reduzida e o tempo de execução das operações de E/S feitas por este processo pode ser aumentado, se o processo estiver executando em um sistema operacional sobre uma máquina virtual? Justifique a sua resposta.

Resp.: A redução da velocidade de execução do processo e o aumento no tempo de execução das operações de E/S ocorre porque uma máquina virtual é uma simulação da máquina real. Em relação à velocidade de execução ser reduzida, isso ocorre porque, como parte da simulação, o monitor de máquina virtual deve alternar o uso do processador do hardware entre os processadores virtuais das diversas máquinas virtuais em execução. Além disso, alguns monitores de máquina virtual podem, ao invés de usar, simular o processador (ou seja, o seu modo de operação) e, com isso, existe um *overhead* adicional no tempo de execução devido a esta simulação. Já em relação à redução da velocidade das operações de E/S, isso ocorre porque o monitor de máquina virtual também precisa simular cada operação de E/S. Esta simulação mapeia cada operação de E/S, feita em uma máquina virtual sobre um dispositivo virtual, na operação de E/S correspondente sobre um dispositivo físico. Logo, existe tambem um aumento no tempo de execução de cada operação de E/S, por causa do *overhead* gerado pelo mapeamento.

4. (1.5) Como os *pipes* podem ser usados para garantir o correto funcionamento de dois processos que cooperem para executar uma tarefa em comum, se um dos processos sempre depende dos dados gerados pelo outro processo?

   Resp.: Como vimos nas aulas 2 e 4, os *pipes* funcionam ligando a saída de um processo, no caso o processo que gera os dados, com a entrada de um outro processo, no caso o processo que depende destes dados. Ao ligarmos estes dois processos pelo *pipe*, quando o processo que gera os dados escrevê-los em sua saída, eles serão copiados para o *pipe* na ordem em que foram escritos. Além disso, quando o processo que depende dos dados ler a sua entrada, os dados serão lidos do *pipe* na mesma ordem em que foram escritos, pois o *pipe* mantém a ordem dos dados. A garantia do correto funcionamento dos processos ocorre porque o processo que gera os dados sempre é bloqueado caso tente escrever dados em sua saída quando o *pipe* estiver cheio, e o processo que depende dos dados sempre é bloqueado caso tente ler dados da sua entrada quando o *pipe* estiver vazio. Note que os processos não precisam se preocupar com a exclusão mútua ao acessar o *pipe*, porque

ele é um tipo especial de arquivo do sistema operacional, para o qual exclusão mútua já é automaticamente garantida.

5. (1.5) Descreva como os semáforos podem ajudar a garantir a correta execução de um conjunto de processos que cooperem para executar uma tarefa em comum.

   Resp.: Os semáforos podem ser usados para garantir a exclusão mútua, ou seja, que somente um processo possa acessar, em um dado momento, a sua seção crítica, e a sincronização destes processos, ou seja, que cada processo somente possa executar depois que certas condições, dependentes dos outros processos, tenham sido satisfeitas. A exclusão mútua é garantida através do uso de um semáforo binário, com valor inicial 1, sobre o qual cada processo deve executar a operação **P** imediatamente antes de entrar em sua seção crítica e executar a operação **V** imediatamente após sair da sua seção crítica. Para garantir a sincronização, precisamos usar um ou mais semáforos (binários e/ou de contagem), sendo que os objetivos destes semáforos e as suas inicializações dependem das condições que devem ser atendidas para que cada processo possa executar corretamente. Por exemplo, no problema do produtor/consumidor precisamos usar dois semáforos de contagem para a sincronização dos processos. O primeiro, inicializado com o tamanho do buffer, é usado para bloquear o processo produtor caso o buffer esteja cheio. Já o segundo, inicializado com 0, é usado para bloquear o processo consumidor caso o buffer esteja vazio. Note que, neste exemplo, o processo produtor deve executar a operação **P** sobre o primeiro semáforo antes de colocar um elemento no buffer, e executar a operação **V** sobre o segundo após colocar este elemento. Já o processo consumidor deve executar a operação **P** sobre o segundo antes de retirar um elemento do buffer, e executar a operação **V** sobre o primeiro após remover este elemento.

6. (1.5) Qual é a diferença entre os algoritmos de escalonamento preemptivos e não-preemptivos? Dê um exemplo para cada um dos casos.

   Resp.: -No algoritmo de escalonamento preemptivo, cada processo

executa no processador ou por um dado intervalo de tempo ou até ser bloqueado esperando por um evento externo (por exemplo, o término de uma operação de E/S). Para poder alternar o processador entre os diversos processos, as interrupções do temporizador são usadas para chamar o algoritmo de escalonamento preemptivo, em intervalos fixos de tempo, para decidir se o processador deve ou não ser cedido a um outro processo. Note que este algoritmo também é chamado quando o processo em execução é bloqueado, para poder escolher um outro processo e evitar que o processador fique ocioso. Agora, no algoritmo de escalonamento não-preemptivo, as interrupções não são usadas e o processo continua a executar até ser bloqueado ou terminar a sua execução. Logo, o algoritmo de escalonamento não-preemptivo somente é chamado quando o processo em execução termina ou bloqueia, para poder alocar o processador para um outro processo.
-Na aula 6 vimos diversos exemplos de algoritmos de escalonamento preemptivos. Os dois mais importantes são o algoritmo por *round robin* e o algoritmo por prioridades, mas também podemos citar, como exemplos, o algoritmo por sorteio e o algoritmo baseado em classes de prioridades. Finalmente, o único algoritmo não-preemptivo que vimos na aula 6 foi o algoritmo do trabalho mais curto primeiro.