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
AD2 - Primeiro Semestre de 2007

Nome -
Assinatura -

---

1. (1.0) Em um sistema de transferência eletrônica de fundos, há centenas de processos idênticos que trabalham da seguinte forma. Cada processo lê uma linha de entrada especificando uma quantidade de dinheiro, a conta a ser creditada e a conta a ser debitada. Então, ele bloqueia ambas as contas e transfere o dinheiro, liberando os bloqueios quando tiver terminado. Com muitos processos executando em paralelo, é possível que, tendo bloqueado a conta x, ele seja incapaz de bloquear y porque y foi bloqueada por um processo agora esperando pelo desbloqueio de x. Esboce um esquema que evite impasses. Não libere um registro de conta até que você tenha completado as transações (em outras palavras, não são permitidas soluções que bloqueiem uma conta e, então, liberem-na imediatamente se a outra estiver bloqueada).

   Resp.: Dos métodos vistos na Aula 7, o que satisfaz a condição do problema é o método visto no final da aula, que evita a condição de espera circular através da numeração dos recursos em ordem crescente.

Neste método, como vimos, um processo somente poderá solicitar um recurso cujo número é maior do que os de todos os recursos que este processo possui. No problema proposto, os recursos são as contas do banco. Agora, suponha que o processo deseja bloquear duas contas x e y (a transferência pode ser de x para y ou de y para x). Então, o processo bloqueará a conta maior somente quando conseguir bloquear antes a menor. Por exemplo, se x for menor do que y, o processo somente bloqueará y quando conseguir bloquear x com sucesso. Note que a ordem usada é a crescente. Logo, os impasses não poderão mais ocorrer, pois a ordenação dos recursos em ordem crescente, e o impedimento de um processo obter um recurso com número maior do que os dos recursos que ele possui, evitará a condição de espera circular, necessária à ocorrência de impasses.

2. (2.0) Um computador tem 1GB de RAM alocada em unidades de 64KB. Quantos bits são necessários se um mapa de bits for usado para gerenciar a memória? Se usarmos uma lista encadeada ao invés de um mapa de bits, quanta memória será necessária no melhor e no pior caso, supondo que o sistema operacional ocupa os 512KB iniciais da memória?

   Resp.:-Como o computador tem 1GB de RAM, isto é, 1048576KB de RAM, e como cada unidade de alocação possui 64KB de tamanho, então o número total de unidades de alocação é de 16384. Logo, o mapa de bits terá 16384 bits, isto é, 2048 bytes (ou 2KB), pois existe um bit no mapa para cada unidade de alocação.
   -Se usarmos uma lista encadeada, primeiramente deveremos determinar o número de bytes usados por cada elemento da lista. Como 1GB é igual a 2^{30} bytes, então são necessários 4 bytes para referenciar um endereço da memória. Além disso, também serão necessários 4 bytes para indicar o tamanho de uma região da memória (usada ou livre). Agora, cada elemento da lista possui o endereço inicial da região da memória, o tamanho da região, o tipo da região (alocada ou disponível), e o endereço do próximo elemento da lista. Logo, cada elemento ocupará 13 bytes da memória, pois podemos representar o tipo em um byte, que indica, por exemplo, se a região está ou não livre para ser usada. Note que, das 16384 unidades de alocações totais, temos somente 16376 unidades disponíveis, pois o sistema operacional possui 512KB de tamanho,

e com isso, ocupa as primeiras 8 unidades de alocação da memória. O melhor caso ocorrerá quando a memória restante estiver toda disponível ou toda alocada a um mesmo processo, e, neste caso, a lista terá somente uma entrada, que ocupará 13 bytes da memória. O pior caso ocorrerá quando a memória estiver completamente fragmentada, ou seja, quando for composta por uma seqüência alternada de unidades livres e usadas (ou vice-versa). Neste último caso, a lista terá 16376 elementos de 13 bytes, e com isso, ocupará 212888 bytes da memória (aproximadamente 207,9KB).

3. (2.0) Um computador cujos processos têm 1024 páginas em seus espaços de endereços mantém suas tabelas de páginas na memória. O *overhead* exigido para ler uma palavra da tabela de páginas é 500ns. Para reduzir esse *overhead*, o computador tem uma TLB, que armazena 32 pares ⟨página virtual, moldura de página física⟩ e pode fazer uma busca em 100ns. Qual a taxa de acertos necessária para reduzir o *overhead* médio para 200ns?

   Resp.: Quando não usamos a TLB, o tempo médio para ler a palavra será sempre de 500ns. Agora, se usarmos a TLB, o tempo médio será reduzido para 100ns se a palavra estiver na TLB. Note que o tempo médio para ler a palavra ainda será de 500ns caso esta palavra não esteja na TLB, pois a procura da palavra na TLB é feita em paralelo com a leitura dela da tabela de páginas. Note também que se a palavra estiver na TLB, o tempo será de 100ns, pois podemos abortar a leitura da tabela na memória. Logo, se a taxa de acerto (isto é, a fração das palavras da tabela de páginas referenciadas que estavam na TLB) for de h, o tempo médio de acesso será de $100h + 500(1 - h)$. Agora, como queremos um tempo médio de 200ns, bastará resolvermos a equação anterior para este valor, ou seja, resolver a equação $100h + 500(1 - h) = 200$. Ao resolvermos esta equação ($100h + 500 - 500h = 200 \Rightarrow 400h = 300 \Rightarrow h = 3/4$), vemos que a taxa de acerto deverá ser de $0,75$, ou seja, uma taxa de acerto de 75%.

4. (2.0) Um computador oferece a cada processo 65536 bytes de espaço de endereço dividido em páginas de 4096 bytes. Um programa particular tem um tamanho de texto de 32768 bytes, um tamanho de dados de

16386 bytes e um tamanho de pilha de 15870 bytes. Esse programa irá se ajustar no espaço de endereçamento? Se o tamanho de página fosse de 512 bytes, ele se ajustaria? Lembre-se de que uma página não pode conter partes de dois segmentos diferentes.

Resp.:-Se o tamanho de cada página for de 4096 bytes, então cada processo receberá 16 páginas do sistema operacional, pois o seu espaço de endereçamento possui 65536 bytes. O texto do programa, com 32768 bytes, ocupará exatamente 8 páginas. Já os dados do programa, com 16386 bytes, ocuparão 5 páginas, mas somente os 4 bytes iniciais da última página serão usados. Finalmente, a pilha do programa, com 15870 bytes, irá ocupar 4 páginas, sendo que também a última página não será completamente usada (serão usados os 3582 bytes iniciais desta página). Logo, o número de páginas necessárias será de 17, pois não podemos colocar duas partes do programa em uma mesma página. Agora, como somente 16 páginas estão disponíveis ao processo, então deveremos necessariamente armazenar uma página no disco e, com isso, o programa não se ajustará no espaço de endereçamento.
-Quando o tamanho de página for de 512 bytes, cada processo receberá 128 páginas do sistema operacional, porque o espaço de endereçamento de cada processo possui 65536 bytes. Neste caso, o texto do programa ocupará 64 páginas (pois o seu tamanho é de 32768 bytes), os dados do programa ocuparão 33 páginas (pois o seu tamanho é de 16386 bytes), e a pilha do programa ocupará 31 páginas (pois o seu tamanho é de 15870 bytes). Assim como antes, os dados e a pilha não usarão completamente a última página (os dados e a pilha usarão, respectivamente, 4 e 510 bytes desta página). Como neste caso também não podemos colocar duas partes do programa em uma mesma página, o número de páginas necessárias será de 128 e, com isso, agora o programa se ajustará na memória.

5. (1.0) O sistema de arquivos do UNIX possui uma chamada ao sistema **chroot** que muda a raiz para um dado diretório. Isso gera problemas de segurança? Se sim, quais são eles?

Resp.: Sim, se a chamada não for limitada ao superusuário, o usuário

administrador do sistema. O problema seria o de que o usuário poderia se tornar o superusuário, que pode fazer qualquer alteração no sistema. Se um usuário pudesse executar esta chamada ao sistema **chroot**, para ele poder definir todas as configurações do sistema, bastaria que ele, usando esta chamada, mudasse o diretório raiz para um diretório seu, como, por exemplo, `/home/alexandre`. Depois disso, cada nome de caminho absoluto seria mapeado a partir do diretório `/home/alexandre`. Para obter o acesso ao superusuário, bastaria que o usuário criasse um diretório `etc` dentro do diretório `/home/alexandre`. Neste diretório, o usuário criaria um arquivo `passwd` com uma senha para o superusuário escolhida por ele, pois o programa de login procura as senhas no arquivo com nome de caminho absoluto `/etc/passwd`, que será mapeado para o nome de caminho `/home/alexandre/etc/passwd`.

6. (2.0) O espaço livre em disco pode ser monitorado utilizando uma lista de blocos livres ou um mapa de bits. Os endereços de disco requerem D bits. Para um disco com B blocos, F dos quais estão livres, declare a condição sob a qual a lista de livres utiliza menos espaço que o mapa de bits. Para D tendo o valor de 16, expresse sua resposta como uma porcentagem do espaço em disco que deve estar livre.

   Resp.:-O espaço gasto pelo mapa de bits será de B bits, pois temos um bit no mapa para cada bloco do disco, e temos B blocos no disco. Já o espaço gasto ao usarmos uma lista de blocos livres será de DF bits, pois temos F blocos livres no disco, uma entrada na lista para cada bloco livre, e o tamanho de um endereço de um bloco do disco é de D bits. Logo, a lista de blocos livres gastará menos espaço do que o mapa de bits se $DF < B$, ou, alternativamente, se $F/B < 1/D$. Note que F/B é a fração de blocos livres do disco, ou seja, $100F/B$ dá a porcentagem de espaço livre no disco.
   -Como vimos na resposta anterior, para a lista ocupar menos espaço, deveremos ter que $F/B < 1/D$. Agora, como F/B é a fração de blocos livres no disco, se $D = 16$ deveremos ter que $F/B < 1/16$, ou seja, $F/B < 0,0625$. Logo, o espaço livre no disco deverá ser menor do que 6,25% do espaço total do disco.