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 - Segundo Semestre de 2006

Nome -
Assinatura -

---

1. Se uma controladora de disco grava os bytes que recebe da memória de disco tão rapidamente quanto ela os recebe, sem nenhuma bufferização interna, a intercalação ainda é útil? Discuta.

   Resp.: Não, pois a interlacação somente é necessária quando a controladora não consegue copiar os bytes de um bloco enviado pelo disco antes de ele enviar os bytes do próximo bloco. Como vimos na Aula 7, na intercalação, os blocos logicamente consecutivos de um trilha do disco não são armazenados consecutivamente nesta trilha. Com esta organização dos blocos no disco, a controladora sempre consegue copiar os bytes lidos do disco para a memória antes que este comece a enviar os bytes do próximo bloco. Se isso não fosse feito, ao lermos dois blocos consecutivos do disco, precisaríamos esperar o tempo de uma rotação para ler o segundo bloco, pois o disco já teria começado a enviar este bloco antes de a controladora copiar o bloco anterior para a memória.

2. Um computador tem espaço suficiente para armazenar quatro programas em sua memória principal. Esses programas ficam inativos esperando E/S metade do tempo. Que fração de tempo da CPU é desperdiçada?

   Resp.: Como cada programa fica inativo esperando E/S por metade do seu tempo de execução, então a probabilidade de um dos programas estar bloqueado é de 1/2. O único modo de o processador estar ocioso é se todos os quatro programas estiverem bloqueados ao mesmo tempo. A probabilidade de isso acontecer é de $(1/2)^4 = 1/16$ (pois a probabilidade de um programa estar bloqueado é de 1/2). Logo, o processador estará ocioso por 1/16 do tempo de processamento, isto é, ele estará ocioso por 6.25% deste tempo.

3. A seguir é apresentada a listagem de um programa curto em linguagem *assembly* para um computador com páginas de 512 bytes. O programa está localizado no endereço 1020, e seu ponteiro de pilha está em 8192 (a pilha cresce em direção a 0). Forneça a cadeia de referências de páginas gerada por esse programa. Cada instrução ocupa 4 bytes (1 palavra) e referências tanto de instruções como de dados contam na cadeia de referências.

   Carregue a palavra 6144 no registrador 0
   Coloque o registrador 0 na pilha
   Chame um procedimento em 5120, colocando o endereço de retorno na pilha
   Subtraia a constante imediata 16 do ponteiro da pilha
   Compare o parâmetro real com a constante imediata 4
   Salte se igual a 5152

   Resp.: Como cada página possui 512 bytes, então podemos inicialmente concluir o seguinte: a primeira instrução do programa, do endereço 1020, está na posição 508 da Página 1; o topo da pilha está no início da Página 16 (na posição 0), pois o ponteiro da pilha está apontando para o endereço 8192. Vamos agora ver quais páginas são acessadas ao executarmos cada uma das instruções:

   (a) **Carregue a palavra 6144 no registrador 0**: O processador

deve ler o endereço 1020 para obter o código da instrução, acessando para isso a posição 508 da Página 1. Depois de obtê-lo, o processador deverá agora acessar o endereço 6144, localizado no início da Página 12 (na posição 0), para obter o valor a ser armazenado no registrador 0. Note que o endereço com a próxima instrução será o 1024, localizado no início da Página 2.

(b) **Coloque o registrador 0 na pilha**: O processador primeiramente lê o endereço 1024 no início da Página 2, para obter o código da instrução. Agora, o ponteiro da pilha é decrementado em pelo menos uma unidade, para armazenarmos o valor do registrador 0 na pilha. Independentemente do valor deste decremento, iremos acessar a Página 15, pois o ponteiro da pilha estava originalmente no início da Página 16. Note que agora o endereço da próxima instrução será o 1028, localizado na posição 4 da Página 2.

(c) **Chame um procedimento em 5120, colocando o endereço de retorno na pilha**: O processador acessará novamente a Página 2 (na posição 4), ao obter o código da instrução armazenado no endereço 1028. Depois disso, o ponteiro da pilha será novamente decrementado, e vamos novamente acessar a Página 15 ao armazenarmos o endereço de retorno da chamada. O endereço da próxima instrução será agora o 5120, localizado no início da Página 10.

(d) **Subtraia a constante imediata 16 do ponteiro da pilha**: O processador primeiramente lerá o endereço 5120, no início da Página 10, para obter o código da instrução. Depois subtrairá o valor 16 do ponteiro da pilha, e como este é um registrador, nenhuma referência será feita a uma página. Agora, o endereço da próxima instrução será o 5124, localizado na posição 4 da Página 10.

(e) **Compare o parâmetro real com a constante imediata 4**: Primeiramente é acessada a posição 4 da Página 10, para obtermos o código da instrução armazenado no endereço 5124. Depois disso, a Página 15 será acessada, para lermos o parâmetro real do topo da pilha. A próxima instrução estará agora no endereço 5128 (localizado na posição 8 da Página 10).

(f) **Salte se igual a 5152**: O processador acessa, para obter o código

da instrução, a posição 8 da Página 10 (o endereço 5128). Note que a próxima instrução a ser executada ainda estará na Página 10, pois o endereço 5134 está na posição 12 desta página, e o endereço 5152 está na posição 32 da página.

Logo, temos a seguinte cadeia de referências, sendo que colocamos, entre parênteses, se a página foi acessada para obter o código da instrução (I) ou um dado (D): 1 (I), 12 (D); 2 (I), 15 (D); 2 (I), 15 (D); 10 (I); 10 (I), 15 (D); 10 (I). Nesta cadeia, separamos as referências de duas instruções consecutivas por um ponto e vírgula, e as referências em uma mesma instrução por uma vírgula.

4. Quanto tempo leva para carregar um programa de 64KB de um disco cujo tempo médio de busca é 10ms, cujo tempo de rotação é 8ms e cujas trilhas armazenam 1MB,

   (a) para um tamanho de página de 2KB?

   (b) para um tamanho de página de 4KB?

   (c) para um tamanho de página de 64KB?

   As páginas são distribuídas aleatoriamente no disco. Baseado nos resultados obtidos anteriormente, por que as páginas são tão pequenas? Cite duas desvantagens de usarmos uma página de 64KB ao invés de uma de 4KB.

   Resp.: Primeiramente vamos notar que, em média, o tempo gasto para ler uma página aleatória do disco é de 14ms, pois deveremos esperar em média pela metade do tempo de rotação do disco para ler esta página (4ms), e o tempo médio de busca é de 10ms. Agora, como cada trilha armazena 1MB, isto é, 1024KB, e como o tempo de rotação do disco é de 8ms, então a taxa de transferência de dados será de 128KB/ms (isto é, transferimos 128KB em 1ms). Então:

   (a) Para uma página com tamanho de 2KB, o tempo de carga desta página será de 14ms mais o tempo necessário para copiá-la, que será de $2/128 = 0.015625$ms. Agora, como o tamanho do programa é de 64KB, este terá 32 páginas, e com isso, o tempo para carregá-lo será de 14.015625×32 ms, isto é, 448.5ms.

(b) Para uma página com tamanho de 4KB, o seu tempo de carga será de 14ms mais o seu tempo de cópia, que será de $4/128 = 0.03125$ms. Agora, como temos 16 páginas no programa, pois o seu tamanho é de 64KB, o tempo para carregar o programa será de 14.03125×16 ms, isto é, 224.5ms.

(c) Para uma página com tamanho de 64KB, o tempo de carga da página será de 14ms mais o tempo gasto para copiá-la, que será de $64/128 = 0.5$ms. Agora, como o programa terá somente uma página, pois o seu tamanho é igual ao tamanho da página, o tempo para carregar o programa será de 14.5ms.

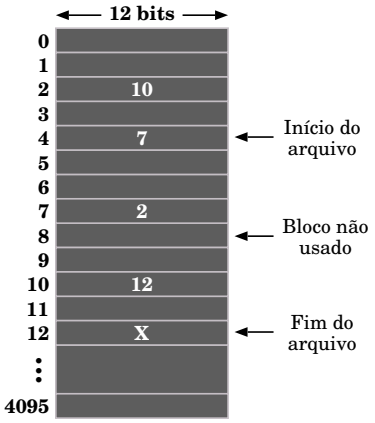
Como podemos notar pelos resultados anteriores, quanto maior a página, menor é o tempo de carga desta página. O motivo de se usar páginas pequenas é devido a dois fatores:

(a) O desperdício com a fragmentação interna, tanto na memória como no disco ao salvar uma página (como vimos na Aula 10, em média, a metade da última página alocada a uma área de um processo não é usada). No caso da página de 64KB (que desperdiça em média 32KB) o desperdício será 16 vezes maior do que o da página de 4KB (que desperdiça em média 2KB);

(b) Quanto maior o tamanho das páginas, menor a sua quantidade, o que pode reduzir o desempenho do sistema, caso o conjunto funcional de um processo esteja disperso por todo o seu espaço de endereçamento. Suponha que temos um espaço de endereçamento virtual de 16MB e um espaço de endereçamento físico de 1MB. Suponha ainda que o processo use ativamente 256 regiões de memória de 4KB distribuídas uniformemente por todo o espaço de endereçamento virtual. Logo, a distância entre as posições iniciais de duas regiões consecutivas no espaço de endereçamento virtual será de 64KB. Além disso, teremos 256 molduras de página se usarmos páginas virtuais com 4KB e somente 16 molduras se usarmos páginas com 64KB. Com isso, a taxa de falhas de página será cerca de 16 vezes maior se usarmos páginas com 64KB, pois poderemos colocar, para ambos os tamanhos de página, somente uma região em cada moldura de página.

5. A figura dada a seguir, que vimos na Aula 11 ao estudarmos a alocação

por lista encadeada utilizando um índice, mostra a estrutura original de um sistema de arquivos FAT usado pelo MS-DOS. Originalmente este sistema de arquivos possuía somente 4096 blocos, e então uma tabela com somente 4096 entradas de 12 bits era suficiente. Se este esquema tivesse que ser diretamente estendido para sistemas de arquivos com 2^{32} blocos, quando espaço a FAT ocuparia?



Exemplo de um arquivo armazenado no disco através do uso de uma FAT. O bloco inicial do arquivo é o 4, e cada entrada indica o próximo bloco, sendo que um X na entrada indica que o bloco é o último do arquivo. Os blocos não usados possuem entradas vazias na FAT.

Resp.: Inicialmente notemos que cada entrada da FAT original possui 12 bits para podermos endereçar todos os 4096 blocos, que são numerados de 0 até 4095. Agora, se o disco possui 2^{32} blocos, então precisaremos de 2^{32} entradas na tabela, cada uma com 32 bits, isto é, 4 bytes, pois o número de um bloco agora varia de 0 até $2^{32} - 1$. Logo, o espaço total necessário para armazenar a tabela será de $4 \times 2^{32} = 2^{34}$ bytes, um valor bem grande, se levarmos em consideração que a maioria dos processadores pode endereçar uma memória com somente 2^{32} bytes (4GB).

6. Um disco tem 4000 cilindros, cada um com 8 trilhas de 512 blocos. Uma busca leva 1ms por cilindro movido. Se nenhuma tentativa for feita para colocar os blocos de um arquivo perto um do outro, dois blocos que são logicamente consecutivos (isto é, um segue o outro no arquivo) irão requerer um tempo médio de busca que levará 5ms. Se, entretanto, o sistema operacional fizer uma tentativa de agrupar os

blocos relacionados, a distância média entre blocos pode ser reduzida a 2 cilindros e o tempo de busca reduzido para 100 microssegundos. Quanto tempo leva para ler um arquivo de 100 blocos em ambos os casos, se a latência rotacional é de 10ms e o tempo de transferência é de 20 microssegundos por bloco?

Resp.: Sempre que transferimos um bloco do disco, perdemos um certo tempo para buscá-lo no disco, mais um outro tempo para esperar que ele passe por uma das cabeças magnéticas do disco, e finalmente mais o tempo necessário para transferi-lo. Os dois últimos tempos são independentes da posição do bloco no disco. No caso de um bloco aleatório, o tempo para que o bloco passe por uma cabeça do disco será de 5ms em média (a metade do tempo de latência rotacional do disco), e o tempo de transferência será de 20 microssegundos, ou seja, 0.02ms. Já o tempo médio de busca dependerá de o sistema operacional colocar ou não os blocos relacionados próximos entre si no disco. Se o sistema operacional não colocar estes blocos próximos uns dos outros, o tempo médio de busca de um bloco aleatório será de 5ms. Logo, neste caso, o tempo total decorrido até lermos um bloco será de 10.02ms, e com isso, o tempo para ler 100 blocos será de 1002ms. Agora, se o sistema colocar os blocos relacionados próximos uns dos outros, o tempo médio de busca será reduzido para 100 microssegundos, isto é, 0.1ms. Logo, o tempo total para ler um bloco será agora de 5.12ms, e com isso, o tempo para ler 100 blocos será de 512ms.

Obs.: A resposta acima vale para o caso em que o tempo de latência rotacional é o maior tempo de espera até que um bloco alcance uma cabeça. Entretanto, caso alguma resposta tenha sido baseada em tomar 10ms como o tempo médio de latência rotacional, ela será também aceita. Nesse caso, seriam necessários 1502ms para ler os 100 blocos se o sistema operacional não colocasse os blocos relacionados próximos uns aos outros, ou 1012ms se o fizesse.