SISTEMAS OPERACIONAIS I
LISTA DE EXERCÍCIOS IV – Sistema de Arquivos e Entrada e Saída
PROF. FRIEDRICH

1) Em um sistema operacional que suporta apenas um único diretório, permitindo que o mesmo tenha um número grande de arquivos com nomes grandes, é possível simular algo parecido com um sistema de arquivos hierárquico? Como?
R:
Jorge: Acredito que sim. A maneira de fazê-lo seria colocando os arquivos contidos nos diretórios de usuário, e dos demais diretórios(usr,tmp, lib, etc), no diretório raiz. Uma restrição grave disso é que o sistema teria apenas um usuário, em minha opinião.

Bremm: Discordo da resposta acima, sistema com um diretório sofre o problema de conflito de nomes, uma solução seria usar nomes de arquivos que simulem caminho absoluto, e até mesmo de usuário. Ex: ls = Thiago/Documentos/listadecompras.txt Deise/Documentos/ listadecompras.txt... etc

Mario: A resposta do Bremm me parece correta.... jogando a estrutura de diretorios direto no nome do arquivo ia gerar uma espécie de simulação de hierarquia. Como o Jorge falou, acho q só copiar os arquivos para a mesma pasta não simularia a hierarquia.

Paulo: Se arbitrariamente nomes longos podem ser usados, então é possível simular uma estrutura de diretório multinível. Isto pode ser feito, por exemplo, usando o caractere "." Para indicar o fim de um subdiretório. Assim, por exemplo, o nome mario.pascal.F1 especifica que F1 é um arquivo dentro do subdiretório pascal que por sua vez está no diretório raiz mario.

2) Compare sistemas que suportam vários tipos de estruturas para os arquivos e outros que suportam apenas um conjunto de bytes. Aponte vantagens e desvantagens.
R:
Ramon: não achei exemplos de sistemas que utilizam vários tipos de estruturas. Mas o Unix e o Win utilizam a abordagem sequência desestruturada de bytes. Ter o sistema operacional tratando arquivos como nada mais que sequências de bytes oferece a máxima flexibilidade. Os programas dos usuários podem pôr qualquer coisa que queiram em seus arquivos e chamá-los do nome que lhes convier. Outras estruturas de arquivos são sequências e registos Árvore.
Bremm: No Tanenbaum existe a justificação organizacional de algumas estruturas de arquivos, como a sequência de registros, que tem motivação histórica para acesso via cartões de tamanho fixo, onde haveria registros estruturados de certa forma. E a árvore de registros que permitiria a classificação não linear de tipos de registros, algo útil para o acesso focado a tipos em específicos, o livro também menciona que esta estrutura ainda é usada em alguns computadores de grande porte para processar dados comerciais. Em resumo, para suporte de classificação, similar a BD.

Paulo: Alguns sistemas permitem diferentes operações de arquivo com base no tipo do arquivo (por exemplo, um arquivo ascii pode ser lido como um stream, enquanto um arquivo database pode ser lido através de um índice para um bloco). Outros sistemas deixam tal interpretação de dados de um arquivo para o processo e fornecer nenhuma ajuda no acesso aos dados. O método que é "melhor" depende das necessidades dos processos no sistema, e as demandas que os usuários envia sobre o sistema operacional. Se um sistema é constituída principalmente aplicações de banco de dados, pode ser mais eficiente para o sistema operacional implementar um arquivo tipo banco de dados e fornecer operações, ao invés de fazer cada programa implementar a mesma coisa (possivelmente de diferentes maneiras). Para sistemas de propósito geral pode ser melhor apenas implementar tipos de arquivos básicos para manter o tamanho do sistema operacional menor e permitir o máximo de liberdade para os processos no sistema.

3) Explique o propósito das operações open e close.
R: p
Deise:
Open: necessário antes de usar, o sistema deve buscar atributos e localização dos dados
Close: os acessos terminam, dados para o acesso não mais necessários, liberar espaço nas tabelas, finalizar escritas
Ramon: Do Tanenbaum
Open. Antes de usar um arquivo, um processo deve abri-lo. O propósito da chamada open é permitir que o sistema busque e coloque na memória principal os atributos e a lista de endereços do disco, para tornar mais rápido o acesso das chamadas posteriores.
Close. Quando todos os acessos terminam, os atributos e os endereços do disco não são mais necessários, portanto o arquivo deve ser fechado para liberar espaço na tabela interna. Muitos sistemas estimulam isso impondo um número máximo de arquivos abertos por processo. Um disco é escrito em blocos e o fechamento de um arquivo força a escrita do último bloco do arquivo, mesmo que o bloco ainda não esteja completo.

4) Considere um sistema que suporta 5000 usuários. Suponha que voce quer permitir 4990 destes usuários acessar um arquivo.
a) Como voce faria isto no UNIX?
R:
Ramon: no Unix temos grupos de usuários assim poderíamos colocar os 10 usuários em um novo grupo “grupo2” e os antigos usuários estariam ainda no grupo “grupo1”. Agora basta dar permissão de leitura ao grupo1 e negar acesso ao grupo2. Obs(assim quem teria permissão seria o dono e o grupo, mas para os outros seria vedado, “chmod 750”)

b) Sugira outro esquema de proteção que pode ser usado mais efetivamente para este propósito, além do UNIX.
R:

Batman: Colocar somente permissão de leitura para todos os 4990?
Diego: Criar uma lista de acesso (com permissões por usuário) e associar ao arquivo.

5) Considere um sistema que suporta as estratégias de alocação contígua, lista ligada e indexada. Qual critério deveria ser usado na decisão de qual estratégia é melhor para um determinado arquivo?
R:
Jorge: Se os arquivos possuem tamanho fixo, não são deletados, e deseja-se máxima performance nas operações de leitura, usa-se alocação contígua.
Para casos de arquivos que não são fixos, são deletados com certa frequência e não há acesso randômico à blocos, então usa-se lista ligada.
E para casos semelhantes aos da lista ligada, porém com acessos randômicos à blocos, usa-se a lista indexada.

6) O SO LQTW-2.0 trabalha com blocos de 4KB. Endereços de bloco ocupam 4 bytes. Esse sistema utiliza alocação indexada para localizar os arquivos no disco. Cada descritor de arquivo possui uma tabela com 16 endereços de blocos. Os primeiros 12 endereços são diretos. Dois endereços são indiretos simples. Os últimos dois endereços são duplamente indiretos. Qual o tamanho máximo de um arquivo nesse sistema?
R:
Jorge: tamanho de bloco = 2^12 bytes; tamanho de endereço = 2^2 bytes.
Tamanho do arquivo=(12*2^12)+{2*[2^(2*12-2)]}+{2*[2^(3**12-4)]} = aprox. 8,6GB
(Mario) *Jorge, pode explicar melhor como fez essa aqui? A minha nao bateu....*

Mario:
Tamanho do arquivo = Endereçamento Direto + End. Indireto Simples + End. Indireto Duplo

[1 bloco de enderecos] = 4096 / 4 = 1024
[1 bloco de dados] = 4096

ED = 12 * [1 bloco de dados] = 12 * 4096 = 49152
EIS = 2 * [1 bloco de endereço] * [1 bloco de dados] = 2 * 1024 * 4096 = 8388608
EID = 2 * [1 bloco de endereco] * [1 bloco de enderecos] * [1 bloco de dados] = 2 * 1024 * 1024 * 4096 = 8589934592

Total = 8598372352 bytes => / 1024^3 => ~ 8.0078 GB

7)Um disco CDROM contém um sistema de arquivos no qual todos os arquivos são imutáveis. Qual método de alocação, entre alocação contígua, encadeada e indexada, é o

mais apropriado? Justifique sua resposta.
R:
Jorge: É a alocação contígua. Porque é a ideal para quando os arquivos possuem tamanho fixo e não são deletados.
Diego: Complementando a resposta do Jorge... A alocação contígua é a que menos desperdiça espaço (com tabelas, encadeamentos e afins), e, como os arquivos não "crescem" ou são apagados, não há a necessidade de efetuar as operações custosas de compactação e expansão (pontos fracos desse método de alocação).

8) Considere um sistema de arquivos em disco que tem blocos, lógico e físico, de 512 bytes. As informações sobre cada um dos arquivos já está em memória. Para cada uma das 3 estratégias de alocação (contígua, lista e indexada) responda:
a) Como é o mapeamento de endereço lógico-físico? (indexado o arquivo é sempre menor que 512 blocos)
R:

b) Se voce está no bloco lógico 10 (último acessado) e deseja acessar o bloco lógico 4, quantos blocos físicos deverão ser lidos?
R:

9) O SO Linux utiliza uma variação do esquema de alocação indexada. Ele contém 15 blocos diferentes de armazenamento. Os primeiros 12 blocos do arquivo são indexados diretamente a partir dos apontadores (12 primeiros) contidos no i-node. Os últimos 3 apontadores são usados como apontadores indiretos (simples, duplo e triplo). Considerando que os blocos são de 4KB e os apontadores são de 32bits, qual é o tamanho que os arquivos podem ter?
R:
Jorge: 12*2^12 + 2^2*12 - 2 + 2^3*12-4 + 2^4*12 - 6 = calcular

10) Compactar o disco periódicamente pode ser produtivo? Explique.
R:
Jorge: Depende. Se ser produtivo é economizar espaço em disco , para poder alocar mais dados então sim. Porém se ser produtivo é ter o máximo de velocidade de acesso e disponibilidade do disco para operações, então não . Porque a operação é cara, ou seja, pode demorar muito e tornar o disco temporariamente indisponível para o uso.
Fabio: Se o tempo utilizado para desfragmentação não for considerado, neste caso, também será produtivo em velocidade de acesso (dados contíguos, menor tempo de deslocamento de cilindro/trilha, etc.).

11) Quando um arquivo é removido, seus blocos são geralmente colocados de volta na lista de livres, mas não são apagados. O que voce acha da idéia do sistema operacional apagar cada bloco antes da liberação? Considere fatores como segurança e desempenho e explique
o efeito de cada um.

R:
Jorge: Em matéria de segurança é o ideal, pois não há como recuperar os arquivos através de algum algoritmo para recuperar as entradas deles na FAT. Em matéria de desempenho é ruim pois é custoso a operação de, literalmente, apagados blocos.

12) Considere um sistema onde o espaço livre é mantido em uma lista de espaços livres.
a) Suponha que o apontador para a lista é perdido. O sistema consegue reconstruí-la. Explique.
R:
Fabio: Creio que seria possível através do complemento da lista de arquivos usados, porém não seria o ideal porquê a lista também pode estar corrompida.

b) Sugira um esquema que assegura que o apontador nunca será perdido por uma falha de memória.
R:
Jorge: Gravar o endereço do apontador em um arquivo, em disco.

13) Considere os seguintes cenários de E/S em um PC – único usuário.
a. Um mouse usado com uma interface de usuário gráfica (GUI)
b. Uma fita em um sistema operacional multitarefa (assumir que não existe pré alocação de dispositivo)
c. Um disco contendo arquivos de usuário
d. Uma placa gráfica com conecção direta ao barramento, acessível através de E/S mapeada em memória.
Para cada um destes cenários de E/S, voce projetaria o sistema operacional para usar buferização, spooling, caching ou uma combinação? Voce usaria E/S programada ou E/S dirigida por interrupção? Justifique suas escolhas.
R:
Jorge:
a. spooling + E/S por interrupção:porque o mouse é um dispositivo que exige uma resposta rápida aos comandos do uusuário.
b.nao sei ainda
c.Caching + E/S programada: caching porque oferece maior desempenho ao disco e E/S programada para somente interromper o CPU quando realizar suas operações pendentes.
d.bufferização + E/S programada: buffer para trabalhar com armazenamento de texturas etc.. e E/S programada pelos mesmo motivos de um disco.

14) Explique de que modo um SO pode facilitar a instalação de um novo dispositivo sem a necessidade de recompilação do sistema.
R:
Jorge: Ler conceito sobre plug & play
Fabio: Com a utilização do VFS, virtual file system, que disponibiliza um meio-termo para os dispositivos. Uma interface em comum com unix, fat, ext2/3, por.ex.

15) Em qual das quato camadas do software de E/S se realiza cada uma das seguintes atividades:
(a) Calcular a trilha, setor e cabeçote para a leitura de disco.
R:
Jorge: Driver
(b) Escrever comandos nos registradores do dispositivo.
R:
Jorge: Tratadores de interrupção
(c) Verificar se o usuário tem permissão para usar o dispositivo.
R:
Jorge: Software independente do dispositivo
(d) Converter inteiros binários em ASCII para impressão.
R:
Jorge: Hardware

16) Considere um disco com 200 trilhas (0-199) sendo que a cabeça de leitura/gravação está atualmente atendendo uma requisição na trilha 143 (última atendida 125). A fila de requisições é mantida em ordem FIFO: 86, 147, 91, 177, 94, 150, 102, 175, 130.
Qual a quantidade de movimentos (trilhas) da cabeça de leitura/gravação para atender os pedidos da fila nos seguintes algoritmos de escalonamento de disco:
a) FCFS R: 565 (acesso na ordem da fila, tem o problema de não racionalizar os movimentos do braço de leitura)
b) SSTF R: 162 (acesso buscando "menor esforço", escolhendo o menor tempo de seek possível a cada iteração, pode gerar concentração em determinada região, nunca dando oportunidade aos processos distantes. Ordem: 147 150 130 102 94 91 86 175 177)
c)SCAN R: 125 (acesso estilo "elevador", vai até o último item em um sentido, para depois percorrer as trilhas no sentido inverso. Ordem: 147 150 175 177 130 102 94 91 86)
d) C-SCAN R: 167 (acesso num único sentido, tratando a fila como uma lista circular, vai até o final e depois retorna para o início, seguindo no mesmo sentido. Ordem: 147 150 175 177 86 91 94 102 130)

17) As requisições de disco não são, normalmente, distribuídas. Por exemplo, os cilindros nos quais as estruturas de um diretório de arquivos estão gravadas são acessadas mais frequentemente que a maioria dos arquivos. Considere que 50% das requisições são para um pequeno número fixo de cilindros.
a) Qual dos algoritmos de escalonamento de disco seria o melhor?
R:
Jorge: Como cerca de 50% das instruções são para um pequeno número FIXO de cilindros, acredito que o algoritmo SSF seja o ideal, porque prioriza o seek time e não haverá um problema de leitura de blocos centralizada.
Talvez N-SCAN também resolva.
Também pode depender da posição onde os cilindros se encontram próximos, o SSF resolve elas, porem se estiverem espalhadas, um N-SCAN talvez seja o ideal.

b) Sugira um novo algoritmo de escalonamento de disco para este caso.
R:

18) Considerando que as memórias tem se tornado cada vez maiores e mais baratas, poderíamos, quando da abertura de um arquivo, buscar uma nova cópia do i-node do arquivo para a tabela de i-nodes, ao invés de procurar pela tabela inteira para ver se o inode está na mesma. Voce acha um procedimento correto? Explique.
R:
Jorge: Não acho que seja correto porque se um arquivo for aberto muitas vezes, e cada uma delas é gerada uma nova cópia do i-node, pode acontecer de haverem muitas cópias rodando e ocasionar problemas de consistência e coerência de dados, não?

Mario: Acho que o problema de fazer o que ele sugere é que voce estara fazendo um acesso a disco a cada nova busca de i-node. Assim, mesmo com uma memoria maior e mais barata, quem estara te limitando sera o disco.
O melhor nesse caso eh manter entradas de inodes em tables (na memoria -> grande/barata) e fazer a busca nela.