SO joão
Walter
Cleverson
Kossmann
Thiago(magrão)
paulo
Fernando
Daniel

Véio

Lembrete pra substitutiva:
Caiu na prova 2 de sexta passada questões referentes aos exercicios
5 (especialmente letra d)
6
7
8 (certeza)
9
13
Teve mais coisas sobre virtualização, mas não sei informar direito

1. Quais as vantagens e desvantagens de guardar o nome do programa criador junto aos atributos do arquivo?
Vantagem: Livra o sistema operacional de saber com qual aplicativo deve ser aberto o arquivo, facilita também a vida do usuário por ter acesso rápido e simples a leitura e edição do arquivo.
Desvantagem: Um terceiro pode alterar o arquivo para apontar para um programa mal intencionado.

2. Na semântica de sistema de arquivos do Unix o que acontece quando um arquivo que está aberto é removido?

(Sei q pra cada arquivo o file system mantem um contador que indica quantos procesos o estão usando, quando é removido o arquivo se o contador não for 0 é apagado somente o acesso ao mesmo, mas o arquivo é mantido no disco até que todos os processos se encerrem)

Quando um arquivo é apagado (deletado) em um sistema GNU/Linux ele é "desconectado" (unlinked) do sistema de arquivos, o inode que contém os dados do arquivo não é apagado até que todos os processos o liberem, é por isso que processo podem continuar usando arquivos que foram apagados.

3. No NFS (Network File System) o que é o "Silly Rename"? Porque surgem os arquivos .nfsXXXXXX ?
É um sistema de arquivo clin

(Pelo q eu lembro o NFS é stateless, e não sabe quais arquivos estão sendo utilizados, assim pra manter compatibilidade com os programas linux (onde um arquivo aberto pode ser excluído sem problemas) ele renomeia o arquivo e deixa ele oculto, quem estava com o arquivo aberto continua lendo deste .nfsX)

Retirado do FAQ do NFS: (http://nfs.sourceforge.net/

Q: What is a "silly rename"? Why do these .nfsXXXXX files keep showing up?
A: Unix applications often open a scratch file and then unlink it. They do this so that the file is not visible in the file system name spas is known as a "silly rename." Note that NFS servers have nothing to do with this behavior. ce to any other applications, and so that the system will automatically clean up (delete) the file when the application exits. This is known as "delete on last close", and is a tradition among Unix applications.
Because of the design of the NFS protocol, there is no way for a file to be deleted from the name space but still remain in use by an application. Thus NFS clients have to emulate this using what already exists in the protocol. If an open file is unlinked, an NFS client renames it to a special name that looks like ".nfsXXXXX". This "hides" the file while it remains in use. Thi
After all applications on a client have closed the silly-renamed file, the client automatically finishes the unlink by deleting the file on the server. Generally this is effective, but if the client crashes before the file is removed, it will leave the .nfsXXXXX file. If you are sure that the applications using these files are no longer running, it is safe to delete these files manually.
The NFS version 4 protocol is stateful, and could actually support delete-on-last-close. Unfortunately there isn't an easy way to do this and remain backwards-compatible with version 2 and 3 accessors.

4. Discuta os fatores envolvidos na escolha do tamanho de bloco utilizado por um filesystem.
(Tamanho dos arquivo que serão salvos, em um bloco de 300 e um arquivo que consome 100, 200 são desperdiçados (mas possibilitam que o arquivo aumente sem criar novos blocos))

O espaço em disco é sempre alocado em blocos, portanto, uma porção do último bloco de cada arquivo é em geral desperdiçada. Se cada bloco tiver 512 bytes, então um arquivo com 1949 bytes seria alocado em quatos (2048 bytes); os últimos 99 bytes seriam desperdiçados. Estes bytes desperdiçados são fragmentação interna. Todos os sistemas de arquivos são afetados pela fragmentação interna; quanto maior o tamanho do bloco, maior a fragmentação interna.
Quanto um arquivo pode ser ampliado dentro de um mesmo espaço alocado é definido, também, pelo tamanho do bloco utilizado, quanto maior o bloco, mais o arquivo pode ser posteriormente ampliado sem a necessidade de alocação de um novo bloco, o que é um problema em alguns métodos de alocação, como a alocação contígua.

Os tamanhos de bloco são definidos durante a criação do sistema de arquivos de

acordo com o uso pretendido para o sistema de arquivos: se muitos arquivos pequenos forem esperados, o tamanho do fragmento deverá ser pequeno; se repetidas transferências de arquivos grandes forem esperadas, o tamanho do bloco básico deve ser grande.

Ou seja, tamanho de bloco maior é desejável por questões de velocidade, mas quanto maior o tamanho do bloco, maior a fragmentação interna.

5. Sobre métodos de alocação de espaço, responder:

5a) Como funciona a alocação de espaço contígua? Que problemas apresenta?

(Pelo q eu lembro fragmentação, tem que ver o que acontece quando um arquivo aumenta seu tamanho)

A alocação de espaço contígua requer que cada arquivo ocupe um conjunto de blocos contíguos no disco. Os endereços de disco definem um ordenamento linear no disco. Com isso, o número de operações de busca no disco, exigidos para acesso a arquivos alocados contiguamente, é mínimo, assim como o tempo de busca quando uma operação de busca for necessária.

Como problemas encontrados na alocação contígua temos:

- a dificuldade de encontrar espaço para um novo arquivo. Sistemas de gerenciamento de espaço livre devem ser utilizados para realizar esta operação, mas tais sistemas geram fragmentação externa;
- a dificuldade de determinar a quantidade de espaço necessário para um arquivo. Pela falta de conhecimento na hora da criação do arquivo, ou para futura ampliação do mesmo. Em geral, o espaço necessário é superestimado. Gerando assim fragmentação interna.

5b) Como funciona a alocação de espaço por encadeamento (linked allocation). Quais os problemas apresentados por este método?

Com a alocação encadeada, cada arquivo é uma lista encadeada de blocos de disco; estes blocos podem estar espalhados por qualquer lugar do disco. O diretório contém um ponteiro para o primeiro e último blocos do arquivo. E cada bloco contém um ponteiro para o proximo bloco. Não gera fragmentação externa.

Como problemas encontrados na alocação por encadeamento temos:

- O principal problema é que ela so pode ser efetivamente utilizada para arquivos de acesso sequencial.
- O espaço perdido com a alocação dos ponteiros, logo, cada arquivo irá requerer um pouco mais de espaço do que necessitaria em outra situação.**va 2 de sexta**
- O problema da confiabilidade, como a integridade do arquivo é mantida por intermedio de ponteiros espalhados por todo o disco, se um desses ponteiros for corrompido, pode-se ter grandes perdas de dados ou de espaço livre.

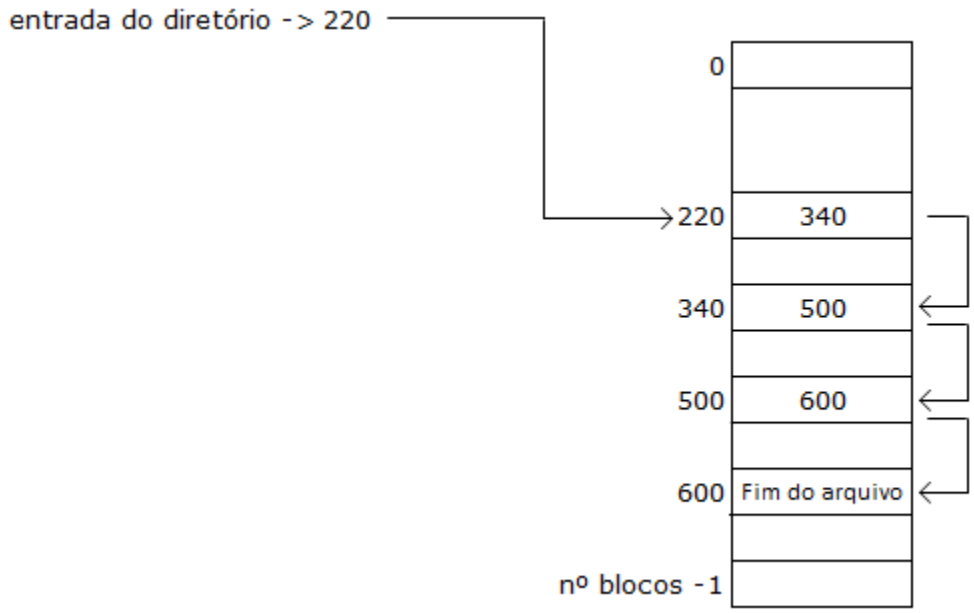
5c) Como funciona a FAT (File-Allocation Table) usado pelo MS-DOS, qual a diferença em relação à alocação por encadeamento simples?

A FAT usa uma tabela de alocação de arquivos, esta tabela fica alocada em uma seção do disco no inicio de cada partição. A FAT tem uma entrada para cada bloco do disco e é indexada pelo número do bloco. A entrada da tabela, indexada pela entrada do arquivo no diretório, contém o número do bloco do próximo bloco no aquivo. Assim continua até o bloco que contém o identificador de fim de arquivo. Ao contrario da alocação por encadeamento a FAT não utiliza um espaço em cada bloco do disco como ponteiro (utiliza os blocos com valor igual a zero na tabela que representam blocos livres).

5d) Mostre como ficaria a tabela FAT para um arquivo composto pelos seguintes blocos: 220, 340, 500, 600.

Isso é fat mesmo? Não é uma lista encadeada? Isso aí parece ter ponteiros nos blocos, apontando para outros blocos, não?

É verdade, mas o fato é que essa tabela tem tamanho fixo (não lembro de quanto era).. Daí não precisa dos ponteiros, mas quando o disco é muito grande você precisa de mais tabelas pra poder mapear o restante do disco... Daí tem que usar a FAT32 ou um outro tipo... Não sei se responde totalmente a pergunta..



Essa imagem tá errada.
O desenho certo tá em "Alocação Encadeada - FAT" desse arquivo:
d
(Eu tive que baixá-lo e converter para pdf pq deu pau)

5e) Em um sistema de arquivos usando uma FAT o que acontece se ocorrer a corrupção total da tabela de alocação de arquivos?
If the FAT table (Table 1) becomes corrupted, or what is known as disk errors, there is a second (backup) table (Table 2) available for programs like Scandisk uses to fix the FAT (Table 1). A

corrupt FAT can cause you to get the message "No Bootable Media". At this point if Scandisk cannot fix it you will need to format and start over.
Não é possível recuperar caso ocorrer um erro na tabela de backup também, pois as ligações de um bloco com outro para formar um arquivo só existe na tabela.

5f1) Como funciona a alocação de espaço indexada? Que problemas da alocação usando encadeamento ela tenta resolver?
A alocação indexada utiliza ponteiros, como a alocação por encadeamento, mas agora todos esses ponteiros são armazenados juntos em um bloco de índices. Cada arquivo possui seu próprio bloco de índices que é um array de endereços de blocos de disco. O diretório contém o endereço do bloco de índices, e a í-esima entrada do bloco de índices aponta para o í-esimo bloco do arquivo. Desta maneira é possível realizar com facilidade o acesso direto ao arquivo.
A alocação indexada tenta resolver o problema da falta de eficiência no acesso direto a arquivos, que ocorre quando se usa a alocação por encadeamento.

5f2) Dentro da alocação indexada quais são os mecanismos usados para guardar os índices?
A alocação indexada causa um desperdício de espaço em disco por conta do armazenamento do bloco de índices. Pois um bloco inteiro de índices deve ser armazenado para cada arquivo, mesmo que apenas um ou dois ponteiros sejam realmente utilizados. Para reduzir esse desperdício devem ser utilizados blocos de tamanho menor para armazenar os índices. Mas blocos menores limitam a indexação de grandes arquivos. Para resolver este problema alguns mecanismos são usados para guardar os índices. Eles são:
- Esquema encadeado: Diversos blocos de índices encadeados, no caso, o ultimo endereço do bloco de índices é nulo (caso o bloco represente todo o arquivo) ou é um ponteiro para outro bloco de índices (para grandes arquivos).
- Índice multinível: Um bloco de índices de primeiro nível aponta para um conjunto de blocos de índices de segundo nível, que por sua vez aponta para os blocos do arquivo. Esta abordagem poderia ser estendida para um terceiro ou quarto níveis dependendo do tamanho máximo de arquivo requerido.
- Esquema combinado: Utiliza uma combinação dos dois esquemas anteriores.

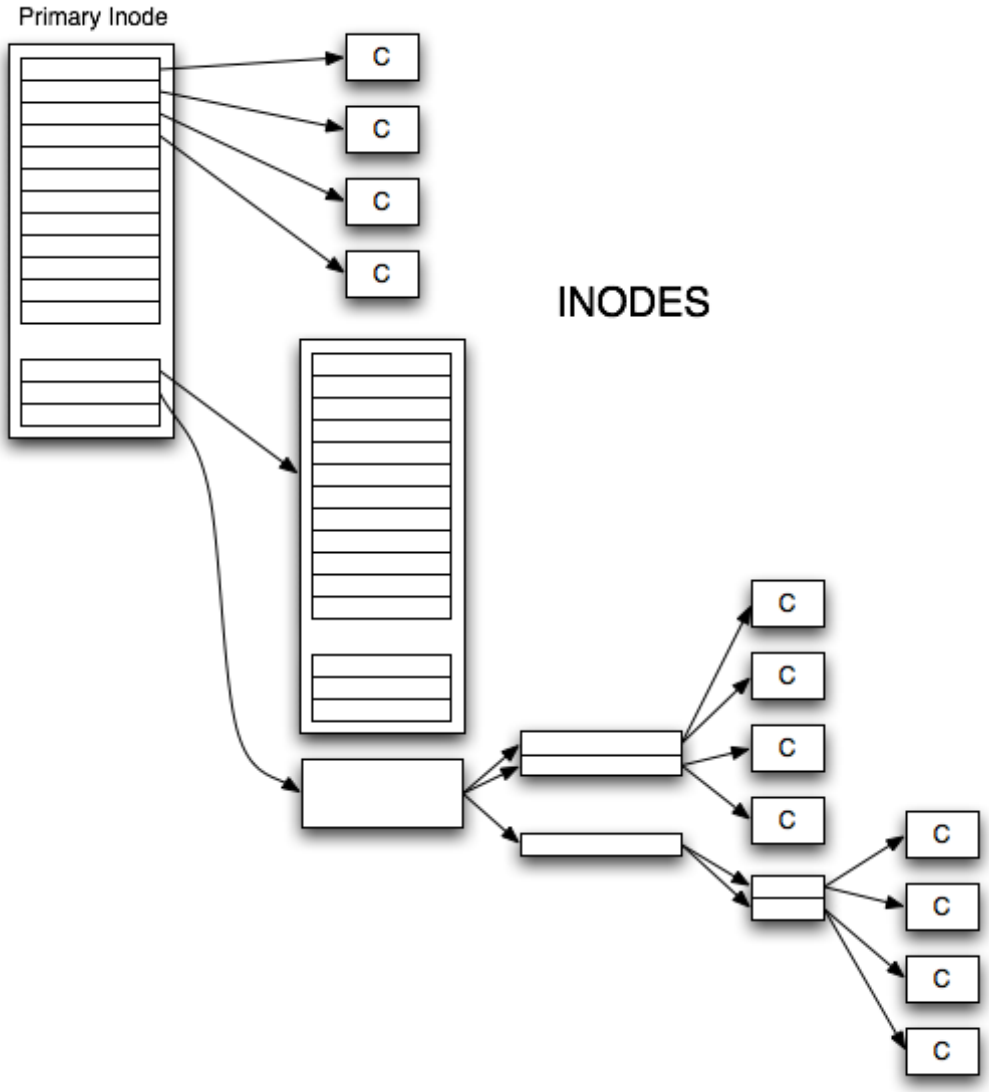
5g) O que são i-nodes? Em quais mecanismos de alocação indexada eles se enquadram?
i-none(index-node) é uma estrutura de dados que relaciona os atributos e os endereços em disco dos blocos do arquivo. Mecanismos: alocação indexada combinada

!!! ACHO QUE A RESPSOTA ABAIXO ESTÁ ERRADA !!!

Acho que tá certo. Pelo menos essa parte de 12 primeiros ponteiros para blocos diretos (o que é bom para caso você esteja lidando com arquivos pequenos), e esses três inodes que começam a envoluir uma hierarquia são bons para armazenar arquivos grandes. Desta maneira você tem uma estrutura capaz de armazenar tanto arquivos grandes, como pequenos.

I-node é o primeiro bloco encontrado quando se faz o acesso a um arquivo e consite em uma estrutura de dados que relaciona os atributos e os endereços em disco dos blocos do arquivo , este bloco de índice utiliza um mecanismo de alocação indexada do tipo combinada. Em geral

utiliza-se um esquema onde os 12 primeiros ponteiros apontam para blocos diretos, ou seja, que contem dados do arquivo; os outros 3 ponteiros apontam para blocos indiretos. O primeiro ponteiro para um bloco indireto é o endereço de um bloco indireto simples (um bloco de índices que não contém dados, mas endereços de blocos que contêm dados). Depois existe um ponteiro para um bloco indireto duplo (contém o endereço de um bloco que contém os endereços de blocos que contêm ponteiros para os blocos de dados reais). O último ponteiro conteria o endereço de um bloco indireto triplo.



Um arquivo é representado por um inode. Um inode é um registro que armazena a maior parte das informações sobre um arquivo específico no disco. O nome inode deriva de index node. O inode contém os identificadores de usuário e o grupo do arquivo, as datas de última modificação e último acesso, um contador do número de hard links ao arquivo, e o tipo do arquivo. Além disso o inode contém 15 ponteiros aos blocos de disco que mantêm os dados do arquivo. Doze apontam para blocos diretos e

3 para blocos indiretos.

6. Considere um sistema com i-nodes com 15 ponteiros. Nestes i-nodes os 12 primeiros são usados para alocação direta; o 13 ponteiro para alocação simples indireta; o 14 para alocação dupla indireta e o 15 para alocação tripla-indireta. Considere ainda que cada bloco tem 512Kb. Como ficaria a estrutura de inodes para armazenar um arquivo de 129024 bytes (252 blocos)?

[1] - 1 bloco
[2] - 1 bloco
[3] - 1 bloco
[4] - 1 bloco
[5] - 1 bloco
[6] - 1 bloco
[7] - 1 bloco
[8] - 1 bloco
[9] - 1 bloco
[10] - 1 bloco
[11] - 1 bloco
[12] - 1 bloco (12 blocos)

[13] - []x15 - 15 blocos

[14] - []x15 - []x15 - 225 blocos

[15] - [1] - [1] - []x10 - 10 blocos

Concordo com o Fernando...

Não entendi sua solução, aí você calculou o tamanho máximo de arquivo a ser salvo, não?

Pra salvar um arquivo de 252 blocos, no caso, seria os primeiros 12 blocos do primeiro i-node ocupado, depois mais 15 blocos do i-node nível 1. Depois vem o i-node de nível 2, que ficaria os quinze níveis dele apontando cada um para um inode distinto, e cada i-node destes teria 15 blocos; Ou seja, mais 225 blocos, e totalizando 242 blocos, faltam 10. O i-node de nível 3 apontaria para um de nível dois, que usaria apenas um de seus apontadores, pra um de nível 1, que salvaria 10 blocos nele. Algo assim. Faz algum sentido? =P

[] -> 1 bloco
[] -> 1 bloco
[] -> 1 bloco

[] -> 1 bloco
[] -> 1 bloco
[] -> 1 bloco
[] -> 1 bloco
[] -> 1 bloco
[] -> 1 bloco
[] -> 1 bloco
[] -> 1 bloco
[] -> 1 bloco
[] -> [] x15-> 12 blocos
[] -> [] x15-> [] x15 -> 225 blocos
[] -> [] x1 -> [] x1 -> [] x10 -> 10 blocos

No total você ocuparia 268 i-nodes, CASO isso esteja correto. Alguém confirma?

Essa é a resposta certa pra mim. Não precisa entrar em detalhes dos kbs que o kossmann colocou aqui embaixo

(mudei a resposta que estava antes pq estava errada)

i-node principal
01 - [-> um bloco de 512 -> ocupa
02 - [-> um bloco de 512 -> ocupa
03 - [-> um bloco de 512 -> ocupa
04 - [-> um bloco de 512 -> ocupa
05 - [-> um bloco de 512 -> ocupa
06 - [-> um bloco de 512 -> ocupa
07 - [-> um bloco de 512 -> ocupa
08 - [-> um bloco de 512 -> ocupa
09 - [-> um bloco de 512 -> ocupa
10 - [-> um bloco de 512 -> ocupa
11 - [-> um bloco de 512 -> ocupa
12 - [-> um bloco de 512 -> ocupa
13 - [-> simples indireta (15 ponteiros, 1 i-node) -> ocupa
14 - [-> duplo indireto (225 ponteiros, 16 i-nodes) -> ocupa
15 - [-> triplo indireto (3375 ponteiros, 241 i-nodes) -> nil (não usa)

Total de ponteiros: 12+15+225+3375= 3627 ponteiros

Tamanho máximo do arquivo: 3627*512Kb=1857024Kb= 1,77Gb

Fazendo um adendo: O tamanho de cada bloco depende do File System, em casos como no UNIX os blocos podem chegar até a 4Kb.

Lembrando que: O deslocamento do arquivo na estrutura de arquivo na memoria principal é mantido em uma palavra de 32 bits. Portanto, os arquivos não podem ser maiores q 2^{32} bytes ou 4Gb.
Em sistemas de 64 bits, o tamanho alcançavel dos arquivos pode chegar a terabytes.

7. Qual a diferença de simbolic links e hard links? E como é a implementação de cada um? Como é feito o controle de remoção de arquivo em cada caso?
Um link simbólico é um novo i-node que tem apenas como conteúdo a referência para o arquivo original (ou seja não ocupa quase nenhum espaço). Uma coisa importante: a referência do link simbólico pode ser relativa, neste caso se mover o link para outro lugar, perde a referência de conteúdo.
O arquivo no qual foi feito o link simbólico pode estar em qualquer lugar pois o mesmo contém um path do mesmo.

Um hard-link possui o mesmo i-node do arquivo original, ou seja é apenas feito uma outra referência no diretório onde o hardlink foi criado para o i-node do arquivo original.
Quando um hard-link é criado, é atualizado um contador no i-node somando +1 para cada hard-link. O arquivo só é removido quando todos os hard-links forem removios.
Só é possível fazer o hard-link de um arquivo do mesmo disco, pois como o mesmo possui o memso i-node do arquivo original não é possível ter o i-node de um arquivo que está em outro disco ou lugar.

8. Considere um sistema onde o espaço livre é gerenciado por uma lista de espaço livre. Suponha que o ponteiro para a lista de espaços-livres foi perdido. Esta lista pode ser reconstruída? Explique sua resposta.

Acredito que sim. Uma possível solução seria veirificar todos os blocos utilizados pelos diretórios/arquivos/sistema e considerar os outros como livres.
Deve-se pensar no fato de que ao um arquivo ser deletado, não necessariamente os dados que ele tem são deletados nos blocos (geralmente que eu saiba não, é apenas deletado a ligação para ele). Então a solução de percorrer por todos os blocos e pegar os livres não funciona. Mas acredito que a solução acima funciona, só que pensando mais nela, ela vai dar um trabalho do cão... mas parece ser possível sim.

Bom, você gerenciar o espaço livre desta maneira, não quer dizer que você não possa ver o espaço ocupado. Uma solução, creio eu, seria você percorrer todos os i-nodes . "Vendo" um i-node você tem como saber se ele está livre ou ocupado. É uma tarefa bem força bruta mesmo. Junto nessa tarefa, você pode ver se um i-node está marcado como "ocupado", e não tem nenhum outro i-node ou diretório apontando/relacionado à ele. Caso isso aconteça, o conteúdo deste i-node e do que mais for relacionado à ele, vai pra pastinha /lost+found lá. (Seria legal se alguém revisasse isso pois não tenho muita certeza de algumas coisas).

9. Quais são os métodos para gerenciamento de espaço livre?
(INCOMPLETO AINDA)
- Vetor de Bits (bitmap)
Cria um vetor onde cada posição corresponde a um bloco do disco, e possui valor 0 para livre e 1 para ocupado.
Vantagem: simplicidade e eficiência para encontrar o primeiro bloco livre.
Desvantagem: ineficiente ao menos que sejam mantido na memória principal (senão fica muito custoso o acesso repetidamente ao disco) e ocupa muito espaço
- Lista Encadeada

Desvantagem:para varrer a lista é preciso ler cada bloco

-Vetor de bits:
A lista de espaço livre é implementada como um mapa de bits ou um vetor de bits. Cada bloco é representado por 1 bit. Se o bloco estiver livre o bit será 1; se o bloco estiver alocado o bit será 0. A principal vantagem é que é relativamente simples e eficiente encontrar o primeiro bloco livre, ou n blocos livres consecutivos no disco.
Mas são ineficientes a não ser que todo o vetor seja mantido na memória principal.
-Lista Encadeada:
Encadear todos os blocos de disco livres, mantendo um ponteiro ao primeiro bloco livre em uma posição especial no disco e armazenando-os em cache na memória. Esse primeiro bloco contém um ponteiro ao próximo bloco livre de disco, e assim por diante.
Não é eficiente, para percorrer a lista, precisamos ler cada bloco, o que requer tempo substancial de I/O.
-Agrupamento:
Armazena os endereços de n blocos livres no primeiro bloco livre. Os primeiros n-1 desses blocos estão realmente livres. O bloco final contém os endereços de outros n blocos livres, e assim por diante. Assim os endereços de um grande número de blocos livres podem ser rapidamente encontrados.
-Contadores:
Em vez de manter uma lista de n endereços de disco livres, mantêm o endereço do primeiro bloco livre e o número n de blocos contíguos livres que seguem esse primeiro bloco. Cada entrada na lista de espaço livre consiste então em um endereço de disco e um contador.Embora cada entrada exija mais espaço que um endereço de disco simples, a lista global fica mais curta. Aproveita o fato de que em geral vários blocos contíguos podem ser alocados ou liberados simultaneamente.

10. Quais as etapas de um checkagem de consistência de um file system?
FALTA ORGANIZAR, SEPARAR, MELHORAR
The consistency checker compares the data in the directory structure with the data blocks on disk, and tris to fix any inconsistencies it finds. The allocation and free-space management algorithms dictate what types of problems the checker can find, and how successful it will be in fixing those problems. For instance, if linked allocation is used and there is a link from any block

to its next block, then the entire file can be reconstructed from the data blocks, and the directory structure can be recreated. The loss of a directory entry on an indexed allocation system could be disastrous, because the data blocks have no knowlegde of one another. For this reason, UNIX caches directory entries for reads, but any data write that results in space allocation generally causes the inode block to be written to disk before the corresponding data blocks are.
Fernando: tem aquele exemplo lá que ele falou dos inodes.. ve se todo inode pertence a um arquivo, se ele não tiver livre e não pertencer a arquivo mapeado algum, ele via pra pasta lost+found lá
tmb tem aquele negócio de contar quantos hard links apontam pra um arquivo
pra qdo for 0 tu deletar ele

A verificação de consistência compara os dados na estrutura de diretório com os blocos de dados no disco e tenta corrigir quaisquer inconsistências encontradas. Os algoritmos de alocação e gerência de espaço livre determinam os tipos de problemas que o verificador poderá encontrar e qual será a taxa de sucesso na correção desses problemas.

11. Como funcionam os sistema de arquivos baseados em Journaling? Quais suas vantagem? O que tentam garantir?

Um dos objetivos de um sistema de arquivos baseado em "journaling" é evitar a execução de verificações de consistência no sistema de arquivos inteiro que consumam muito tempo, substituído isso pela verificação a uma área especial de disco que contém as operações mais recentes de escrita no disco, esta área é chamada "journal". Remontar um sistema de arquivos após uma falha de sistema é uma questão de poucos segundos.

Eles salvam em uma área especial do disco as operações que vão realizar, antes de efetivar elas. Isso é bom pois evita a corrupção de dados. Numa queda de luz durante uma operação, por exemplo, ele poderia verificar no journal o que estava fazendo, o que já foi feito, e o que falta fazer. Seja para continuar a operação ou reverter ela.

12. Explique os seguintes algoritmos de escalonamento de requisições de disco: FCFS, SSTF (Shortest-Seek First), SCAN, C-SCAN e LOOK.

(Este aqui inclui o esqueminha do elevador, são os métodos para ordenar as requisições de leitura do disco de forma que a agulha do hd se movimente menos / leia tudo rapidamente.
Lembrem que aqui pode dar estarvation)

FCSF, provavelmente é FCFS: First Come First Serve

Equivalente a FIFO (First In First Out), as requisições são atendidas à medida que elas são recebidas, não é feito nenhum tipo de ordenamento.

Atende todas as requisições ao contrário do SSTF que pode levar a "starvation" tendo pedidos não atendidos.

calculo de custo dado a seguinte sequencia: 98 183 37 122 14 124 65 67

iniciando no 53: 98-53= 35; 183-98=85; 183-37=146; 122-14=108; 124-14=110; 124-65=59; 67-65=2;
O custo então será 35+85+146+108+110+59+2

SSTF

A política do SSTF é atender à requisição que necessite da menor movimentação a partir do ponto atual. Este algoritmo é uma melhoria direta do FCFS (FIFO) e mantém um buffer de requisições que chegam, junto com cada requisição é aramzenado o número do cilindro da requisição, números de cilindros menores indicam que o cilindro está próximo ao eixo (spindle) enquanto que números maiores indicam que o cilindro está distante do centro.

O benefício direto do SSTF é a simplicidade e ele é claramente mais vantajoso que o FIFO em termos de movimentação da cabeça, resultando em um menor tempo médio de resposta. No entanto, como o buffer está sempre recebendo novas requisições, essas podem elevar o tempo de resposta para os dados que estão distantes da posição atual da cabeça, se todas as requisições novas forem próximas da posição atual. O resultado pode ser "starvation" e as requisições distantes nunca conseguirão progredir.

O "algoritmo do elevador" é uma das formas de reduzir o tempo de resposta e o movimento e garantir a consistência do atendimento das requisições.

Usando a mesma sequencia do FCFS:
53->65->67->37->14->98->122->124->183
  12+ 2+ 30+ 23+ 84+ 24+   2+   59=custo

SCAN e C-SCAN (algoritmo do elevador)

O braço do disco move-se em uma única direção a

A implementação mantém um buffer de requisições de leitura/escrita pendentes juntamente com o número do cilindro associado a cada requisição. Quando uma nova requisição chega e o drive está "idle", a movimentação da cabeça/braço será na direçãodo cilindro onde o dado está armazenado. Quando as próximas requisições chegarem, elas serão atendidas somente na direção atual do movimento do braço, até que o mesmo atinga o fim do disco, nesta ocasião o braço muda de direção e as requisições remanescentes são atendidas, e assim por diante.

Nem sempre o algotirmo do elevador é melhor que o SSTF, que é mais próximo do ótimo, mas pode resultar em grande variação do tempo de resposta e mesmo em situações de starvation onde novas requisições continuamente são atendidas antes das requisições já existentes. Técnicas anti-

starvation podem ser aplicadas ao SSTF para garantir um tempo de resposta ótimo.

Uma das variações do método SCAN é o C-SCAN (Circular Elevator Algorithm), nele todas as requisições são atendidas em uma única direção, quando a cabeça chega no limite mais externo do disco, ela retorna para o início e recomeça a atendar as requisições (é possível que a direção única seja de fora para dentro, mas é sempre uma direção única). O resultado é um desempenho mais próximo do igual para todas as posições da cabeça, pois a distância esperada a partir da cabeça é sempre metade da distância máxima, ao contrário do algortimo do elevator padrão onde os cilindros no meio serão atendidos duas vezes mais do que os cilindros das extremidades.

!!! É preciso verificar com o Bona, em sala ele disse que no SCAN as pontas são atendidas mais vezes, as referências on-line apontam isso como "ruim", o meio é atendido a tempos constantes enquanto as pontas tem que esperar duas transições completas para serem atendidas novamente, ou seja, as pontas levam vantagem ou não no SCAN? (O C-SCAN procura corrigir exatamente esse problema).

Desvantagem: Assim, você tem os dois extremos A e B. Você passa pelo discos atendendo a requisições de A até B, e depois de B até A no Scan. O que acontece, é que você atende os dois extremos duas vezes, com um intervalo de tempo muito pequeno entre elas - neste tempo novas requisições mal aparecem. E depois de uma extremidade atendida, vai demorar até que as leituras voltem para esta região do disco. Então no final, acaba se tornando desvantajoso.

É como um sofá com várias pessoas comendo pipoca, os que estão no meio comem mais do que os que estão na ponta, pois eles recebem o pote de pipoca na ida e na volta.

SCAN(bate e volta)
Usando a mesma sequencia do FCFS: 0-199

53->65->67->98->122->124->183->200(final)->37->14

C-SCAN
53->65->67->98->122->124->183->200->0->14->37

LOOK e C-LOOK

Similar ao SCAN, a diferençaé que o LOOK muda a direção quando alcança a última requisição de uma dada direção, ao contrário do SCAN, que varre o disco até o final dele.

Uma das variações do LOOK é o C-LOOK, que atende requisições em uma única direção, começando no cilindro requisitado mais internamente e movendo-se para fora para atender as requisições até atingir o requisição "mais externa", daí a cabeça/braço é movida de volta para a requisição mais interna e o ciclo recomeça.

O tempo de acesso médio do LOOK é ligeraimente melhor que o do SCAN. O C-LOOK possui uma variação ligeiramente menor no tempo de acesso que o LOOK já que no pior caso o tempo

de acesso é aproximadamente dividido pela metade.

Referências:

Wikipedia
http://en.wikipedia.org/wiki/FIFO_(computing)
http://en.wikipedia.org/wiki/Shortest_seek_first

Operating Systems by Stallings
http://books.google.com/books?id=dBQFXs5NPEYC&pg=PA530&lpg=PA530&dq=Linux+disk+scheduling&source=bl&ots=CtpP2SiAVEWHld0WktgQQoikYtpHFQEGE&hl=en&ei=-C0aSqTLMYeJtgfZ1tT1DA&sa=X&oi=book_result&ct=result&resnum=6#PPA513,M1

http://www.dcs.ed.ac.uk/home/stg/pub/D/disk.html

http://www.kom.e-technik.tu-darmstadt.de/projects/iteach/itbeankit/lessons/scheduling/SCAN/scan.html

13. Quais são os algoritmos de escalonamentos disponíveis atualmente do kernel Linux? Descreva cada um deles.

NOOP

É o escalonador de I/O mais simples do Linux. Ele insere todas as requisições de I/O é uma fila FIFO (First In First Out) não ordenada.

O escalonador assume que a otimização do desempenho de I/O será feito em alguma outra camada da hierarquia de I/O (por exemplo, no dispositivo de blocos, por um controlador RAID "inteligente" ou por um controlador externo como um subsistema de armazenamento (*storage*).

O NOOP é melhor para dispositivos de estado sólido (*SSD - Solid State Devices*) como memória flash ou, em linhas gerais, dispositivo que não dependam de movimento mecânico para acesso aos dados (como a tecnologia tradicional de "disco rígido" que depende do tempo de acesso - *seek time* - e da latência rotacional). Dispositivos não-mecânicos não necessitam de reordenamento de múltiplas requisições de I/O, uma técnica que agrupa as requisições de I/O que estão fisicamente próximas no disco, dessa forma reduzindo o tempo médio de acesso e a variabililidade do tempo de "serviço" (pense em servir algo) de I/O.

ANTICIPATORY

O objetivo é aumentar a eficiência de utilização do disco "antecipando" operações de leitura síncrona.

"Deceptive idleness" é uma situação onde um processo parece ter terminado a leitura do disco quando na verdade está processando dados enquanto se prepara para a próxima operação de leitura. Isto fará com que um escalonador que procura reduzir trabalho mude para outro processo não relacionado. Esta situação é prejudicial à vazão de leituras síncronas, pois sobrecarrega o trabalho de "acesso". A solução do escalonador Anticipatory para o "Deceptive idleness" é fazer uma pausa por um pequeno tempo (alguns milisegundos) após uma operação de leitura em antecipação a outra requisição de leitura fisicamente próxima.

O Anticipatory pode reduzir o desempenho em discos usando TCQ (Tagged Command Queuing), discos de alto desempenho e conjuntos (arrays) RAID feitos via hardware. Este escalonador foi o padrão no Linux entre o kernel 2.6.0 e o 2.6.18, sendo substituído pelo CFQ.

DEADLINE

A principal meta deste escalonador é tentar garantir um tempo inicial para atender uma requisição. Ele faz isso impondo uma "deadline" em todas as operações de I/O para evitar "*starvation"* dos recursos. São mantidas duas filas de "deadline" além das filas ordenadas (ambas de leitura e escrita). As filas de "deadline" são minimamente ordenadas por seu tempo de expiração, enquanto que as filas ordenadas usam o número do setor para definir a ordem.

Antes de atender a próxima requisição, o Deadline decide qual fila usará. Filas de leitura recebem prioridade maior porque os processos usalmente "param" em operações de leitura. Em seguida, o escalonador verifica se a primeira requisição na fila "deadline" expirou, se não ele serve uma sequência de requisições da fila ordenada. Nos dois casos, o escalonador também serve uma sequência de requisições seguindo a requisição escolhida na fila ordenada.

Por padrão, as requisições de leitura expiram em 500ms, requisições de escrita em 5 segundos.

A documentação indica que este é o escalonador recomendado para sistemas de bancos de dados, especialmente em discos que trabalham com TCQ (Tagged Command Queuing) ou quaisquer sistemas com discos de alto desempenho.

CFQ (Completely Fair Queuing): 1 fila de I/O por processo, o objetivo é ser justo entre os processos.

O CFQ funciona colocando requisições síncronas enviadas pelos processos em uma certa quantidade de filas por-processo e então alocando espaços de tempo para que cada uma das filas acesse o disco. A quantidade de tempo e o número de requisições que uma fila pode enviar depende da prioridade de IO de um determinado processo.

Requisições assíncronas para todos os processos são processadas em conjunto em um número menor de filas que estão separadas por prioridade (uma fila por prioridade).

Embora o CFQ não faça explicitamente escalonamento de I/O anticipatório, ele atinge o mesmo efeito de ter boa vazão para o sistema como um todo, permitindo que uma fila de processo "não faça nada" (idle) ao final de um IO síncrono e, dessa forma, "antecipando" IOs próximos vindos do mesmo processo.

O CFQ pode ser considerado uma extensão natural da ação de conceder "fatias de tempo de IO" a um processo.

Referências:

Wikipedia:
http://en.wikipedia.org/wiki/Noop_scheduler
  http://en.wikipedia.org/wiki/Anticipatory_scheduling
http://en.wikipedia.org/wiki/Deadline_residentescheduler
http://en.wikipedia.org/wiki/CFQ

Artigo da RedHat comparando escalonadores:
https://news.wideopen.com/magazine/008jun05/features/schedulers/

Linux Documentation:
http://lxr.linux.no/linux+v2.6.29/Documentation/block/as-iosched.txt
http://lxr.linux.no/linux+v2.6.29/Documentation/block/deadline-iosched.txt
http://lxr.linux.no/linux+v2.6.29/Documentation/block/switching-sched.txt

14. Compare RAID0, RAID1, RAID5 e RAID6.

RAIDO (striped disks): distribui os dados entre vários discos de forma a aumentar a velocidade e não perder capacidade, mas todos os dados em todos os discos será perdido se um disco falhar. Embora este tipo de arranjo (array) não tenha redundância, é geralmente chamado de RAID 0. Também é possível encontrar a opção "linear", neste caso não há ganho de desempenho, os discos são arranjados de forma sequencial como um grande disco, os dados não são distribuídos entre os vários discos.

RAID1 (mirrored settings/disks): duplica os dados entre cada disco do array, fornecendo total redundância. Dois (ou mais) discos armazenam exatamente os mesmos dados, ao mesmo tempo e o tempo todo. Dados não são perdidos desde que um dos discos sobreviva. A capacidade total do array é igual à capacidade do menor disco no array. A qualquer momento, o conteúdo de cada disco no array é idêntico ao conteúdo dos outros discos do array.

RAID5 (striped disks with parity): combina três ou mais discos de forma a proteger os dados contra perda de um disco; a capacidade de armazenamento do array é igual à soma de todos os discos menos o espaço de um dos discos (por causa da paridade). Pode se recuperar em caso de perda de um disco. Mínimo de 3 discos.

RAID6 (striped disks with dual parity): menos comum. Pode se recuperar em caso de perdas de até dois discos. Mínimo de 4 discos.

Não está na questão mas foi falado em sala de aula:

RAID 0+1
Primeiro é feito RAID0 (strip) de dois conjuntos de disco, cada conjunto é então espelhado usando RAID1. (Espelhamento de conjuntos RAID0). São necessários pelo menos 4 discos (quantidades maiores devem ser pares) e a falha de um único disco fará com que todo o conjunto passe a se comportar como um RAID0, é possível ter múltiplas falhas em um mesmo conjunto, mas se um disco falhar em cada conjunto todos os dados do array estarão perdidos.

RAID 1+0 (RAID10)
Primeiro é feito RAID1 (espelho) de dois conjuntos de disco, cada conjunto é então dividido em "strips" usando RAID0. São necessários pelo menos 4 discos (quantidades maiores devem ser pares). É possível perder vários discos desde que nenhum dos "espelhos" perca todas as suas cópias.

RAID 5+0 (RAID50)
Divide (strip) os dados entre sistemas RAID de paridade distribuída.

Referência:

Wikipedia:
http://en.wikipedia.org/wiki/Redundant_array_of_independent_disks

Excelente tutorial de RAID:
http://www.raid.com/04_00.html

15. Suponha que você possui 4 discos rígidos de 750GB. Quanto espaço disponível você obteria usando RAID5 e quanto espaço você obteria com RAID6? Quantos discos poderiam falhar em cada um destes casos?

RAID5: 3 * 750 GB (+1 disco para paridade): 2250 GB e é possível perder um disco.
RAID6: 2 * 750 GB (+2 discos para paridade): 1500 GB e é possível perder até dois discos.

16. Virtualização
Xen/KVM, Paravirtualização
Artigo sobre virtualização: http://www.inf.ufrgs.br/~asc/sodr/pdf/SODR-ManuelaFerreira-rev.pdf
Fernando: hmmm.... não lembro direito.. é algo que intermedia o HW e o SO, o que é meio

estranho. é como se os SOs rodassem em cima disso, como máquinas virtuais, ou algo assim. dái essa camada virtual ficaria no anel 0 lá, e os SOs em niveis ainda mais a cima, sendo que antes eles que ficavam em nivel 0
me: vc tem isso no caderno?
Fernando: deixa eu ver
um pouco, em nivel 0 tenho desenhado a virtual machine, e acima dela os SOs
dai tem virtualização completa -> reescrever instruções; paravirutalização -> modificar o sistema convidado

17. Real-Time

Aula do Todt
Sistemas Operacionais de tempo real