Sistemas Operacionais

Prof. Rafael Obelheiro
rafael.obelheiro@udesc.br

Sistemas de Arquivos

Introdução

- Em muitas situações, é necessário armazenar os dados além do período de vida do processo que os usa
- Existem conjuntos de dados que simplesmente não cabem na memória principal
  - mesmo quando é usada memória virtual
- Outra necessidade importante é permitir que múltiplos processos acessem informações de forma concorrente
  - uma saída é tornar a informação independente de processos
- O armazenamento persistente de dados em um SO é responsabilidade do sistema de arquivos
  - os arquivos são a unidade de armazenamento

Camadas de armazenamento

Aplicações

↓ ↑ *operações com arquivos e diretórios*

Sistema de arquivos

↓ ↑ *operações com setores (blocos físicos)*

Gerência de E/S

↓ ↑ *interação com dispositivos*

Memória secundária

Responsabilidades do sistema de arquivos

- Existem diversos aspectos que precisam ser definidos pelo SA
  - como os arquivos são nomeados
  - como os arquivos são estruturados internamente
  - tipos de arquivos suportados
  - métodos de acesso aos arquivos
  - quais atributos são associados a um arquivo
  - operações com arquivos suportadas
  - organização em diretórios
  - implementação de arquivos e diretórios
  - gerenciamento do espaço em disco
- A interface oferecida pelo subsistema de E/S é leitura e escrita de setores (blocos físicos)
  - blocos de tamanho fixo (tipicamente 512 bytes), numerados sequencialmente

Sumário

1. Introdução
2. Arquivos
3. Diretórios
4. Implementação de sistemas de arquivos
5. Tópicos adicionais
6. Sistemas de arquivos no Linux

Nomeação de arquivos

- Permitem ao usuário acessar informações sem precisar saber onde elas estão localizadas no disco
- Aspectos a considerar
  - tamanho máximo do nome
  - caracteres permitidos, sensibilidade a caso
  - extensões de arquivo
    - ★ interpretadas pelo SO, pelas aplicações, ou mera conveniência para o usuário

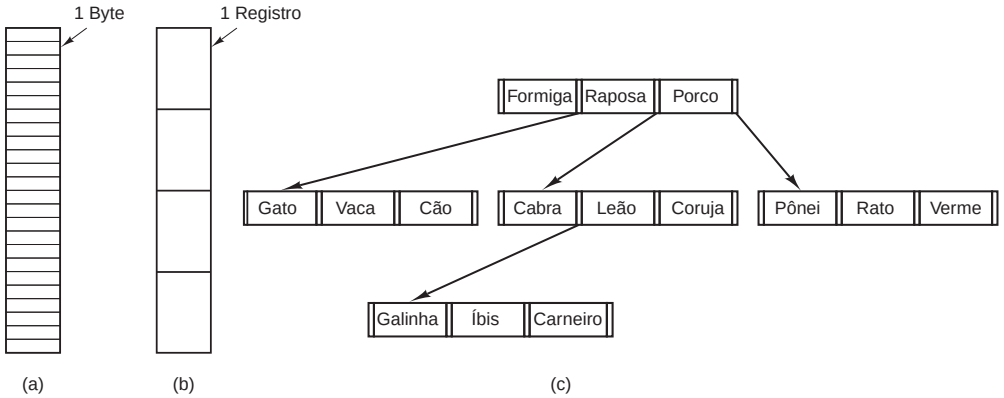
Extensões típicas de arquivo

Extensão	Significado
file.bak	Arquivo de cópia de segurança
file.c	Programa fonte em C
file.gif	Imagem no formato de intercâmbio gráfico da Compuserve (graphical interchange format)
file.hlp	Arquivo de auxílio
file.html	Documento da World Wide Web em Linguagem de Marcação de Hipertexto (*hypertext markup language* — HTML)
file.jpg	Imagem codificada com o padrão JPEG
file.mp3	Música codificada no formato de áudio MPEG — camada 3
file.mpg	Filme codificado com o padrão MPEG
file.o	Arquivo-objeto (saída do compilador, ainda não ligado)
file.pdf	Arquivo no formato portátil de documentos (*portable document format* — PDF)
file.ps	Arquivo no formato PostScript
file.tex	Entrada para o programa de formatação TEX
file.txt	Arquivo de textos
file.zip	Arquivo comprimido

Estrutura de arquivos (1/2)

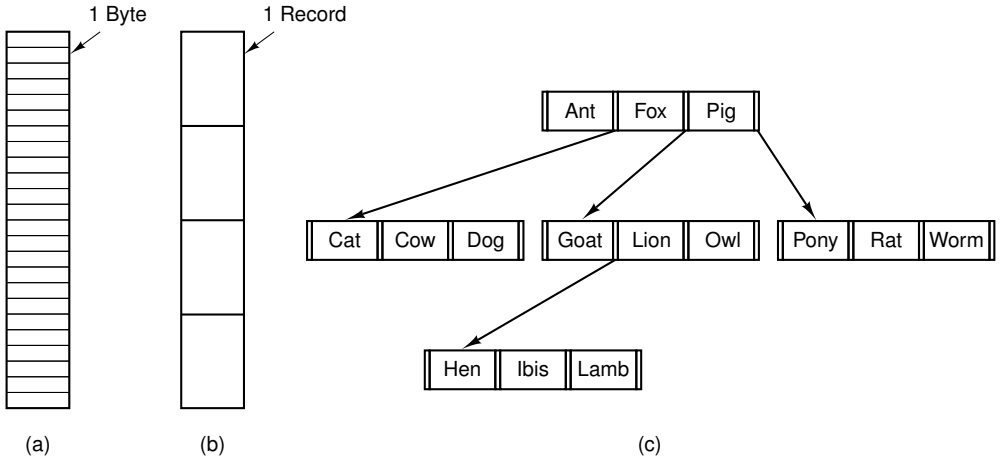
- Como os arquivos são estruturados internamente
- **Sequências de bytes**: estrutura conhecida apenas pelas aplicações, o SO só enxerga sequências de bytes
  - usada no UNIX e no Windows
- **Registros de tamanho fixo**: operações de leitura e escrita operam sobre registros com alguma estrutura
  - arquivo é uma sequência de registros
  - mais comum em mainframes da década de 60
- **Árvore**: cada registro possui uma chave, que é usada para indexação
  - usado em SOs que suportam BDs de grande porte

Estrutura de arquivos (2/2)



(a) Sequência de bytes (b) Registros (c) Árvore

Estrutura de arquivos (2/2)



(a) Sequência de bytes (b) Registros (c) Árvore

Tipos de arquivos

- **Arquivos regulares**: contêm dados de usuários
- **Diretórios**: arquivos do sistema que contêm a estrutura do sistema de arquivos
- **Arquivos especiais de caracteres**: usados para acessar dispositivos orientados a caracter
- **Arquivos especiais de bloco**: usados para acessar dispositivos de E/S orientados a bloco
  - filosofia do UNIX: arquivos são a interface de acesso aos dispositivos de E/S para os usuários
- Variam de acordo com o SO

Acesso a arquivos

- **Sequencial**: arquivo precisa ser lido/escrito em sequência
  - para ler o byte 1 milhão, é preciso ler os 999.999 anteriores
  - único modo de acesso compatível com fita magnética, p. ex.
- **Direto (aleatório)**: bytes/registros podem ser acessados em qualquer ordem
  - a operação de E/S pode especificar a posição ou pode haver uma operação específica de posicionamento
  - modo mais comum de acesso a disco

Atributos de arquivos

Atributo	Significado
Proteção	Quem pode ter acesso ao arquivo e de que maneira
Senha	Senha necessária para ter acesso ao arquivo
Criador	ID da pessoa que criou o arquivo
Proprietário	Atual proprietário
Flag de apenas para leitura	0 para leitura/escrita; 1 se apenas para leitura
Flag de oculto	0 para normal; 1 para não exibir nas listagens
Flag de sistema	0 para arquivos normais; 1 para arquivos do sistema
Flag de repositório (*archive*)	0 se foi feita cópia de segurança; 1 se precisar fazer cópia de segurança
Flag ASCII/binário	0 para arquivo ASCII; 1 para arquivo binário
Flag de acesso aleatório	0 se apenas para acesso seqüencial; 1 para acesso aleatório
Flag de temporário	0 para normal; 1 para remover o arquivo na saída do processo
Flag de impedimento	0 para desimpedido; diferente de zero para impedido
Tamanho do registro	Número de bytes em um registro
Posição da chave	Deslocamento da chave dentro de cada registro
Tamanho da chave	Número de bytes no campo-chave
Momento da criação	Data e horário em que o arquivo foi criado
Momento do último acesso	Data e horário do último acesso ao arquivo
Momento da última mudança	Data e horário da última mudança ocorrida no arquivo
Tamanho atual	Número de bytes no arquivo
Tamanho máximo	Número de bytes que o arquivo pode vir a ter

Operações com arquivos

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get Attributes
10. Set Attributes
11. Rename

Arquivos mapeados em memória

- Operações de acesso a arquivos são razoavelmente mais complicadas do que operações de acesso à memória
- Muitos sistemas suportam a noção de **arquivos mapeados em memória**
  - acessos a uma determinada região de memória são equivalentes a acessos ao arquivo
  - muitas vezes é possível mapear apenas partes de um arquivo
    - ★ evita problemas com arquivos maiores que o espaço de endereçamento do processo
  - o que ocorre quando um processo A mapeia um arquivo em memória e outro processo B realiza uma leitura do modo tradicional?
    - ★ modificações feitas por A só serão refletidas no arquivo quando a página correspondente for salva no disco

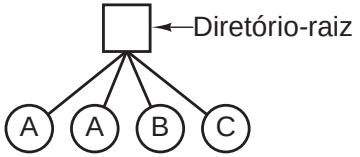
Sumário

1 Introdução
2 Arquivos
3 Diretórios
4 Implementação de sistemas de arquivos
5 Tópicos adicionais
6 Sistemas de arquivos no Linux

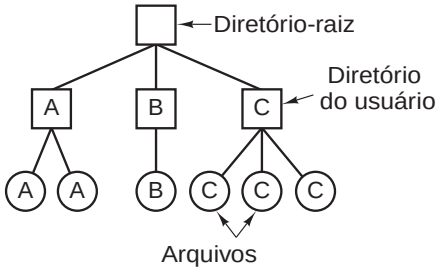
Estruturas de diretórios

- Diretório em nível único
  - simplicidade
  - rapidez na busca
  - problemas na colisão de nomes
- Diretório em dois níveis
  - cada usuário possui seu diretório privado
- Diretórios hierárquicos
  - árvore de diretórios
  - facilita organização dos arquivos

Diretório em nível único

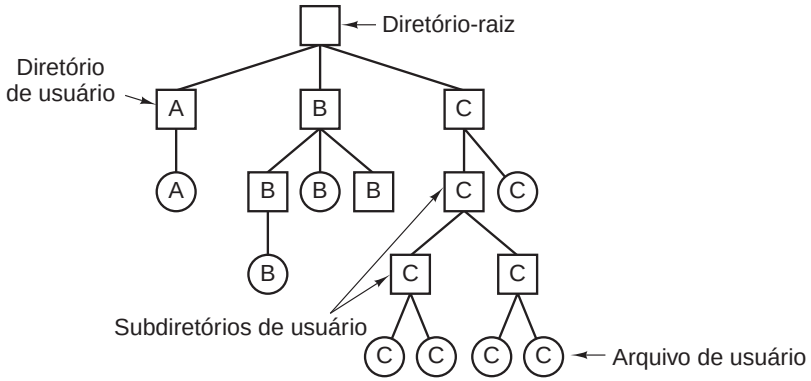


Diretório em dois níveis



a letra indica o usuário dono do diretório/arquivo

Diretórios hierárquicos



Operações com diretórios

1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir: retorna a próxima entrada de diretório
   - evita que o programador precise conhecer a estrutura interna dos diretórios
6. Rename
7. Link
   - cria uma nova entrada no diretório, apontando para algum arquivo
8. Unlink

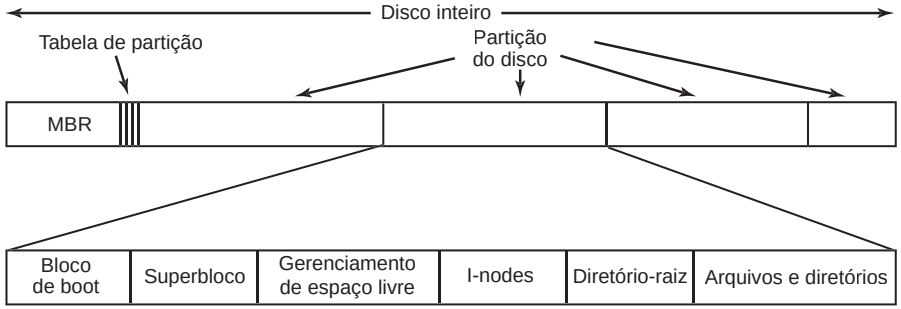
Sumário

1. Introdução
2. Arquivos
3. Diretórios
4. Implementação de sistemas de arquivos
5. Tópicos adicionais
6. Sistemas de arquivos no Linux

Esquema do sistema de arquivos (1/2)

- Sistemas de arquivos são armazenados em “discos”
- Um disco contém uma ou mais **partições**
- **MBR** (setor 0): controla a inicialização (boot)
  - localiza a partição ativa e carrega seu bloco de boot
- **Tabela de partições** (após o MBR): define endereço inicial e final de cada partição
- Estrutura de uma partição
  - bloco de boot: programa que carrega o SO contido na partição
  - superbloco: parâmetros do sistema de arquivos
    - ★ tamanho de bloco, tipo do SA, . . .
  - informações de gerenciamento
    - ★ espaço livre, tabelas de alocação, . . .
  - dados
- Alguns sistemas usam **volume** para descrever um “disco” lógico
  - um volume contém uma ou mais partições, e abstrai o dispositivo físico subjacente

Esquema do sistema de arquivos (2/2)



Implementação de arquivos

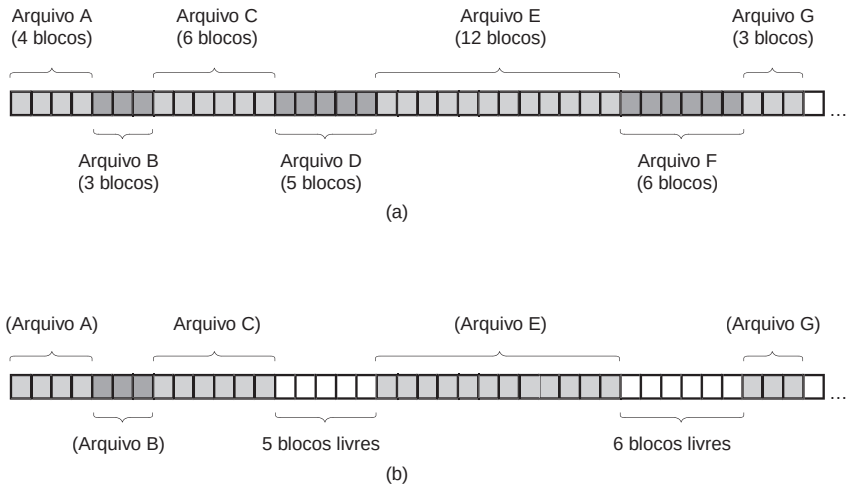
Como é alocado o espaço em disco para os arquivos?

- alocação contígua
- alocação por lista encadeada
- alocação por lista encadeada com tabela em memória
- i-nodes

Alocação contígua (1/2)

- Arquivos são armazenados em blocos contíguos no disco
- Vantagens
  - fácil de implementar
    - basta saber o bloco inicial e o número de blocos do arquivo
  - rapidez de acesso
- Desvantagens
  - fragmentação do disco com o tempo
    - análoga à fragmentação externa de memória
  - solução: compactação
- Usada em sistemas de arquivos somente de leitura
  - CDs, DVDs, ...

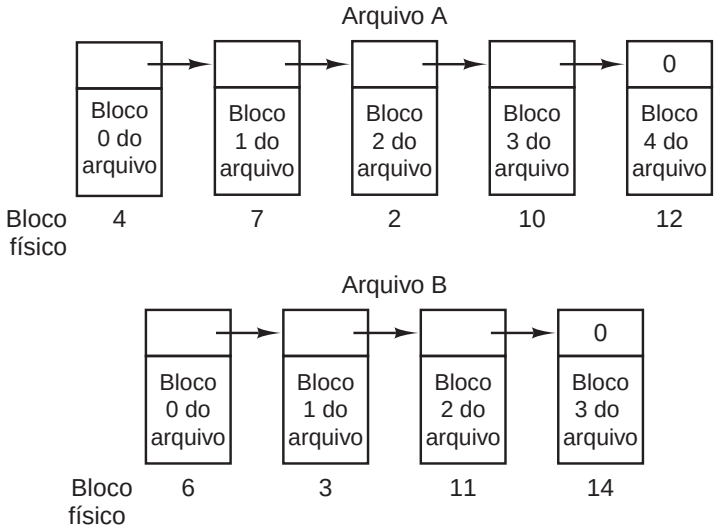
Alocação contígua (2/2)



(a) alocação contígua para os arquivos A–G

(b) estado do disco após a remoção de D e F

Alocação por lista encadeada

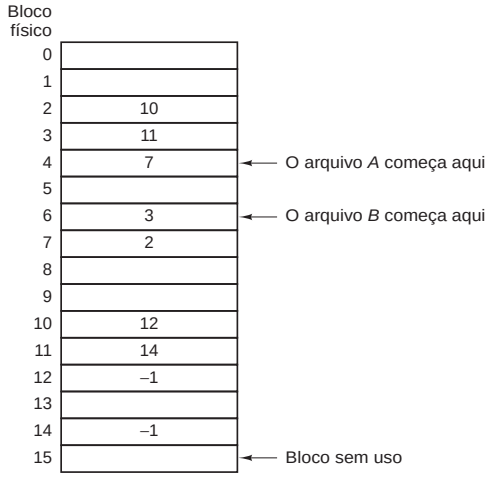


- principal desvantagem é no acesso direto

Lista encadeada com tabela na memória (1/2)

- Elimina-se as desvantagens da lista encadeada acrescentando-se uma tabela de correspondência na memória
- FAT (*file allocation table*)
  - o bloco todo fica disponível para dados
  - acesso aleatório facilitado
- Desvantagem: toda a tabela deve estar em memória o tempo todo
  - para um disco com 200 GB e blocos de 1 KB
    - ★ 200 milhões de entradas × 4 bytes (por entrada) = 800 MB
- Existe no disco uma cópia da FAT em memória

Lista encadeada com tabela na memória (2/2)



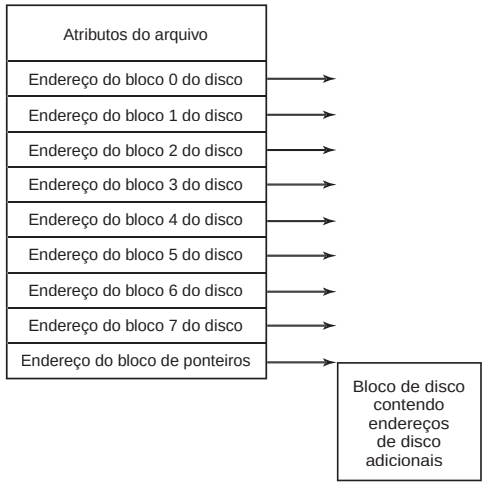
arquivo A: blocos 4, 7, 2, 10, 12

arquivo B: blocos 6, 3, 11, 14

I-nodes (1/5)

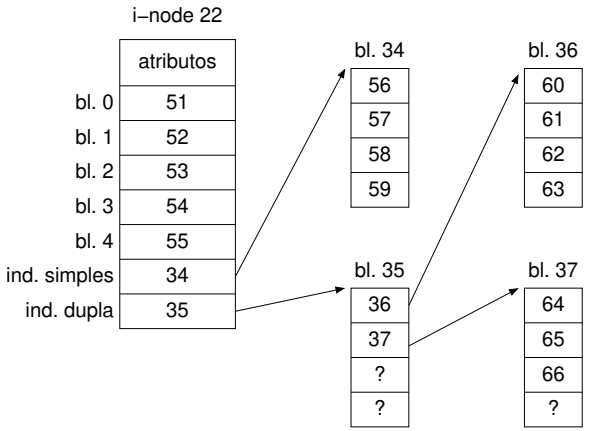
- Um i-node é uma estrutura que representa um arquivo, contendo os endereços dos blocos e os atributos
- Ocupa menos memória do que a tabela de alocação
  - apenas os i-nodes dos arquivos abertos precisam estar na memória
- Mecanismo empregado no UNIX

I-nodes (2/5)

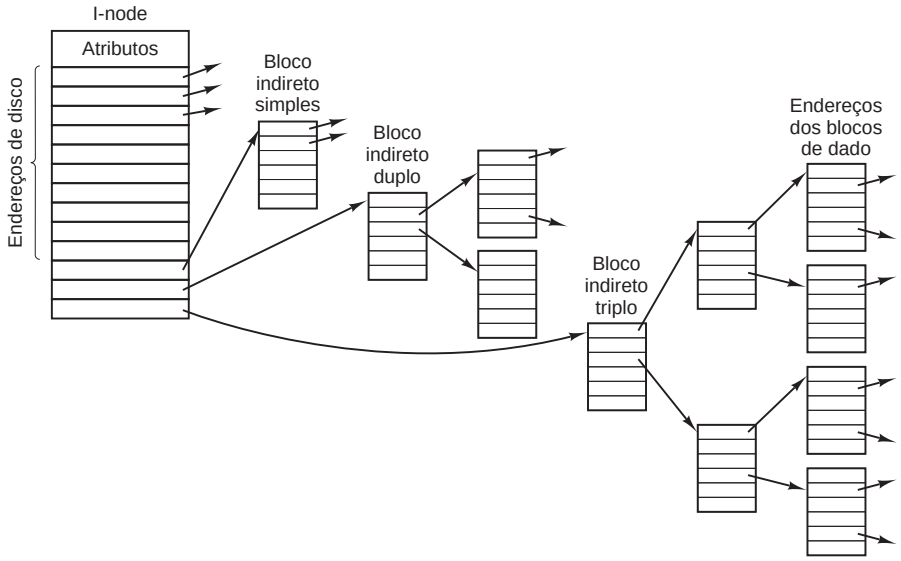


I-nodes (3/5)

- Exemplo
  - i-node com 5 ponteiros diretos, indireção simples e indireção dupla, e cada bloco de dados pode conter 4 ponteiros
  - arquivo ocupa os blocos 51–66, e seu descritor é o i-node 22
    - ★ blocos 34–37 são usados como blocos de índices



I-nodes (4/5)



Um i-node do UNIX V7 (1979)

I-nodes (5/5)

- Se existem 10 blocos diretos em um i-node, os blocos de dados são de 512 bytes e o endereço do bloco tem 32 bits, qual o tamanho máximo de um arquivo usando a estrutura de i-nodes do slide anterior?

Cálculo

$TamArq_{max} = blocos_{diretos} + blocos_{simples} + blocos_{duplo} + blocos_{triplo}$

$blocos_{diretos} = 10 \times 512 = 5120$ bytes $= 5$ KB

$blocos_{simples} = (512/4) \times 512 = 65.536$ bytes $= 64$ KB

$blocos_{duplo} = (512/4) \times (512/4) \times 512 = 8.388.608$ bytes $= 8.192$ KB

$blocos_{triplo} = (512/4) \times (512/4) \times (512/4) \times 512 = 1.048.576$ KB

Total = 1.056.837 KB = 1,008 GB

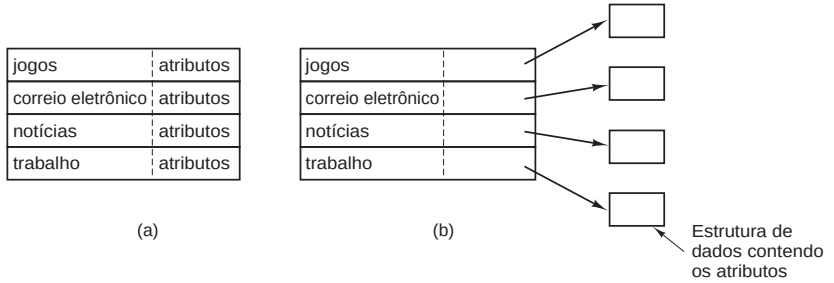
Implementação de diretórios (1/4)

- A entrada de um diretório fornece as informações para encontrar os blocos de disco correspondentes ao arquivo
  - endereço de todo o arquivo (alocação contígua)
  - número do primeiro bloco (lista encadeada)
  - número do i-node
- Função do diretório é mapear o nome do arquivo à informação necessária para localizá-lo

Implementação de diretórios (2/4)

- Projeto simples: lista de entradas de tamanho fixo contendo o nome do arquivo e seus atributos
  - uma entrada por arquivo
  - solução usada no MS-DOS
- Projeto com i-nodes: as entradas contêm o nome do arquivo e um ponteiro para o descritor com os atributos

Implementação de diretórios (3/4)

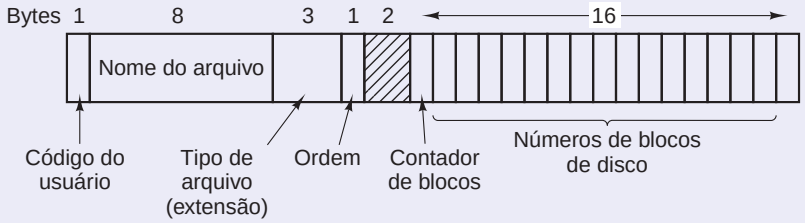


(a) Um diretório simples
  - entradas de tamanho fixo
  - endereços de disco e atributos na entrada de diretório

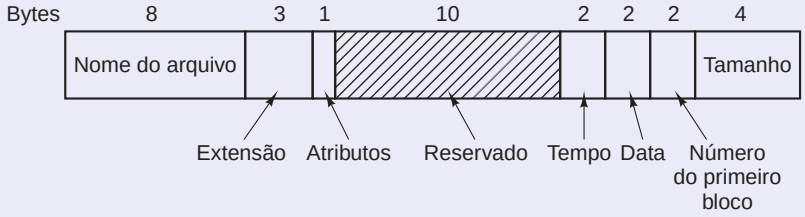
(b) Diretório no qual cada entrada se refere apenas a um i-node

Implementação de diretórios (4/4)

Implementação de diretórios no CP/M



Implementação de diretórios MS/DOS (FAT)



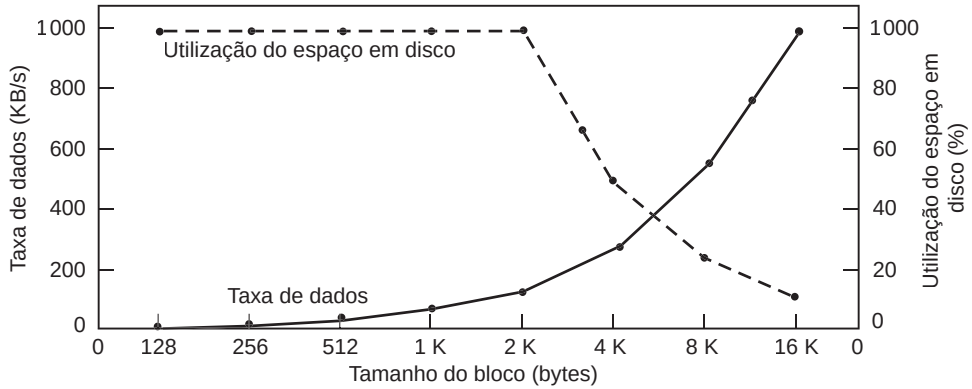
Gerência de espaço em disco

- Como alocar *n* bytes no disco?
  - usando uma área contígua de *n* bytes
  - dividindo os *n* bytes em blocos, não necessariamente contíguos
- Problema semelhante a segmentação vs paginação
  - soluções de compactação em disco são mais caras que a compactação de memória

Tamanho do bloco (1/2)

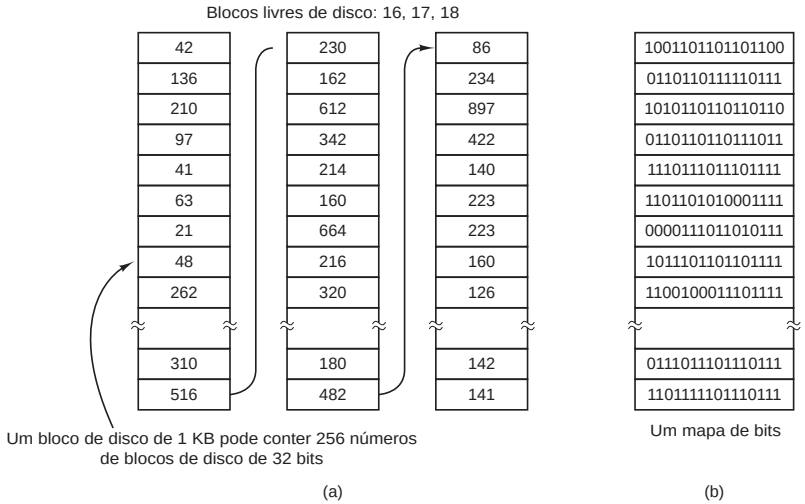
- Um bloco lógico corresponde a 2^k blocos físicos (setores) consecutivos no disco
  - tamanho típico de setor: 512 bytes
- Qual o tamanho de bloco mais adequado?
- Blocos grandes têm melhor desempenho, mas desperdiçam mais espaço
  - fragmentação interna
- Blocos pequenos têm pior desempenho (muitos blocos a serem lidos/escritos), mas aproveitam melhor o espaço
- Estudos mostram que o tamanho médio de um arquivo no UNIX é de 2 KB

Tamanho do bloco (2/2)



arquivos de 2 KB

Gerência de espaço livre



(a) lista encadeada (b) mapa de bits

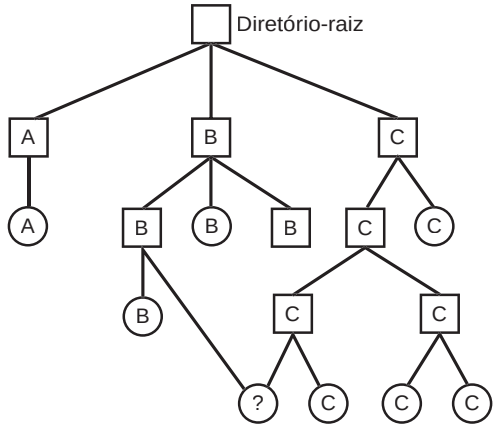
Sumário

1. Introdução
2. Arquivos
3. Diretórios
4. Implementação de sistemas de arquivos
5. Tópicos adicionais
6. Sistemas de arquivos no Linux

Arquivos compartilhados (1)

- Frequentemente é desejável que um mesmo arquivo esteja presente em vários diretórios diferentes
  - múltiplas cópias desperdiçam espaço em disco e introduzem problemas de consistência
- **Ligações** (*links*) permitem que um arquivo seja referenciado por múltiplos nomes
  - árvore de diretórios se torna um grafo acíclico dirigido

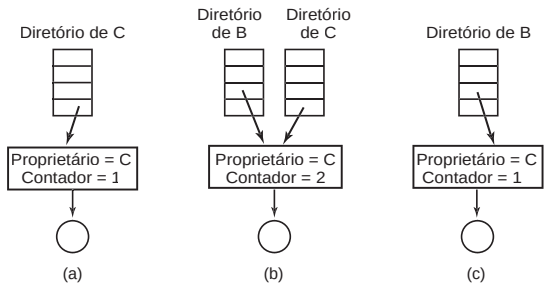
Arquivos compartilhados (2)



Arquivo compartilhado

Ligações estritas (*hard links*)

- Ligação estrita: entrada no diretório aponta para o mesmo i-node do arquivo original
  - i-node possui um contador de ligações
    - ★ incrementado quando uma nova ligação é criada
    - ★ decrementado quando uma ligação é removida
    - ★ i-node e dados só são removidos quando o contador chega a zero
- Não pode ser usada com diretórios → ciclos no grafo
- Problemas na contabilização do espaço ocupado pelos usuários



Ligações simbólicas

- Ligação simbólica: entrada no diretório aponta para o i-node de um arquivo do tipo ligação, que contém o nome de caminho do arquivo original
  - atalhos no Windows
- Pode ser usado com diretórios
- Problemas
  - overhead na tradução de nomes de caminho → acessos extras a disco
  - se o arquivo original for removido, as ligações se tornam inválidas
  - percurso recursivo de diretórios requer tratamento especial para evitar duplicação de conteúdo e laços
    - ★ duplicação também se aplica a ligações estritas

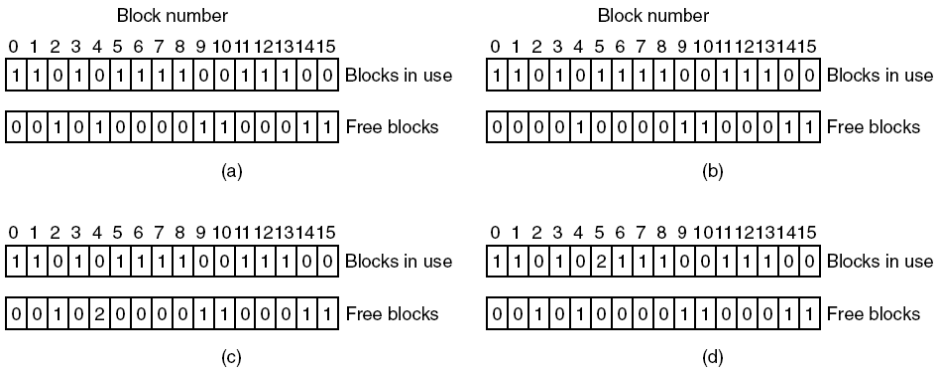
Consistência do sistema de arquivos

- Escritas em disco não são operações atômicas, e falhas (falta de energia, p.ex.) podem deixar o SA em um estado inconsistente
  - se a falha for durante a escrita de metadados o problema pode ser agravado
- Geralmente há programas que analisam o SA para detectar e corrigir eventuais inconsistências
  - `fsck` (UNIX), `Scandisk/CHKDSK` (Windows)
- Esses programas se baseiam na redundância interna do SA para repará-lo

Funcionamento básico do `fsck` (1)

- Executado no boot quando o SA não foi desmontado corretamente, e periodicamente como prevenção
- Dois tipos principais de verificação de consistência: por blocos e por arquivos
- Verificação por blocos
  - duas tabelas, cada uma com um contador para cada bloco do SA
    - ★ quantas vezes o bloco aparece como pertencente a um arquivo
    - ★ quantas vezes o bloco aparece na lista/mapa de blocos livres
  - lê todos os blocos do SA
    - ★ preenche a 1ª tabela com base nos i-nodes
    - ★ preenche a 2ª tabela com base na lista/mapa de blocos livres
    - ★ compara as duas tabelas

Funcionamento básico do `fsck` (2)



Funcionamento básico do `fsck` (3)

(a) SA consistente: cada bloco está ou livre ou em uso
  - tem 1 em uma tabela e 0 na outra

(b) SA inconsistente: bloco 2 não está nem livre nem em uso
  - bloco desaparecido
  - solução: incluir na lista de blocos livres

(c) SA inconsistente: bloco 4 aparece duas vezes na lista de livres
  - só pode ocorrer com lista, não com mapa de bits
  - solução: remover as ocorrências extras

(d) SA inconsistente: bloco 5 é usado por dois arquivos
  - solução: copiar o bloco 5 para um bloco livre e corrigir o ponteiro em um dos i-nodes
  - provavelmente o conteúdo ficará inconsistente, e o usuário terá de resolver

Funcionamento básico do `fsck` (4)

- Verificação por arquivos
  - percorre recursivamente a árvore de diretórios, contando o número de referências a cada i-node
    - ★ pode ser > 1 devido às ligações estritas
  - compara o número de referências observadas com a contagem de ligações presente em cada i-node
    - ★ idealmente, são iguais
    - ★ se a contagem do i-node for maior, o i-node e os blocos do arquivo não serão liberados depois que a última ligação for removida
    - ★ se a contagem do i-node for menor, o i-node e os blocos serão liberados quando ainda estiverem em uso
    - ★ em ambos os casos, a solução é atualizar a contagem do i-node com o número observado
- `fsck` ainda realiza outras verificações
  - compara informações resumidas no superbloco/i-nodes com resultados da análise estrutural
  - pode analisar permissões dos arquivos em busca de permissões sem sentido ou inseguras

Sistemas de arquivos *journaling* (1)

- Uma solução para manter a consistência dos dados e metadados diante de falhas do SA
- Exemplo: remoção de um arquivo no UNIX
  1. Remove a entrada de diretório
  2. Coloca o i-node na lista de i-nodes livres
  3. Coloca os blocos de dados na lista de blocos livres
  - se o sistema parar entre 2 e 3, os blocos ficam em um limbo
    - ★ indisponíveis sem estarem associados a nenhum arquivo
  - se o sistema parar entre 1 e 2, os i-nodes também são afetados
  - reordenar os passos muda o problema, mas não o resolve
- Journaling: operações são gravadas em um log (*journal*) antes de serem realizadas no disco
  - caso ocorra uma falha, o log permite refazer as operações pendentes
  - caso a falha ocorra durante a escrita no log, a operação toda é abortada

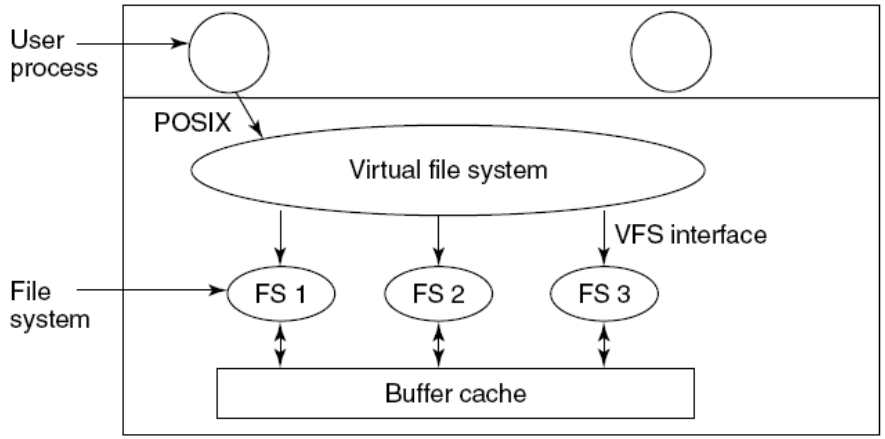
Sistemas de arquivos *journaling* (2)

- Journaling exige que as operações registradas no log sejam **idempotentes**
  - podem ser repetidas várias vezes sem afetar o resultado final
  - `x=171` (idempotente) vs. `x++` (não idempotente)
- Exemplos de operações idempotentes
  - marcar o bloco *k* como livre no mapa de bits
  - remover a entrada `bar` no diretório `foo`
- Exemplos de operações não idempotentes
  - colocar o bloco *k* na lista de blocos livres (ele pode já estar lá)
  - decrementar o contador de ligações do i-node *i*
- Muitos sistemas de arquivos atuais usam journaling
  - ext3/ext4, XFS, JFS, NTFS

Sistemas de arquivos virtuais (1)

- Um SO pode ter de lidar com diferentes sistemas de arquivos simultaneamente
  - Windows: pendrive com FAT-16, partição com FAT-32, partição com NTFS, CD-ROM com ISO 9660
  - Windows usa letras de unidade diferentes para cada dispositivo
  - UNIX incorpora todos os SAs em uma única hierarquia de diretórios
  - essa heterogeneidade pode ser transparente para o usuário, mas é totalmente visível para a implementação
- Sun introduziu o conceito de sistema de arquivos virtual (VFS, *Virtual File System*)
  - uma camada que oferece uma interface bem definida para as aplicações e para os diferentes SAs
    - ★ aplicações funcionam independente do SA
    - ★ SA sabe quais operações precisa implementar

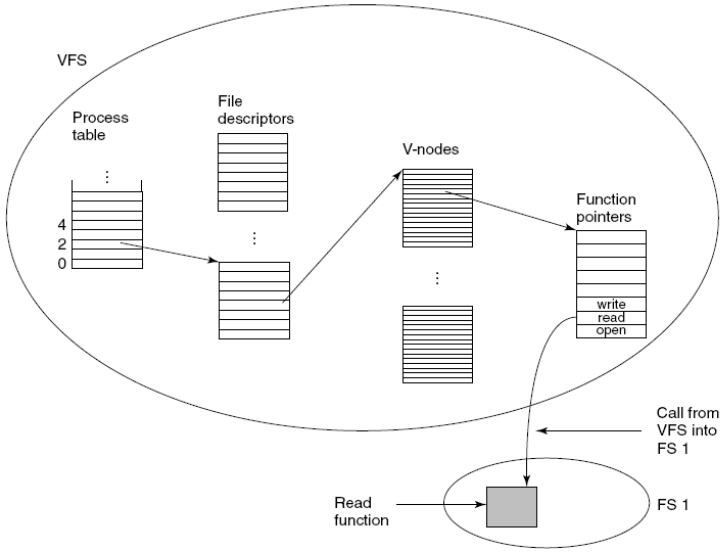
Sistemas de arquivos virtuais (2)



Implementação do VFS (1)

- VFS mantém várias estruturas de dados
  - tabelas de descritores de arquivos em uso
    - ★ para cada arquivo aberto é mantido um v-node em memória
  - v-nodes (equivalente a i-nodes)
    - ★ contém dados que permitem acessar o SA subjacente
  - tabela de ponteiros de função para as operações de cada SA
    - ★ funções que implementam `open()`, `read()`, `write()`, `close()`, `link()`, `unlink()`, etc.
    - ★ projeto OO implementado em C

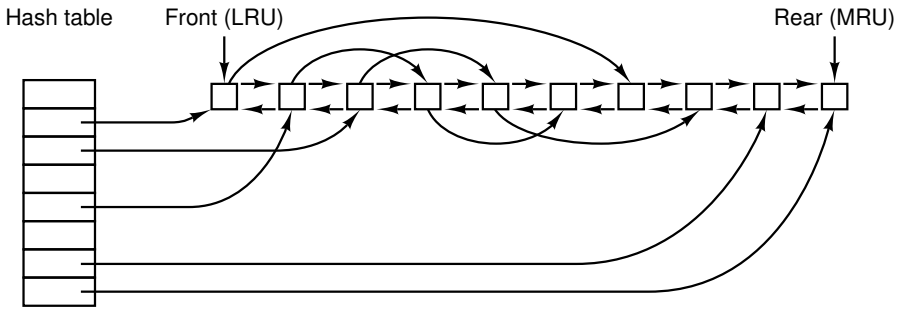
Implementação do VFS (2)



Cache de blocos (1)

- Para reduzir o tempo das operações de disco, o SO mantém uma cache dos blocos mais acessados em memória
  - cache de blocos ou *buffer cache*
- Estrutura típica usa uma tabela hash com lista de colisão, indexada pelo dispositivo e pelo número do bloco
  - blocos são mantidos em uma lista duplamente encadeada ordenada pelo algoritmo MRU (menos recentemente usado)
    - ★ LRU (*least recently used*)
    - ★ bloco acessado por último vai para o final da fila
    - ★ blocos acessados há mais tempo migram para o início da fila
    - ★ quando for necessário substituir um bloco na cache (cache cheia), pega-se o primeiro da fila, que é gravado se foi modificado (→ sujo)

Cache de blocos (2)



Cache de blocos (3)

- Heurísticas podem ser usadas para determinar se um bloco é inserido no início ou no final da fila
  - blocos com baixa probabilidade de reuso vão no início
    - ★ são rapidamente escritos no disco e "reciclados"
  - blocos com maior probabilidade de reuso vão no final
    - ★ blocos que estão sendo preenchidos, p.ex.
- Esperar até que um bloco de metadados chegue ao início da fila para ser gravado pode deixar o SA inconsistente
  - blocos de metadados são escritos rapidamente, em muitos casos de forma síncrona
- Podem ser usadas chamadas de sistema que forçam a escrita de todos os blocos sujos
  - `sync` (UNIX), `FlushFileBuffers` (Windows)
  - versões anteriores do Windows usavam cache de escrita direta (*write-through*)
    - ★ modificações na cache eram imediatamente gravadas em disco
- Alguns sistemas integram cache de blocos e cache de páginas

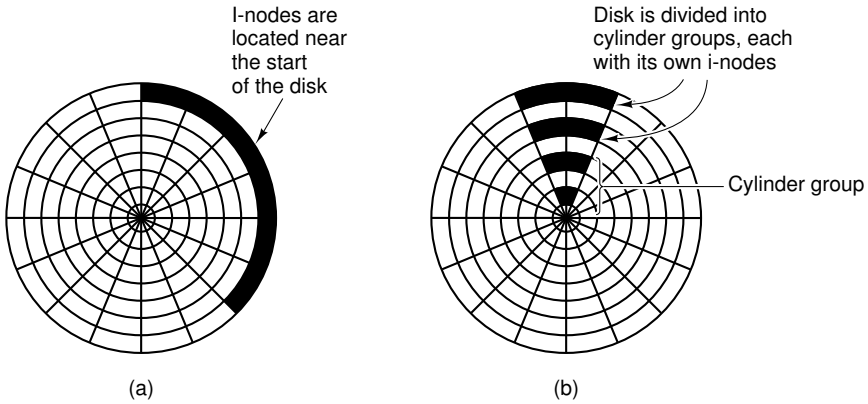
Leitura antecipada de blocos

- Tenta trazer para a cache blocos que provavelmente serão necessárias em breve
- Exemplo: se uma aplicação está lendo um arquivo sequencialmente e solicita o bloco k, o SA já escalona a leitura do bloco $k + 1$
- Em acesso aleatório, degrada o desempenho
- Pode ser usada uma heurística para analisar o comportamento da aplicação e decidir se vale a pena fazer leitura antecipada
  - inicialmente considera-se que o acesso é sequencial, e se faz leitura antecipada
  - se a aplicação reposiciona o ponteiro de arquivo, considera-se que o acesso é aleatório, e não há antecipação
  - se o acesso voltar a ser sequencial, pode-se voltar à leitura antecipada
  - tipo corrente de acesso pode ser indicado por um bit na entrada correspondente na tabela de arquivos abertos

Redução do movimento do braço do disco (1)

- Usar estratégias de alocação que minimizem deslocamentos (*seeks*)
- Originalmente, o sistema de arquivos do UNIX mantinha todos os i-nodes no início do disco, seguidos pelos blocos de dados
- Grupos de cilindros: espalha i-nodes pelo disco, mantendo-os próximos aos seus blocos de dados
  - cada grupo contém uma cópia do superbloco, i-nodes, um mapa de bits e blocos de dados
  - o SA tenta alocar todos os i-nodes de um diretório dentro do grupo
  - quando um bloco é requisitado para um i-node, o SA tenta alocar um bloco dentro do grupo
  - se não houver blocos disponíveis no mesmo grupo, recorre-se aos grupos vizinhos
  - blocos para arquivos grandes são alocados em diversos grupos, usando uma heurística

Redução do movimento do braço do disco (2)



(a) Layout original

(b) Grupos de cilindros

Desfragmentação

- Com a passagem do tempo, o sistema de arquivos se torna fragmentado
  - blocos do mesmo arquivo e blocos livres espalhados pelo disco
- Fragmentação prejudica o desempenho do SA
- Ferramentas de desfragmentação reorganizam o SA, agrupando os blocos de dados de arquivos em regiões contíguas no disco e concentrando os blocos livres
- A desfragmentação pode ser necessária antes do redimensionamento de uma partição
- Estratégias como grupos de cilindros podem reduzir/eliminar a necessidade de desfragmentação

Sumário

1. Introdução
2. Arquivos
3. Diretórios
4. Implementação de sistemas de arquivos
5. Tópicos adicionais
6. Sistemas de arquivos no Linux

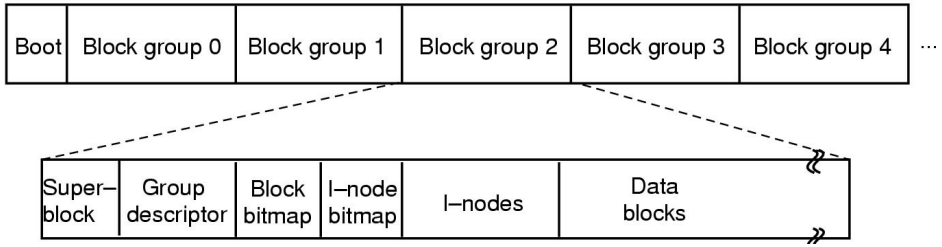
Introdução

- O Linux oferece suporte a diversos tipos de sistemas de arquivos, tanto nativos quanto de outros SOs
- O primeiro SA do Linux foi o mesmo do MINIX 1
  - nomes de arquivos com até 14 caracteres, arquivos de até 64 MB
- A linha mais tradicional é a do *Extended Filesystem* e seus descendentes
  - ext (1992) → ext2 (1993) → ext3 (2001) → ext4 (2008)
  - a principal diferença é que ext3 e ext4 incorporam *journaling*
    - além de melhorias de desempenho e suporte a tamanhos maiores

Layout de disco no ext2 (1/2)

- O ext2 suporta partições de até 4 TB (2^{42} bytes)
- Cada partição é dividida em grupos de blocos
  - análogos aos grupos de cilindros do FFS (BSD UNIX)
  - no. de grupos depende do tamanho do SA
- Cada grupo de blocos é subdividido em
  - **superbloco**: contém os parâmetros e informações de controle do SA como um todo
    - ★ cada grupo tem uma cópia do superbloco, que é único para o SA
  - **descritor do grupo**: contém parâmetros e informações de controle do grupo
    - ★ localização dos mapas de bits, no. i-nodes/blocos livres, no. diretórios
  - **mapas de bits**: controlam alocação de i-nodes e blocos de dados
    - ★ cada mapa ocupa um bloco → limitador do tamanho do grupo
  - **i-nodes**: descritores de arquivo
  - **blocos de dados**

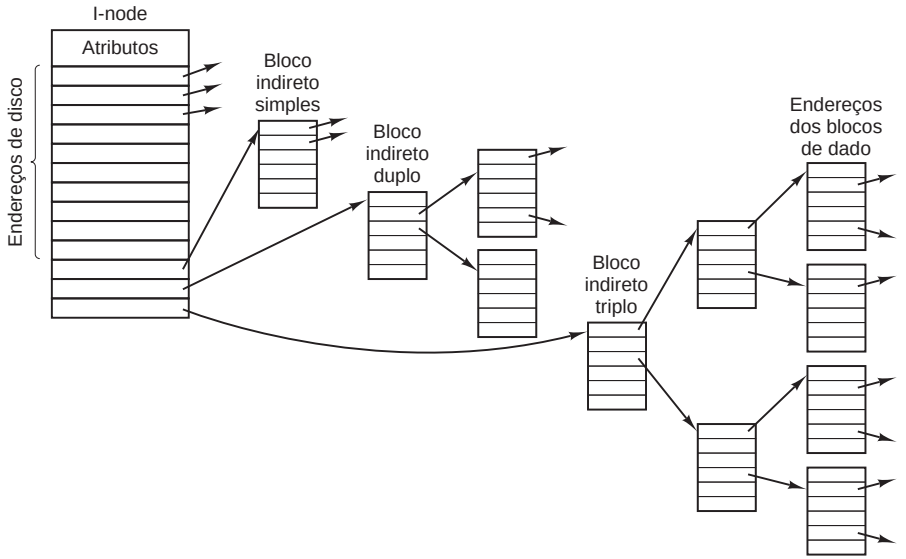
Layout de disco no ext2 (2/2)



I-Nodes no ext2 (1/2)

- O ext2 segue a estrutura clássica do UNIX
  - 12 ponteiros diretos
  - indireção simples/dupla/tripla
  - ponteiros de 32 bits (4 bytes)
  - blocos de 1, 2 ou 4 KB
    - ★ o tamanho de bloco é definido na formatação do SA, por escolha do usuário ou em função do tamanho da partição
  - cada i-node ocupa 128 bytes

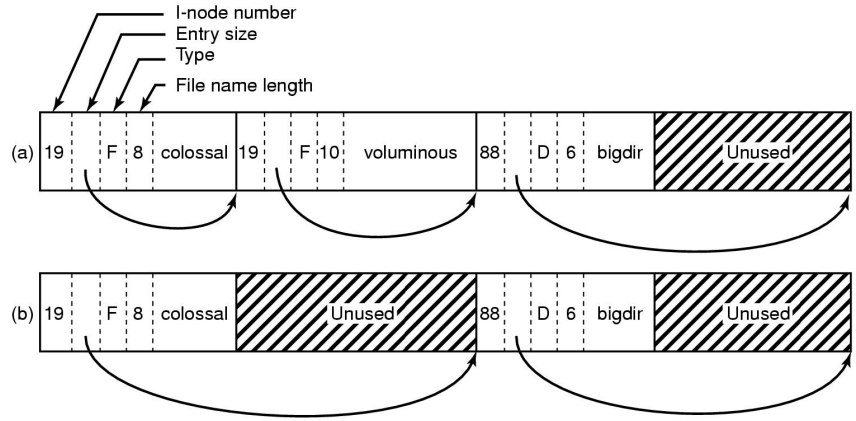
I-Nodes no ext2 (2/2)



Diretórios no ext2 (1/2)

- Um diretório é uma lista encadeada de entradas de tamanho variável
- Cada entrada possui
  - i-node
  - tamanho da entrada
  - tamanho do nome
  - tipo de arquivo
  - nome
- O encadeamento é implementado usando o tamanho da entrada
  - quando uma entrada é removida, o tamanho da entrada anterior aumenta de modo a apontar para a próxima
    - ★ o i-node também é zerado

Diretórios no ext2 (2/2)



(a) diretório com 3 entradas

(b) após a remoção da entrada `voluminous`

Bibliografia Básica

Andrew S. Tanenbaum.
Sistemas Operacionais Modernos, 4ª Edição. Capítulo 4.
Pearson Prentice Hall, 2016.

Carlos A. Maziero.
Sistemas Operacionais: Conceitos e Mecanismos. Capítulos 22–25.
Editora da UFPR, 2019.
http://wiki.inf.ufpr.br/maziero/doku.php?id=socm:start