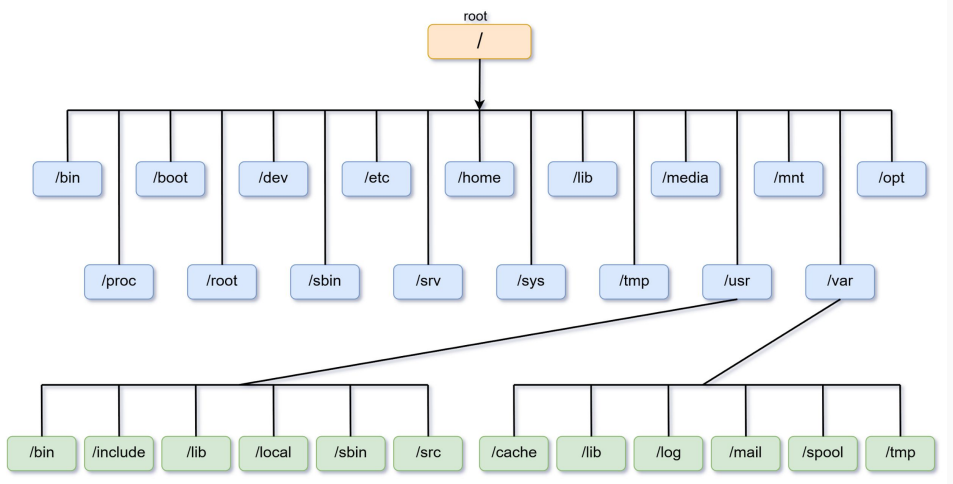
Aula 09 - Sistemas de Arquivos

Sistemas Operacionais
Ciência da Computação
IFB - Campus Taguatinga

Professor João Victor de A. Oliveira

Hoje

- **Sistemas de Arquivos**
  - Arquivos
  - Diretórios
- **Implementação do Sistema de Arquivos**
  - Esquema do Sistema de Arquivos
  - Implementando arquivos
  - Implementando diretórios
  - Arquivos compartilhados



Sistemas de Arquivos

- **3 requisitos essenciais para o armazenamento de informações por longo prazo**
  1. Deve ser possível armazenar uma quantidade muito grande de informações
  2. As informações devem sobreviver ao término do processo que a está executando
  3. Múltiplos processos têm de ser capazes de acessá-las ao mesmo tempo

Sistemas de Arquivos

- Abstrações realizadas pelo SO
  - Conceito de processador ⇒ Processo
  - Conceito da memória física ⇒ Espaços de endereçamento (virtual)
  - Conceito de armazenamento de informações em disco ⇒ *Arquivos*
- **Arquivos: unidades lógicas de informação criadas por processos**
  - Podemos pensar em arquivos como espaços de endereçamentos usados para modelar o disco, ao invés da RAM

Sistemas de Arquivos

- Informações armazenadas em arquivos devem ser **persistentes**
  - Não devem ser afetadas pela criação e término de processos
  - Um arquivo deve desaparecer apenas quando o seu proprietário o remove explicitamente
- Arquivos são gerenciados pelo SO
  - Este deve definir como os arquivos são:
    - Nomeados
    - Estruturados
    - Acessados
    - Protegidos
    - Implementados
    - Gerenciados
  - A parte do SO responsável por lidar como arquivos é chamada **Sistema de Arquivo**

Nomeação de Arquivos

- Um arquivo é um mecanismo de abstração
  - Fornece uma maneira para armazenar informações sobre o disco e lê-las depois
  - Deve ser feito de tal modo que isole do usuário os detalhes de como e onde as informações estão armazenadas e como os discos realmente funcionam
- Quando um processo cria um arquivo ele lhe dá um nome
  - Quando o processo é concluído, o arquivo continua a existir e pode ser usado por outros processos através de seu nome
  - Regras exatas para a nomeação de arquivos varia de certa maneira de sistema para sistema

Nomeação de Arquivos

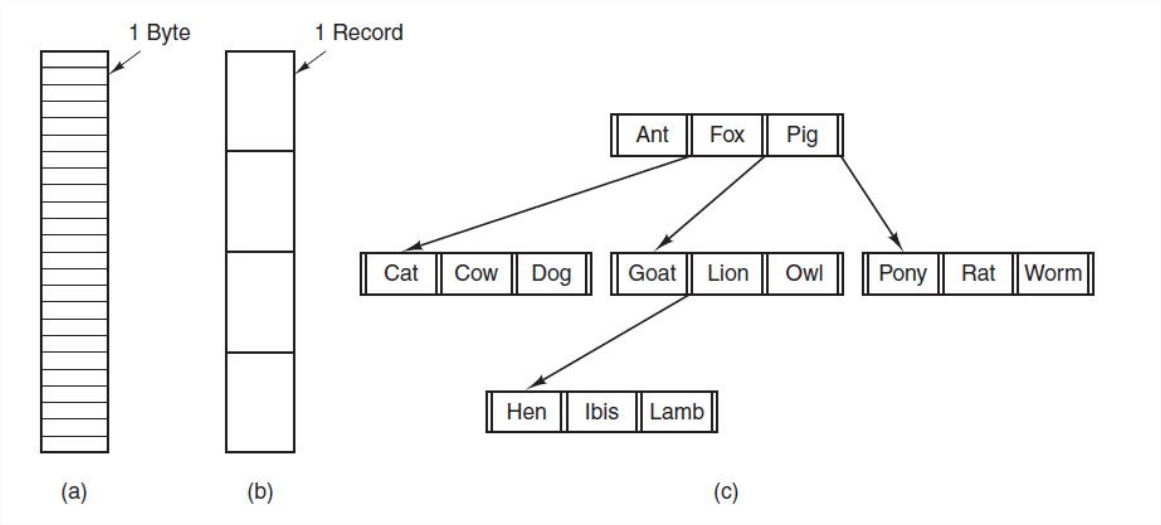
- Muitos SOs aceitam nomes de arquivos de duas partes, com as partes separadas por ponto.
  - A parte que vem seguida ao ponto é chamada **extensão do arquivo**
    - Pode ser obrigatória (MS-DOS)
    - Optativa (UNIX)
  - Note que o uso de extensões auxilia a interação entre homem e computador
    - Ex.: no Windows é designado um significado para extensões conhecidas, de forma que ao abrir um arquivo de formato conhecido, é buscado o programa que melhor se adequa a ele.

Exemplos de extensões de arquivos

Extension	Meaning
.bak	Backup file
.c	C source program
.gif	Compuserve Graphical Interchange Format image
.hlp	Help file
.html	World Wide Web HyperText Markup Language document
.jpg	Still picture encoded with the JPEG standard
.mp3	Music encoded in MPEG layer 3 audio format
.mpg	Movie encoded with the MPEG standard
.o	Object file (compiler output, not yet linked)
.pdf	Portable Document Format file
.ps	PostScript file
.tex	Input for the TEX formatting program
.txt	General text file
.zip	Compressed archive

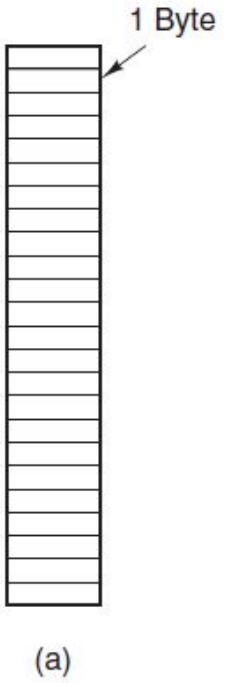
Estruturas de arquivos

- Arquivos podem ser estruturado de várias maneiras:
  - Sequência de bytes
  - Sequências de registros
  - Árvore



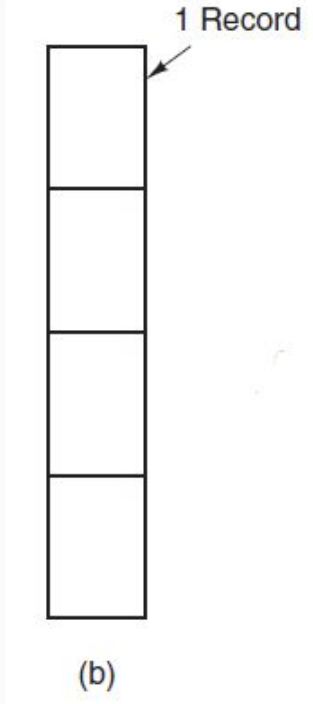
Estruturas de arquivos

- Sequência de bytes
  - O SO não sabe ou não se importa sobre o que há no arquivo
    - Tudo o que ele vê são bytes
  - Qualquer significado deve ser imposto por programas em nível de usuário
  - **Tanto o UNIX (OS X também) quanto o Windows usam essa abordagem**
  - Esta estrutura (ou ausência de estrutura) tem a vantagem de oferecer a máxima **flexibilidade**
    - Programas de usuário podem colocar qualquer coisa que quiserem em seus arquivos e nomeá-los do jeito que acharem conveniente



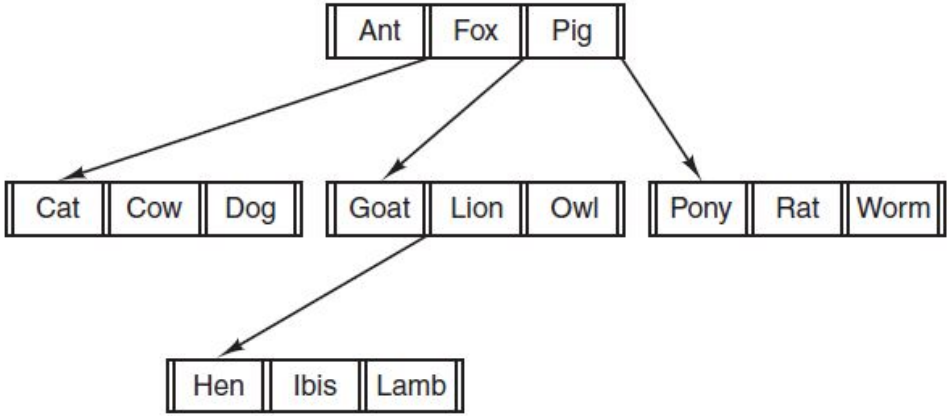
Estruturas de arquivos

- Sequência de registros
  - Registros de tamanho fixo, com alguma estrutura interna
  - Operação de leitura retorna um registro e a operação de escrita sobrepõe ou anexa um registro
  - Ex. de uso: armazenamento de cartões perfurados de 80 colunas
    - Programas liam a entrada em unidades de 80 caracteres (registro) e escreviam em arquivos com registros de 132 caracteres, usados em impressoras de linha
  - Atualmente, nenhum sistema de propósito geral atual usa esse modelo



Estruturas de arquivos

- Árvore
  - Arquivo consiste em uma árvore de registros, não necessariamente de mesmo tamanho, cada um contendo um campo **chave** em uma posição fixa no registro
  - A árvore é ordenada no campo chave, a fim de permitir uma busca rápida por uma chave específica
  - Operação básica não é obter o próximo registro, mas aquele com a chave específica
  - Novos registros podem ser adicionados à árvore com o SO, e não com o usuário
  - Usado em alguns computadores de grande porte para o processamento de dados comerciais



(c)

Tipos de Arquivos

- Windows e UNIX aceitam dois tipos principais de arquivos
  - **Arquivos regulares**
    - Contém informações de usuários
  - **Diretórios**
    - Arquivos do sistema para manter a estrutura do sistema de arquivos
- UNIX também tem arquivos especiais de **caracteres** e **blocos**
  - Arquivos especiais de caracteres: relacionados com a entrada/saída e usados para modelar dispositivos de E/S seriais
  - Arquivos especiais de blocos: usados para modelar discos

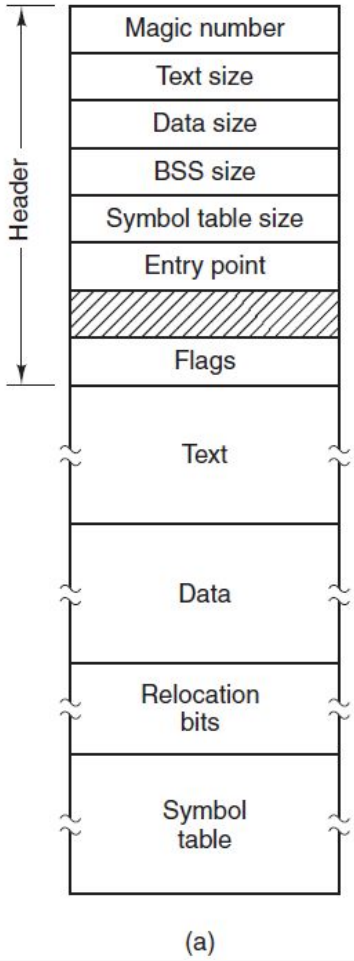
Arquivos regulares

- **Divididos em arquivos ASCII ou arquivos binários**
  - **Arquivos ASCII:** armazenam linhas de texto
  - **Arquivos binários:** não são arquivos ASCII
    - Em geral possuem estrutura interna conhecida pelos programas que os usam

Arquivos regulares

- Na figura ao lado (a) vemos um exemplo de arquivo binário executável dividido em cinco seções:
  - Cabeçalho
  - Texto,
  - Dados,
  - Bits de realocação
  - Tabela de símbolos
- Todo SO deve reconhecer pelo menos um tipo de arquivo
  - O seu próprio arquivo executável
  - Pode reconhecer outros tipos de arquivos…

Ex.: Java Bytecode → https://en.wikipedia.org/wiki/Java_class_file



Acesso aos arquivos

- Acesso sequencial
  - Um processo podia ler todos os bytes ou registros em um arquivo em ordem
    - Não era possível pular nenhum byte ou lê-los fora de ordem
    - No entanto, era possível ser trazido de volta para o ponto de partida
  - Conveniente quando o meio de armazenamento era uma fita magnética ao invés de um disco

Acesso aos arquivos

- Acesso fora de ordem
  - Com o advento dos discos, tornou-se possível ler os bytes ou registros de um arquivo fora de ordem, ou acessar registros pela chave em vez de pela posição
  - Arquivos ou registros que podem ser lidos em qualquer ordem são chamados de **arquivos de acesso aleatório**
  - **Dois métodos para especificar onde começar a leitura:**
    - ***read***: fornece a posição no arquivo onde começar a leitura
    - ***seek:*** estabelece a posição atual na qual um arquivo pode ser lido sequencialmente (usado tanto no UNIX quanto no Windows)

Atributos de arquivos

- Todos SOs associam diversas informações com cada arquivo:
  - Ex.: nome, data de criação...
  - Chamaremos esses itens de atributos do arquivo (ou **metadados)**
    - Existem diversas classes de atributos:
      - Atributos de proteção
      - Atributos de sinalizações (flags)
      - Atributos de tamanho do arquivo/registro
      - Atributos de tempo

Atributos de arquivos

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 4-4. Some possible file attributes.

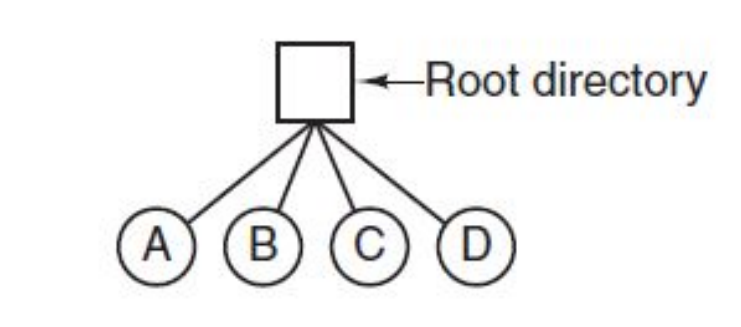
Operações com arquivos

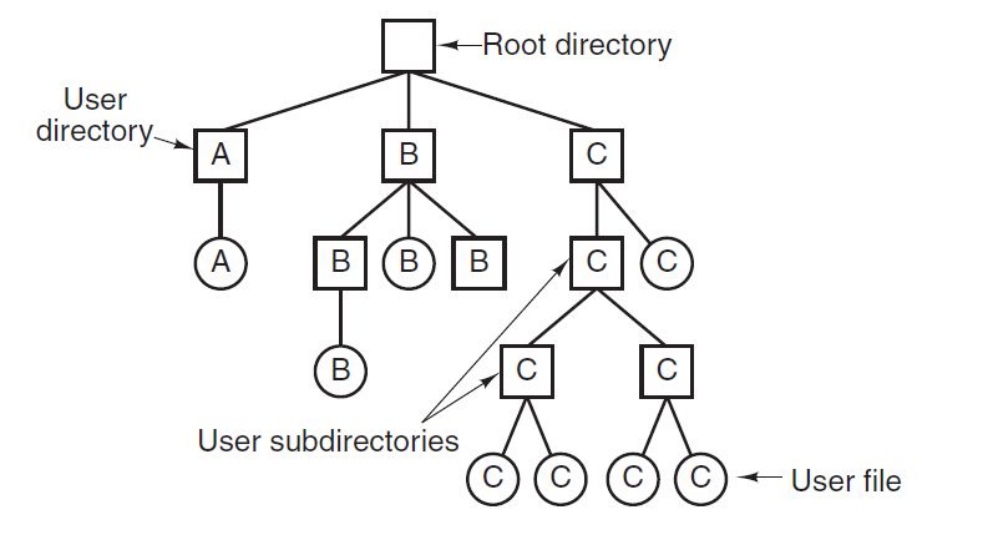
- Arquivos são manipulados através de chamadas de sistemas
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write
  - Append
  - Seek
  - Get attributes
  - Set Attributes
  - Rename

Diretórios

- Para controlar os arquivos, o sistema de arquivos normalmente têm **diretórios ou pastas,** que são em si arquivos
  - Podem ser organizados de duas formas
    - Sistemas de diretório em nível único
    - Sistemas de diretórios hierárquicos

Diretórios


Root directory
A
B
C
D


Root directory
User directory
A
B
C
A
B
B
B
C
C
B
C
C
User subdirectories
C
C
C
C
User file

Nomes de caminhos

- Quando o sistema de arquivos é organizado como uma **árvore de diretórios,** alguma maneira é necessária para especificar nomes de arquivos
  - Dois métodos mais usados:
    - **Nome de caminho absoluto**
      - Ex.: **/usr/ast/caixapostal**
      - Sempre começam pelo diretório raiz e são únicos
    - **Nome de caminho relativo**
      - Usado em conjunção com o conceito de **diretório de trabalho (ou diretório atual)**
      - Todos os nomes de caminho não começando no diretório raiz, são presumidos como **relativos ao diretório de trabalho**
      - Ex.: caso o diretório de trabalho atual é **/usr/ast**, então o arquivo caixapostal pode ser acessado por: ***caixapostal***
      - **Cada processo tem seu próprio diretório de trabalho**

Operações com diretórios

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- link
- unlink

Implementação do Sistema de Arquivos

- Vimos anteriormente a visão do usuário do sistemas de arquivos
- Agora veremos a visão de um implementador
  - Implementadores estão interessados em:
    - Como os arquivos e diretórios são armazenados
    - Como o espaço de disco é gerenciado
    - Como fazer tudo funcionar de maneira eficiente e confiável

Esquema do Sistemas de arquivos

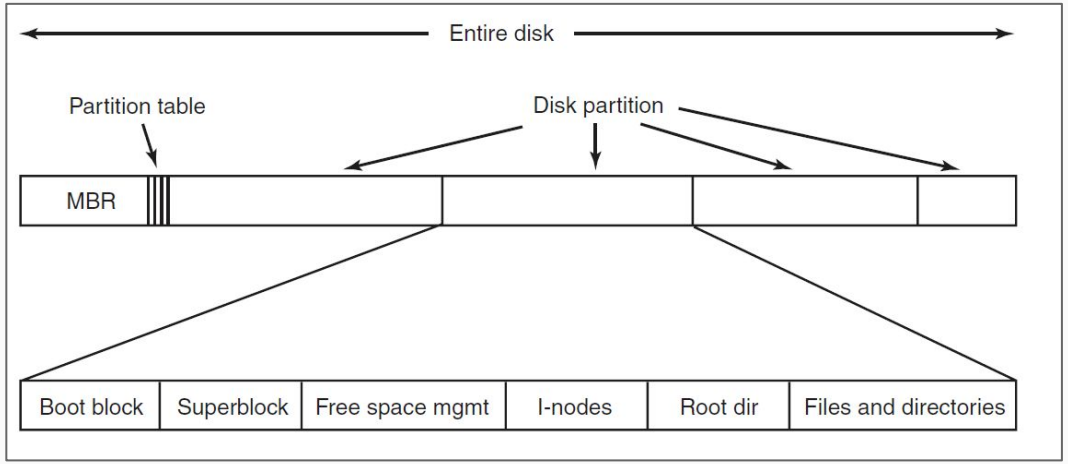
- Sistemas de arquivos são armazenados em discos
  - Maioria dos discos podem ser divididos em uma ou mais **partições,** com sistemas de arquivos independentes em cada partição
  - Setor 0 do disco é chamado de **MBR (*Master Boot Record* - Registro mestre de inicialização)**
    - Usado para inicializar o computador
    - O fim do MBR contém a **tabela de partição**
      - Responsável por dar os endereços de início e fim de cada partição

Esquema do Sistemas de arquivos

- Quando um computador é inicializado
  - A BIOS lê e executa o MBR
  - A primeira coisa que o programa em MBR faz é localizar a **partição ativa**, ler o seu primeiro bloco (**chamado de bloco de inicialização)** e executá-lo
  - Programa do bloco de inicialização carrega o SO contido naquela partição

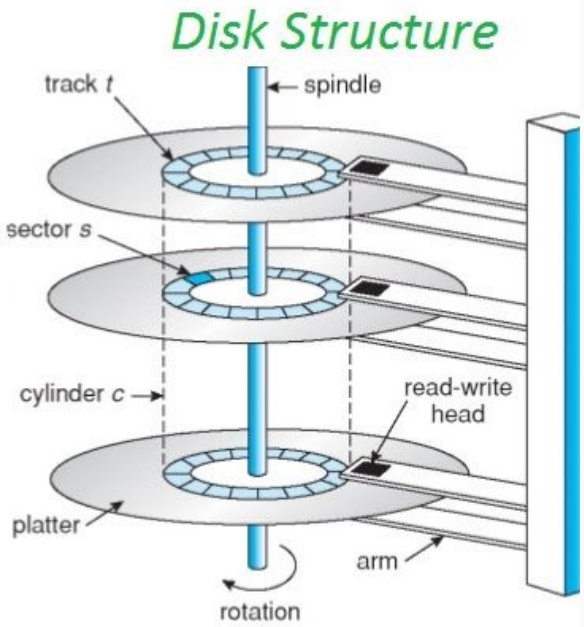
Esquema do Sistemas de arquivos

- Esquema de uma partição de disco varia bastante entre sistemas de arquivo e normalmente contém alguns dos itens mostrados na figura abaixo:
  - **Bloco de iniciação** (*boot block*)
  - **Superbloco:** Contém parâmetros chave a respeito do sistema de arquivos e é lido para a memória quando o computador é inicializado ou o sistemas de arquivos é tocado pela primeira vez
    - Ex. de informações: número mágico, número de blocos e outras informações administrativas fundamentais
  - **Gerenciamento de espaço livre**
  - **I-nodes**
  - **Diretório raiz**
  - **Restante do disco**



Implementando arquivos

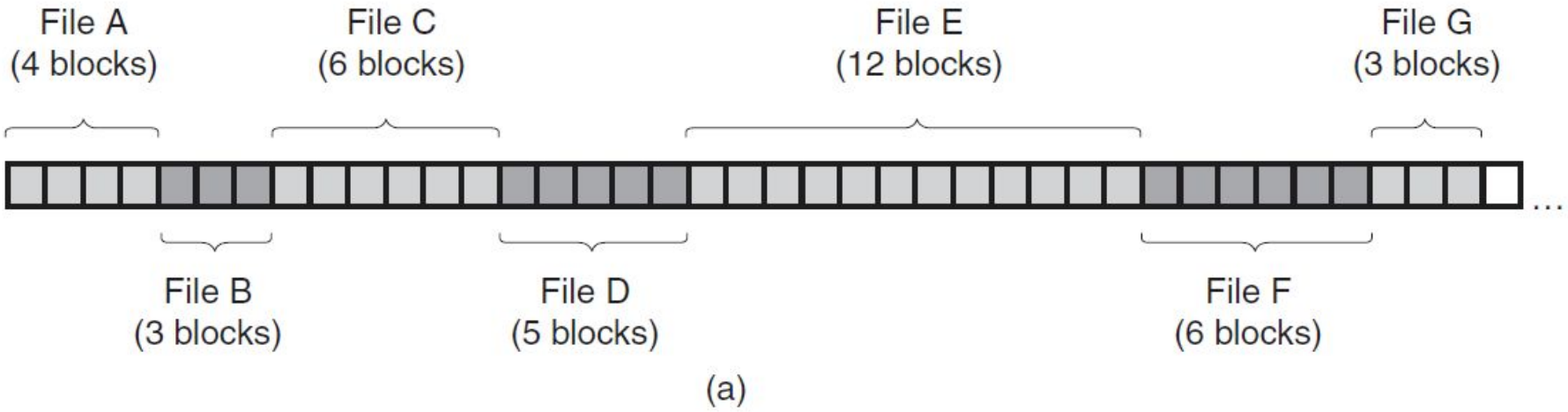
- **Questão mais importante na implementação do armazenamento de arquivo:**
  - **Controlar quais blocos de disco vão com quais arquivos**
- Examinaremos 4 métodos
  - Alocação contígua
  - Alocação por lista encadeada
  - Alocação por lista encadeada usando uma tabela na memória
  - I-nodes



http://www.dbmsinternals.com/database-fundamentals/data-storage/data-storage-magnetic-disk-dbms-perspective/

Alocação contígua

- **Alocação mais simples:** armazenar cada arquivo como uma execução contígua de blocos de disco
  - Assim, em um disco com blocos de 1KB, um arquivo de 50KB seria alocado em 50 blocos consecutivos



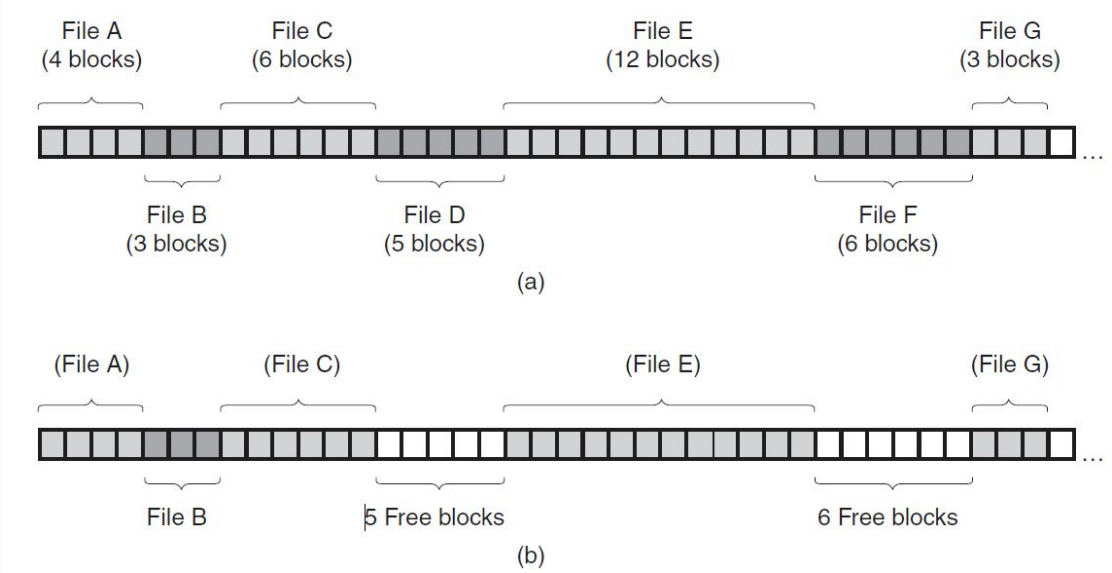
Alocação contígua

Vantagens

- Simples de implementar (endereço do primeiro bloco e número de blocos no arquivo)
- Alto Desempenho da leitura
  - Arquivo pode ser lido do disco em uma única operação

Desvantagens

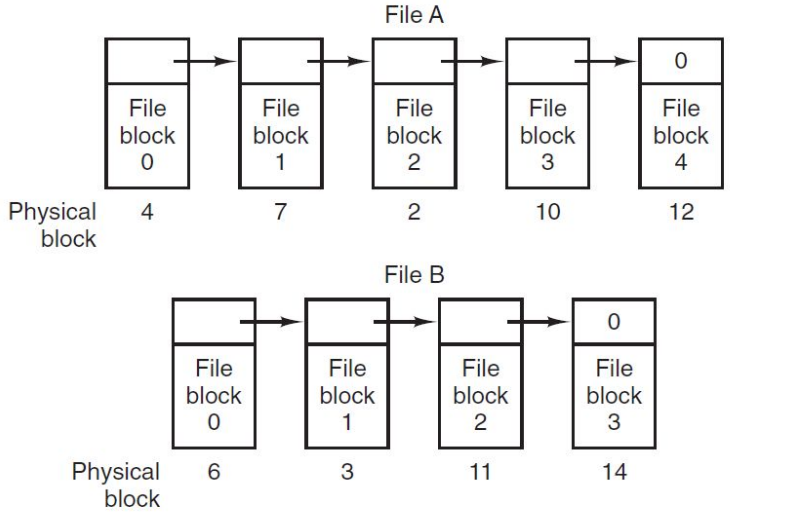
- Fragmentação interna
- Fragmentação externa



Situação onde a alocação contígua é possível: CD/DVD/Bluetooth “ROMs”

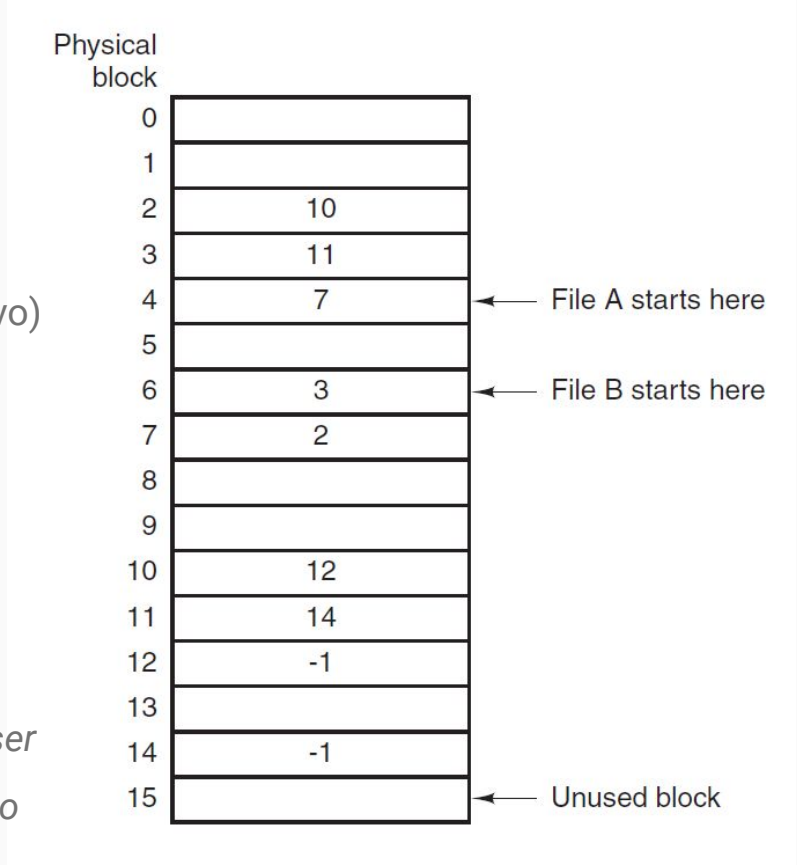
Alocação por lista encadeada

- Ideia: manter cada arquivo como uma lista encadeada de blocos de disco
  - **Primeira palavra de cada bloco:** ponteiro para o próximo bloco usado
  - **Restante do bloco:** reservado para dados
- **Vantagens**
  - Evita fragmentação externa
  - Entrada de diretório é suficiente apenas armazenar endereço de disco do primeiro bloco
- **Desvantagens**
  - Leitura de um arquivo dolorosamente lento!
  - Quantidade de dados que um bloco pode armazenar não é mais uma potência de dois
    - Maioria dos programas lêem e escrevem blocos de tamanho em potência de dois
    - Leitura de todo bloco exigiria a concatenação de dois blocos (sobrecarga extra)



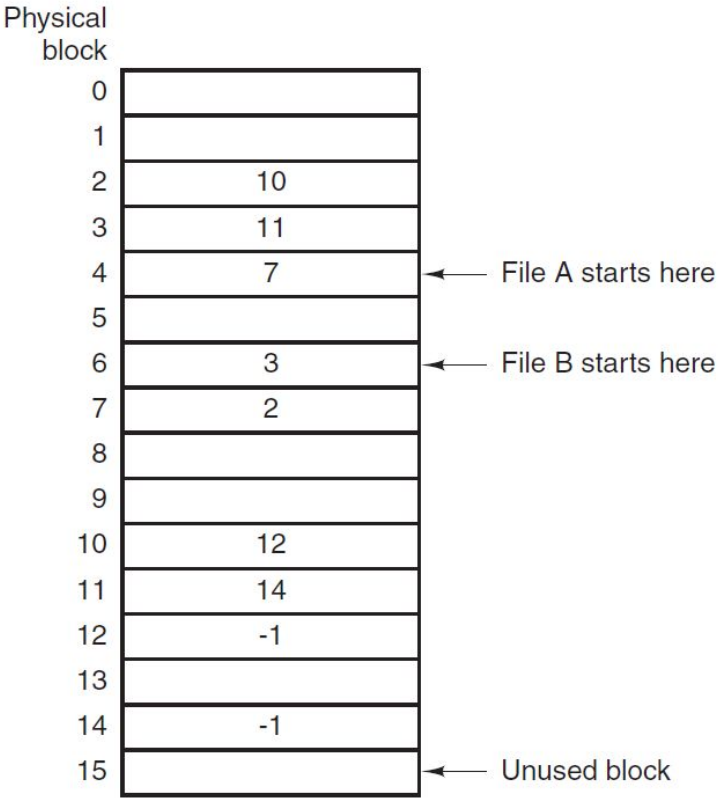
Alocação por lista encadeada usando uma tabela na memória

- Ideia: Ambas as desvantagens anteriores podem ser eliminadas
  - Colocando as palavras do ponteiro de cada bloco de disco em uma tabela na memória principal
  - -1 indica um bloco inválido (último bloco de um arquivo)
- Essa tabela na memória é chamada de **FAT**
  - **FAT (*File allocation Table*- Tabela de Alocação de arquivo)**
  - O bloco inteiro fica disponível para os dados
  - *Acesso aleatório muito mais fácil*
    - *Todo o encadeamento está na memória e pode ser seguido sem fazer quaisquer referências ao disco*



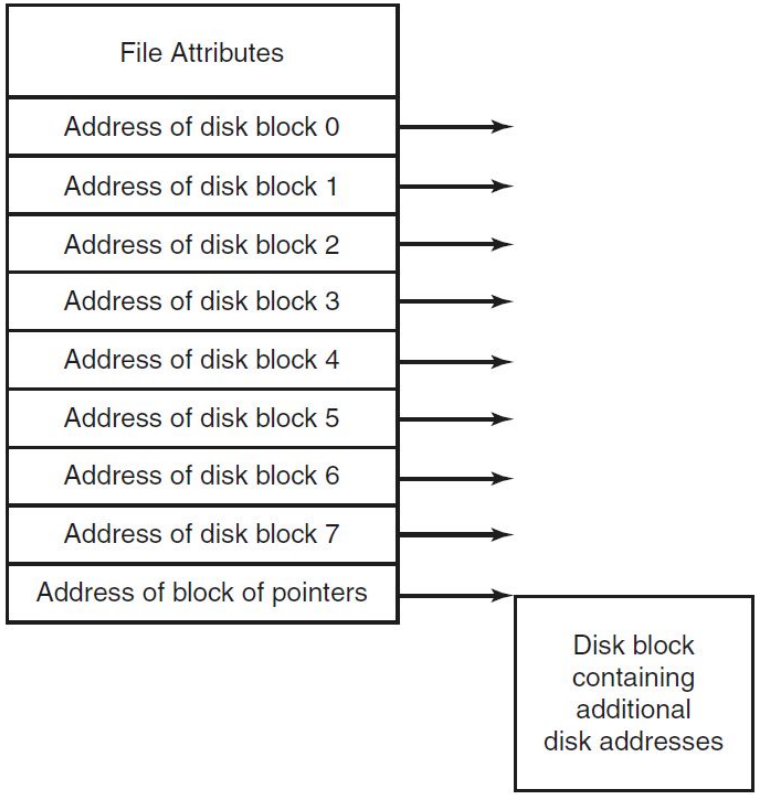
Alocação por lista encadeada usando uma tabela na memória

- Principal desvantagem
  - A tabela inteira precisa estar na memória o tempo todo para fazê-la funcionar
  - Ex.: um disco de 1 TB e um tamanho de bloco de 1KB tem que ter **1 bilhão de entradas para cada 1 bilhão de blocos de disco**
    - A tabela ocupará algo em torno de 2,4 GB a 3GB da memória principal o tempo inteiro!!
  - **FAT não se adapta bem para discos grandes**
    - Usado no MS-DOS, mas é aceito em todas as versões do Windows



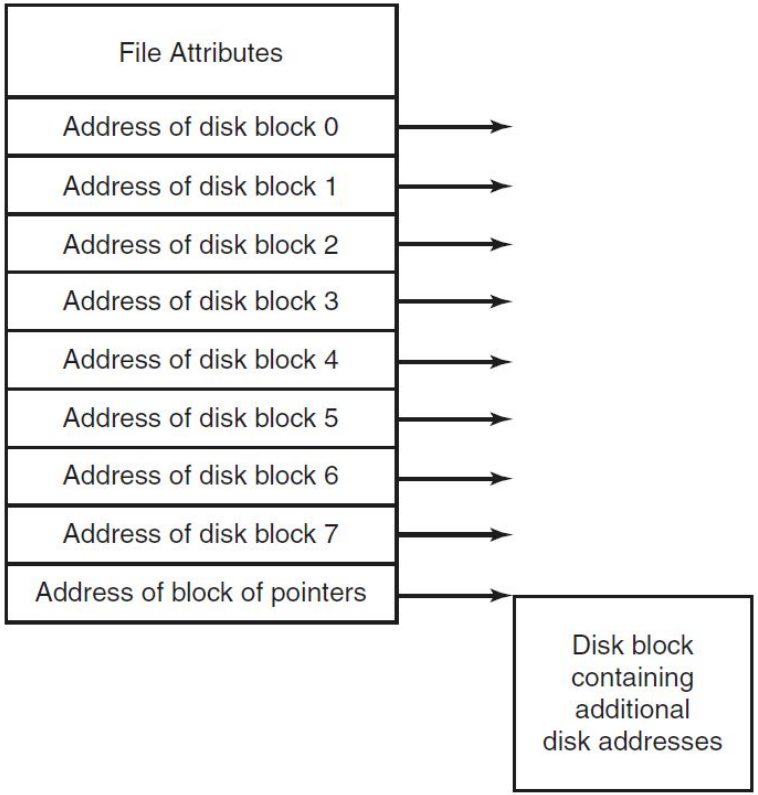
I-nodes

- **Ideia:** associar cada arquivo a uma estrutura de dados chamada de **i-node (nó índice)**
  - Lista os atributos e os endereços dos blocos do disco
  - Dado um i-node, é possível encontrar todos os blocos do arquivo
- Vantagem
  - I-node precisa estar na memória apenas quando o arquivo correspondente estiver aberto
    - Se cada i-node ocupa *n* byte e um máximo de *k* arquivos puderem estar abertos simultaneamente, a memória total usada pelos arquivos abertos será de ***k.n bytes***



I-nodes

- Tamanho ocupado em memória é proporcional ao número máximo de arquivos que podem ser abertos ao mesmo tempo
  - Não importa se meu disco tem 100 GB, 1 TB ou 10 TB
- **Problema**: um i-node tem espaço para um número fixo de endereços de disco
  - O que fazer quando um arquivo cresce além de seu limite?
  - **Solução**: reservar o último endereço de disco não para um bloco de dados, mas para um **endereço de blocos de disco apontando para outros blocos cheios de endereços.**
- UNIX em geral usam essa abordagem (Windows usa idéia semelhante - NTFS)



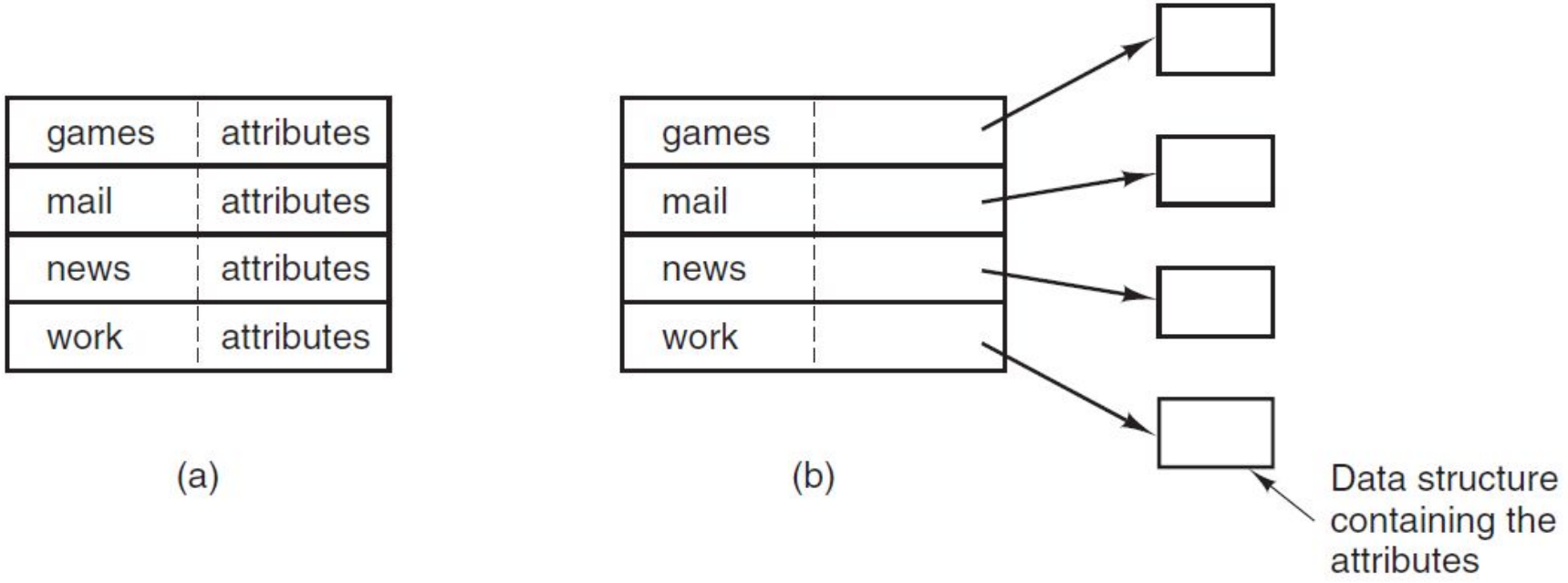
Implementando diretórios

- Quando um arquivo é aberto, o SO usa o nome do caminho fornecido pelo usuário para localizar a entrada de diretório no disco
  - **A entrada de diretório** fornece a informação necessária para encontrar os blocos de disco
  - Essa informação pode ser
    - Endereço de disco do arquivo inteiro (Alocação contígua)
    - Número do primeiro bloco (listas encadeadas)
    - Número do i-node
  - Em todos os casos, a **principal função do sistema de diretórios** é mapear o nome do arquivo em ASCII na informação necessária para localizar os dados

Implementando diretórios

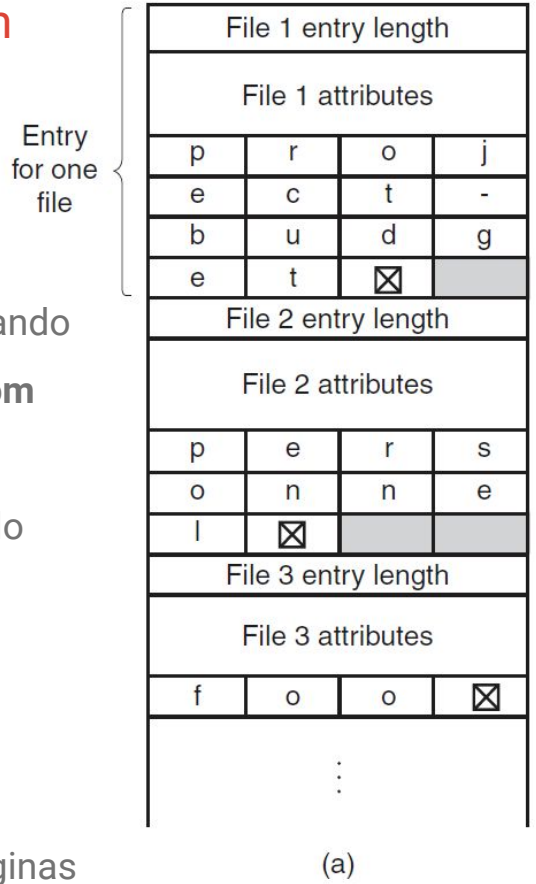
- Questão relacionada: **Onde os atributos de arquivos devem ser armazenados?**
  - **Diretamente na entrada do diretório**
    - Diretório consistirá de uma lista de entradas de tamanho fixo, um por arquivo
    - Deve conter em cada entrada um **nome de arquivo** (tamanho fixo), uma **estrutura dos atributos do arquivo** e um ou mais endereços de disco (dizendo onde estão os blocos de disco)
  - **Armazenar os atributos nos próprios i-nodes**
    - Nesse caso, a entrada do diretório pode ser curta: apenas o nome de arquivo e um número de i-node (mais vantajoso)

Implementando diretórios



Implementando diretórios

- **Outro Problema**: quase todos os SO modernos aceitam nomes de arquivos maiores e de tamanho variável
  - **Alternativa:** abrir mão da ideia de que todas as entradas de diretório sejam do mesmo tamanho
    - Cada entrada de diretório contém uma porção fixa, começando com o **tamanho da entrada** e, então seguidos por **dados com um formato fixo (atributos)**
    - Após este cabeçalho de tamanho fixo, é inserido o nome do arquivo real
  - Desvantagem:
    - Quando um arquivo é removido do diretório...(similar ao problema da fragmentação externa)
    - Única entrada de diretório pode estender por múltiplas páginas



Implementando diretórios - Busca

- Diretórios normalmente são pesquisados linearmente, do início ao fim quando o nome de um arquivo precisa ser procurado
  - Para diretórios extremamente longos, a busca linear pode ser bem lenta
- Maneira de acelerar a busca: usar **tabela de espalhamento** em cada diretório
  - Defina uma tabela de espalhamento de tamanho **n**
    - Ao entrar com um nome de arquivo, o nome é mapeado em um valor entre 0 e n-1 (mod n)
  - **Entradas de arquivos seguem a tabela de espalhamento**
    - Se aquela vaga já estiver em uso, uma lista encadeada é construída, inicializada naquela entrada da tabela, unindo todas as entradas com o mesmo valor de espalhamento
    - **Vantagem:** busca muito mais rápida de arquivos e diretórios
    - **Desvantagem:** Administração da tabela é complexa
  - É uma séria alternativa para sistemas em que é esperado que os diretórios contenham centenas ou milhares de arquivos

Implementando diretórios

- **Outra maneira de acelerar a busca:** colocar os resultados em uma cache de buscas
  - Antes de começar a busca, é feito primeiro uma verificação para ver se o nome do arquivo está na cache
    - Se estiver, pode ser localizado imediatamente
  - Esta abordagem só é útil se um número relativamente pequeno de arquivos compreende a maioria das buscas

Próxima aula

- Arquivos Compartilhados
- Tipos de Sistemas de Arquivos
  - Sistemas de arquivos estruturados em diário (log)
  - Sistemas de Arquivos *journailing*
  - Sistemas de arquivos virtuais

Referências

Tanenbaum, A. S. e Bos, H.. Sistemas Operacionais Modernos. 4.ed. Pearson/Prentice-Hall. 2016.