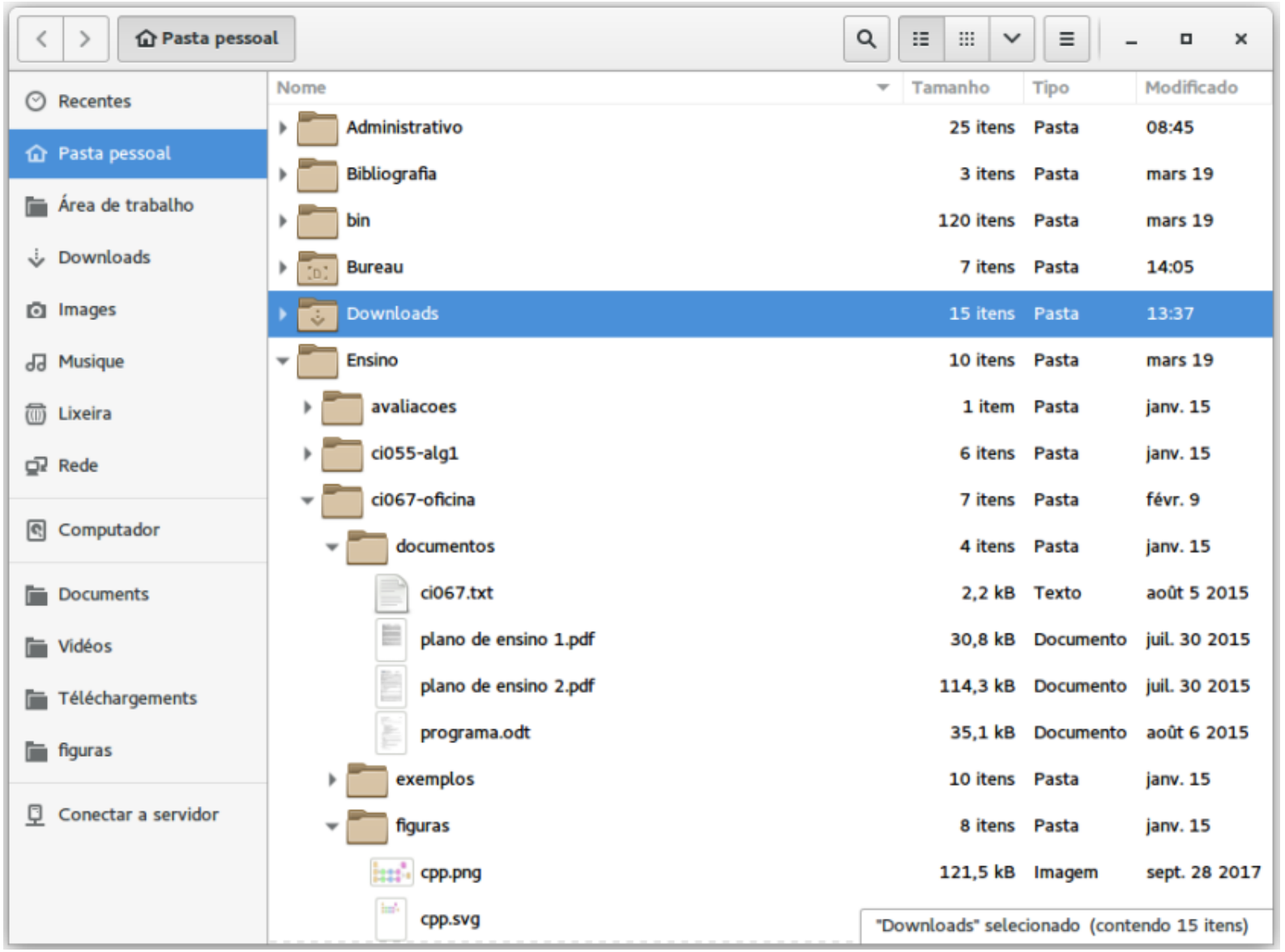
Sistemas de Arquivos

Prof. Paulo Valim

Introdução

- Conceito de Arquivo: um **arquivo** é essencialmente uma sequência de bytes armazenada em um dispositivo físico não volátil, como um disco rígido ou de estado sólido, que preserva seu conteúdo mesmo quando desligado.
- Cada arquivo possui um nome, ou outra referência, que permite sua localização e acesso.
- Como um dispositivo de armazenamento pode conter milhões de arquivos, estes são organizados em estruturas hierárquicas denominadas **diretórios**, para facilitar sua localização e acesso pelos usuários.
  A organização do conteúdo dos arquivos e diretórios dentro de um dispositivo físico é denominada **sistema de arquivos**.
- Um sistema de arquivos pode ser visto como uma imensa estrutura de dados armazenada de forma persistente no dispositivo físico.
- Finalmente, um dispositivo físico é estruturado em um ou mais **volumes** (ou partições); cada volume pode armazenar um sistema de arquivos próprio. Assim, um mesmo disco pode conter volumes com diferentes sistemas de arquivos, como FAT, NTFS ou EXT4, por exemplo.


Pasta pessoal
Recentes
Pasta pessoal
Área de trabalho
Downloads
Images
Musique
Lixeira
Rede
Computador
Documents
Vidéos
Téléchargements
figuras
Conectar a servidor
Nome
Tamanho
Tipo
Modificado
Administrativo 25 itens Pasta 08:45
Bibliografia 3 itens Pasta mars 19
bin 120 itens Pasta mars 19
Bureau 7 itens Pasta 14:05
Downloads 15 itens Pasta 13:37
Ensino 10 itens Pasta mars 19
avaliacoes 1 item Pasta janv. 15
ci055-alg1 6 itens Pasta janv. 15
ci067-oficina 7 itens Pasta févr. 9
documentos 4 itens Pasta janv. 15
ci067.txt 2,2 kB Texto août 5 2015
plano de ensino 1.pdf 30,8 kB Documento juil. 30 2015
plano de ensino 2.pdf 114,3 kB Documento juil. 30 2015
programa.odt 35,1 kB Documento août 6 2015
exemplos 10 itens Pasta janv. 15
figuras 8 itens Pasta janv. 15
cpp.png 121,5 kB Imagem sept. 28 2017
cpp.svg
"Downloads" selecionado (contendo 15 itens)

Arquivos: atributos

- **Nome:** *string* que identifica o arquivo para o usuário, como “foto1.jpg”, “documento. pdf”, “hello.c”, etc.;
- **Tipo:** indicação do formato dos dados contidos no arquivo, como áudio, vídeo, imagem, texto, etc. Muitos sistemas operacionais usam parte do nome do arquivo para identificar o tipo de seu conteúdo, na forma de uma extensão: “.doc”, “.jpg”, “.mp3”, etc.;
- **Tamanho:** indicação do tamanho do conteúdo do arquivo, geralmente em bytes;
- **Datas:** para fins de gerência, é importante manter as datas mais importantes relacionadas ao arquivo, como suas datas de criação, de último acesso e de última modificação do conteúdo;
- **Proprietário:** em sistemas multiusuários, cada arquivo tem um proprietário, que deve estar corretamente identificado;
- **Permissões de acesso:** indicam que usuários têm acesso àquele arquivo e que formas de acesso são permitidas (leitura, escrita, remoção, etc.);
- **Localização:** indicação do dispositivo físico onde o arquivo se encontra e da posição do arquivo dentro do mesmo;

Arquivos: operações

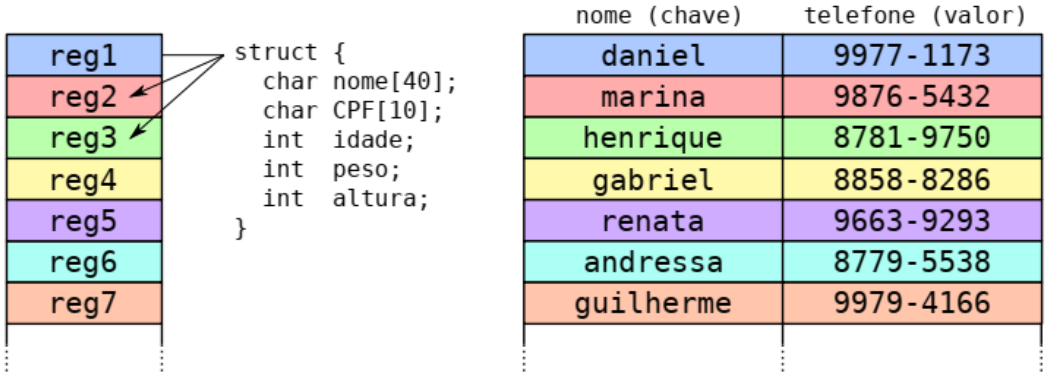
- **Criar:** a criação de um novo arquivo implica em alocar espaço para ele no dispositivo de armazenamento e definir valores para seus atributos (nome, localização, proprietário, permissões de acesso, datas, etc.);
- **Abrir:** antes que uma aplicação possa ler ou escrever dados em um arquivo, ela deve solicitar ao sistema operacional a “abertura” desse arquivo. O sistema irá então verificar se o arquivo desejado existe, verificar se as permissões associadas ao arquivo permitem aquele acesso, localizar seu conteúdo no dispositivo de armazenamento e criar uma referência para ele na memória da aplicação;
- **Ler:** permite transferir dados presentes no arquivo para uma área de memória da aplicação;
- **Escrever:** permite transferir dados na memória da aplicação para o arquivo no dispositivo físico; os novos dados podem ser adicionados ao final do arquivo ou sobrescrever dados já existentes;
- **Fechar:** ao concluir o uso do arquivo, a aplicação deve informar ao sistema operacional que o mesmo não é mais necessário, a fim de liberar as estruturas de gerência do arquivo mantidas na memória do núcleo;
- **Remover:** para eliminar o arquivo do dispositivo, descartando seus dados e liberando o espaço ocupado por ele.
- **Alterar atributos:** para modificar os valores dos atributos do arquivo, como nome, proprietário, permissões, datas, etc.

Arquivos: formatos

- **Sequência de bytes:** Em sua forma mais simples, um arquivo contém basicamente uma sequência de bytes. Essa sequência de bytes pode ser estruturada de forma a representar diferentes tipos de informação, como imagens, música, textos ou código executável.

  Uma aplicação pode definir um formato próprio para armazenar seus dados, ou pode seguir formatos padronizados. Por exemplo, há um grande número de formatos padronizados para o armazenamento de imagens, como JPEG, GIF, PNG e TIFF, mas também existem formatos de arquivos proprietários, definidos por algumas aplicações específicas.

- **Arquivos de registros:** Alguns núcleos de sistemas operacionais oferecem arquivos com estruturas internas que vão além da simples sequência de bytes. Por exemplo, o sistema *OpenVMS* [Rice, 2000] proporciona *arquivos baseados em registros*, cujo conteúdo é visto pelas aplicações como uma sequência linear de registros de tamanho fixo ou variável, e também *arquivos indexados*, nos quais podem ser armazenados pares {*chave*/*valor*}, de forma similar a um banco de dados relacional.



Arquivos: formatos

- Arquivos de texto: Um tipo de arquivo de uso muito frequente é o arquivo de *texto puro* (ou *plain text*). Esse tipo de arquivo é usado para armazenar informações textuais simples, como códigos-fonte de programas, arquivos de configuração, páginas HTML, dados em XML, etc.

```
1 int␣main()
2 {
3 ␣␣printf("Hello,␣world\n");
4 ␣␣exit(0);
5 }
```

Unix

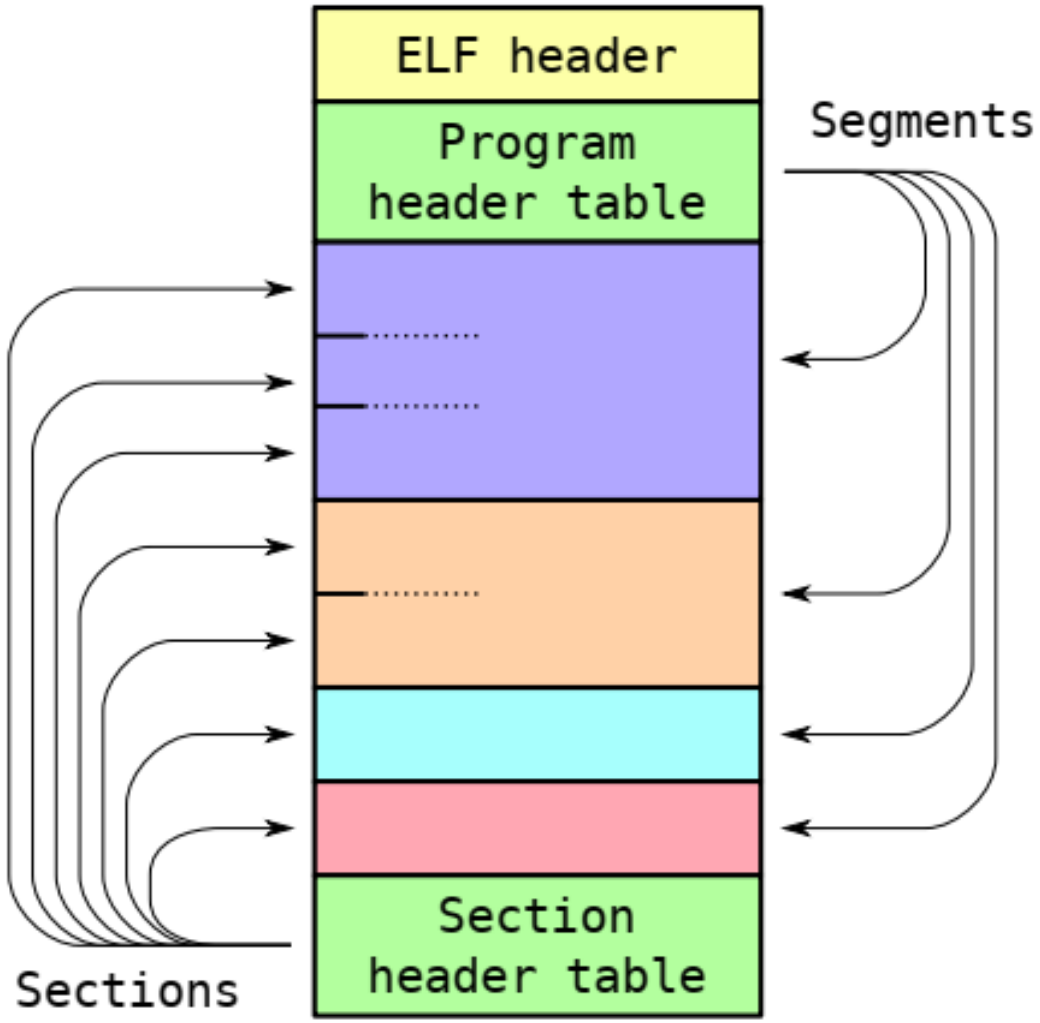
```
0000  69 6e 74 20 6d 61 69 6e 28 29 0a 7b 0a 20 20 70
       i  n  t  ␣  m  a  i  n  (  ) \n  {  \n ␣  ␣  p
0010  72 69 6e 74 66 28 22 48 65 6c 6c 6f 2c 20 77 6f
       r  i  n  t  f  (  "  H  e  l  l  o  ,  ␣  w  o
0020  72 6c 64 5c 6e 22 29 3b 0a 20 20 65 78 69 74 28
       r  l  d  \  n  "  )  ;  \n ␣  ␣  e  x  i  t  (
0030  30 29 3b 0a 7d 0a
       0  )  ;  \n  }  \n
```

DOS/Windows

```
0000  69 6e 74 20 6d 61 69 6e 28 29 0d 0a 7b 0d 0a 20
       i  n  t  ␣  m  a  i  n  (  ) \r \n  {  \r \n ␣
0010  20 70 72 69 6e 74 66 28 22 48 65 6c 6c 6f 2c 20
       ␣  p  r  i  n  t  f  (  "  H  e  l  l  o  ,  ␣
0020  77 6f 72 6c 64 5c 6e 22 29 3b 0d 0a 20 20 65 78
       w  o  r  l  d  \  n  "  )  ;  \r \n ␣  ␣  e  x
0030  69 74 28 30 29 3b 0d 0a 7d 0d 0a
       i  t  (  0  )  ;  \r \n  }  \r \n
```

Arquivos: formatos

- Arquivos de códigos: Em um sistema operacional moderno, um arquivo de código (programa executável ou biblioteca) é dividido internamente em várias seções, para conter código, tabelas de símbolos (variáveis e funções), listas de dependências (bibliotecas necessárias) e outras informações de configuração. A organização o interna de um arquivo de código depende do sistema operacional para o qual foi definido.
  - **ELF** (*Executable and Linking Format*): formato de arquivo usado para programas executáveis e bibliotecas na maior parte das plataformas UNIX modernas. É composto por um cabeçalho e várias seções de dados, contendo código executável, tabelas de símbolos e informações sobre relocação de código, usadas quando o código executável é carregado na memória.
  - **PE** (*Portable Executable*): é o formato usado para executáveis e bibliotecas na plataforma Windows. Consiste basicamente em uma extensão do formato COFF (Common Object File Format), usado em plataformas UNIX mais antigas.



Estrutura interna de um arquivo em formato ELF [Levine, 2000].

Arquivos: identificação de conteúdos

- Um problema importante relacionado aos formatos de arquivos é a correta identificação de seu conteúdo pelos usuários e aplicações. Já que um arquivo de dados pode ser visto como uma simples sequência de bytes, como é possível reconhecer que tipo de informação essa sequência representa?
- Uma solução simples para esse problema consiste em usar parte do nome do arquivo para indicar o tipo do conteúdo. A estratégia de **extensão do nome**, utilizada ainda hoje na maioria dos sistemas operacionais, foi introduzida nos anos 1980 pelo sistema operacional DOS.
- Outra abordagem, frequentemente usada em sistemas UNIX, é usar alguns bytes no início do conteúdo do arquivo para a definição de seu tipo. Esses bytes iniciais do conteúdo são denominados números mágicos (magic numbers), e são convencionados para muitos tipos de arquivos.

Tipo de arquivo	bytes iniciais	Tipo de arquivo	bytes iniciais
Documento PostScript	`%!`	Documento PDF	`%PDF`
Imagem GIF	`GIF89a`	Imagem JPEG	`0xFF D8 FF`
Música MIDI	`MThd`	Classes Java	`0xCA FE BA BE`
Arquivo ZIP	`0x50 4B 03 04`	Documento RTF	`{\rtf1`

Arquivos: arquivos especiais

- O conceito de arquivo é ao mesmo tempo simples e poderoso, o que motivou sua utilização de forma quase universal. Além do armazenamento de dados do sistema operacional e de aplicações, como mostrado na seção anterior, o conceito de arquivo também pode ser usado como:
    - **Abstração de dispositivos de entrada/saída**
    - **Abstração de dispositivos de entrada/saída**
    - **Abstração de interfaces do núcleo: Canais de comunicação**

Arquivos: interface de acesso

- A interface de acesso a um arquivo normalmente é composta por uma representação lógica do arquivo, denominada **descritor de arquivo** (*file descriptor* ou *file handle*), e um conjunto de funções para manipular o arquivo.
- Existem dois níveis de interface de acesso: uma **interface de baixo nível**, oferecida pelo sistema operacional aos processos através de chamadas de sistema, e uma interface de alto nível, composta de funções na linguagem de programação usada para implementar cada aplicação.

Operação	Linux	Windows
Abrir arquivo	`OPEN`	`NtOpenFile`
Ler dados	`READ`	`NtReadRequestData`
Escrever dados	`WRITE`	`NtWriteRequestData`
Fechar arquivo	`CLOSE`	`NtClose`
Remover arquivo	`UNLINK`	`NtDeleteFile`
Criar diretório	`MKDIR`	`NtCreateDirectoryObject`

Operação	C (padrão C99)	Java (classe `File`)
Abrir arquivo	`fd = fopen(...)`	`obj = File(...)`
Ler dados	`fread(fd, ...)`	`obj.read()`
Escrever dados	`fwrite(fd, ...)`	`obj.write()`
Fechar arquivo	`fclose(fd)`	`obj.close()`
Remover arquivo	`remove(...)`	`obj.delete()`
Criar diretório	`mkdir(...)`	`obj.mkdir()`

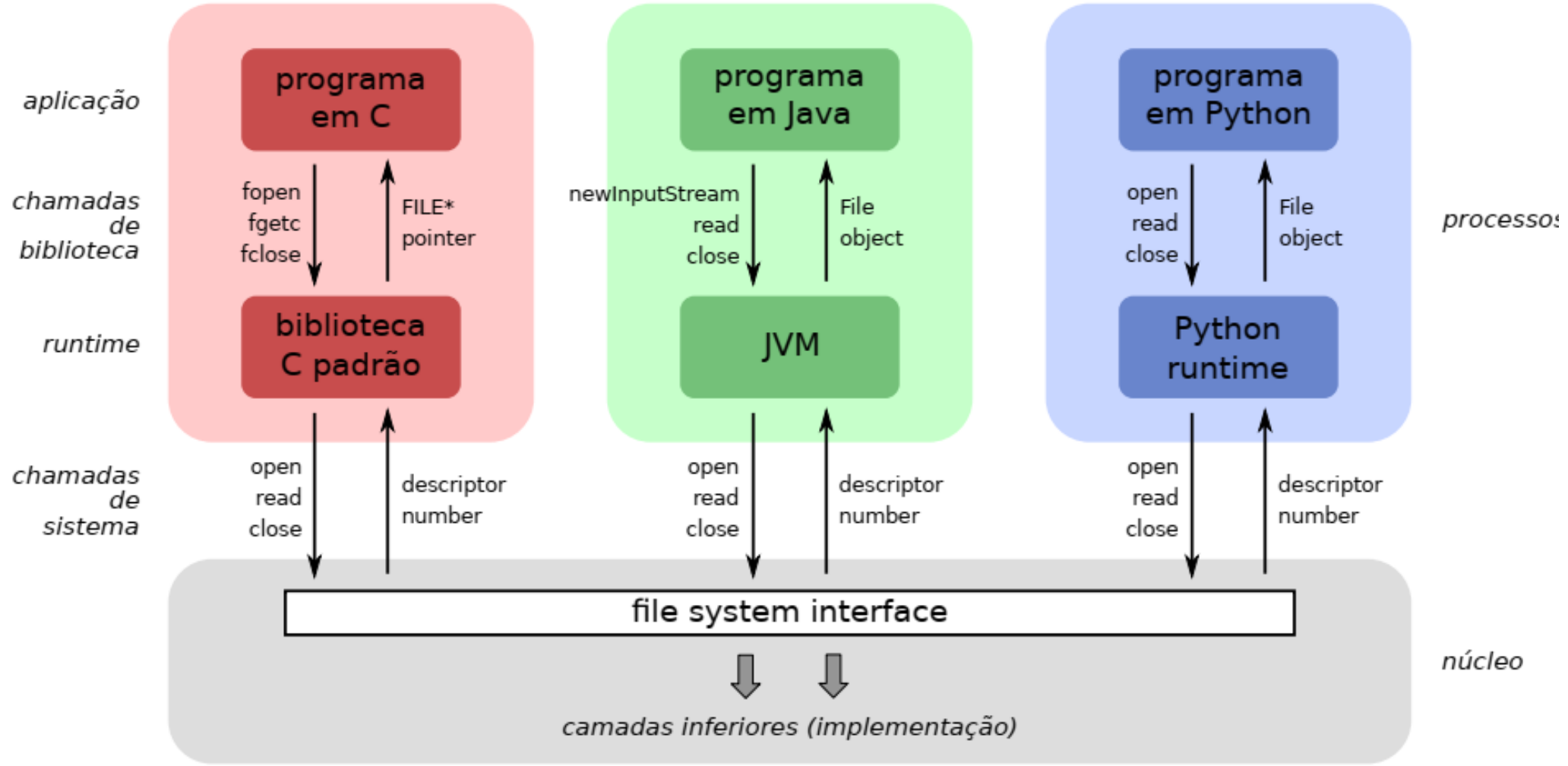
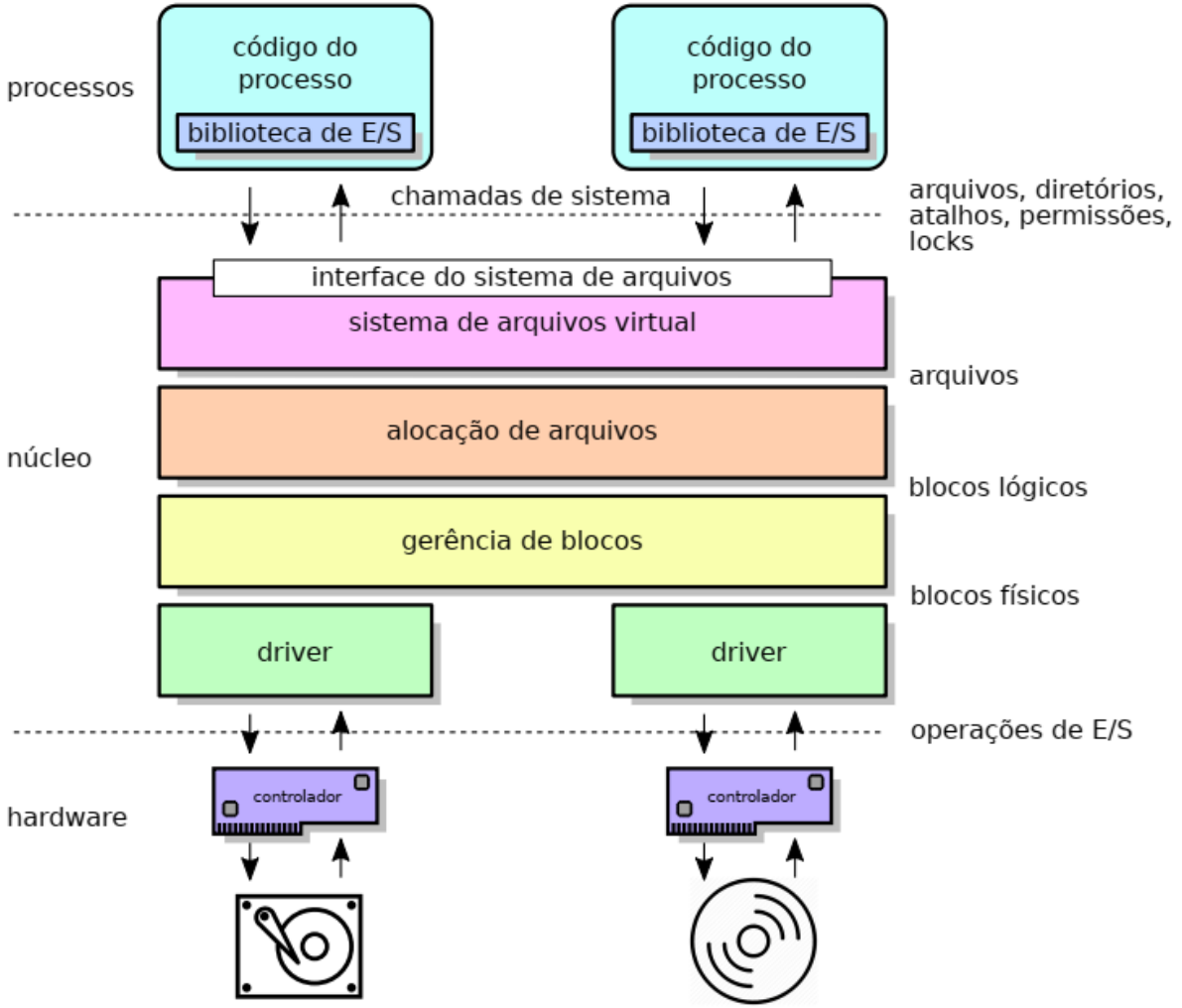


Figura 23.1: Relação entre funções de linguagem e chamadas de sistema.

Sistemas de Arquivos

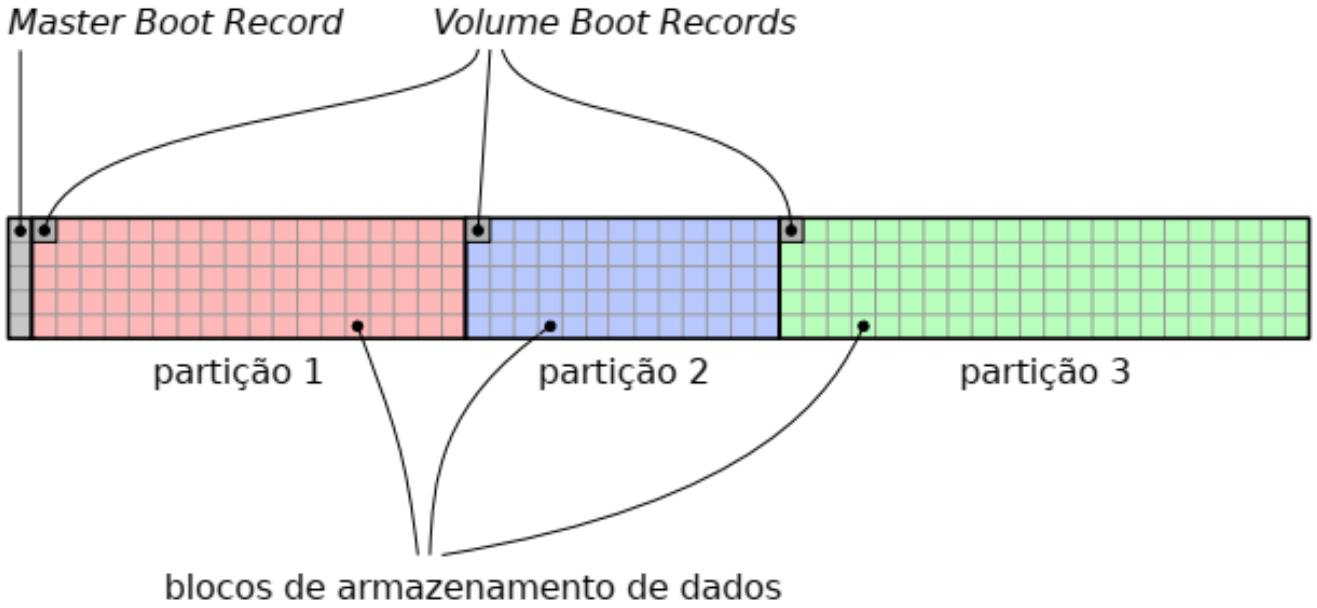
- Vários problemas importantes devem ser resolvidos para a implementação eficiente de arquivos e diretórios, que vão dos aspectos de baixo nível, como o acesso aos dispositivos físicos, a aspectos mais abstratos, como a implementação da interface de acesso a arquivos para os programadores.



Arquitetura Geral

Sistemas de Arquivos

- Discos e partições: Em linhas gerais, um disco é visto pelo sistema operacional como um grande vetor de blocos de dados de tamanho fixo, numerados sequencialmente. As operações de leitura e escrita de dados nesses dispositivos são feitas bloco a bloco, por essa razão esses dispositivos são chamados dispositivos de blocos (block devices ou block-oriented devices).



Sistemas de Arquivos

- Montagem de volumes: Para que o sistema operacional possa acessar os arquivos armazenados em um volume, ele deve ler os dados presentes em seu bloco de inicialização, que descrevem o tipo de sistema de arquivos do volume, e criar as estruturas em memória que representam esse volume dentro do núcleo do SO. Além disso, ele deve definir um identificador para o volume, de forma que os processos possam acessar seus arquivos.

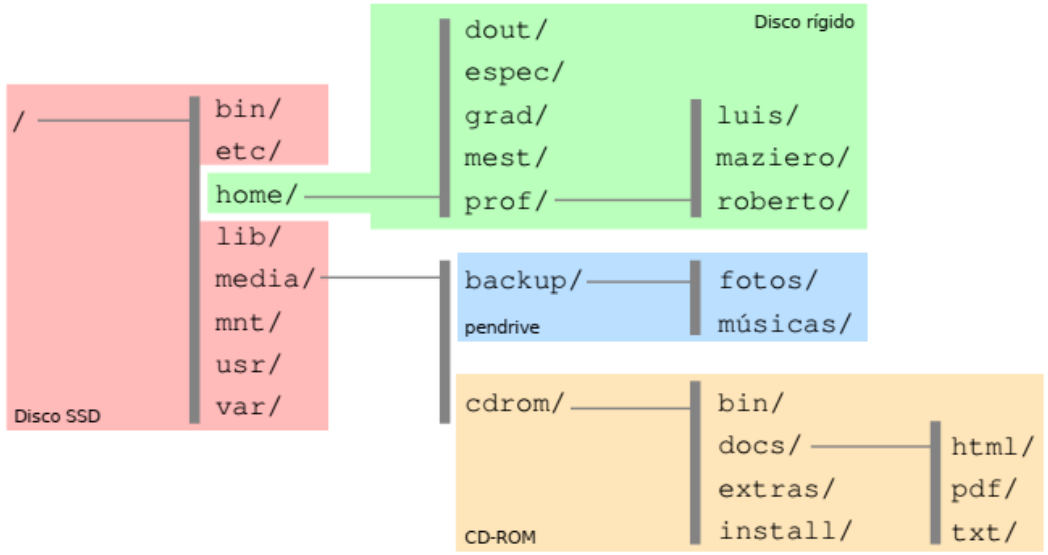


Figura 24.3: Montagem de volumes em UNIX.

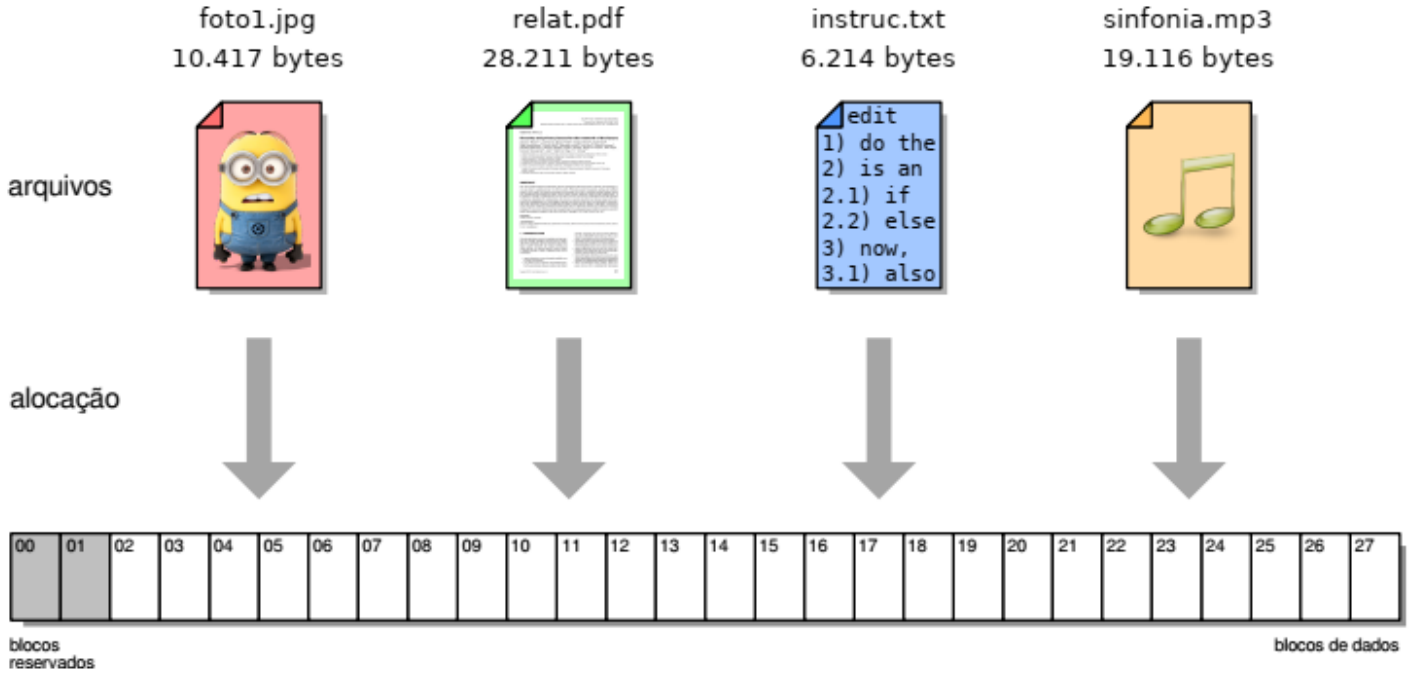
Sistemas de Arquivos

Blocos físicos e blocos lógicos

- um disco rígido pode ser visto como um conjunto de blocos de tamanho fixo (geralmente de 512 ou 4.096 bytes). Os blocos do disco rígido são normalmente denominados *blocos físicos*.
- Para simplificar a gerência da imensa quantidade de blocos físicos e melhorar o desempenho das operações de leitura/escrita, os sistemas operacionais costumam agrupar os blocos físicos em *blocos lógicos* ou *clusters*, que são grupos de 2*n* blocos físicos consecutivos. Definem a unidade mínima de alocação de arquivos e diretórios: cada arquivo ou diretório ocupa um ou mais blocos lógicos para seu armazenamento.
- O número de blocos físicos em cada bloco lógico é fixo e definido pelo sistema operacional ao formatar a partição, em função de vários parâmetros, como o tamanho da partição, o sistema de arquivos etc
- Blocos lógicos maiores (32 KB ou 64 KB) levam a uma menor quantidade de blocos lógicos a gerenciar pelo SO em cada disco e implicam em mais eficiência de entrada/saída, pois mais dados são transferidos em cada operação. Entretanto, blocos grandes podem gerar muita *fragmentação interna*.

Sistemas de Arquivos

- Alocação de arquivos: um espaço de armazenamento é visto pelas camadas superiores do sistema operacional como um grande vetor de blocos lógicos de tamanho fixo. O problema da alocação de arquivos consiste em dispor (alocar) o conteúdo e os metadados dos arquivos dentro desses blocos.

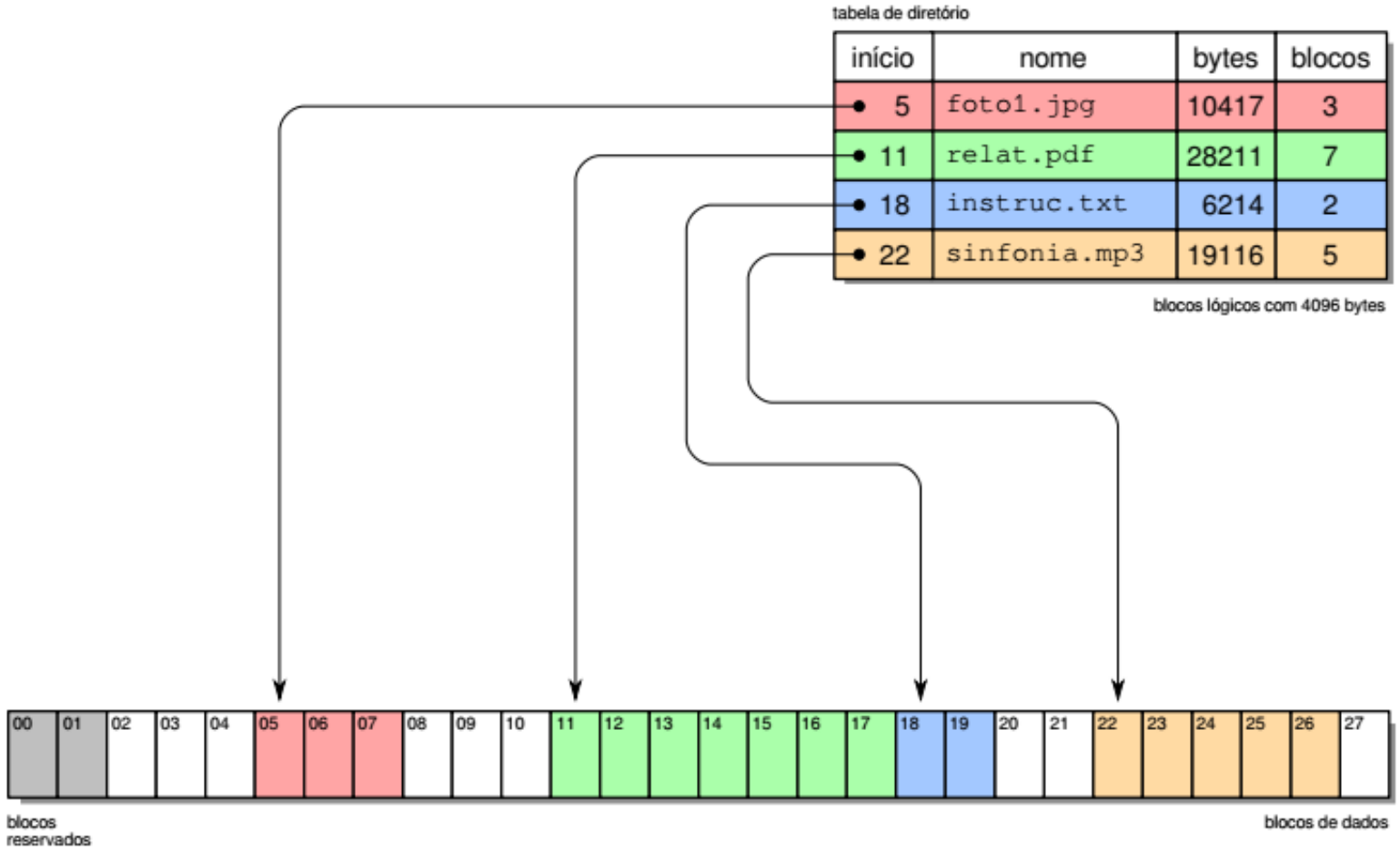


Alocação de Arquivos

- 3 estratégias a serem seguidas: as alocações **contígua**, **encadeada** e **indexada**

- O que deve ser considerado para a escolha: **rapidez, robustez e flexibilidade**
- Um conceito importante na alocação de arquivos é o **bloco de controle de arquivo** (FCB - *File Control Block*), que nada mais é que uma estrutura contendo os metadados do arquivo e uma referência para a localização de seu conteúdo no disco.
- A implementação do FCB depende do sistema de arquivos: em alguns pode ser uma simples entrada na tabela de diretório, enquanto em outros é uma estrutura de dados separada, como a *Master File Table* do sistema NTFS e os *i-nodes* dos sistemas UNIX.

Alocação contígua

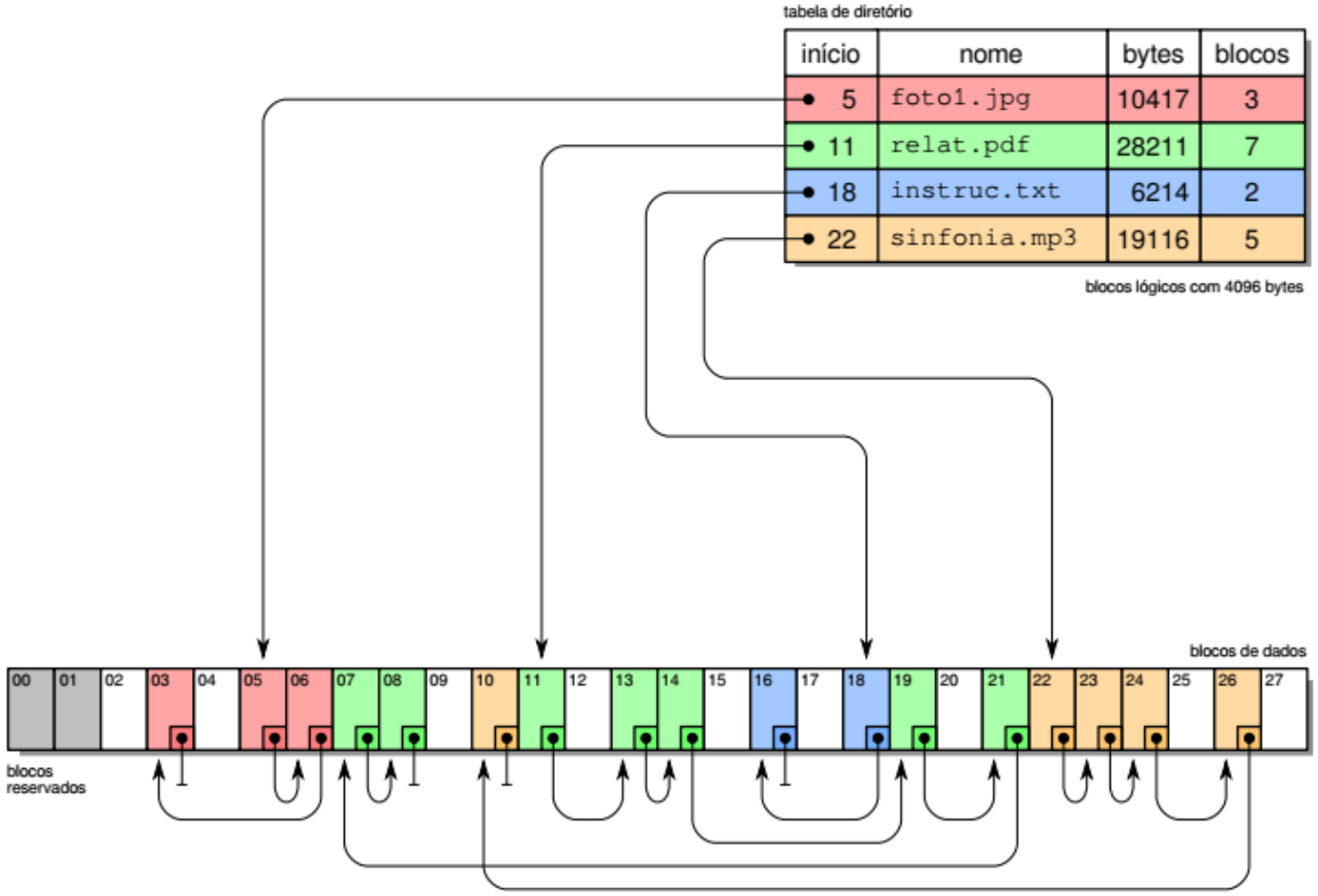


Alocação contígua

- boa robustez a falhas de disco: caso um bloco do disco apresente defeito e não permita a leitura dos dados contidos nele, apenas o conteúdo daquele bloco é perdido: o conteúdo do arquivo nos blocos anteriores e posteriores ao bloco defeituoso ainda poderão ser acessados normalmente.
- baixa flexibilidade, pois o tamanho máximo de cada arquivo precisa ser conhecido no momento de sua criação.
- Outro problema desta estratégia é a fragmentação externa: à medida em que arquivos são criados e destruídos, as áreas livres do disco vão sendo divididas em pequenas áreas isoladas (os fragmentos) que diminuem a capacidade de alocação de arquivos maiores.
- A baixa flexibilidade desta estratégia e a possibilidade de fragmentação externa limitam muito seu uso em sistemas operacionais de propósito geral, nos quais arquivos são constantemente criados, modificados e destruídos. Todavia, ela pode encontrar uso em situações específicas, nas quais os arquivos não sejam modificados constantemente e seja necessário rapidez nos acessos sequenciais e aleatórios aos dados.

Alocação encadeada simples

- cada bloco do arquivo no disco contém dados do arquivo e também um ponteiro para o próximo bloco, ou seja, um campo indicando a posição no disco do próximo bloco do arquivo. Desta forma é construída uma lista encadeada de blocos para cada arquivo, não sendo mais necessário manter os blocos do arquivo lado a lado no disco.

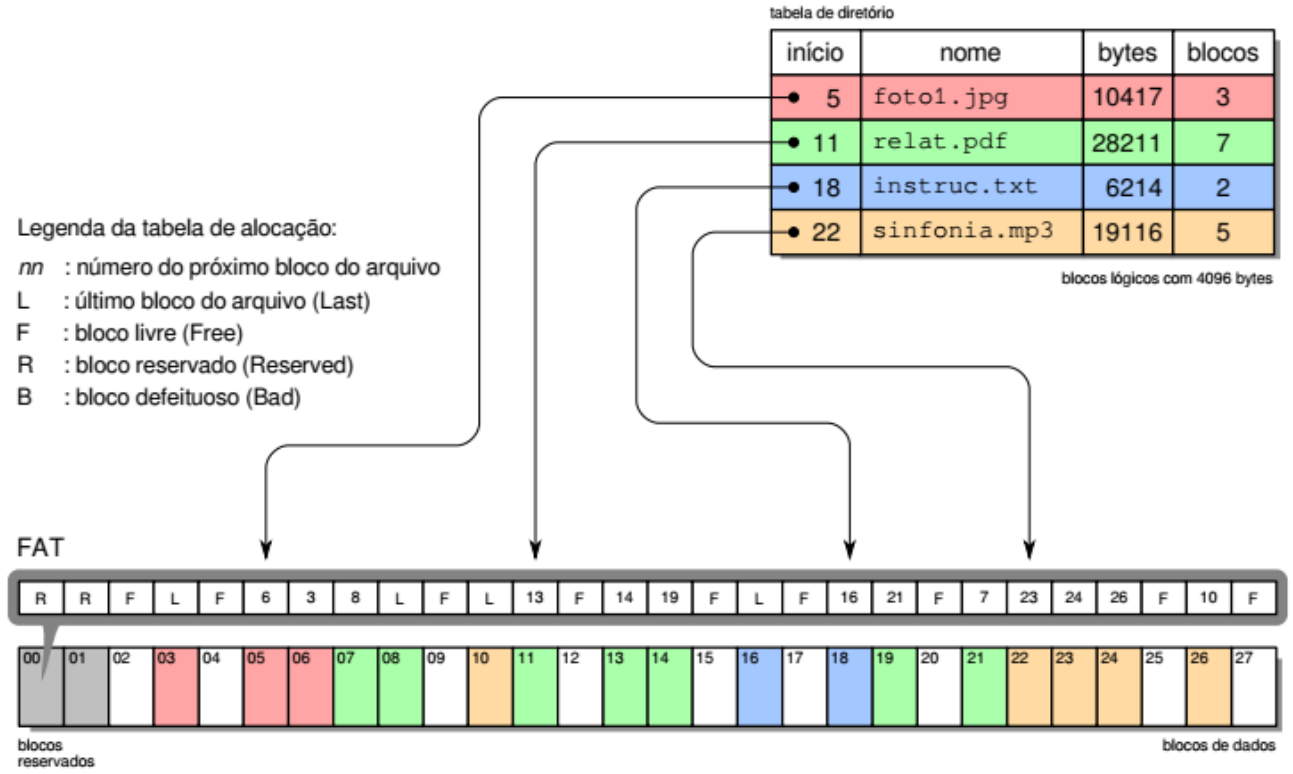


Alocação encadeada simples

- Esta estratégia elimina a fragmentação externa, pois todos os blocos livres do disco podem ser utilizados sem restrições, e permite que arquivos sejam criados sem a necessidade de definir seu tamanho final.
- o acesso sequencial aos dados do arquivo é simples e rápido, pois cada bloco do arquivo contém um “ponteiro” para o próximo bloco.
- o acesso aleatório ao arquivo fica muito prejudicado com esta abordagem: caso se deseje acessar o *n*-ésimo bloco do arquivo, os *n* - 1 blocos anteriores terão de ser lidos em sequência, para poder encontrar os ponteiros que levam ao bloco desejado.
- A dependência dos ponteiros de blocos também acarreta problemas de robustez: caso um bloco do arquivo seja corrompido ou se torne defeituoso, todos os blocos posteriores a este também ficarão inacessíveis.
- esta abordagem é muito flexível, pois não há necessidade de se definir o tamanho máximo do arquivo durante sua criação

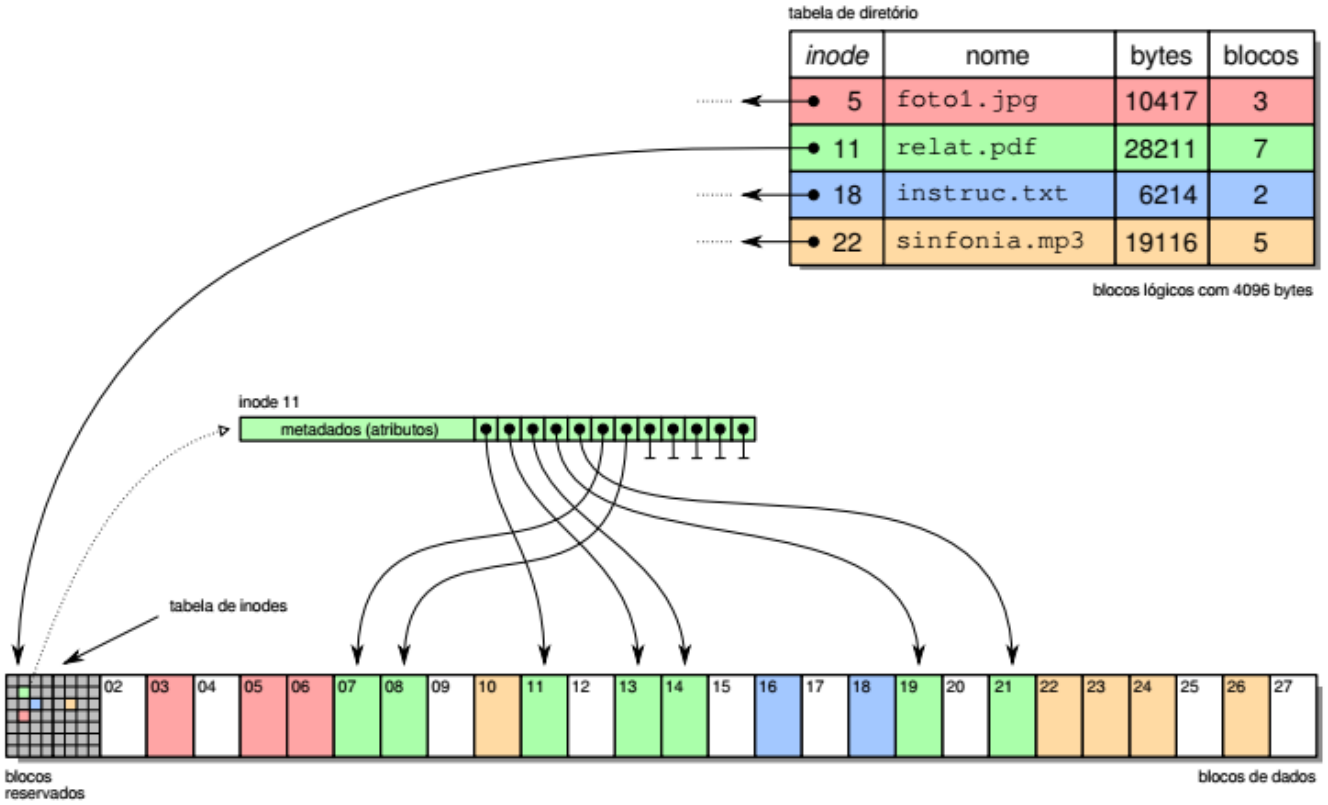
Alocação encadeada FAT

- Os principais problemas da alocação encadeada simples são o baixo desempenho nos acessos aleatórios e a relativa fragilidade em relação a erros nos blocos do disco. Ambos os problemas provêm do fato de que os ponteiros dos blocos são armazenados nos próprios blocos, junto dos dados do arquivo. Para resolver esse problema, os ponteiros podem ser retirados dos blocos de dados e armazenados em uma tabela separada.
- Essa tabela é denominada **Tabela de Alocação de Arquivos** (FAT - *File Allocation Table*), sendo a base dos sistemas de arquivos FAT12, FAT16 e FAT32 usados nos sistemas operacionais MS-DOS, Windows e em muitos dispositivos de armazenamento portáteis, como *pendrives*, reprodutores MP3 e câmeras fotográficas digitais.



Alocação indexada simples

- Nesta abordagem, a estrutura em lista encadeada da estratégia anterior é substituída por um vetor contendo um *índice de blocos* do arquivo. Cada entrada desse índice corresponde a um bloco do arquivo e aponta para a posição desse bloco no disco. O índice de blocos de cada arquivo é mantido no disco em uma estrutura denominada *nó de índice* (*index node*) ou simplesmente *nó-i* (*i-node*). O *i-node* de cada arquivo contém, além de seu índice de blocos, os principais atributos do mesmo, como tamanho, permissões, datas de acesso, etc. Os *i-nodes* de todos os arquivos são agrupados em uma tabela de *i-nodes*, mantida em uma área reservada do disco, separada dos blocos de dados dos arquivos.

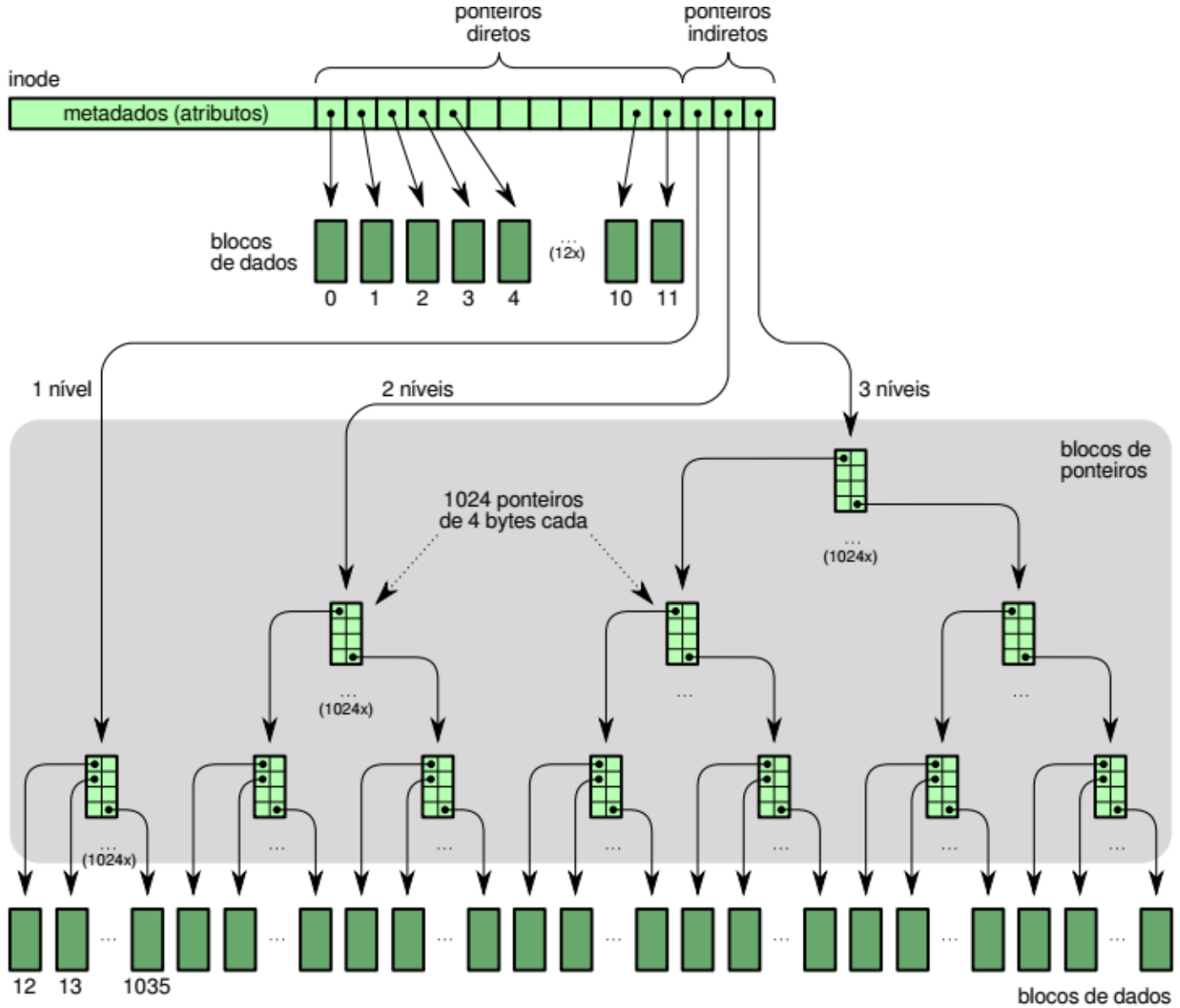


Alocação indexada simples

- Como os *i-nodes* também têm tamanho fixo, o número de entradas no índice de blocos de um arquivo é limitado. Por isso, esta estratégia de alocação impõe um tamanho máximo para os arquivos. Por exemplo, se o sistema usar blocos de 4 KBytes e o índice de blocos suportar 64 ponteiros, só poderão ser armazenados arquivos com até 256 KBytes (64 × 4).
- Além disso, a tabela de i-nodes também tem um tamanho fixo, determinado durante a formatação do sistema de arquivos, o que limita o número máximo de arquivos ou diretórios que podem ser criados na partição (pois cada arquivo ou diretório consome um i-node).

Alocação indexada multinível

- Para aumentar o tamanho máximo dos arquivos armazenados, algumas das entradas do índice de blocos podem ser transformadas em ponteiros indiretos. Essas entradas apontam para blocos do disco que contêm outros ponteiros, criando assim uma estrutura em árvore. Considerando um sistema com blocos lógicos de 4 Kbytes e ponteiros de 32 bits (4 bytes), cada bloco lógico pode conter 1.024 ponteiros para outros blocos, o que aumenta muito a capacidade do índice de blocos.
- Considerando um sistema com blocos lógicos de 4 Kbytes e ponteiros de 32 bits (4 bytes), cada bloco lógico pode conter 1.024 ponteiros para outros blocos, o que aumenta muito a capacidade do índice de blocos. Além de ponteiros indiretos, podem ser usados ponteiros dupla e triplamente indiretos.



Alocação indexada multinível

- Apesar dessa estrutura aparentemente complexa, a localização e acesso de um bloco do arquivo no disco permanece relativamente simples, pois a estrutura homogênea de ponteiros permite calcular rapidamente a localização dos blocos.
- Em relação ao desempenho, pode-se afirmar que esta estratégia é bastante rápida, tanto para acessos sequenciais quanto para acessos aleatórios a blocos, devido aos índices de ponteiros dos blocos presentes nos i-nodes. Contudo, no caso de blocos situados no final de arquivos muito grandes, podem ser necessários três ou quatro acessos a disco adicionais para localizar o bloco desejado, devido à necessidade de ler os blocos com ponteiros indiretos.
- Defeitos em blocos de dados não afetam os demais blocos de dados, o que torna esta estratégia robusta. Todavia, defeitos nos metadados (o *i-node* ou os blocos de ponteiros) podem danificar grandes extensões do arquivo; por isso, muitos sistemas *que usam esta estratégia implementam técnicas de redundância de i-nodes e metadados para melhorar a robustez.*
- Em relação à flexibilidade, pode-se afirmar que esta forma de alocação é tão flexível quanto a alocação encadeada, não apresentando fragmentação externa e permitindo o uso de todas as áreas livres do disco para armazenar dados.

FAT16 - Descrição

- Trata-se da versão de 16 bits do sistema de arquivos FAT.
- Utiliza um número de 16 bits para identificar cada unidade de alocação (cluster), o que dá um total de 65536 (2^{16}) unidades de alocação.
- O tamanho de cada cluster da partição é definido no setor de boot.
- O identificador do sistema de arquivos associados com a FAT16 geralmente são 04h e 06h (o primeiro usado para volumes com menos que 65536 setores, tipicamente em volumes menores que 32M).

Leiaute básico

Area description	Area size
Boot block	1 block
File Allocation Table (may be multiple copies)	Depends on file system size
Disk root directory	Variable (selected when disk is formatted)
File data area	The rest of the disk

Boot block

Offset from start	Length	Description
0x00	3 bytes	Part of the bootstrap program.
0x03	8 bytes	Optional manufacturer description.
0x0b	2 bytes	Number of bytes per block (almost always 512).
0x0d	1 byte	Number of blocks per allocation unit.
0x0e	2 bytes	Number of reserved blocks. This is the number of blocks on the disk that are not actually part of the file system; in most cases this is exactly 1, being the allowance for the boot block.
0x10	1 byte	Number of File Allocation Tables.
0x11	2 bytes	Number of root directory entries (including unused ones).
0x13	2 bytes	Total number of blocks in the entire disk. If the disk size is larger than 65535 blocks (and thus will not fit in these two bytes), this value is set to zero, and the true size is stored at offset 0x20.
0x15	1 byte	Media Descriptor. This is rarely used, but still exists. .
0x16	2 bytes	The number of blocks occupied by one copy of the File Allocation Table.
0x18	2 bytes	The number of blocks per track. This information is present primarily for the use of the bootstrap program, and need not concern us further here.
0x1a	2 bytes	The number of heads (disk surfaces). This information is present primarily for the use of the bootstrap program, and need not concern us further here.
0x1c	4 bytes	The number of *hidden blocks*. The use of this is largely historical, and it is nearly always set to 0; thus it can be ignored.
0x20	4 bytes	Total number of blocks in the entire disk (see also offset 0x13).
0x24	2 bytes	Physical drive number. This information is present primarily for the use of the bootstrap program, and need not concern us further here.
0x26	1 byte	Extended Boot Record Signature This information is present primarily for the use of the bootstrap program, and need not concern us further here.
0x27	4 bytes	Volume Serial Number. Unique number used for identification of a particular disk.
0x2b	11 bytes	Volume Label. This is a string of characters for human-readable identification of the disk (padded with spaces if shorter); it is selected when the disk is formatted.
0x36	8 bytes	File system identifier (padded at the end with spaces if shorter).
0x3e	0x1c0 bytes	The remainder of the bootstrap program.
0x1fe	2 bytes	Boot block 'signature' (0x55 followed by 0xaa).

Tabela de Alocação de Arquivos (FAT)

- O FAT ocupa um ou mais blocos imediatamente após os Blocos reservados iniciais. Comumente, parte de seu último bloco permanecerá sem uso, já que é improvável que o número necessário de entradas preencherá exatamente um número completo de blocos. Se houver um segundo FAT, vai estar na sequência do primeiro.
- No caso do sistema de arquivos FAT de 16 bits, cada entrada no FAT tem dois bytes de comprimento (i.e. 16 bits), uma entrada para cada cluster no disco.

Valid FAT16 values

Value	Description
0000h	Free cluster
0001h - 0002h	Not allowed
0003h - FFEFh	Number of the next cluster
FFF7h	One or more bad sectors in cluster
FFF8h - FFFFh	End-of-file

Disk Root Directory

Offset	Length	Description
0x00	8 bytes	Filename
0x08	3 bytes	Filename extension
0x0b	1 byte	File attributes
0x0c	10 bytes	Reserved
0x16	2 bytes	Time created or last updated
0x18	2 bytes	Date created or last updated
0x1a	2 bytes	Starting cluster number for file
0x1c	4 bytes	File size in bytes

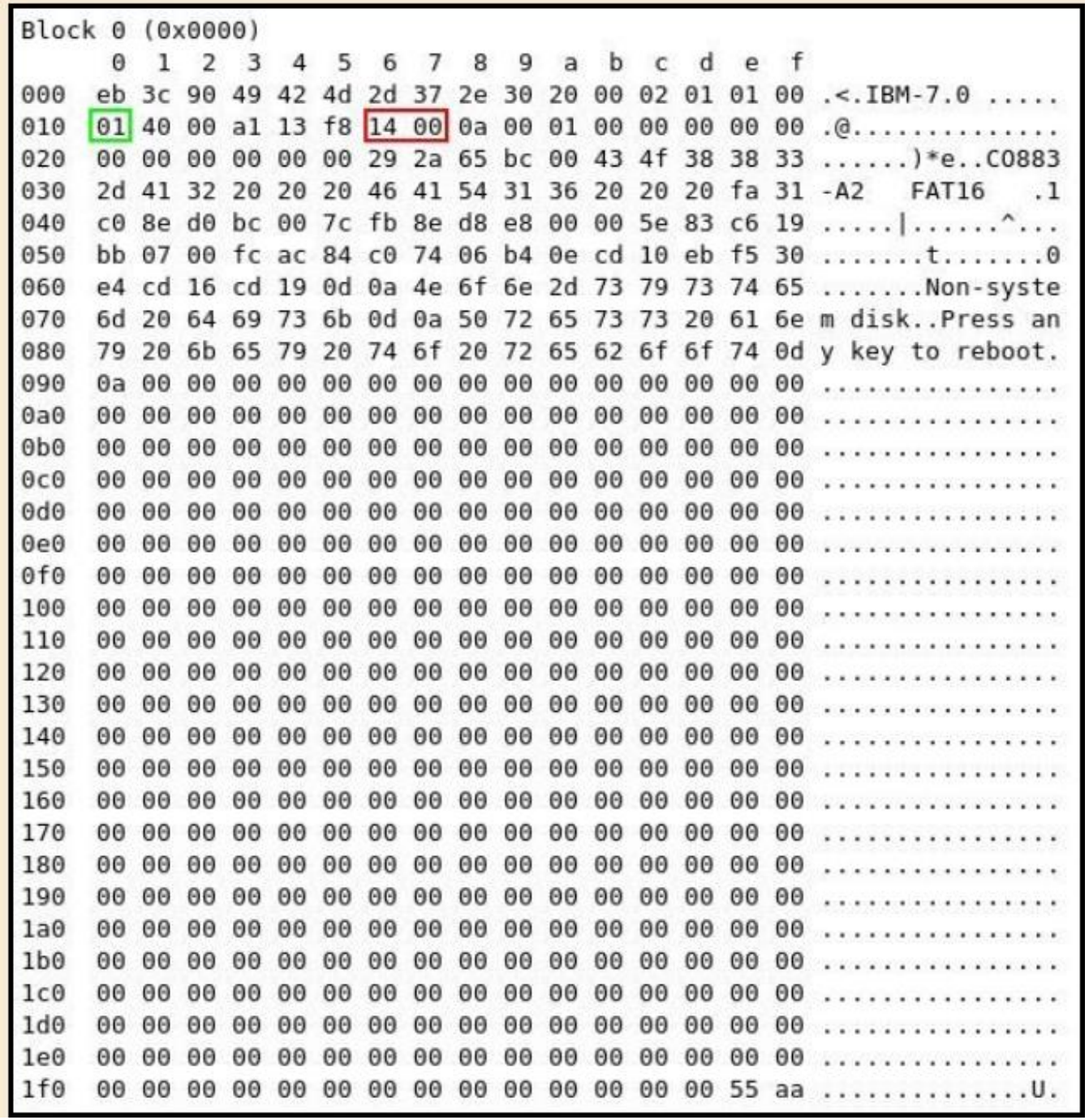
Explorando o conteúdo do Boot Block

```
Block 0 (0x0000)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  eb 3c 90 49 42 4d 2d 37 2e 30 20 00 02 01 01 00 .<.IBM-7.0 .....
010  01 40 00 a1 13 f8 14 00 0a 00 01 00 00 00 00 00 .@..............
020  00 00 00 00 00 00 29 2a 65 bc 00 43 4f 38 38 33 ......)*e..CO883
030  2d 41 32 20 20 20 46 41 54 31 36 20 20 20 fa 31 -A2   FAT16   .1
040  c0 8e d0 bc 00 7c fb 8e d8 e8 00 00 5e 83 c6 19 .....|......^...
050  bb 07 00 fc ac 84 c0 74 06 b4 0e cd 10 eb f5 30 .......t.......0
060  e4 cd 16 cd 19 0d 0a 4e 6f 6e 2d 73 79 73 74 65 .......Non-syste
070  6d 20 64 69 73 6b 0d 0a 50 72 65 73 73 20 61 6e m disk..Press an
080  79 20 6b 65 79 20 74 6f 20 72 65 62 6f 6f 74 0d y key to reboot.
090  0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa ..............U.
```

Localização do diretório raiz:

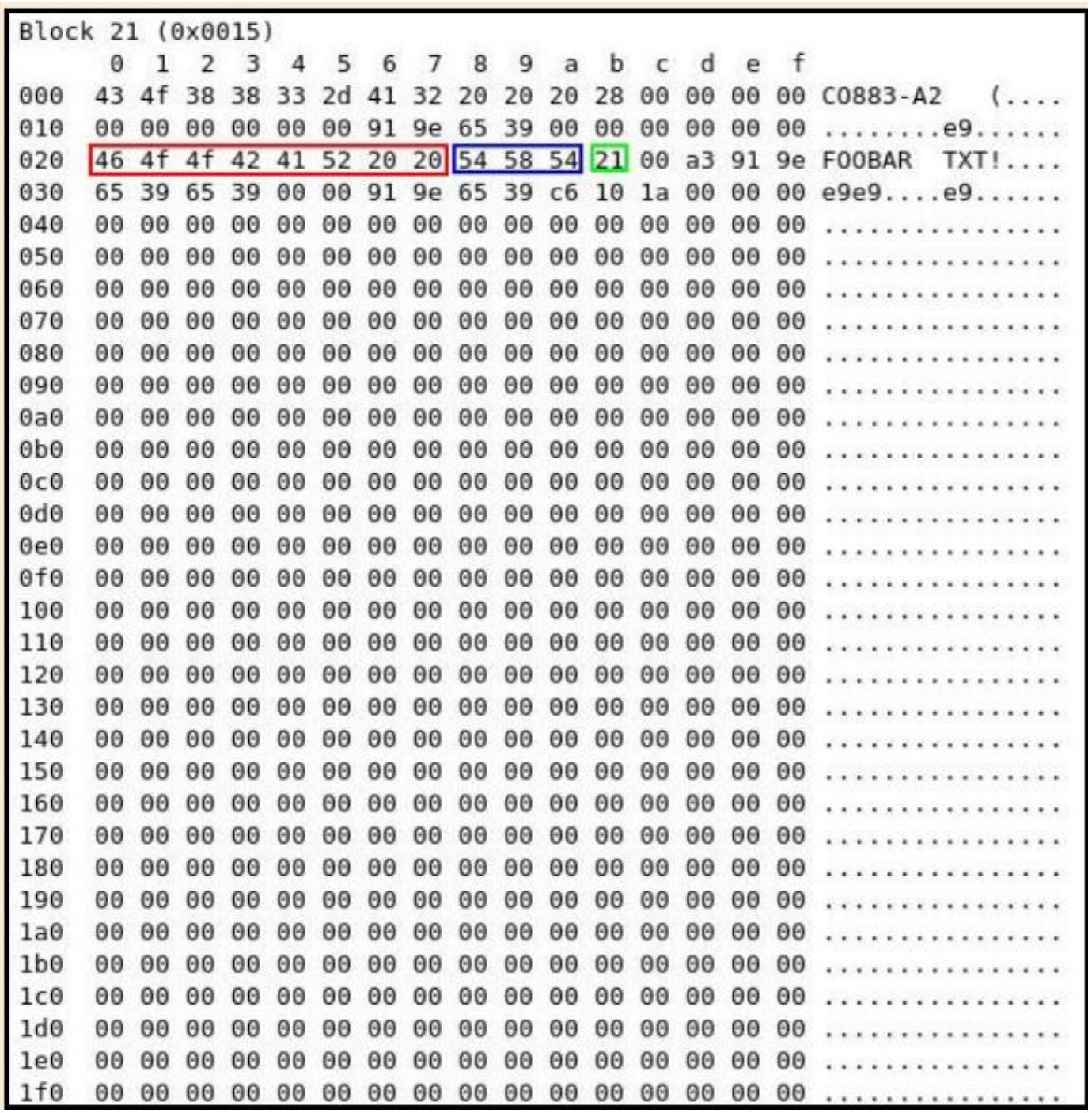
(size of FAT)*(nro of FATs)+1

no exemplo resulta no bloco 21 em decimal.
Se cada bloco tem 512 bytes, então é possível definir um deslocamento em bytes até o início do diretório raiz.



```
Block 0 (0x0000)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  eb 3c 90 49 42 4d 2d 37 2e 30 20 00 02 01 01 00 .<.IBM-7.0 .....
010  01 40 00 a1 13 f8 14 00 0a 00 01 00 00 00 00 00 .@..............
020  00 00 00 00 00 00 29 2a 65 bc 00 43 4f 38 38 33 ......)*e..CO883
030  2d 41 32 20 20 20 46 41 54 31 36 20 20 20 fa 31 -A2   FAT16   .1
040  c0 8e d0 bc 00 7c fb 8e d8 e8 00 00 5e 83 c6 19 .....|......^...
050  bb 07 00 fc ac 84 c0 74 06 b4 0e cd 10 eb f5 30 .......t.......0
060  e4 cd 16 cd 19 0d 0a 4e 6f 6e 2d 73 79 73 74 65 .......Non-syste
070  6d 20 64 69 73 6b 0d 0a 50 72 65 73 73 20 61 6e m disk..Press an
080  79 20 6b 65 79 20 74 6f 20 72 65 62 6f 6f 74 0d y key to reboot.
090  0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa ..............U.
```

- Neste exemplo é possível observar alguma coisa armazenada no diretório raiz



```
Block 21 (0x0015)
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
000  43 4f 38 38 33 2d 41 32 20 20 20 28 00 00 00 00 CO883-A2   (....
010  00 00 00 00 00 00 91 9e 65 39 00 00 00 00 00 00 ........e9......
020  46 4f 4f 42 41 52 20 20 54 58 54 21 00 a3 91 9e FOOBAR  TXT!....
030  65 39 65 39 00 00 91 9e 65 39 c6 10 1a 00 00 00 e9e9....e9......
040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Atividade válida para a M2

- Implementação de funções de acesso a um arquivo contendo uma imagem de um disco no formato F16. Estas funções devem permitir:
- Listar o arquivos armazenados (apenas diretório raiz, sem subdiretórios);
- Listar o conteúdo de um arquivo armazenado;
- Inserir um novo arquivo;
- Apagar um arquivo;
- Renomear um arquivo;
- Obter os metadados associados a um arquivo (tamanho, data, permissões)

- O arquivo com uma imagem de um disco está disponível no AVA Univali.
- Ferramenta útil para explorar a imagem do disco: https://hexed.it/

- Links para informações sobre a FAT16
- http://www.maverick-os.dk/FileSystemFormats/FAT16_FileSystem.html
- http://www.tavi.co.uk/phobos/fat.html