

Aula 11

Professores:

Felipe M. G. França
Valmir C. Barbosa

Conteúdo:

Sistema de arquivos

- Arquivos
- Diretórios
- Implementação (arquivos e diretórios)

Introdução

- Um processo precisa armazenar e recuperar informações:
 - Deve ser possível o armazenamento de uma grande quantidade de informações.
 - As informações não devem ser perdidas após o término do processo que as utilizou.
 - As informações devem poder ser acessadas simultaneamente por mais de um processo.
- A solução é a de armazenar estas informações em meios (*media*) externos, como o disco, usando unidades chamadas de **arquivos**:
 - As informações de um arquivo são **persistentes**, ou seja, não são perdidas após o término do processo que as usa.
- O **sistema de arquivos** define como os arquivos são nomeados, estruturados, acessados, protegidos e implementados:
 - O usuário está interessado na **interface** com o sistema.
 - O projetista está interessado na **implementação** do sistema.

Arquivos

- ➡ Um **arquivo** é uma abstração que oferece um modo de armazenar as informações em um dispositivo, como o disco.
- ➡ Os usuários devem ser capazes de usar os arquivos sem saber como o sistema os implementa.
- ➡ A parte mais importante da abstração é a **nomeação** dos arquivos:
 - Um arquivo é sempre acessado através do seu nome.
 - As regras para a nomeação do arquivo dependem do sistema, mas, em geral, o nome tem pelo menos 8 letras:
 - Em alguns sistemas, outros caracteres podem ser usados, como nos nomes **2, urgente!** ou **Figura 2.14.**
 - Em alguns sistemas, o nome é **sensível ao uso** de letras minúsculas ou maiúsculas:
 - No UNIX, **SO.txt** e **so.txt** são dois nomes diferentes de um arquivo, e no MS-DOS são o mesmo nome.

Arquivos

- Em alguns sistemas, o nome do arquivo pode ser dividido em duas partes, separadas por um '':
 - Uma cadeia é usada para diferenciar o arquivo dos outros.
 - Uma outra cadeia, chamada de **extensão**, usada para indicar alguma informação sobre o conteúdo do arquivo.
 - Em alguns sistemas, uma extensão pode possuir extensões:
 - O nome **prog.c** refere-se a um programa em C, e o nome **prog.c.Z** a uma versão compactada deste programa.
- As extensões de um arquivo são importantes, pois são usadas pelos programas que podem manipular vários tipos de arquivo:
 - Um compilador C pode usá-las para distinguir entre um arquivo fonte em C, e um código objeto já compilado.
- As extensões também podem ser usadas pelos usuários para estes terem uma idéia do conteúdo de cada um de seus arquivos.

Arquivos

- Um arquivo pode ser estruturado por um dos três modos:
- **Seqüência de bytes não estruturada:** maior flexibilidade, pois o significado do arquivo é definido pelo programa.
 - **Estruturação baseada em registros:** o arquivo é composto por um conjunto de registros iguais com comprimento fixo.
 - **Estruturação baseada em árvore:** o arquivo é uma árvore cujos registros são diferentes, e possuem comprimento variável.

Seqüência de bytes não estruturada

Arquivo →

A/C



Byte

Nesta organização, o arquivo nada mais é, para o sistema operacional, do que uma seqüência de bytes não estruturada. O sistema somente gerencia o armazenamento dos arquivos no disco, e não se importa com o conteúdo dos arquivos, apesar de que, como veremos, o sistema poder reconhecer alguns arquivos, como os executáveis com os programas. O UNIX e o MS-DOS usam esta organização.

Arquivos

- Um arquivo pode ser estruturado por um dos três modos:
- **Seqüência de bytes não estruturada:** maior flexibilidade, pois o significado do arquivo é definido pelo programa.
 - **Estruturação baseada em registros:** o arquivo é composto por um conjunto de registros iguais com comprimento fixo.
 - **Estruturação baseada em árvore:** o arquivo é uma árvore cujos registros são diferentes, e possuem comprimento variável.

Estruturação baseada em registros

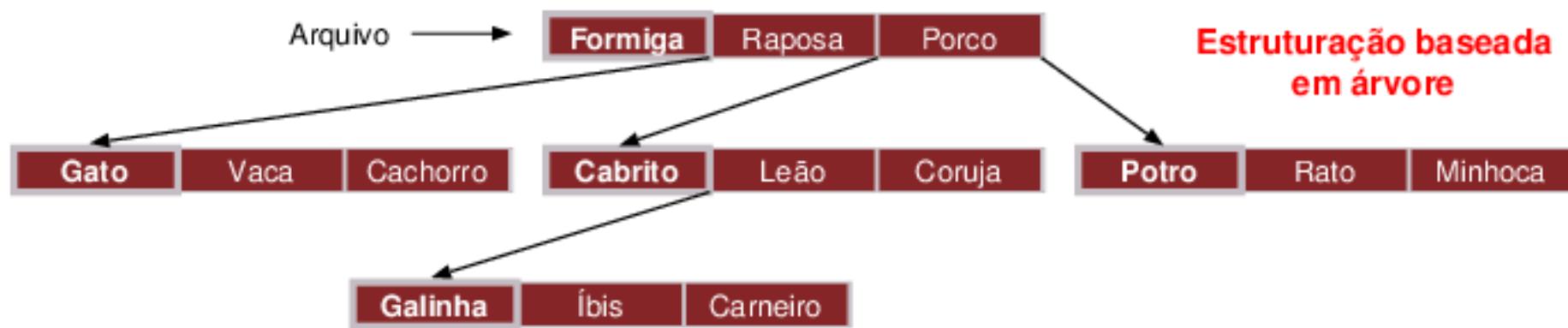


→ Registro

Nesta estruturação, cada arquivo é organizado, pelo sistema operacional, como um conjunto de registros com o mesmo comprimento e estrutura interna. Como o sistema impõe a estruturação, as operações sobre arquivos operam sobre registros, ou seja, por exemplo, uma operação de leitura retorna um registro, e uma operação de gravação grava ou sobrescreve um registro. Um exemplo de um sistema com esta estruturação era o CP/M, cujos registros possuíam 128 caracteres.

Arquivos

- Um arquivo pode ser estruturado por um dos três modos:
- **Seqüência de bytes não estruturada:** maior flexibilidade, pois o significado do arquivo é definido pelo programa.
 - **Estruturação baseada em registros:** o arquivo é composto por um conjunto de registros iguais com comprimento fixo.
 - **Estruturação baseada em árvore:** o arquivo é uma árvore cujos registros são diferentes, e possuem comprimento variável.



O sistema estrutura os arquivos como uma árvore cujos registros podem possuir tamanho e estrutura interna diferentes. Em cada registro existe um campo chave em uma posição fixa do registro, que é usado pelo sistema nas operações sobre o arquivo, e para classificar a árvore. Na figura, as chaves estão destacadas nos registros. Os micros de grande porte usam esta estruturação.



C/C

Arquivos

- ➡ Existem vários tipos de arquivos em um sistema de arquivos:
 - Os arquivos **comuns** com as informações dos programas:
 - Os arquivos **ASCII** são compostos por linhas de texto.
 - Os arquivos **binários** são compostos por seqüências arbitrárias de bytes.
 - Os **diretórios** são usados para manter a estrutura hierárquica do sistema de arquivos.
 - Os arquivos especiais de **bloco** e de **caractere** são usados para fornecer um acesso abstrato aos dispositivos físicos.
- ➡ Os arquivos ASCII possuem as seguintes propriedades:
 - Cada linha é finalizada com um **terminador**, que pode ser um retorno de carro, uma quebra de linha, ou ambos.
 - O arquivo pode ser editado, e o seu conteúdo pode ser exibido e impresso.
 - Como muitos programas usam arquivos ASCII, a comunicação entre processos via pipe é facilitada.

Arquivos

Os arquivos binários não podem ser exibidos e impressos, e, em geral, possuem alguma estrutura interna:

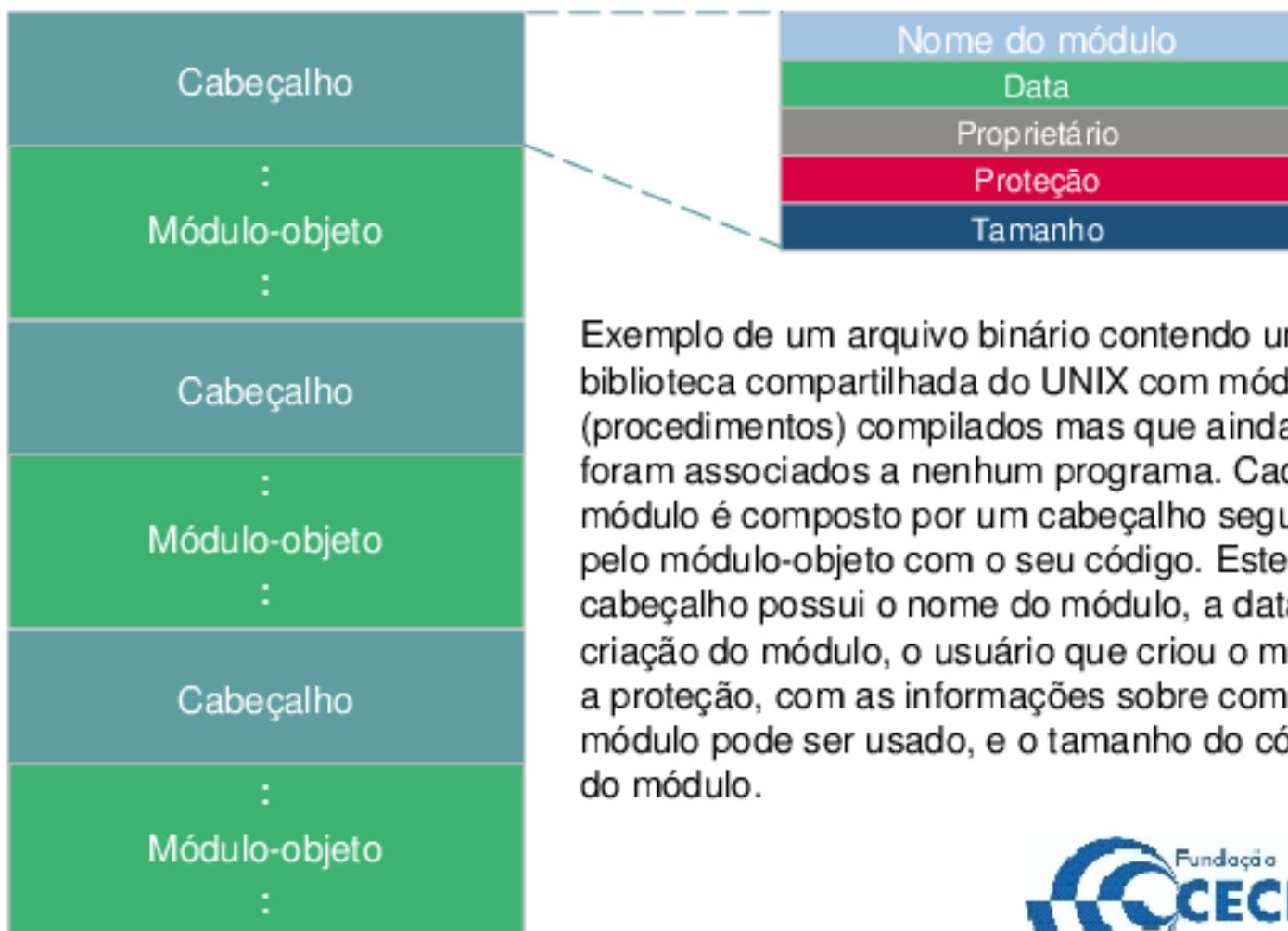
Cabeçalho →



Exemplo de um arquivo binário contendo um programa executável de uma versão preliminar do UNIX. Os primeiros campos destacados formam o cabeçalho do arquivo, sendo que os campos principais são o **número mágico**, que identifica o arquivo binário como um programa executável, e o **ponto de entrada**, que define o endereço em que deveremos começar a executar o programa. Depois do cabeçalho, vem as áreas com o texto do programa, os dados do programa, os bits usados para fazer a relocação do programa na memória, e a tabela de símbolos usada pela depuração.

Arquivos

Os arquivos binários não podem ser exibidos e impressos, e, em geral, possuem alguma estrutura interna:



B/B



Arquivos

- ➡ O sistema operacional deve sempre reconhecer pelo menos um tipo de arquivo, o que possui o seu código executável.
- ➡ O sistema também pode reconhecer outros tipos de arquivos:
 - O Windows, por exemplo, reconhece o tipo do arquivo pela sua extensão, e chama o programa correto para visualizá-lo.
 - Este reconhecimento automático nem sempre é desejável, pois a extensão de um arquivo pode não refletir o seu conteúdo:

```
program HelloWorld;
var
  i: integer;
begin
  for i := 1 to 4 do
    writeln("Hello World!");
end;
end.
```

Arquivo helloworld.pas
Será compilado

Programa de
formatação

```
program HelloWorld;
var
  i: integer;
begin
  for i := 1 to 4 do
    writeln("Hello World!");
end;
end.
```

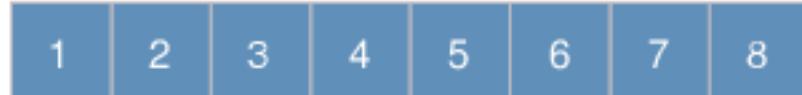
Arquivo helloworld.dat
Não será compilado

Se o sistema operacional reconhecer os tipos dos arquivos, e impor o seu uso com os programas corretos, e se helloworld.dat for a saída de um programa formatador para o código pascal helloworld.pas, então o compilador pascal não compilará este arquivo.

Arquivos

- O acesso aos bytes (ou registros) do arquivo pode ser feito por um dos dois modos:
- **Acesso seqüencial:** os bytes (ou registros) são acessados em ordem, começando pelo primeiro byte (ou registro).
 - **Acesso aleatório:** os bytes (ou registros) podem ser acessados em qualquer ordem (por posição, ou usando uma chave).
 - Em alguns sistemas, como os computadores de grande porte, o tipo de acesso deve ser fornecido ao criarmos o arquivo.

Arquivo com 8 bytes (ou registros)



No exemplo temos um arquivo com 8 bytes (ou registros). Vamos ver agora qual é a diferença entre acessar este arquivo de modo seqüencial, e acessar este arquivo de modo aleatório.



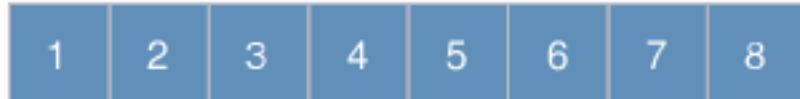
A/C

Arquivos

→ O acesso aos bytes (ou registros) do arquivo pode ser feito por um dos dois modos:

- **Acesso seqüencial:** os bytes (ou registros) são acessados em ordem, começando pelo primeiro byte (ou registro).
- **Acesso aleatório:** os bytes (ou registros) podem ser acessados em qualquer ordem (por posição, ou usando uma chave).
- Em alguns sistemas, como os computadores de grande porte, o tipo de acesso deve ser fornecido ao criarmos o arquivo.

Arquivo com 8 bytes (ou registros)



No acesso seqüencial, ao abrirmos o arquivo, a posição atual a ser lida é a 1. Depois desta posição, leremos a 2, a 3, e assim por diante. Ou seja, a próxima posição a ser lida é sempre adjacente a que foi lida anteriormente, e com isso, os bytes (ou registros) do arquivo devem ser lidos na ordem em que foram armazenados. Se necessário, podemos retroceder no arquivo, e começar a ler novamente a partir da primeira posição.



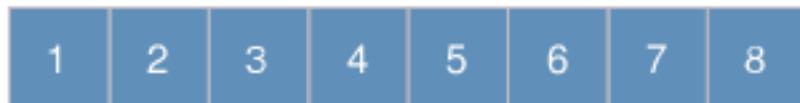
B/C

Arquivos

→ O acesso aos bytes (ou registros) do arquivo pode ser feito por um dos dois modos:

- **Acesso seqüencial:** os bytes (ou registros) são acessados em ordem, começando pelo primeiro byte (ou registro).
- **Acesso aleatório:** os bytes (ou registros) podem ser acessados em qualquer ordem (por posição, ou usando uma chave).
- Em alguns sistemas, como os computadores de grande porte, o tipo de acesso deve ser fornecido ao criarmos o arquivo.

Arquivo com 8 bytes (ou registros)



No acesso aleatório, existe uma operação, a **seek**, que permite saltarmos para qualquer posição do arquivo. Ao abrir o arquivo, a primeira posição a ser lida será a 1, assim como no caso do acesso seqüencial, e se leremos um conjunto de bytes (ou registros) em seqüência, estes bytes (ou registros) serão lidos de posições consecutivas. Para alterar a posição atual a ser lida, deveremos usar a operação **seek** para saltar para esta posição.



C/C

Arquivos

- Os arquivos, além de possuirem nome e dados, possuem **atributos**, que dão algumas informações adicionais sobre o arquivo.
- A lista dos possíveis atributos depende do sistema operacional. A seguir, vamos ver alguns dos atributos de um arquivo:

Atributo	Significado
Proteção	Controle de acesso (usuário e o tipo de acesso)
Criador	UID do usuário que criou o arquivo
Proprietário	UID do usuário que atualmente possui o arquivo
Sinalizador de somente-leitura	0 - leitura/gravação; 1 - somente leitura
Sinalizador de oculto	0 - visível ; 1 - invisível (nas listagens de diretório)
Sinalizador de sistema	0 - normal ; 1 - sistema (arquivos do sistema)
Sinalizador de arquivo	0 - salvo; 1 - não salvo (backup)
Sinalizador de temporário	0 - normal; 1 - temporário (remover ao terminar o processo)
Sinalizador de bloqueio	0 - desbloqueado; q - bloqueado (acesso exclusivo)
Tempo de criação	Data e hora da criação do arquivo
Tempo do último acesso	Date e hora do último acesso ao arquivo
Tempo da última alteração	Data e hora da última alteração do arquivo
Tamanho	Tamanho do arquivo em bytes

Arquivos

→ O sistema define várias chamadas para gerenciar os arquivos (abaixo é dada uma descrição das chamadas mais comuns):

Operação	Descrição
CREATE	Cria um novo arquivo vazio (sem dados)
DELETE	Deleta um arquivo
OPEN	Abre um arquivo (leitura, gravação, e anexação)
CLOSE	Fecha um arquivo aberto
READ	Lê um conjunto de dados do arquivo (em geral, na posição atual)
WRITE	Grava um conjunto de dados no arquivo (em geral, na posição atual)
APPEND	Grava um conjunto de dados ao final do arquivo
SEEK	Define a posição da próxima leitura ou gravação no arquivo
GET ATTRIBUTES	Lê os atributos de um arquivo
SET ATTRIBUTES	Altera os atributos de um arquivo
RENAME	Altera o nome de um arquivo

Principais chamadas ao sistema, sendo que em geral usamos as funções da biblioteca que implementam estas chamadas. É interessante sabermos porque as chamadas CREATE, OPEN e CLOSE existem: a chamada CREATE objetiva inicializar alguns atributos do arquivo que foi criado; a chamada OPEN objetiva copiar os atributos do arquivo para memória em uma tabela interna, para um acesso mais rápido (no UNIX, o nó-l é copiado para a memória); e a chamada CLOSE objetiva liberar a memória ocupada pelos atributos (o espaço na tabela interna é limitado).

Diretórios

- Um **diretório** é um arquivo especial composto por um certo número de entradas, com os nomes dos arquivos e seus atributos:
- Cada uma das entradas possui o nome do arquivo.
 - Os atributos do arquivo podem estar na entrada do diretório, ou em uma estrutura separada.

A/B



jogos	atributos
correo	atributos
notícias	atributos
trabalho	atributos

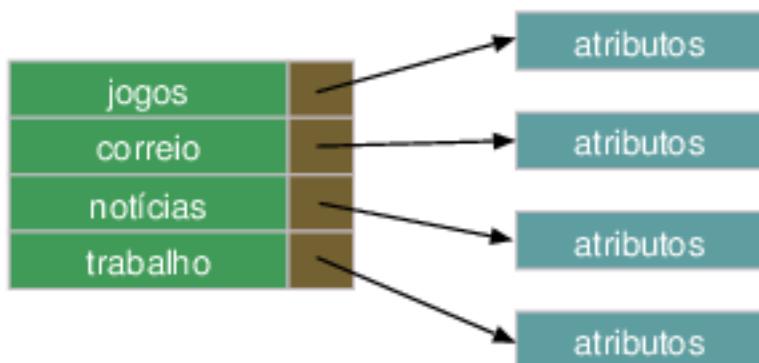
Nesta implementação, os atributos do arquivo estão na entrada do diretório. Como veremos, a entrada também possui, em geral, as informações necessárias para localizar o arquivo no disco.

- Quando usamos a função para abrir um arquivo, o sistema:
- Procura pela entrada do arquivo no diretório com este arquivo.
 - Lê as informações dadas nesta entrada, como os atributos e os endereços do disco, em uma estrutura interna do sistema.

Diretórios

Um **diretório** é um arquivo especial composto por um certo número de entradas, com os nomes dos arquivos e seus atributos:

- Cada uma das entradas possui o nome do arquivo.
- Os atributos do arquivo podem estar na entrada do diretório, ou em uma estrutura separada.



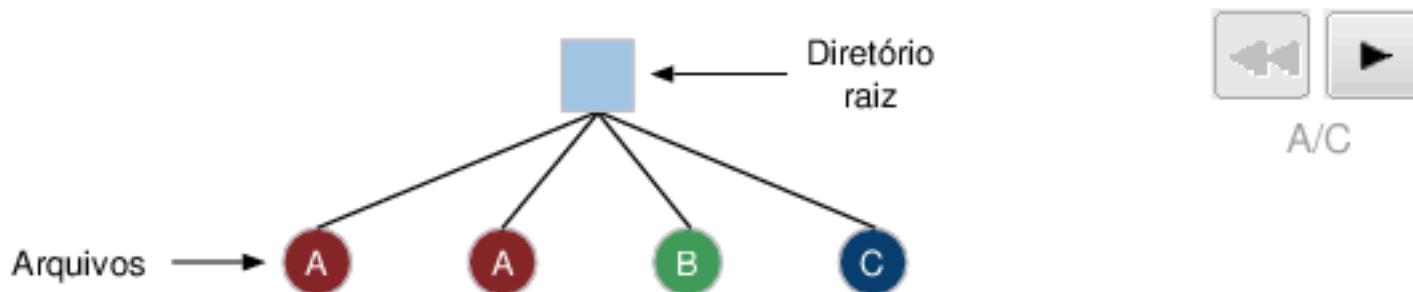
Neste caso, os atributos estão em uma estrutura separada, e o diretório somente possui um ponteiro para a estrutura. A estrutura também possuirá, em geral, as informações necessárias para localizar os arquivos no disco (exemplo: os nós-i usados pelo UNIX).

Quando usamos a função para abrir um arquivo, o sistema:

- Procura pela entrada do arquivo no diretório com este arquivo.
- Lê as informações dadas nesta entrada, como os atributos e os endereços do disco, em uma estrutura interna do sistema.

Diretórios

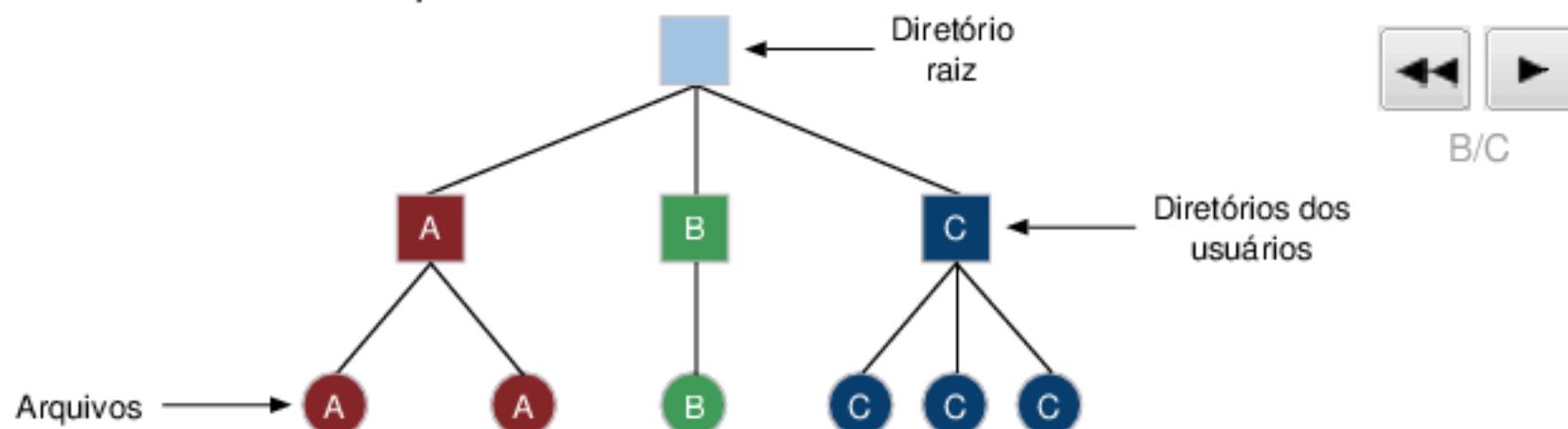
- O número total de diretórios disponíveis depende do sistema, mas as seguintes implementações são comuns:
- Somente um diretório com todos os arquivos dos usuários.
 - Um diretório para cada um dos usuários do sistema.
 - Uma hierarquia de diretórios, onde cada usuário possui um diretório e pode criar um número arbitário de diretórios.



A abordagem mais simples, usada pelo antigo sistema CP/M, é de o sistema possuir um único diretório, com todos os arquivos dos usuários. O problema neste caso é que se, por exemplo, um usuário criar um arquivo com o nome *teste.txt*, nenhum outro usuário poderá criar um arquivo com este nome.

Diretórios

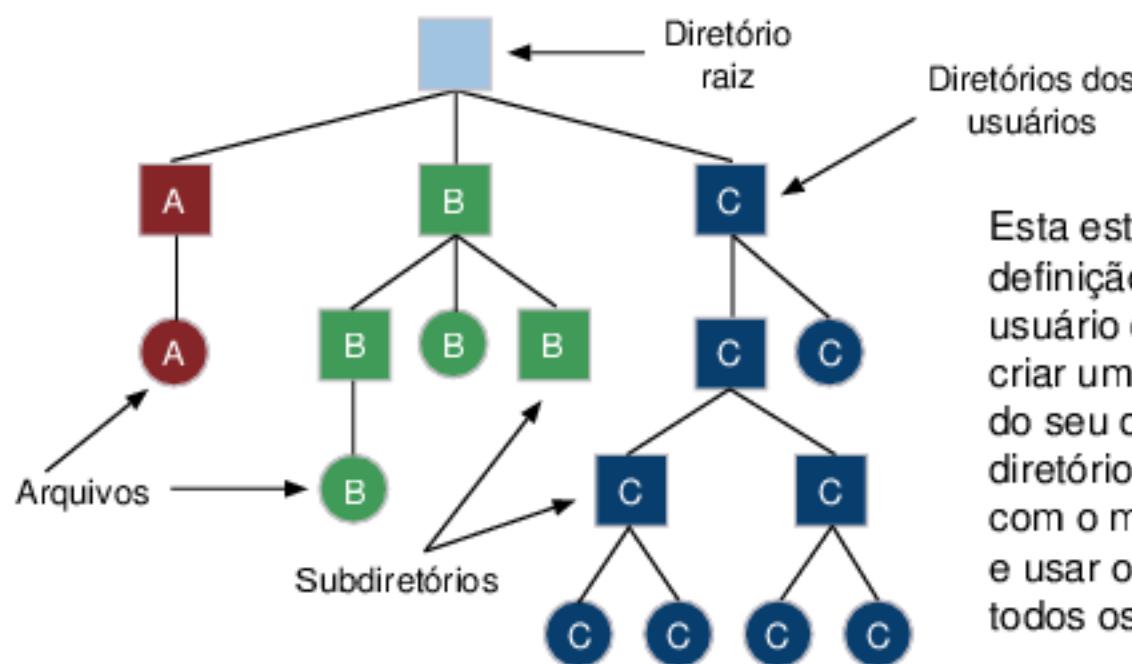
- O número total de diretórios disponíveis depende do sistema, mas as seguintes implementações são comuns:
- Somente um diretório com todos os arquivos dos usuários.
 - Um diretório para cada um dos usuários do sistema.
 - Uma hierarquia de diretórios, onde cada usuário possui um diretório e pode criar um número arbitário de diretórios.



Neste caso, existe um diretório para cada um dos usuários do sistema. Agora, dois ou mais usuários poderão criar um arquivo com o mesmo nome. Porém, a estruturação ainda não é ideal, pois os usuários não poderão organizar logicamente os seus arquivos (de acordo com o conteúdo e a relação com os outros arquivos), e nem poderão criar vários arquivos com o mesmo nome.

Diretórios

- O número total de diretórios disponíveis depende do sistema, mas as seguintes implementações são comuns:
- Somente um diretório com todos os arquivos dos usuários.
 - Um diretório para cada um dos usuários do sistema.
 - Uma hierarquia de diretórios, onde cada usuário possui um diretório e pode criar um número arbitrário de diretórios.

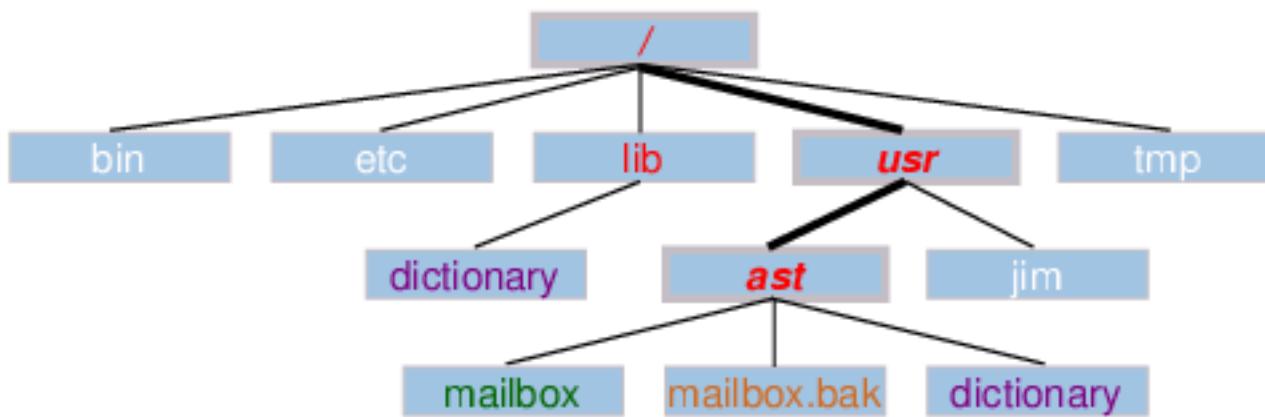


C/C

Esta estruturação é a mais geral, e permite a definição de uma hierarquia de diretórios. Cada usuário do sistema possui um diretório, e pode criar um número ilimitado de diretórios dentro do seu diretório, e dentro dos diretórios do seu diretório. Agora, o usuário pode criar arquivos com o mesmo nome (em diretórios diferentes), e usar os diretórios para organizar logicamente todos os seus arquivos.

Diretórios

- Na organização baseada em uma hierarquia de diretórios, os arquivos são acessados usando o seu **nome de caminho**:
- O nome de caminho **absoluto** dá o caminho do diretório raiz até o diretório que contém o arquivo.
 - O nome de caminho **relativo** dá o caminho do diretório de trabalho do processo até o diretório com o arquivo:
 - Um '.' é usado para indicar o diretório atual, e um '..' é usado para indicar o diretório pai do atual.



Cópia de `mailbox` para `mailbox.bak`:

```
cp /usr/ast/mailbox /usr/ast/mailbox.bak
cp mailbox mailbox.bak
```

Cópia do arquivo `dictionary`:

```
cp /lib/dictionary dictionary
cp /lib/dictionary .
cp ../../lib/dictionary .
```

Diretório de trabalho: `/usr/ast`.

Acesso ao arquivo `mailbox`:

absoluto: `/usr/ast/mailbox`
relativo: `mailbox`

Acesso ao arquivo `dictionary`:

absoluto: `/lib/dictionary`
relativo: `../../lib/dictionary`

Diretórios

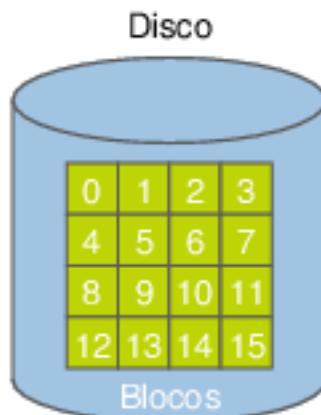
→ São definidas várias chamadas ao sistema para gerenciar os diretórios, sendo que as mais comuns são dadas na tabela a seguir:

Operação	Descrição
CREATE	Cria um diretório vazio (com um . e um ..)
DELETE	Deleta um diretório vazio (somente com os arquivos . e ..)
OPENDIR	Abre um diretório (para a busca de arquivos)
CLOSEDIR	Fechá um diretório aberto
REaddir	Lê a próxima entrada de um diretório aberto
RENAME	Alterar o nome de um diretório
LINK	Cria um novo vínculo para um arquivo no diretório
UNLINK	Remove um vínculo de um arquivo do diretório

Principais chamadas ao sistema, sendo que em geral usamos as funções da biblioteca que implementam estas chamadas. Das chamadas da tabela, a chamada REaddir lê as entradas do diretório, e é usada, por exemplo, pelos programas que listam os arquivos de um diretório. Já a chamada LINK permite a criação de um nome de caminho alternativo para um arquivo. Ao executá-la, damos o nome de caminho do arquivo e um diretório. A chamada UNLINK remove um vínculo criado para um arquivo. Se existir somente um vínculo para o arquivo, ele será deletado.

Implementação

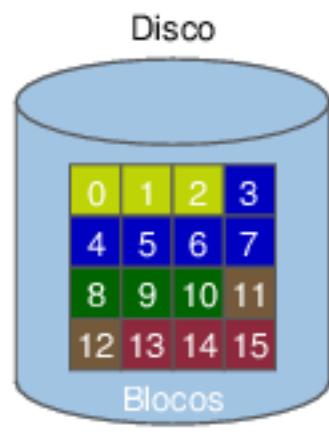
- ➡ Vamos começar o nosso estudo de implementação do sistema de arquivos vendo as seguintes técnicas para implementar arquivos:
 - Alocação contígua.
 - Alocação por lista encadeada.
 - Alocação por lista encadeada utilizando um índice.
 - Nós-i.
- ➡ Na implementação, vamos usar a versão abstrata do disco, em que o disco é composto por um conjunto de blocos.



Exemplo de uma versão abstrata de um disco composto por 16 blocos, numerados de 0 até 15. Como vimos, os blocos deste disco podem ser acessados através de um arquivo especial de bloco, de modo aleatório, ou seja, os blocos podem ser lidos e escritos em qualquer ordem.

Implementação

- Na alocação contígua, o arquivo é armazenado no disco em uma seqüência consecutiva de blocos do disco:
- O esquema é simples de implementar, pois precisamos somente saber em que bloco o arquivo inicia.
 - A leitura do arquivo será eficiente, pois somente uma operação sobre o disco é necessária para ler todo o arquivo.
 - Uma desvantagem é a de que devemos necessariamente saber o tamanho máximo do arquivo ao criá-lo.
 - Outra desvantagem é que este modo de alocação pode gerar uma fragmentação no disco, similar a que vimos na memória.

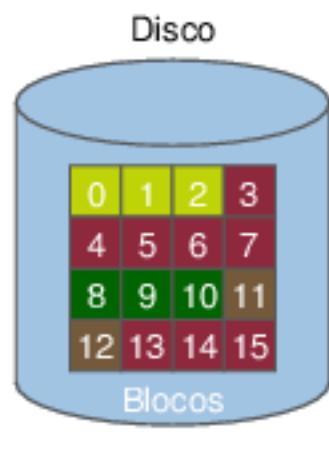


Exemplo de um disco com 16 blocos de 4K, em que usamos a alocação contígua para armazenar os arquivos. Temos atualmente 4 arquivos, A, B, C e D no disco, que foram alocados em blocos consecutivos do disco, e 12K bytes disponíveis no disco (3 blocos):

- | | | | |
|--|---------------------------|--|--------------------------|
| | Arquivo A: 12K (3 blocos) | | Arquivo D: 8K (2 blocos) |
| | Arquivo B: 20K (5 blocos) | | Livre: 12K (3 blocos) |
| | Arquivo C: 12K (3 blocos) | | |

Implementação

- Na alocação contígua, o arquivo é armazenado no disco em uma seqüência consecutiva de blocos do disco:
- O esquema é simples de implementar, pois precisamos somente saber em que bloco o arquivo inicia.
 - A leitura do arquivo será eficiente, pois somente uma operação sobre o disco é necessária para ler todo o arquivo.
 - Uma desvantagem é a de que devemos necessariamente saber o tamanho máximo do arquivo ao criá-lo.
 - Outra desvantagem é que este modo de alocação pode gerar uma fragmentação no disco, similar a que vimos na memória.

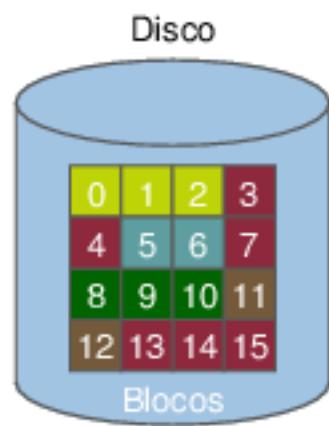


Alocação do disco depois de deletarmos o arquivo B com 20K. Agora, temos 3 arquivos no disco A, C, e D, e temos 32K livres no disco, isto é, 8 blocos no disco. Note que estes blocos não são consecutivos. Se desejarmos alojar um arquivo com 24K, não poderemos alojá-lo no disco, pois não temos 6 blocos consecutivos, apesar de termos 8 blocos (32K) disponíveis no disco

- | | | | |
|--|---------------------------|--|--------------------------|
| | Arquivo A: 12K (3 blocos) | | Arquivo D: 8K (2 blocos) |
| | Arquivo C: 12K (3 blocos) | | Livre: 32K (8 blocos) |

Implementação

- Na alocação contígua, o arquivo é armazenado no disco em uma seqüência consecutiva de blocos do disco:
- O esquema é simples de implementar, pois precisamos somente saber em que bloco o arquivo inicia.
 - A leitura do arquivo será eficiente, pois somente uma operação sobre o disco é necessária para ler todo o arquivo.
 - Uma desvantagem é a de que devemos necessariamente saber o tamanho máximo do arquivo ao criá-lo.
 - Outra desvantagem é que este modo de alocação pode gerar uma fragmentação no disco, similar a que vimos na memória.

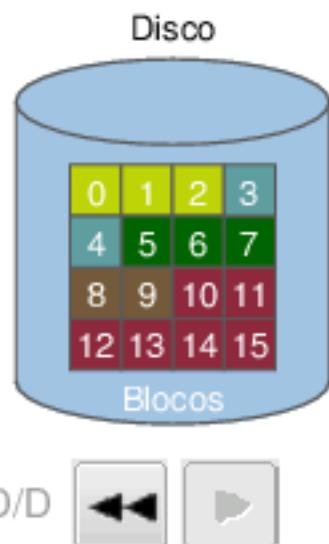


Exemplo depois da criação de um novo arquivo E, com 8K (2 blocos), que foi alocado nos blocos 5 e 6 do disco. Como podemos ver pela figura, agora temos 6 blocos livres (24K), mas que estão dispersos por todo o disco, o que limita o tamanho do arquivo a no máximo 3 blocos (12K).

- | | | | |
|--|---------------------------|--|--------------------------|
| | Arquivo A: 12K (3 blocos) | | Arquivo E: 8K (2 blocos) |
| | Arquivo C: 12K (3 blocos) | | Livre: 24K (6 blocos) |
| | Arquivo D: 8K (2 blocos) | | |

Implementação

- Na alocação contígua, o arquivo é armazenado no disco em uma seqüência consecutiva de blocos do disco:
- O esquema é simples de implementar, pois precisamos somente saber em que bloco o arquivo inicia.
 - A leitura do arquivo será eficiente, pois somente uma operação sobre o disco é necessária para ler todo o arquivo.
 - Uma desvantagem é a de que devemos necessariamente saber o tamanho máximo do arquivo ao criá-lo.
 - Outra desvantagem é que este modo de alocação pode gerar uma fragmentação no disco, similar a que vimos na memória.

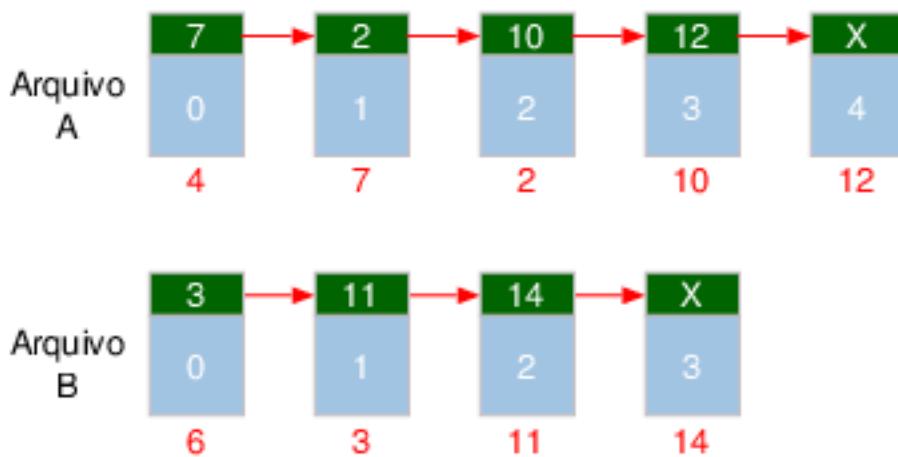


Para resolver o problema de blocos dispersos pelo disco, que na verdade é uma fragmentação externa dentro do disco, podemos, assim como no gerenciamento de memória, usar a técnica de compactação, para colocar todos os blocos livres ao final do disco. Agora, temos 6 blocos livres consecutivos no final do disco, e poderemos criar um arquivo com até 24K.

- | | | | |
|--|---------------------------|--|--------------------------|
| | Arquivo A: 12K (3 blocos) | | Arquivo E: 8K (2 blocos) |
| | Arquivo C: 12K (3 blocos) | | Livre: 24K (6 blocos) |
| | Arquivo D: 8K (2 blocos) | | |

Implementação

- Na alocação por lista encadeada, o arquivo é armazenado em blocos não contíguos do disco:
- Devemos armazenar somente o ponteiro para o primeiro bloco.
 - A primeira palavra de cada bloco armazena um ponteiro para o próximo bloco do arquivo.
 - A vantagem é que agora todos os blocos do disco podem ser usados, e temos somente desperdício dentro de um bloco.
 - Existem duas desvantagens nesta técnica:
 - O acesso aos arquivos agora será lento.
 - O espaço para os dados não possui o tamanho do bloco.

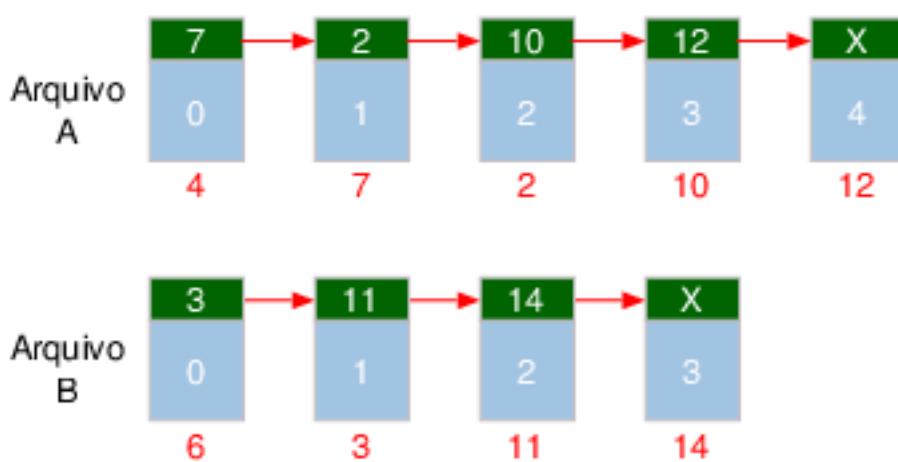


No exemplo dado ao lado, mostramos a lista um arquivo A, que ocupa os blocos 4, 7, 2, 10, 12, e um arquivo B, que ocupa os blocos 6, 3, 11 e 14. Em cada um dos blocos, a palavra inicial do bloco é usada para apontar para o próximo bloco da lista, sendo que o 'X' indica que o bloco é o último do arquivo. Os arquivos são seqüências de bytes



Implementação

- Na alocação por lista encadeada, o arquivo é armazenado em blocos não contíguos do disco:
- Devemos armazenar somente o ponteiro para o primeiro bloco.
 - A primeira palavra de cada bloco armazena um ponteiro para o próximo bloco do arquivo.
 - A vantagem é que agora todos os blocos do disco podem ser usados, e temos somente desperdício dentro de um bloco.
 - Existem duas desvantagens nesta técnica:
 - O acesso aos arquivos agora será lento.
 - O espaço para os dados não possui o tamanho do bloco.

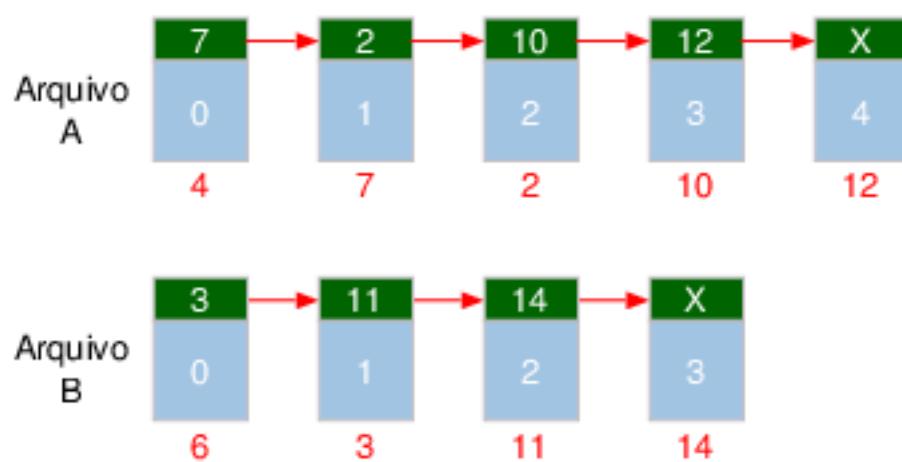


Se desejarmos acessar todo o arquivo A, de um modo seqüencial, deveríamos, ler, em ordem, os blocos 4, 7, 2, 10, e 12, o que fará o acesso ao arquivo ser mais lento do que, se, por exemplo, o arquivo estivesse armazenado contiguamente nos blocos 4, 5, 6, 7, e 8, pois os blocos podem estar dispersos por todo o disco.



Implementação

- Na alocação por lista encadeada, o arquivo é armazenado em blocos não contíguos do disco:
- Devemos armazenar somente o ponteiro para o primeiro bloco.
 - A primeira palavra de cada bloco armazena um ponteiro para o próximo bloco do arquivo.
 - A vantagem é que agora todos os blocos do disco podem ser usados, e temos somente desperdício dentro de um bloco.
 - Existem duas desvantagens nesta técnica:
 - O acesso aos arquivos agora será lento.
 - O espaço para os dados não possui o tamanho do bloco.

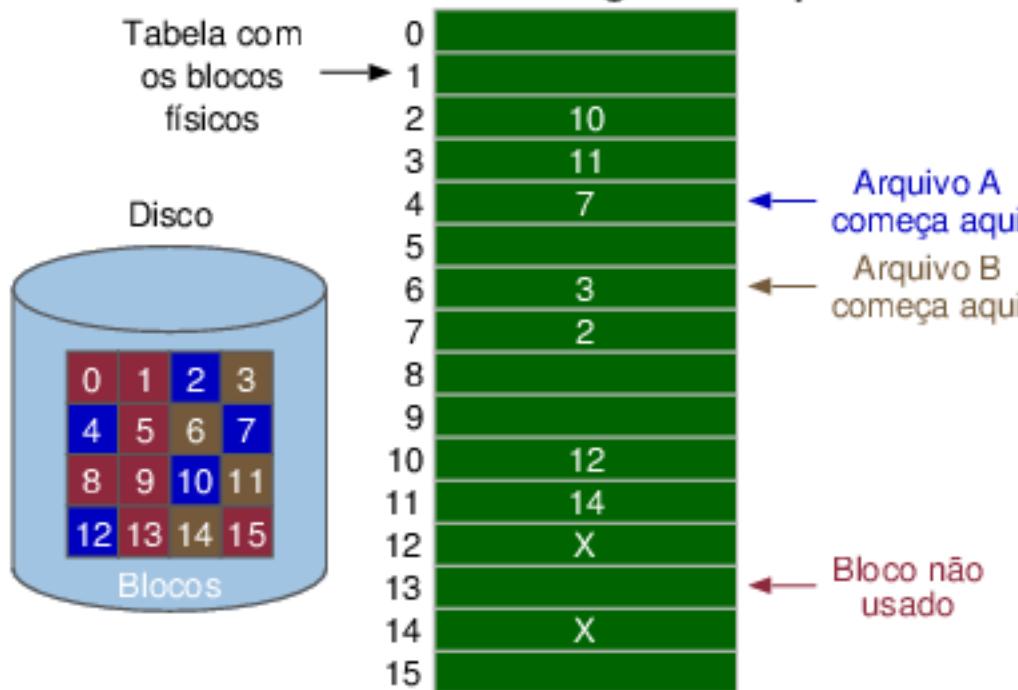


Se desejarmos acessar a posição 14K do arquivo B, se cada bloco possuir um tamanho de 4K, e se o ponteiro para o próximo bloco ocupar 1 byte, então deveríamos ler os blocos 6, 3, e 11, nesta ordem, e acessar a posição 2052 do bloco, com o byte desejado, pois cada bloco terá espaço para o armazenamento de 4095 bytes do arquivo B.



Implementação

- A técnica de alocação por lista encadeada utilizando um índice tenta corrigir os erros da técnica anterior:
- Usa uma tabela na memória com uma entrada para cada um dos blocos do disco.
 - Os números dos próximos blocos serão salvos nesta tabela, ao invés de dentro dos blocos, que armazenarão somente dados.
 - O acesso aleatório agora será mais rápido.
 - A desvantagem é que toda a tabela deve estar na memória.

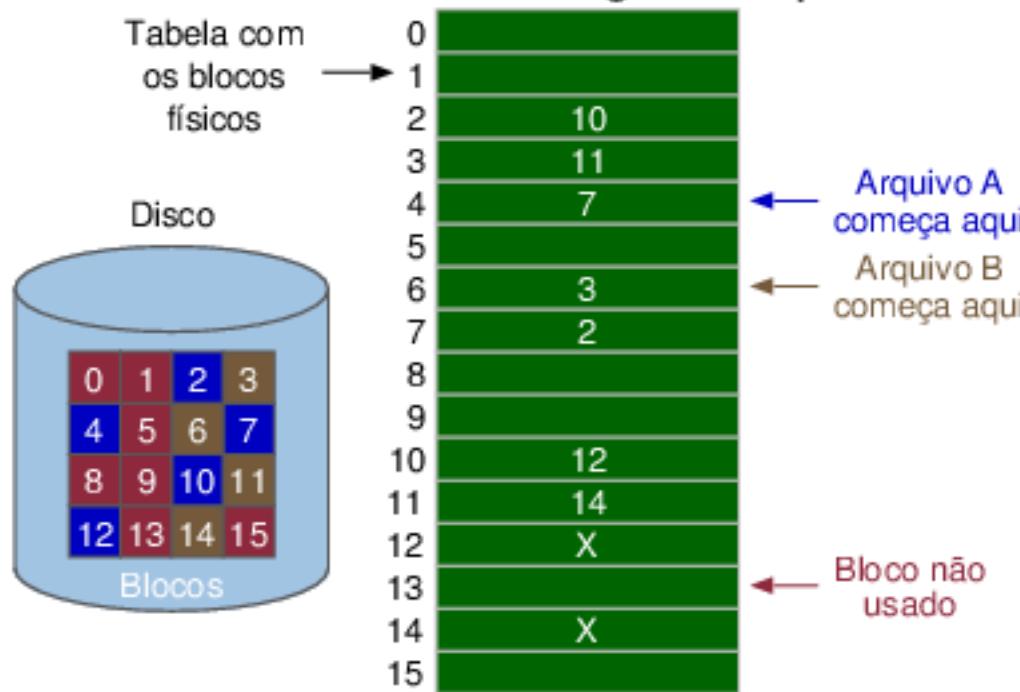


Exemplo anterior com os dois arquivos, A e B, num disco com 16 blocos numerados de 0 à 15. O arquivo A usa, em ordem, os blocos 4, 7, 2, 10, e 12, e o arquivo B usa, em ordem, os blocos 6, 3, 11, e 14. Assim como antes, um 'X' indica que o bloco é o último do arquivo.



Implementação

- A técnica de alocação por lista encadeada utilizando um índice tenta corrigir os erros da técnica anterior:
- Usa uma tabela na memória com uma entrada para cada um dos blocos do disco.
 - Os números dos próximos blocos serão salvos nesta tabela, ao invés de dentro dos blocos, que armazenarão somente dados.
 - O acesso aleatório agora será mais rápido.
 - A desvantagem é que toda a tabela deve estar na memória.

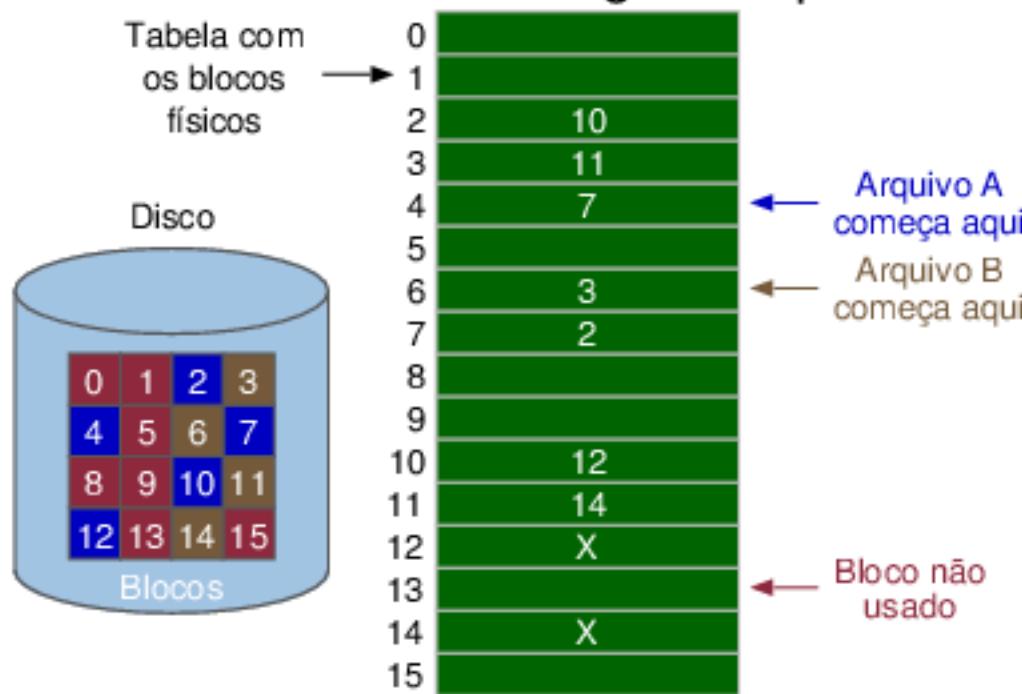


Agora, ao acessarmos todo o arquivo A, vamos acessar a tabela, a partir do bloco inicial 4 do arquivo A, para obter cada um dos outros blocos do arquivo, e acessá-los, pois estamos lendo o arquivo A de modo seqüencial.



Implementação

- A técnica de alocação por lista encadeada utilizando um índice tenta corrigir os erros da técnica anterior:
- Usa uma tabela na memória com uma entrada para cada um dos blocos do disco.
 - Os números dos próximos blocos serão salvos nesta tabela, ao invés de dentro dos blocos, que armazenarão somente dados.
 - O acesso aleatório agora será mais rápido.
 - A desvantagem é que toda a tabela deve estar na memória.

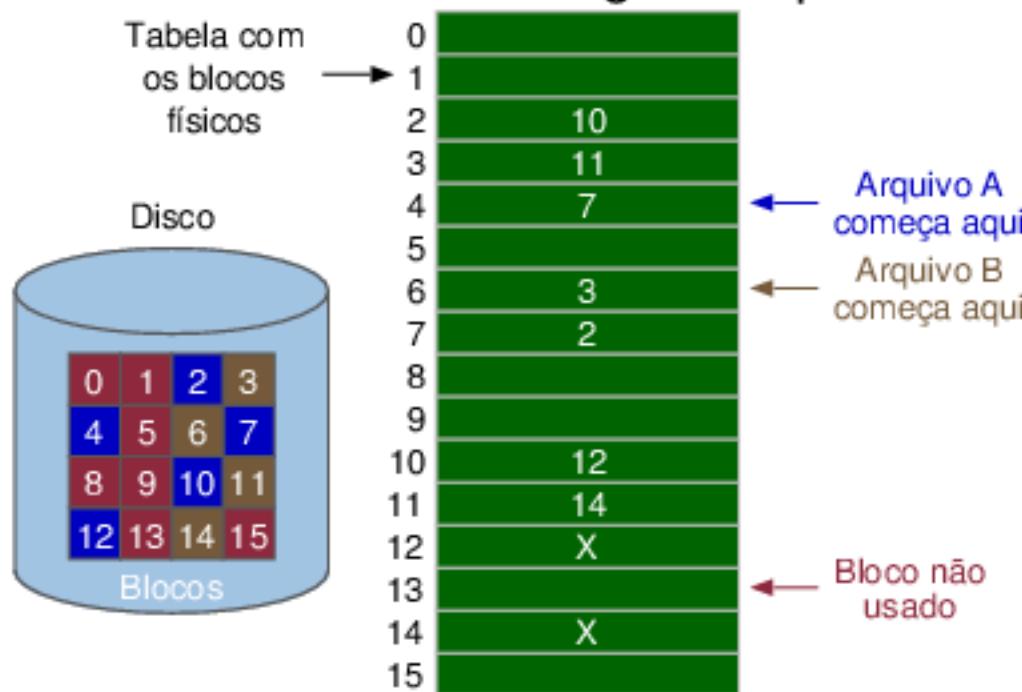


No caso do arquivo B, em que desejamos acessar a posição 14K que está no 3 bloco lógico do arquivo (o bloco físico 11), vamos acessar as entradas 6, 3, e 11 da tabela, mas vamos acessar somente o bloco físico 11 do disco.



Implementação

- A técnica de alocação por lista encadeada utilizando um índice tenta corrigir os erros da técnica anterior:
- Usa uma tabela na memória com uma entrada para cada um dos blocos do disco.
 - Os números dos próximos blocos serão salvos nesta tabela, ao invés de dentro dos blocos, que armazenarão somente dados.
 - O acesso aleatório agora será mais rápido.
 - A desvantagem é que toda a tabela deve estar na memória.



O MS-DOS é um exemplo de um sistema que usa esta abordagem. O problema com a abordagem é o espaço gasto pela tabela. pois, se o disco tiver 500.000 blocos, então cada entrada deverá ter pelo menos 3 bytes, e com isso, o tamanho da tabela é de pelo menos 1.5M. O acesso aos arquivos será mais rápido, pois lemos somente os blocos do arquivo que são acessados.



D/D

Implementação



Na técnica de alocação baseada em **nós-i**:

- Um nó-i é associado a cada arquivo do sistema.
- Cada nó-i contém os atributos do arquivo, e os endereços dos blocos do disco com o arquivo.
- Os endereços dos primeiros blocos estão dentro do nó-i:
 - Se necessário, podemos usar os endereços dos blocos indiretos **simples**, **duplo**, e **triplo**:

Nó-i



Nó-i usado pelo sistema UNIX. Os primeiros campos do nó-i contém os atributos do arquivo que vimos anteriormente. Depois disso, temos 10 campos (em vermelho) que podem armazenar os endereços dos primeiros 10 blocos do arquivo no disco. Se o arquivo possui 10 ou menos blocos, então todos os blocos estarão no nó-i. Se o arquivo possuir mais de 10 blocos, então devemos usar pelo menos um dos três últimos campos, os endereços de blocos indiretos **simples**, **duplo**, e **triplo**.

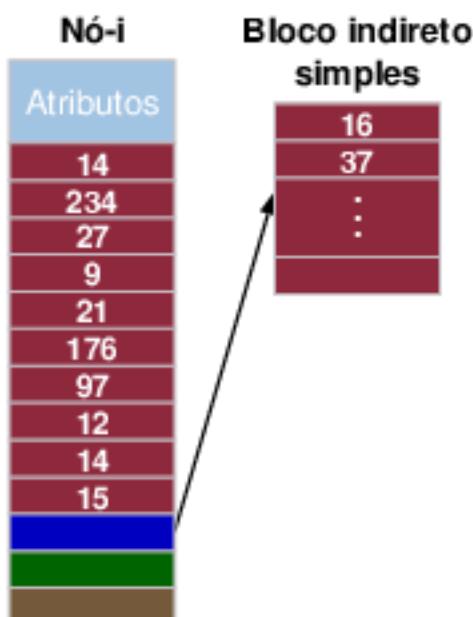


Implementação



Na técnica de alocação baseada em **nós-i**:

- Um nó-i é associado a cada arquivo do sistema.
- Cada nó-i contém os atributos do arquivo, e os endereços dos blocos do disco com o arquivo.
- Os endereços dos primeiros blocos estão dentro do nó-i:
 - Se necessário, podemos usar os endereços dos blocos indiretos **simples**, **duplo**, e **triplo**:



Caso o arquivo possua mais de 10 blocos, deveremos usar o campo **bloco indireto simples** (em azul na figura), que nos fornece um endereço de um bloco com endereços de blocos adicionais. Este número é igual ao de endereços que cabem em um bloco, isto é, se, por exemplo, o tamanho do bloco é de 4K, e o tamanho do endereço do bloco é de 4 bytes, teremos 1024 blocos adicionais, além dos 10 blocos que estão no nó-i.

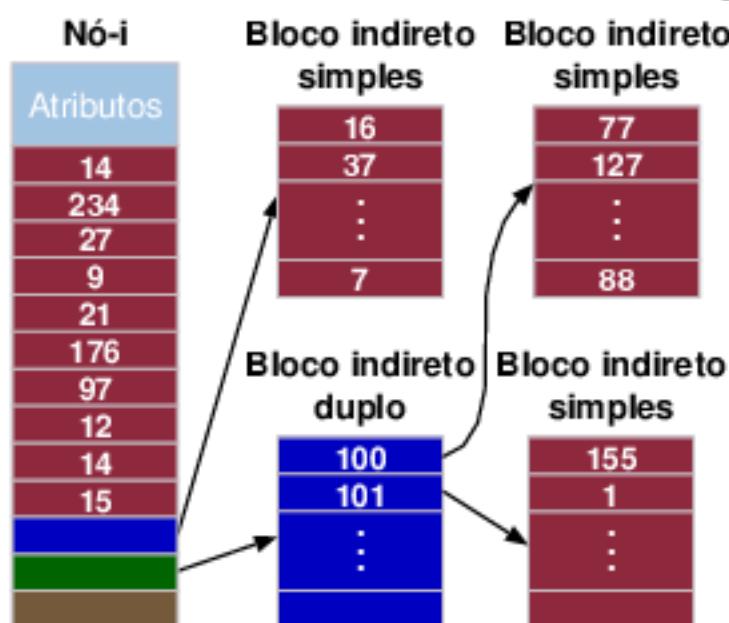


Implementação



Na técnica de alocação baseada em **nós-i**:

- Um nó-i é associado a cada arquivo do sistema.
- Cada nó-i contém os atributos do arquivo, e os endereços dos blocos do disco com o arquivo.
- Os endereços dos primeiros blocos estão dentro do nó-i:
 - Se necessário, podemos usar os endereços dos blocos indiretos **simples**, **duplo**, e **triplo**:

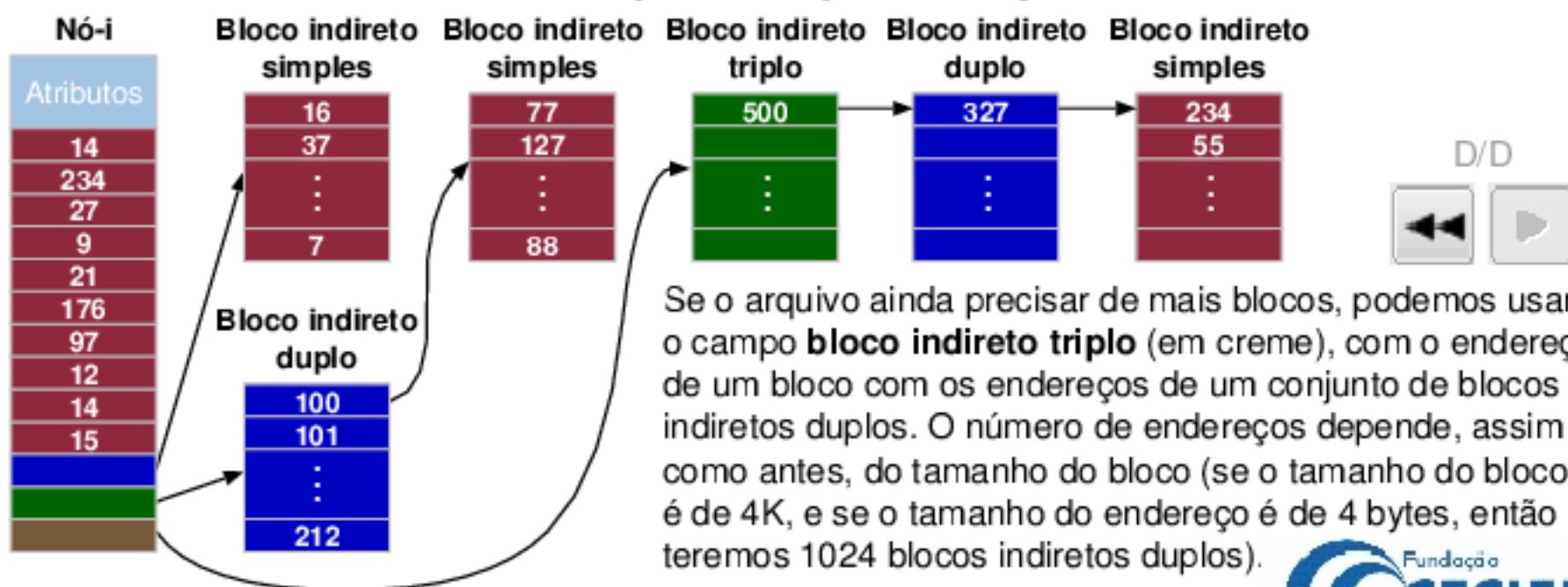


Se o arquivo precisar de mais blocos, então deveremos usar o campo de **bloco indireto duplo** (em verde na figura), que possui o endereço de um bloco com os endereços de um certo número adicional de blocos indiretos simples, que cabem no tamanho do bloco. Se o tamanho do bloco é de 4K, e o do endereço é de 4 bytes, teremos 1024 blocos indiretos simples adicionais.



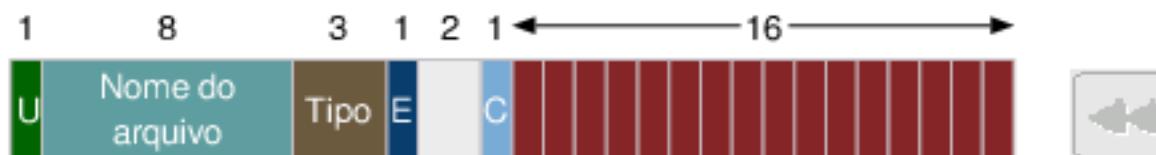
Implementação

- Na técnica de alocação baseada em **nós-i**:
- Um nó-i é associado a cada arquivo do sistema.
 - Cada nó-i contém os atributos do arquivo, e os endereços dos blocos do disco com o arquivo.
 - Os endereços dos primeiros blocos estão dentro do nó-i:
 - Se necessário, podemos usar os endereços dos blocos indiretos **simples**, **duplo**, e **triplo**:



Implementação

- O nome do caminho do arquivo é usado para obter a entrada do diretório com os atributos e os blocos usados pelo arquivo:
 - As informações, como vimos, podem estar no diretório, ou em uma estrutura especial, como os nós-i, apontada pelo diretório.
 - A principal função dos sistemas de diretório é a de mapear o nome do caminho aos seus atributos e blocos no disco.
- Um exemplo bem simples é o do sistema CP/M, que possui somente um diretório, cujas entradas possuem o seguinte formato:



Formato de uma entrada do único diretório do sistema CP/M. Cada entrada possui exatamente 32 bytes de comprimento. Pressione o mouse sobre cada um dos campos para ver o seu significado.

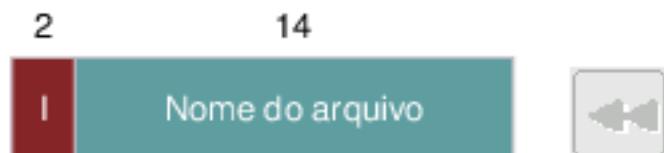
Implementação

→ O sistema MS-DOS permite uma árvore hierárquica de diretórios, sendo que as entradas dos diretórios possuem o seguinte formato:



Entrada de um diretório do MS-DOS. Cada entrada de um diretório possui um tamanho igual a 32 bytes. As entradas podem conter tanto arquivos, como diretórios, o que permite que seja criada uma hierarquia de diretórios. Pressione o mouse sobre cada um dos campos para ver o seu significado.

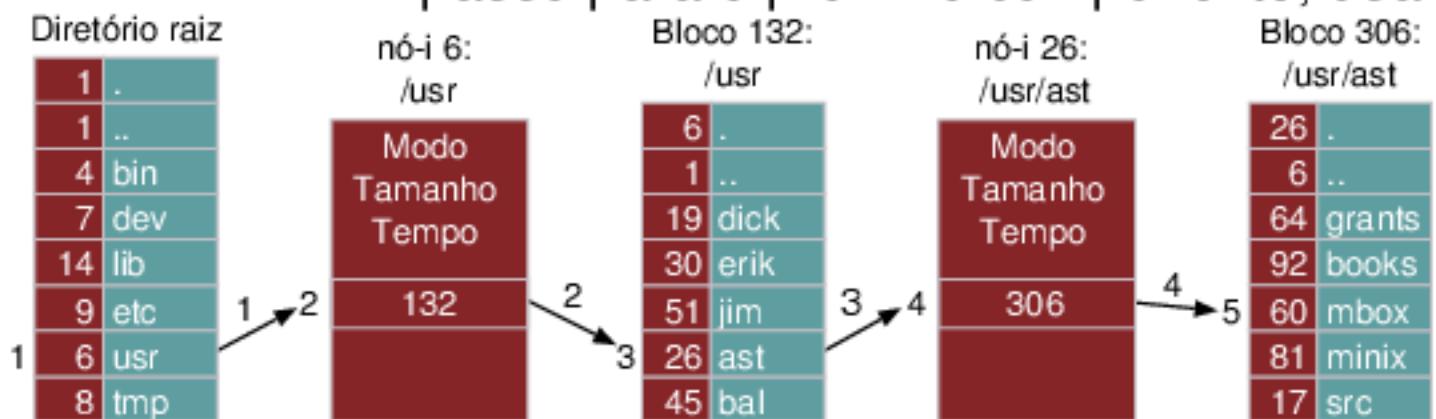
→ O sistema UNIX também permite uma hierarquia de diretórios, e possui uma entrada de diretório bem simples:



Formato da entrada de um diretório de uma versão preliminar do UNIX. A entrada, com 16 bytes de comprimento, é bem simples, e, assim como o MS-DOS, esta pode conter tanto um arquivo, como um diretório, o que possibilita a criação de uma hierarquia de diretórios. Temos sempre pelo menos duas entradas: a '.' e a '..'.

Implementação

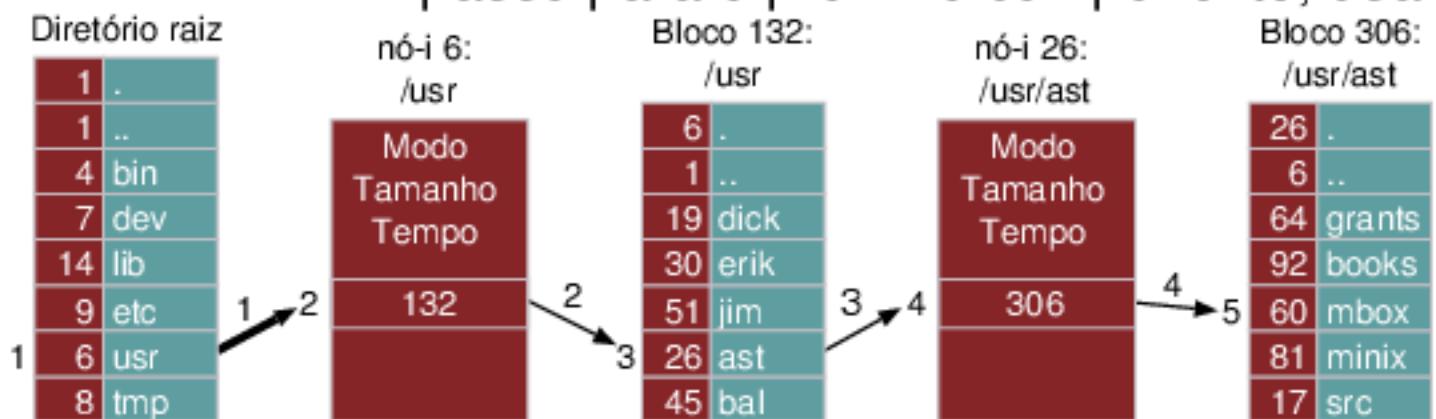
- ➡ Quando o arquivo é aberto, o sistema usa o nome de caminho do arquivo para localizar os seus atributos e os seus blocos.
- ➡ No UNIX, primeiramente acessamos o nó-i do diretório raiz, para obter os blocos com este diretório, e:
 - Procuramos pela entrada com o primeiro componente do caminho, obtemos o nó-i deste componente, e:
 - Se o componente for o nome do arquivo, este nó-i será o do arquivo, e será copiado para a memória.
 - Se o componente for um diretório, então repetimos este passo para o próximo componente, usando este nó-i.



Exemplo do acesso ao arquivo mbox do diretório /usr/ast (cujo caminho absoluto é /usr/ast/mbox), usando o algoritmo descrito acima.

Implementação

- Quando o arquivo é aberto, o sistema usa o nome de caminho do arquivo para localizar os seus atributos e os seus blocos.
- No UNIX, primeiramente acessamos o nó-i do diretório raiz, para obter os blocos com este diretório, e:
 - Procuramos pela entrada com o primeiro componente do caminho, obtemos o nó-i deste componente, e:
 - Se o componente for o nome do arquivo, este nó-i será o do arquivo, e será copiado para a memória.
 - Se o componente for um diretório, então repetimos este passo para o próximo componente, usando este nó-i.

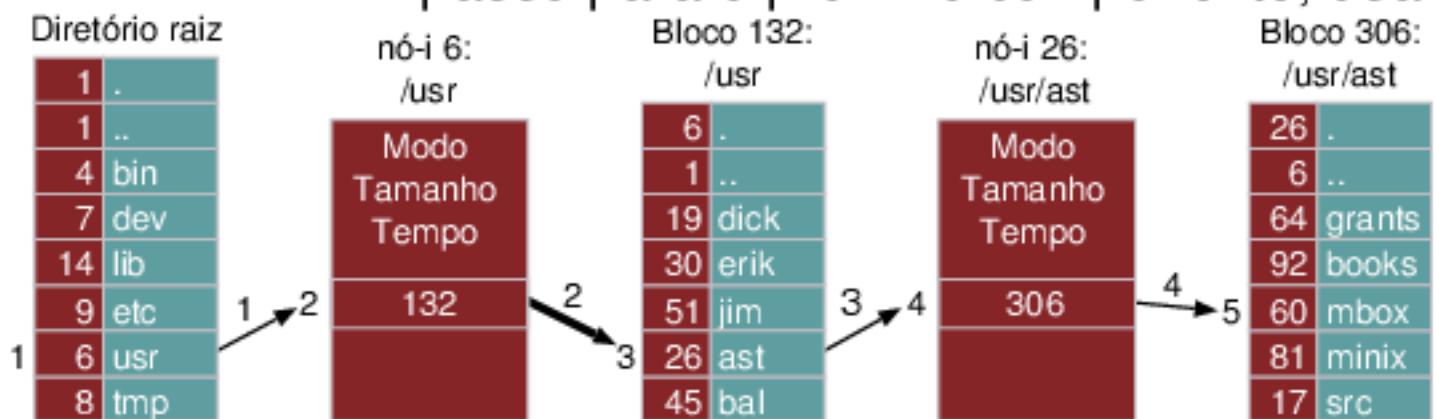


1: Procuramos pela entrada do diretório **usr** no diretório raiz, e descobrimos que o nó-i deste diretório é o 6 (caminho: **/usr/ast/mbox**).



Implementação

- Quando o arquivo é aberto, o sistema usa o nome de caminho do arquivo para localizar os seus atributos e os seus blocos.
- No UNIX, primeiramente acessamos o nó-i do diretório raiz, para obter os blocos com este diretório, e:
 - Procuramos pela entrada com o primeiro componente do caminho, obtemos o nó-i deste componente, e:
 - Se o componente for o nome do arquivo, este nó-i será o do arquivo, e será copiado para a memória.
 - Se o componente for um diretório, então repetimos este passo para o próximo componente, usando este nó-i.



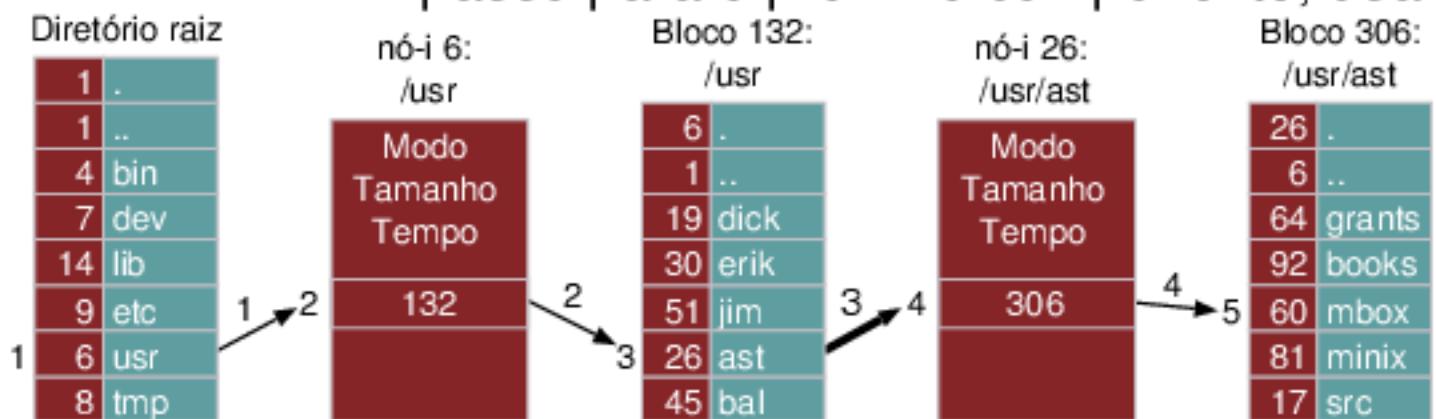
2: Verificando o nó-i 6, que contém o diretório **/usr**, vemos que as suas entradas estão dentro do bloco 132 do disco (caminho: **/usr/ast/mbox**).



C/F

Implementação

- Quando o arquivo é aberto, o sistema usa o nome de caminho do arquivo para localizar os seus atributos e os seus blocos.
- No UNIX, primeiramente acessamos o nó-i do diretório raiz, para obter os blocos com este diretório, e:
 - Procuramos pela entrada com o primeiro componente do caminho, obtemos o nó-i deste componente, e:
 - Se o componente for o nome do arquivo, este nó-i será o do arquivo, e será copiado para a memória.
 - Se o componente for um diretório, então repetimos este passo para o próximo componente, usando este nó-i.

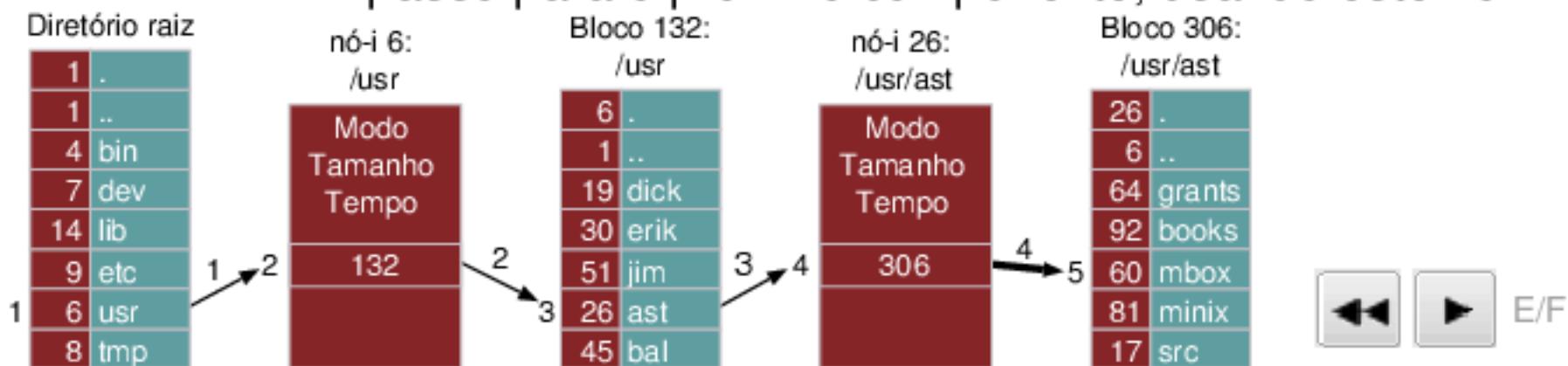


3: Agora procuramos pelo diretório **ast** dentro do diretório **/usr**, para descobrir que este diretório está associado ao nó-i 26 (caminho: **/usr/ast/mbox**).



Implementação

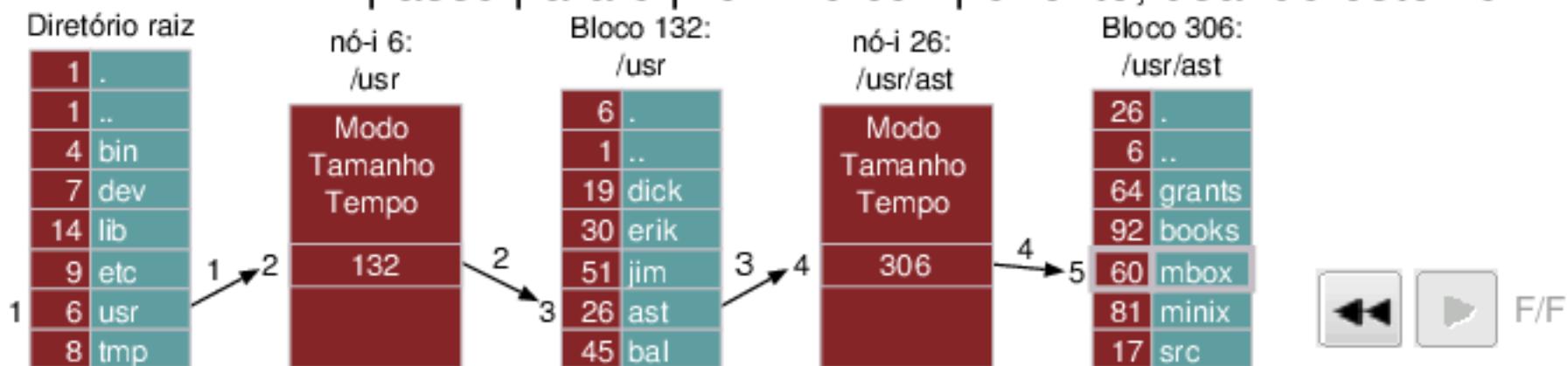
- Quando o arquivo é aberto, o sistema usa o nome de caminho do arquivo para localizar os seus atributos e os seus blocos.
- No UNIX, primeiramente acessamos o nó-i do diretório raiz, para obter os blocos com este diretório, e:
 - Procuramos pela entrada com o primeiro componente do caminho, obtemos o nó-i deste componente, e:
 - Se o componente for o nome do arquivo, este nó-i será o do arquivo, e será copiado para a memória.
 - Se o componente for um diretório, então repetimos este passo para o próximo componente, usando este nó-i.



4: Do nó-i 26, associado ao diretório **/usr/ast**, descobrimos que as suas entradas estão no bloco 306 do disco (caminho: **/usr/ast/mbox**).

Implementação

- Quando o arquivo é aberto, o sistema usa o nome de caminho do arquivo para localizar os seus atributos e os seus blocos.
- No UNIX, primeiramente acessamos o nó-i do diretório raiz, para obter os blocos com este diretório, e:
 - Procuramos pela entrada com o primeiro componente do caminho, obtemos o nó-i deste componente, e:
 - Se o componente for o nome do arquivo, este nó-i será o do arquivo, e será copiado para a memória.
 - Se o componente for um diretório, então repetimos este passo para o próximo componente, usando este nó-i.



5: Pelas entradas de /usr/ast, vemos que o arquivo **mbox** está associado ao nó-i 60. Este nó-i então será copiado para a memória (caminho: /usr/ast/mbox).