

Sistemas Operacionais

Notas de Aula

Francisco Rapchan
rapchan@writeme.com

CAPÍTULO 4 – SISTEMAS DE ARQUIVO

- A parte do sistema operacional responsável pelo tratamento dos arquivos é denominada **sistema de arquivos**.
- sistema de arquivos tem dois níveis de apresentação: o do usuário e do desenvolvedor.
- Ao usuário comum não interessam os detalhes da implementação dos arquivos. Estas informações entretanto são muito importantes para os desenvolvedores de aplicações que fazem acesso a informações de arquivos.

4.1 ARQUIVOS

Nesta seção vamos examinar os arquivos sob a ótica de seus usuários ou seja, estamos interessados em descobrir como os arquivos são usados e quais as suas propriedades.

Identificação de arquivos

O nome do arquivo é o elemento mais importante para sua identificação. Sistemas antigos só aceitavam poucos caracteres para identificar o arquivo. Hoje em dia é comum os sistemas de arquivos aceitarem nomes com até 256 caracteres.

Estrutura do arquivo

Há 3 formas comuns de estrutura de arquivos:

- Seqüência de bytes (UNIX e DOS).
- Registros de tamanho fixo com 80, 128 ou 132 bytes (o CP/M usava registros de 128 bytes).
- Árvore. Um campo do registro é feito como chave. Permite a implementação de bancos de dados mais rápidos. Usado em sistemas operacionais mais antigos de mainframes.

Tipos de arquivos

Os sistemas operacionais costumam distinguir alguns tipos de arquivos.

- Arquivos regulares (ASCII ou Binários). São os arquivos gerados diretamente pelos usuários.
- Arquivos de diretórios.
- Arquivos de I/O.
- Arquivos especiais blocados que simulam discos.

Todos os sistemas operacionais devem reconhecer, no mínimo um tipo de arquivo, mais precisamente o tipo de arquivo executável.

Tipos de acesso

- Sequencial. Os registros do arquivo são lidos um a um, do início ao fim do arquivo, de forma seqüencial.
- Randômico ou aleatório. Os registros são acessados através de uma chave em vez de sua posição. Este tipo de acesso é fundamental para a implementação de bancos de dados.

Atributos dos arquivos

Alem do nome, é conveniente guardar algumas outras informações dos arquivos tais como data de criação e tamanho.

[Ver figura 4.4 pg106 para uma lista de atributos de arquivos]

Operações sobre arquivos

Os sistemas operacionais implementam diversas operações diferentes para armazenamento e recuperação de informações em arquivos.

[Ver pg106-107 para uma lista de chamadas de sistema mais comuns relacionadas ao sistema de arquivos]

O autor mostra um exemplo de programa que copia um arquivo com outro nome:

```
copyfile abc xyz
```

[Ver figura 4.5 pg108 Um programa simples para copiar arquivos]

Arquivos mapeados em memória

[Apenas leitura.]

4.2 DIRETÓRIOS

Para tratar dos arquivos, a maioria dos sistemas operacionais utiliza o conceito de **diretório**, que em muitos sistemas também é um arquivo.

Um diretório contém tipicamente um conjunto de entradas, cada uma a respeito de um dos arquivos.

Sistemas de diretório hierárquico

Pode-se ter uma estrutura de diretórios em árvore. Eventualmente cada usuário pode ter sua própria árvore de diretórios.

Nomes de arquivos

Muitos sistemas operacionais que suportam o conceito de sistema de arquivos hierárquico têm duas entradas especiais em cada diretório, "." (dot) que referencia o diretório corrente e ".." (dotdot) que referencia o diretório pai.

Operações sobre diretórios

[Apenas leitura.]

4.3 IMPLEMENTAÇÃO DO SISTEMA DE ARQUIVO

Vamos agora discutir aspectos internos dos sistemas de arquivos.

4.3.1 Implementação de arquivos

Vamos estudar 4 métodos para controlar a correspondência entre blocos no disco e arquivos.

1 - Alocação Contígua

Coloca todo o arquivo em blocos contíguos

- Só é preciso guardar o endereço do primeiro bloco.
- A performance de nenhum outro método de alocação chega perto da performance de alocação contígua.

Dois problemas:

- Só pode ser usada se o tamanho máximo do arquivo for conhecido no momento de sua criação.
- Perde-se muito espaço útil com a fragmentação.

2 - Alocação com listas ligadas

A primeira palavra de cada bloco é usada como um ponteiro para o próximo bloco.

Há duas desvantagens básicas:

- O acesso randômico é extremamente lento.
- A quantidade de informação armazenada em um bloco não é mais uma potência de dois.

3 - Alocação com lista ligada usando um índice

Ao invés de colocar o ponteiro no próprio bloco, cria-se um vetor que relacione os ponteiros para o próximo bloco com o índice do vetor.

[Ver Fig 4.11 pg 114 para ver uma tabela de alocação]

- Apesar do acesso ser randômico ele é muito mais fácil de ser implementado.
- MS-DOS usa este método de alocação de espaço em disco.

A maior desvantagem deste esquema é que toda a tabela deve estar na memória durante todo o tempo. Com um disco de 500.000 blocos de 1K a tabela terá 500.000 entradas, cada uma delas com 3 ou 4 bytes. Em sua totalidade a tabela ocupa 1,5 a 2 Mbytes.

4 - Nós Índices (i-Node)

É uma tabela de índices que lista os atributos e os endereços em disco dos blocos do arquivo.

Os primeiros endereços de disco são armazenados no próprio i-node, de maneira que para arquivos pequenos, toda a informação necessária é encontrada diretamente no nó.

Há ainda 3 outras entradas:

- Ponteiro para bloco indireto simples.
- Ponteiro para bloco indireto duplo.
- Ponteiro para bloco indireto triplo.

Cada bloco endereça 256 entradas.

4.3.2 Implementação de diretórios

A principal função do diretório é mapear o nome ASCII do arquivo na informação necessário à localização do dado.

Os atributos podem ser armazenados na entrada do diretório ou, para os sistemas baseados em i-node, os atributos podem ser armazenados no próprio i-node.

Diretório no CP/M

No CP/M só existe um diretório.

Suporta apenas 16 blocos do disco. Os índices são armazenados na entrada do diretório (se o arquivo usa mais de 16 blocos, são alocados mais entradas de diretório para o mesmo arquivo).

[Ver figura 4.13 pg115 Uma entrada de diretório do CP/M]

Diretório no MS-DOS

[Ver figura 4.14 pg116 Uma entrada de diretório do MS-DOS]

[Ver figura 8.20 pg241 Uma entrada de diretório do MS-DOS – detalhada]

Diretório no UNIX

A entrada de diretório do UNIX possui apenas o número do i-node e o nome do arquivo. Todas as outras informações estão no i-node.

[Ver figura 4.16 pg116 mostra a localização de um arquivo nos i-node UNIX]

4.3.3 Arquivos compartilhados

[Apenas leitura.]

4.3.4 Gerência do espaço em disco

Tamanho do bloco

Se o bloco for muito grande e a média dos arquivos for de tamanho pequeno haverá muito desperdício de espaço.

Se o bloco for pequeno demais, os arquivos maiores ocuparão muitos blocos.

Controle dos blocos livres

Pode ser feito por:

- lista ligada dos blocos livres
- mapa de bits

[Ver figura 4.20 pg119 Blocos livres numa lista ligada e um mapa de bits]

Cotas do Disco

O administrador do sistema atribui a cada usuário um lote máximo de arquivos e blocos, e o sistema operacional controle os usuários de forma que eles não possam exceder suas cotas.

4.3.5 Confiabilidade dos sistemas de arquivos

A destruição de um sistema de arquivos é o maior desastre que pode acontecer em um computador.

- **Gerência de blocos ruins**
- **Backups.** Usar fitas streamer, disco espelho. A cópia deve ser feita de forma incremental ou seja, deve ser feito o backup apenas dos arquivos que foram alterados.
- **Consistência do sistema de arquivos.** Deve haver mecanismos que garantam a consistência mesmo se houver uma pane durante as atualizações das estruturas usadas pelo sistema de arquivos tais como i-nodes, diretórios ou lista de blocos livres.
- **Proteção contra comandos perigosos:** O sistema deve evitar que o usuário cometa erros tais como apagar arquivos de sistema, apagar o programa shell, etc.
[Ver pg123 o exemplo do `rm *`]

4.3.6 Performance do sistema de arquivos

- autor diz que um acesso a disco é uma operação até 100.000 vezes mais lenta que um acesso a memória.
- Devido a esta demora no acesso, muitos sistemas de arquivos são projetados para reduzir ao mínimo necessário seus acessos ao disco.
- A técnica mais comum a utilização de **caches** (a palavra cache vem do francês *acher*, que significa esconder).
- Neste contexto, cache é um conjunto de blocos do disco que são mantidos na memória para melhorar a performance.
- Um problema com cache é manter a integridade do sistema. Não se deve manter os dados no cache por muito tempo sem atualizar o disco.
- No UNIX, o programa **update** grava as informações do cache no disco (executando a chamada SYNC) a cada 30 segundos. Você pode também executar diretamente o comando **sync**.
- No DOS os blocos do cache que são alterados são imediatamente gravados no disco.

4.4 SEGURANÇA

O item neste item ou autor do livro texto traz uma série de fatos e histórias interessantes sobre a segurança de sistemas operacionais. Entre os assuntos relacionados estão:

- Falhas de segurança famosas;
- Violação da Internet;
- Ataques à segurança de um computador (inclusive Vírus);
- Sistemas de passwords, etc.

4.5 MECANISMOS DE PROTEÇÃO

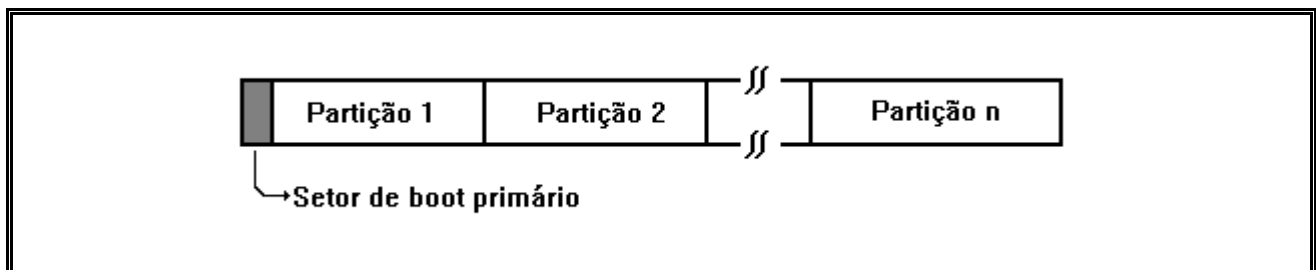
[Apenas Leitura]

APÊNDICE – O SISTEMA DE ARQUIVOS DO DOS

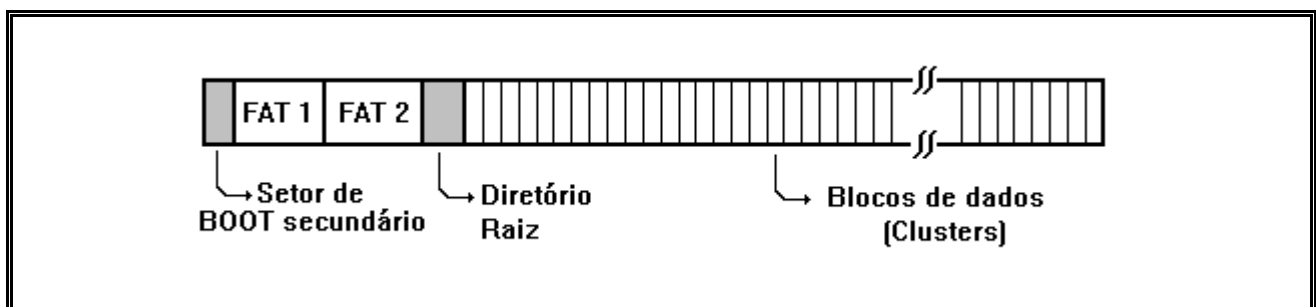
OBS: O sistema de arquivos descrito aqui aplica-se tanto ao DOS quanto ao Windows (95, 98, e algumas versões do NT). Este sistema de arquivos é chamado de FAT.

Um sistema de arquivos DOS pode ser composto de vários discos rígidos (mais conhecidos como *winchester*) e discos flexíveis (conhecidos como *floppy*). No DOS, cada disco é representado por uma letra de **A** a **Z**. As letras **A** e **B** são reservadas para discos flexíveis, enquanto que as letras **C** e **D** são reservadas para discos rígidos. As letras restantes podem ser usadas livremente.

Um disco rígido pode ser dividido em discos lógicos, chamados **partições**. A estrutura de um disco particionado é mostrada a seguir.



Cada partição pode conter um sistema operacional diferente, com estruturas internas diferentes. O setor de *boot* primário tem informações sobre quantas partições existem no disco e qual delas é a de *boot*. Uma partição DOS tem a seguinte estrutura:



O setor de *boot* secundário contém informações sobre a partição tais como: número de *clusters*, número de FATs (a FAT 2 é opcional), número de setores por *cluster*, número de bytes por setor, etc.

Após estes parâmetros vêm o código de inicialização (*bootstrap*). O *bootstrap* sabe que é exatamente o DOS que está sendo carregado, já que esta é uma partição DOS "*bootável*". Ele procura no diretório raiz pela existência dos arquivos IO.SYS e MSDOS.SYS e usa o BIOS para carregá-los na memória. Desta forma, o *bootstrap* da BIOS (ROM *bootstrap*) precisa apenas carregar a informação do setor de *boot* e executar o

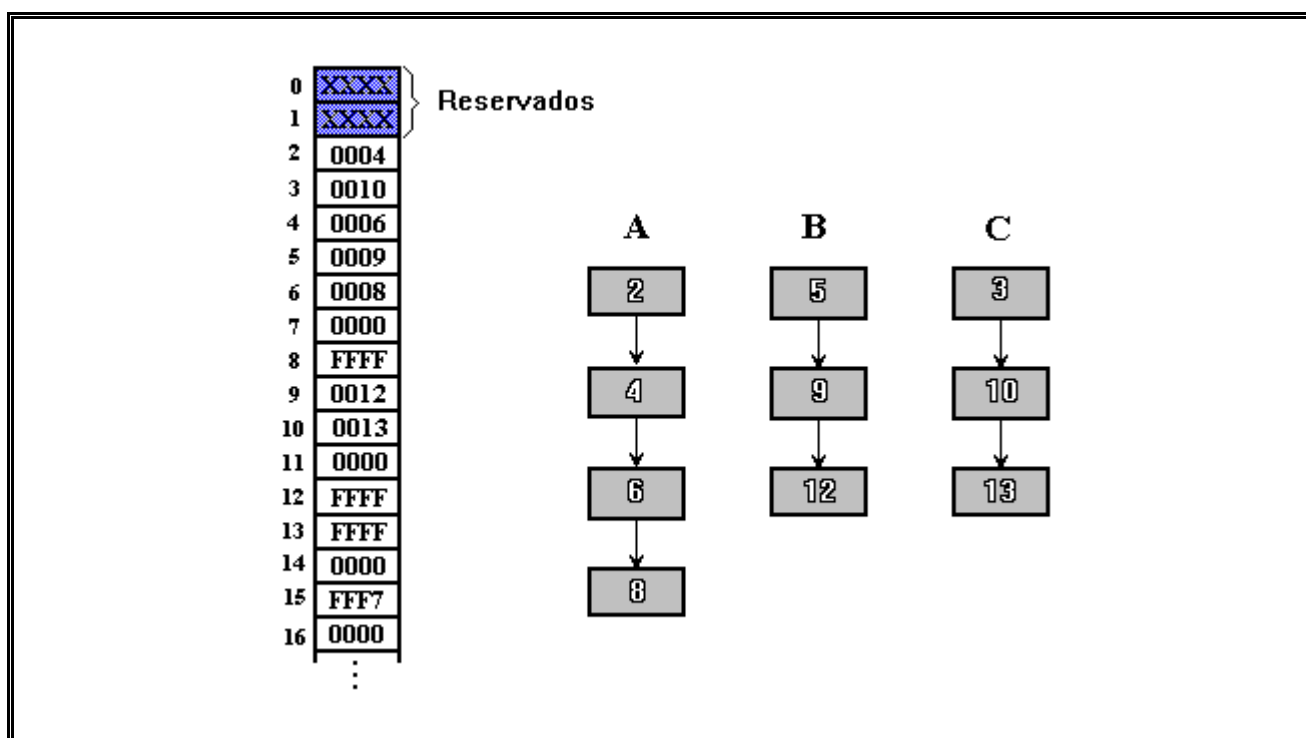
bootstrap. Com esse esquema, é possível carregar qualquer sistema operacional, já que é o *bootstrap* quem se encarrega das nuances da carga de um S.O.

Depois do setor de *boot* secundário vem a **FAT** (*File Allocation Table*), que mantém informação de todo espaço no disco. Por questões de confiabilidade, algumas versões mantêm duas FATs idênticas.

A FAT contém uma entrada para cada *cluster* (1 *cluster* é a menor unidade de alocação do disco). O valor de cada uma destas entradas diz qual a situação do *cluster*, exceto as duas primeiras entradas da FAT que são usadas para indicar o formato e tamanho do disco. Os valores de cada entrada podem ser os seguintes:

Valor Hexa	Significado
0	<i>Cluster</i> não usado e disponível
FFF8 a FFFF	Último <i>cluster</i> de um arquivo
FFF7	Setor defeituoso (<i>bad cluster</i>)
Outro valor	Número do proximo <i>cluster</i> do arquivo

Como exemplo de como funciona a FAT, imagine que existam três arquivos no disco, chamados **A**, **B** e **C**. O arquivo A ocupa os *clusters* 2,4,6 e 8, nesta ordem, ou seja, o primeiro bloco de dados está no *cluster* 2 é o ultimo no *cluster* 8. O arquivo B ocupa os *clusters* 5, 9 e 12. O arquivo C os *clusters* 3, 10 e 13. Suponha também que o *cluster* 15 esteja marcado como defeituoso. A nossa FAT seria então:



Um diretório DOS é uma estrutura com uma entrada por cada arquivo. Cada entrada possui 32 bytes e tem o seguinte formato:



Um arquivo é designado por no máximo 11 caracteres, sendo 8 para o nome e 3 para a extensão. Note que o ponto entre o nome e a extensão não é guardado. Ele só é usado na sua representação.

O byte de atributos possui bits para as seguintes representações:

- A - Aceso quando o arquivo é modificado (Archive)
- D - Aceso indica que aquela entrada é para um diretório
- V - Aceso indica que a entrada é para o nome do disco (Volume)
- S - Aceso para os arquivos do sistema. Não podem ser deletados
- H - Aceso para esconder arquivos (não são listados no comando DIR)
- R - Aceso indica que o arquivo é apenas de leitura

Os dois campos seguintes armazenam a hora e a data da última modificação do arquivo. A codificação é feita com 6 bits para os segundos, 6 bits para os minutos, 4 bits para a hora, 5 bits para o dia, 4 bits para o mês e 7 bits para o ano (a partir de 1980).

O próximo campo indica o *cluster* inicial do arquivo. É usando este valor na FAT que é possível encontrar a sequência de *clusters* onde está o arquivo no disco. O campo seguinte informa o tamanho do arquivo em bytes.

Um subdiretório no DOS, assim como no UNIX, é um arquivo com características especiais. Ele ocupa um ou mais bloco de dados mas possui suas informações (dados) organizadas na forma de diretório. Com isso é possível criar uma estrutura em árvore com infinitos subdiretórios. A limitação será imposta pela limitação física do disco, no caso sua capacidade.

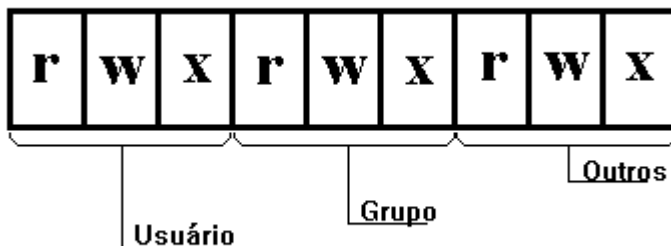
Um diretório no DOS possui duas entradas especiais, chamadas • e •• que, assim como no UNIX, representam o diretório corrente e o diretório pai (aquele imediatamente acima do corrente na árvore de diretórios). Cada uma destas entradas tem como *cluster* inicial aquele correspondente ao seu próprio ou do diretório pai, respectivamente. Da mesma forma que no UNIX, estas duas entradas permitem o encadeamento na árvore de diretórios. Este encadeamento é importante, pois o sistema precisa saber em que ponto da árvore de diretórios está o diretório atual quando o usuário pedir uma mudança de diretório (usando o comando CD).

ARQUIVOS NO UNIX

A organização dos arquivos no UNIX é feita em forma de uma árvore de diretórios. Partindo-se de um diretório raiz tem-se vários arquivos e subdiretórios. Descendo a estes subdiretórios encontramos também vários arquivos e outros subdiretórios, e assim sucessivamente. Esta organização permite que se estruture melhor os arquivos, criando-se subdiretórios diferentes para agrupar arquivos correlatos, evitando assim que eles fiquem misturados.

Cada usuário do UNIX possui um subdiretório próprio, chamado diretório base (home directory), onde todos os seus arquivos são colocados. Só o próprio usuário e o superuser tem direito de acesso aos arquivos no diretório base.

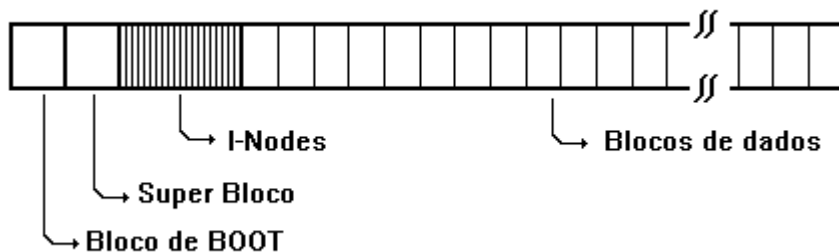
Os direitos de acesso de um arquivo ou diretório são guardados numa seqüência de 9 bits, dispostos da seguinte forma:



De acordo com os valores destes bits, são atribuídos direitos de ler (r) escrever (w) e executar (x) para o próprio usuário, usuários do mesmo grupo e outros usuários.

SISTEMA DE ARQUIVOS

Todos os discos que contém um sistema de arquivos do UNIX possuem o seguinte formato:



O bloco zero não é usado pelo UNIX e geralmente contém informações sobre o *boot* do sistema. Bloco 1 é o **super bloco**. Ele contém todas as informações a respeito do sistema de arquivos como número de blocos de dados, tamanho dos blocos de dados, número de *i-nodes*, etc.

Depois do super bloco vêm os ***i-nodes***. Cada *i-node* descreve exatamente um arquivo, mantendo informações sobre o proprietário, bits de proteção e a localização do arquivo no disco.

Em seguida aos *i-nodes* vêm os blocos de dados. Todos os arquivos e diretórios estão guardados aí. O bloco de dados 1 corresponde ao diretório raiz. O bloco de dados zero não é usado e é nele que são mantidas as informações de blocos não utilizáveis do disco (*bad blocks*).

A estrutura do diretório no UNIX é muito simples. Cada linha contém apenas o nome do arquivo ou diretório e seu *i-node* correspondente. Todas as outras informações sobre este arquivo ou diretório são guardadas no *i-node*. Como exemplo, vamos procurar no disco pelo arquivo cujo *path* absoluto é */usr/ast/a.c*, ou seja, o arquivo **a.c** no diretório **/usr/ast**.

Olhando no bloco de dados 1 (diretório raiz) encontramos os seguintes dados:

1	###
1	#####
4	bin
7	dev
6	usr
8	tmp

O sub-diretório **usr** é indicado pelo *i-node* 6. Olhando para este *i-node* vemos o seguinte:

Modo Tamanho, etc...
132
(outras informações)

O *i-node* 6 diz que o diretório correspondente está no bloco de dados 132. Buscando este bloco temos:

6	###
1	#####
19	pedro
30	texto
26	ast
51	prog.c

O sub-diretório **ast** está indicado pelo *i-node* 26. Buscando este *i-node*, temos:

Modo Tamanho, etc...
406
(outras informações)

O sub-diretório **/usr/ast** está no bloco de dados 406. Buscando este bloco temos:

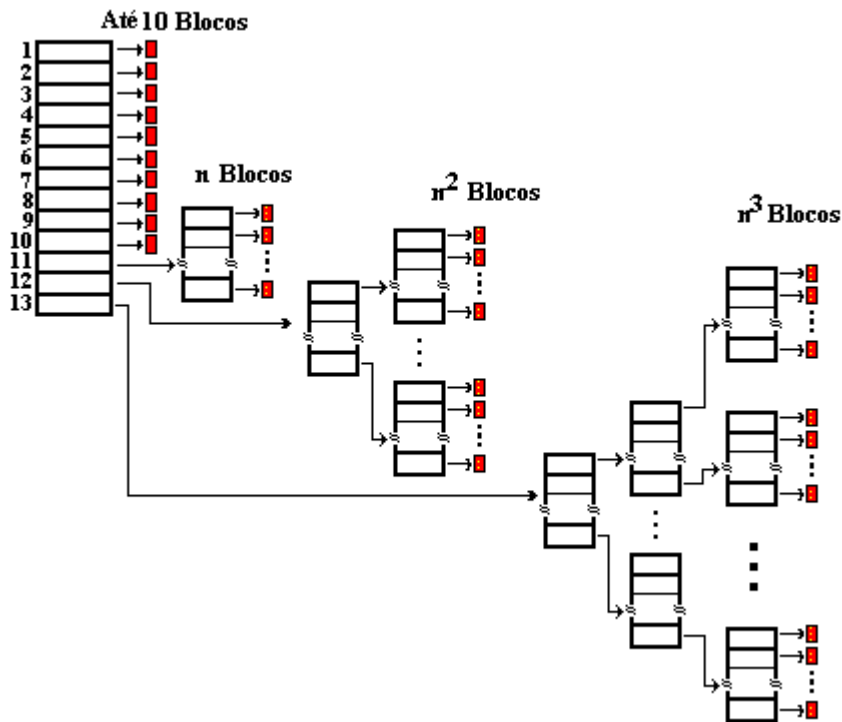
26	###
6	#####
64	livro
92	mail
81	bin
17	a.c

Vemos então que o arquivo que procuramos está indicado pelo *i-node* 17.

O mapeamento do arquivo no disco é feito da seguinte forma: Existe no *i-node* uma tabela com 13 entradas, onde as 10 primeiras entradas apontam para os 10 primeiros blocos de dados do disco onde está o arquivo. Caso o arquivo seja maior que 10 blocos de dados, a décima primeira entrada desta tabela aponta para um

bloco de dados que contém uma tabela com n entradas, onde cada uma dessas entradas aponta para um bloco de dados do arquivo. O valor de n vai depender do tamanho do bloco de dados.

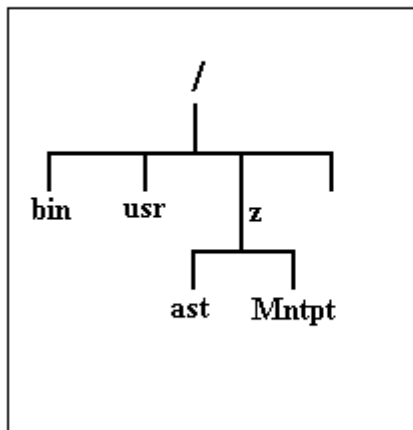
Caso o arquivo seja maior que $n + 10$ blocos, a décima segunda entrada da tabela de alocação no *i-node* aponta para um bloco de dados que contém uma tabela com n entradas que apontam para blocos de dados que contêm tabelas com n entradas que apontam para os blocos de dados do arquivo. Se o arquivo for maior que $n^2 + n + 10$, então a décima terceira entrada da tabela de alocação no *i-node* aponta para um bloco de dados que possui uma tabela com n entradas que aponta para blocos de dados que são tabelas com n entradas que apontam para blocos de dados que também são tabelas com n entradas que apontam para os blocos de dados do arquivo. Desta forma o tamanho máximo de um arquivo é $n^3 + n^2 + n + 10$ blocos de dados. Este esquema é representado na figura a seguir:



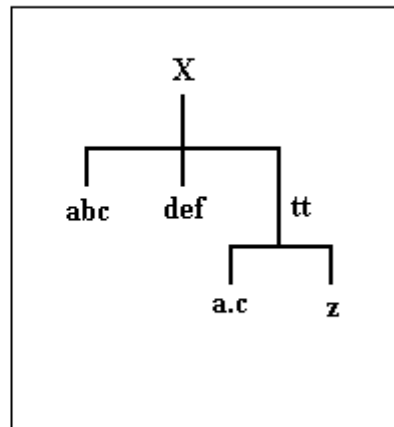
Os subdiretórios ### e ##### são usados para manter informação do diretório corrente e do diretório anterior, respectivamente. O diretório ##### é usado para saber de qual diretório o atual veio. O diretório ### é usado para manter a informação do diretório corrente, informação esta que será usada pelos subdiretórios abaixo dele. Com este esquema, temos uma lista encadeada, que permite que se volte na árvore de diretórios exatamente como se desceu.

No UNIX não existe o conceito de múltiplos discos. Um mesmo disco físico pode ser logicamente dividido em várias partições (camadas *file systems*) ou pode-se ter vários *file systems* em discos físicos diferentes. No UNIX você não tem como referenciar estas partições. Elas são **montadas** numa parte de um outro *file system*, de forma que passam a ser vistos como pertencentes àquele ramo da árvore. A figura abaixo exemplifica.

Disco 1



Disco 2



Se o *file system* do disco dois (que pode ser físico ou uma partição) for montado no subdiretório **Mntpt** do disco 1, teremos que o arquivo **a.c** terá *path* absoluto

/z/Mntpt/tt/a.c

O diretório **Mntpt** deve ser um diretório vazio.

Os dispositivos de E/S também são montados no *file system*. Cada dispositivo é visto no UNIX como se fosse um arquivo. Para fazer qualquer operação de entrada/saída, basta ler/escrever do/no arquivo daquele dispositivo.