

Sistemas Operacionais

Sistemas de Arquivos

Implementação de Sistemas de Arquivos

Sistema de Arquivos

- ◆ Gerência de Espaço Livre
 - ❖ Mapa de Bits (Vetor de Bits)
 - ❖ Lista Ligada Simples
 - ❖ Lista Ligada com Agrupamento
 - ❖ Tabela de Blocos Livres Agrupados
- ◆ Alocação de Espaço em Disco
 - ❖ Alocação Contígua
 - ❖ Alocação Encadeada (Lista Ligada)
 - ❖ Lista Ligada usando Índice
 - ❖ Alocação Indexada (Nó-I)
- ◆ Desempenho dos Sistemas de Arquivos
- ◆ Implementações de Diretórios

Gerência de Espaço Livre

- ◆ São possíveis duas estratégias genéricas para armazenamento de arquivos:
 - ❖ alocação de n bytes consecutivos no disco
 - 📄 apresenta um problema óbvio quando o arquivo cresce...
 - ❖ alocação de blocos não necessariamente contíguos
- ◆ Tamanho do Bloco:
 - ❖ Candidatos a serem a unidade de alocação: o tamanho do setor, da trilha, do cilindro e da página (se SO paginado)
 - ❖ Unid. de alocação grande: provável desperdício de memória
 - ❖ Unid. de alocação pequena: baixa taxa de transferência
- ◆ Para poder criar arquivos, o SO precisa saber se há blocos livres e quais são → lista de espaço livre

Mapa de Bits (Vetor de Bits)

- ◆ Cada entrada da tabela aponta para um bloco
- ◆ Define-se: $\begin{cases} \text{Livre} = 1 \\ \text{ocupado} = 0 \end{cases}$
- ◆ Simples
- ◆ Consome muito espaço de memória: $\begin{cases} \text{Disco: 1,3 Gbytes} \\ \text{Bloco: 512 bytes} \\ \text{Bitmap: 333k} \end{cases}$
- ◆ Exemplo:
2, 3, 4, 5,
8, 9, 10, 11, 12, 13,
17, 18,
25, 26 e 27 livres

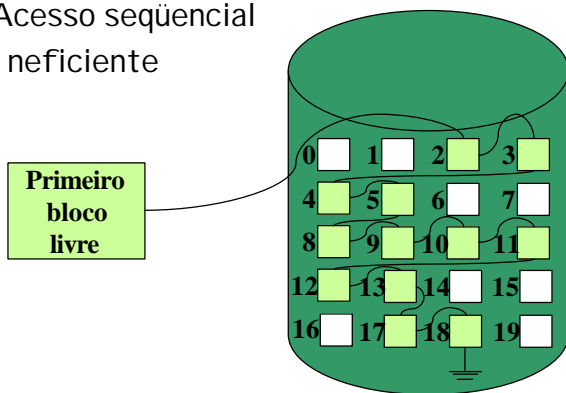


00111100
11111100
01100000
01110000

Usado quando
se tem espaço
na memória
principal para
todo o vetor

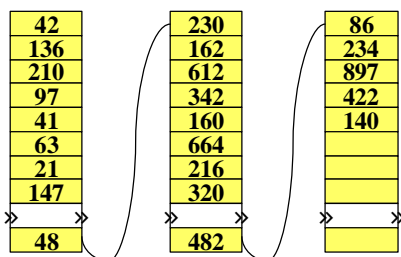
Lista Ligada Simples

- ◆ Cada bloco livre aponta para o próximo livre
- ◆ Acesso seqüencial
- ◆ Ineficiente



Lista Ligada com Agrupamento

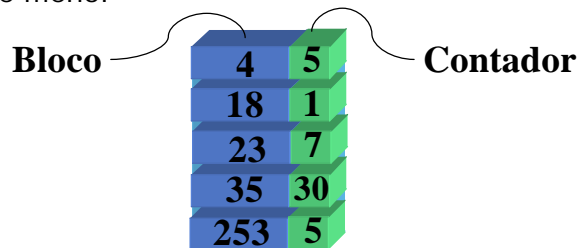
- ◆ Os endereços dos n primeiros blocos livres são armazenados no primeiro bloco. Os primeiros $n-1$ endereços estão disponíveis e o último endereço aponta para outro bloco com endereços livres



Ex.: Blocos de 1K
 End. do bloco: 16 bits
 → cada bloco guarda 511 blocos livres
 Disco de 20M
 20K blocos
 → precisa, no máximo, de uma lista ligada de 40 blocos

Tabela de Blocos Livres (Contagem)

- ◆ Geralmente blocos contíguos são alocados ou liberados simultaneamente
- ◆ Encontrar um número grande de blocos livres é mais rápido
- ◆ Tabela é menor



Métodos de Alocação de Espaço em Disco

- ◆ A natureza do acesso direto aos discos permite flexibilidade na implementação de arquivos
- ◆ A questão principal do projeto de um Sistema de Arquivos é:

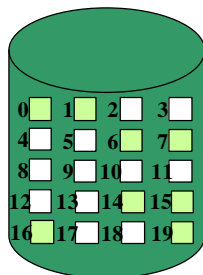
Como alocar espaço aos arquivos de modo que o uso do espaço em disco seja eficaz e que o acesso aos dados seja rápido?

Alocação Contígua

- ◆ Cada arquivo deve ocupar um conjunto de blocos contíguo no disco
- ◆ Vantagens:
 - ❖ Simples de implementar → guardar apenas o end. do 1º bloco
 - ❖ Acessos seqüencial e direto facilitados
 - ❖ Excelente performance → leitura em uma única operação
- ◆ Dificuldade: achar espaço para novo arquivo → seqüência de blocos livres igual ou maior que o arquivo
- ◆ A alocação de blocos em uma área contígua pode ser feita através de algoritmos de estratégia de alocação como o *First-Fit*, *Best-Fit* e *Worst-Fit*.

Alocação Contígua

- ◆ Desvantagens:
 - ❖ Necessário conhecer o tamanho dos arquivos no instante de sua criação (se reservar pouco → necessidade de cópia, se reservar muito → desperdício)
 - ❖ Fragmentação do disco → compactação é cara.

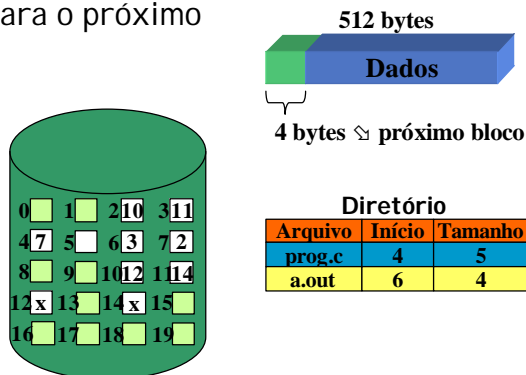


Diretório

Arquivo	Início	Tamanho
prog.c	0	2
a.out	14	3
leiamc	19	1
arg.txt	6	2

Alocação com Lista Ligada

- ◆ Espaço alocado é mantido através de uma lista ligada de blocos → uma parte do bloco guarda a referência para o próximo



Para criar um arquivo:

Início (1º bloco) = -1

Tamanho = 0

Para escrever:

Aloca-se um bloco

etc

Alocação com Lista Ligada

- ◆ Vantagens:
 - ❖ Não existe fragmentação
 - ❖ Entrada do diretório tb armazena apenas o end. do 1º bloco
- ◆ Desvantagens:
 - ❖ Acesso randômico (mais lento)
 - ❖ O acesso direto é feito sequencialmente
 - ❖ Tamanho útil do bloco (\neq potência de 2)
 - ❖ Espaço perdido com os ponteiros para próximo bloco → solução: *clusters*
 - ❖ Confiabilidade: erro em qualquer um dos blocos do arquivo torna impossível recuperação do restante → solução parcial: lista duplamente ligada

Alocação com Lista Ligada usando um Índice

- ◆ Similar ao método anterior, colocando-se os ponteiros que encadeiam os blocos em tabela ou índice na memória
- ◆ A tabela fica no início de cada partição e é usada como uma lista ligada

➔ FAT (*File Allocation Table*)

Entrada no diretório

prog.c	...	4
a.out	...	6
nome		bloco inicial

0	0
1	0
2	10
3	11
4	7
5	0
6	3
7	2
8	0
9	0
10	12
11	14
12	eof
13	0
14	eof

} FAT

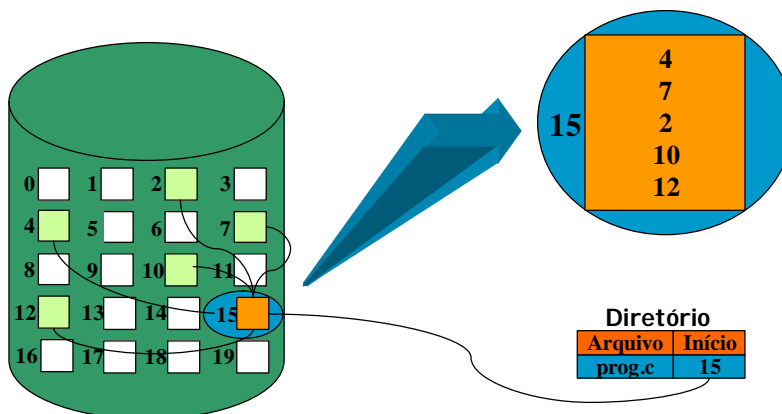
Alocação com Lista Ligada usando um Índice

- ◆ Vantagens:
 - ❖ Bloco fica livre para armazenamento apenas da informação
 - ❖ Apesar de randômico, o acesso é baseado na cadeia que está inteiramente na memória principal (basta realizar a busca na FAT)
 - ❖ Entrada no diretório precisa conter apenas o número do bloco inicial
- ◆ Desvantagem:
 - ❖ Tabela deve permanecer na memória durante todo o tempo
 - 📁 Disco de 500.000 blocos } 500 MBytes
 - 📁 Blocos de 1K
 - 📁 Tabela com 500.000 entradas vezes 4 bytes = 2 MBytes

Alocação Indexada (Nó-I)

- ◆ Assim como lista ligada, resolve problemas relativos à:
 - ❖ tamanho do arquivo
 - ❖ fragmentação externa
 - ❖ suporte a acesso direto → alocação encadeada
- ◆ ... além do problema da grande tabela em memória...
- ◆ Consiste em associar a cada arquivo uma pequena tabela de índices (nó-i), que lista seus atributos e endereços em disco (um vetor de endereços)
- ◆ O diretório contém o endereço da tabela de índices
- ◆ Permite acesso direto ao n-ésimo bloco
 - ❖ é só ler a n-ésima entrada do nó-i.

Alocação Indexada (Nó-I)



Alocação Indexada (Nó-I)

- ◆ Desvantagem: Se o arquivo for pequeno, gasta-se um bloco por índice
- ◆ Solução: índices encadeados (vários níveis)
 - ❖ Os primeiros endereços são armazenados no próprio nó-i → arquivos pequenos
 - ❖ Um dos endereços apontados pelo nó-i é o endereço do **bloco indireto simples**, que aponta para endereços do arquivo no disco
 - ❖ Outro apontador do nó-i é para o **bloco indireto duplo**, que contém uma lista de blocos de endereço simples
 - ❖ Existe ainda o **bloco indireto triplo**

Desempenho dos Sistemas de Arquivos

- ◆ *Cache* de bloco ou *buffer cache*: mantém-se um conjunto de blocos do disco na memória principal para reduzir o tempo de acesso
- ◆ Verificar se o bloco está na memória, se não estiver, ler o bloco para memória
- ◆ Se houver a necessidade de remover algum bloco → algoritmo semelhante aos de troca de página
- ◆ E se houver pane (falta de energia) ?
 - ❖ salvar periodicamente
 - ❖ *caches write-through*

Implementação de Diretórios

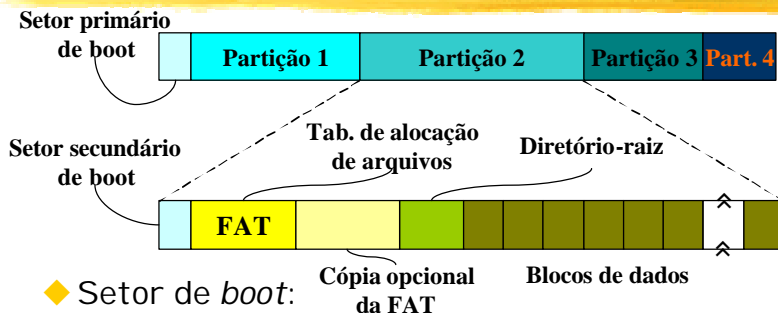
- ◆ Para abrir um arquivo o SO usa o nome do caminho para localizar sua entrada no diretório
- ◆ Dependendo do sistema, tal entrada fornece informações do tipo:
 - ❖ endereço do bloco (alocação contígua)
 - ❖ número do primeiro bloco (alocação com listas ligadas)
 - ❖ número do nó-i relativo ao arquivo (alocação indexada)
- ◆ Outro aspecto a ser considerado é onde os atributos dos arquivos devem estar armazenados: diretamente na entrada do diretório ou nos nós-i, no caso dos sistemas que usam alocação indexada

Diretórios no CP/M



- ◆ Só existe um diretório
- ◆ Campos da entrada do diretório:
 - ❖ código de usuário: identificação do usuário → são verificadas apenas as entradas pertencentes àquele usuário
 - ❖ nome e tipo do arquivo
 - ❖ extensão: informa qual das entradas usadas vem em primeiro lugar (para arquivos que ocupam mais de 16 blocos)
 - ❖ contador de blocos: nº de blocos em uso

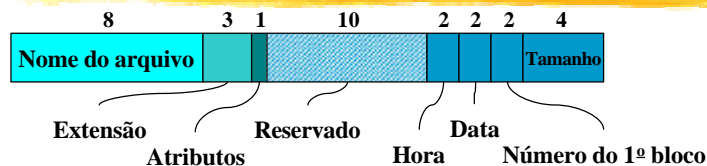
Diretórios no MS-DOS



◆ Setor de *boot*:

- ❖ começa com um desvio incondicional para código de *boot*
- ❖ lista de parâmetros: nº de bytes por setor, nº de setor por bloco, nº de arquivo na tabela de alocação, tamanho do diretório raiz, ...
- ❖ Tabela de partição (máximo 4): uma deve ser ativa

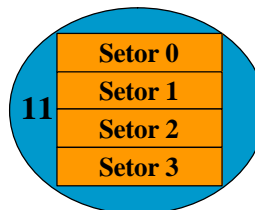
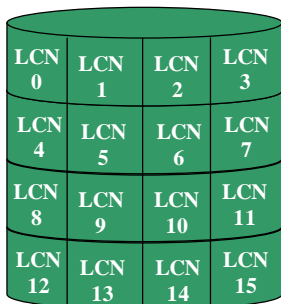
Diretórios no MS-DOS



- ◆ Nome e extensão: o ponto não é armazenado
- ◆ Atributos: contém os seguintes bits
 - ❖ A - 1 quando arquivo é modificado
 - ❖ D - 1 quando for diretório
 - ❖ V - 1 quando for nome do volume
 - ❖ S - 1 arquivo do sistema
 - ❖ H - 1 arquivo escondido
 - ❖ R - 1 arquivo não pode ser escrito
- ◆ Não se pode ter *link* → não se pode ter 2 entradas com mesmo número inicial de bloco

Diretórios no Windows 2000/NT

Logical Cluster Number



- ❖ Nomes de arquivos com até 255 caracteres
- ❖ Partições dispensam ferramentas de recuperação de erros
- ❖ Proteção por grupos e ACLs
- ❖ Criptografia e compressão de arquivos
- ❖ Suporte a volumes de até 2^{64} bytes
- ❖ Ferramentas de desfrag. e gerência de quotas em disco

Diretórios no Windows 2000/NT

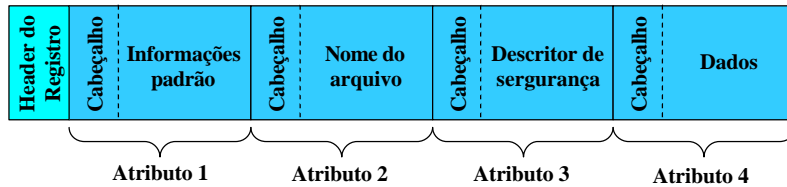
Master File Table

0	MasterFile Table	Arquivos de Metadados
1	Cópia do MFT	
2	Arquivo de Log	
3	Volume	
4	Atributos	
5	Diretório Raiz	
6	Arquivo Bitmap	
7	Arquivo de Boot	
8	Arquivo de Clusters Ruins	Arquivos e Diretórios de Usuários
	...	
16		

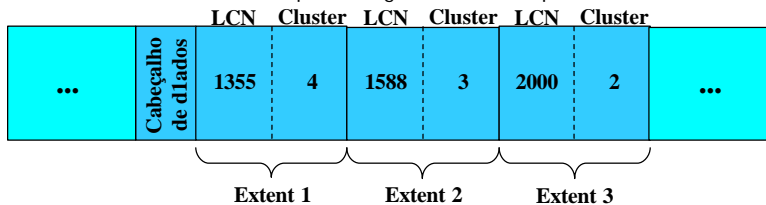
- ❖ MFT é formado por diversos registros de 1K
- ❖ Arquivos de metadados: utilizados para mapear os arquivos de controle do S.A.
- ❖ Registros apresentam formatos diferentes:
 - header
 - 1 ou mais atributos (13 tipos)
 - Cabeçalho
 - Valor
- ❖ Arquivos pequenos tem atributos e dados no próprio registro na MFT
- ❖ Arquivos maiores são formados por *extents* (conjuntos de clusters contíguos)

Diretórios no Windows 2000/NT

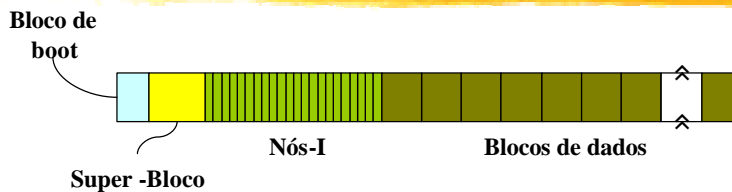
Exemplo de registro para um pequeno arquivo



Exemplo de registro de um arquivo

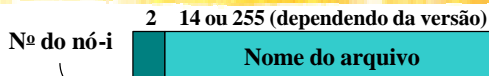


Diretórios no UNIX



- ◆ Bloco de *boot*: não é usado pelo UNIX e muitas vezes contém o código de *boot*
- ◆ Super-bloco: nº de nós-i, nº de blocos e início da lista de blocos livres (crítico para o sistema)
- ◆ Nós-i: atrib. do arquivo mais a localização dos blocos
- ◆ Blocos de dados: armazenam arquivos e diretórios

Diretórios no UNIX



- ◆ Os atributos não estão armazenados na entrada do diretório, mas no nó-i

◆ Ex.: /usr/enfz/mbox

Diretório raiz

1	.
1	..
4	bin
7	dev
14	Arq.txt
9	etc
6	usr
8	tmp

Procura usr
Obtém nó-i 6

Nó-i 6

Modo
Tamanho
Tempos
132

usr está no
bloco 132

Bloco 132: /usr

6	.
1	..
19	ast
30	enfz
51	lff
26	lgsj
45	rcmp

Procura /usr/enfz
Obtém nó-i 30

Nó-i 30

Modo
Tamanho
Tempos
406

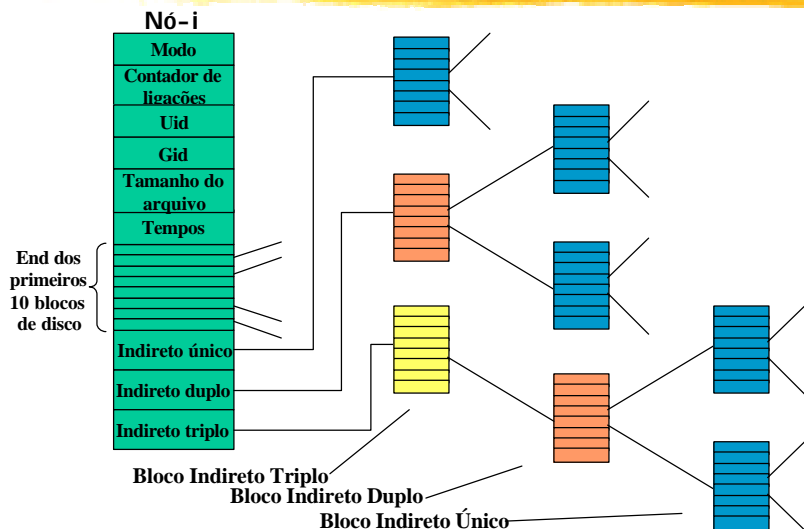
/usr/enfz está
no bloco 406

Bl. 406: /usr/enfz

30	.
6	..
64	Prog1.c
92	books
60	mbox
81	minix
17	src

Procura /usr/enfz/mbox
Obtém nó-i 60

Diretórios no UNIX



Diretórios no UNIX

Montagem do Sistema de Arquivos

