- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1300 bytes
 Cria arquivo B com 200 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 300 bytes
 Expande ou Cria arquivo A em/com 200 bytes
 Expande ou Cria arquivo A em/com 700 bytes
 Expande ou Cria arquivo D em/com 300 bytes
 Apaga arquivo D
 Expande ou Cria arquivo C em/com 900 bytes
 Expande ou Cria arquivo C em/com 200 bytes
 Expande ou Cria arquivo A em/com 1000 bytes
 Expande ou Cria arquivo B em/com 1100 bytes
 Cria arquivo G com 1200 bytes
 Apaga arquivo A

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1400 bytes
 Cria arquivo B com 100 bytes
 Cria arquivo C com 200 bytes
 Cria arquivo D com 300 bytes
 Apaga arquivo A
 Apaga arquivo B
 Cria arquivo G com 400 bytes
 Cria arquivo H com 100 bytes
 Expande ou Cria arquivo A em/com 700 bytes
 Expande ou Cria arquivo D em/com 1000 bytes
 Expande ou Cria arquivo A em/com 200 bytes
 Cria arquivo E com 1000 bytes
 Expande ou Cria arquivo B em/com 1300 bytes
 Expande ou Cria arquivo A em/com 200 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterà 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 700 bytes
 Cria arquivo B com 300 bytes
 Cria arquivo C com 1000 bytes
 Cria arquivo D com 700 bytes
 Apaga arquivo D
 Cria arquivo F com 400 bytes
 Expande ou Cria arquivo C em/com 500 bytes
 Expande ou Cria arquivo A em/com 900 bytes
 Expande ou Cria arquivo A em/com 400 bytes
 Expande ou Cria arquivo B em/com 1100 bytes
 Expande ou Cria arquivo A em/com 1100 bytes
 Expande ou Cria arquivo C em/com 100 bytes
 Apaga arquivo B
 Apaga arquivo A

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar no próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1000 bytes
 Cria arquivo B com 800 bytes
 Cria arquivo C com 1400 bytes
 Cria arquivo D com 400 bytes
 Expande ou Cria arquivo A em/com 1100 bytes
 Expande ou Cria arquivo A em/com 300 bytes
 Expande ou Cria arquivo B em/com 600 bytes
 Cria arquivo E com 1000 bytes
 Expande ou Cria arquivo C em/com 300 bytes
 Cria arquivo F com 1300 bytes
 Apaga arquivo C
 Apaga arquivo D
 Expande ou Cria arquivo B em/com 200 bytes
 Cria arquivo H com 1200 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar no próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 500 bytes
 Cria arquivo B com 1400 bytes
 Cria arquivo C com 400 bytes
 Cria arquivo D com 800 bytes
 Expande ou Cria arquivo D em/com 1000 bytes
 Cria arquivo E com 500 bytes
 Apaga arquivo A
 Apaga arquivo D
 Expande ou Cria arquivo A em/com 700 bytes
 Expande ou Cria arquivo B em/com 300 bytes
 Expande ou Cria arquivo A em/com 200 bytes
 Apaga arquivo C
 Expande ou Cria arquivo A em/com 800 bytes
 Expande ou Cria arquivo A em/com 200 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar no próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl.inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 200 bytes
 Cria arquivo B com 600 bytes
 Cria arquivo C com 500 bytes
 Cria arquivo D com 1200 bytes
 Expande ou Cria arquivo B em/com 1200 bytes
 Apaga arquivo D
 Cria arquivo F com 700 bytes
 Cria arquivo H com 1300 bytes
 Expande ou Cria arquivo A em/com 200 bytes
 Expande ou Cria arquivo D em/com 400 bytes
 Cria arquivo E com 600 bytes
 Expande ou Cria arquivo C em/com 500 bytes
 Expande ou Cria arquivo A em/com 1100 bytes
 Expande ou Cria arquivo B em/com 500 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 600 bytes
 Cria arquivo B com 800 bytes
 Cria arquivo C com 1200 bytes
 Cria arquivo D com 300 bytes
 Expande ou Cria arquivo A em/com 100 bytes
 Expande ou Cria arquivo B em/com 500 bytes
 Expande ou Cria arquivo D em/com 100 bytes
 Cria arquivo G com 1300 bytes
 Apaga arquivo A
 Cria arquivo E com 200 bytes
 Apaga arquivo B
 Expande ou Cria arquivo C em/com 200 bytes
 Expande ou Cria arquivo B em/com 300 bytes
 Cria arquivo F com 1100 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl.inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 700 bytes
 Cria arquivo B com 900 bytes
 Cria arquivo C com 1400 bytes
 Cria arquivo D com 800 bytes
 Apaga arquivo A
 Expande ou Cria arquivo D em/com 1200 bytes
 Apaga arquivo B
 Apaga arquivo D
 Apaga arquivo C
 Cria arquivo F com 1000 bytes
 Cria arquivo E com 1100 bytes
 Expande ou Cria arquivo C em/com 300 bytes
 Expande ou Cria arquivo A em/com 100 bytes
 Cria arquivo H com 400 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começamos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 200 bytes
 Cria arquivo B com 500 bytes
 Cria arquivo C com 600 bytes
 Cria arquivo D com 900 bytes
 Expande ou Cria arquivo A em/com 100 bytes
 Apaga arquivo A
 Expande ou Cria arquivo D em/com 1100 bytes
 Expande ou Cria arquivo C em/com 1400 bytes
 Cria arquivo G com 400 bytes
 Expande ou Cria arquivo B em/com 200 bytes
 Cria arquivo E com 1000 bytes
 Expande ou Cria arquivo A em/com 500 bytes
 Expande ou Cria arquivo A em/com 900 bytes
 Cria arquivo F com 200 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1400 bytes
 Cria arquivo B com 1400 bytes
 Cria arquivo C com 600 bytes
 Cria arquivo D com 1400 bytes
 Expande ou Cria arquivo D em/com 700 bytes
 Expande ou Cria arquivo B em/com 1400 bytes
 Expande ou Cria arquivo A em/com 1200 bytes
 Expande ou Cria arquivo C em/com 300 bytes
 Apaga arquivo B
 Cria arquivo G com 1300 bytes
 Expande ou Cria arquivo B em/com 300 bytes
 Cria arquivo H com 200 bytes
 Expande ou Cria arquivo A em/com 1200 bytes
 Expande ou Cria arquivo A em/com 1100 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1400 bytes
 Cria arquivo B com 700 bytes
 Cria arquivo C com 1400 bytes
 Cria arquivo D com 200 bytes
 Cria arquivo H com 600 bytes
 Cria arquivo G com 800 bytes
 Cria arquivo E com 1400 bytes
 Apaga arquivo A
 Expande ou Cria arquivo B em/com 200 bytes
 Expande ou Cria arquivo B em/com 800 bytes
 Expande ou Cria arquivo D em/com 900 bytes
 Expande ou Cria arquivo A em/com 700 bytes
 Expande ou Cria arquivo C em/com 1400 bytes
 Expande ou Cria arquivo C em/com 300 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 200 bytes
 Cria arquivo B com 600 bytes
 Cria arquivo C com 1300 bytes
 Cria arquivo D com 1200 bytes
 Apaga arquivo A
 Expande ou Cria arquivo C em/com 1400 bytes
 Cria arquivo F com 300 bytes
 Expande ou Cria arquivo B em/com 1200 bytes
 Apaga arquivo B
 Cria arquivo H com 900 bytes
 Expande ou Cria arquivo D em/com 1400 bytes
 Expande ou Cria arquivo B em/com 100 bytes
 Expande ou Cria arquivo A em/com 1200 bytes
 Apaga arquivo C

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1400 bytes
 Cria arquivo B com 900 bytes
 Cria arquivo C com 700 bytes
 Cria arquivo D com 900 bytes
 Expande ou Cria arquivo B em/com 700 bytes
 Expande ou Cria arquivo B em/com 1100 bytes
 Expande ou Cria arquivo A em/com 1000 bytes
 Apaga arquivo B
 Cria arquivo E com 900 bytes
 Apaga arquivo A
 Cria arquivo G com 900 bytes
 Expande ou Cria arquivo B em/com 500 bytes
 Expande ou Cria arquivo C em/com 700 bytes
 Apaga arquivo D

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterà 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 800 bytes
 Cria arquivo B com 1100 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 1300 bytes
 Cria arquivo F com 800 bytes
 Cria arquivo G com 400 bytes
 Expande ou Cria arquivo B em/com 300 bytes
 Apaga arquivo A
 Expande ou Cria arquivo D em/com 800 bytes
 Expande ou Cria arquivo A em/com 1300 bytes
 Expande ou Cria arquivo B em/com 200 bytes
 Cria arquivo H com 600 bytes
 Expande ou Cria arquivo B em/com 200 bytes
 Apaga arquivo D

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 700 bytes
 Cria arquivo B com 200 bytes
 Cria arquivo C com 900 bytes
 Cria arquivo D com 800 bytes
 Apaga arquivo C
 Cria arquivo G com 900 bytes
 Expande ou Cria arquivo B em/com 1100 bytes
 Expande ou Cria arquivo A em/com 1000 bytes
 Apaga arquivo B
 Expande ou Cria arquivo D em/com 300 bytes
 Expande ou Cria arquivo C em/com 800 bytes
 Expande ou Cria arquivo C em/com 700 bytes
 Cria arquivo E com 1100 bytes
 Cria arquivo H com 300 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 900 bytes
 Cria arquivo B com 1200 bytes
 Cria arquivo C com 1000 bytes
 Cria arquivo D com 800 bytes
 Expande ou Cria arquivo B em/com 900 bytes
 Apaga arquivo C
 Cria arquivo F com 1400 bytes
 Apaga arquivo B
 Expande ou Cria arquivo A em/com 1200 bytes
 Apaga arquivo D
 Expande ou Cria arquivo C em/com 100 bytes
 Apaga arquivo A
 Expande ou Cria arquivo C em/com 800 bytes
 Expande ou Cria arquivo A em/com 600 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl.inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1100 bytes
 Cria arquivo B com 600 bytes
 Cria arquivo C com 700 bytes
 Cria arquivo D com 1300 bytes
 Expande ou Cria arquivo A em/com 100 bytes
 Expande ou Cria arquivo A em/com 1100 bytes
 Expande ou Cria arquivo B em/com 500 bytes
 Expande ou Cria arquivo B em/com 1400 bytes
 Expande ou Cria arquivo A em/com 1300 bytes
 Cria arquivo E com 1200 bytes
 Cria arquivo H com 1300 bytes
 Apaga arquivo D
 Expande ou Cria arquivo C em/com 300 bytes
 Expande ou Cria arquivo D em/com 1200 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1000 bytes
 Cria arquivo B com 600 bytes
 Cria arquivo C com 400 bytes
 Cria arquivo D com 800 bytes
 Cria arquivo E com 1100 bytes
 Expande ou Cria arquivo B em/com 1000 bytes
 Cria arquivo F com 600 bytes
 Expande ou Cria arquivo A em/com 700 bytes
 Apaga arquivo C
 Expande ou Cria arquivo A em/com 700 bytes
 Expande ou Cria arquivo A em/com 600 bytes
 Apaga arquivo D
 Expande ou Cria arquivo B em/com 800 bytes
 Expande ou Cria arquivo A em/com 100 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 800 bytes
 Cria arquivo B com 900 bytes
 Cria arquivo C com 1000 bytes
 Cria arquivo D com 1100 bytes
 Expande ou Cria arquivo B em/com 400 bytes
 Cria arquivo E com 1300 bytes
 Expande ou Cria arquivo B em/com 600 bytes
 Expande ou Cria arquivo A em/com 300 bytes
 Apaga arquivo A
 Expande ou Cria arquivo C em/com 300 bytes
 Apaga arquivo B
 Expande ou Cria arquivo A em/com 400 bytes
 Expande ou Cria arquivo B em/com 300 bytes
 Cria arquivo H com 1100 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar no próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1400 bytes
 Cria arquivo B com 800 bytes
 Cria arquivo C com 600 bytes
 Cria arquivo D com 1300 bytes
 Expande ou Cria arquivo A em/com 400 bytes
 Expande ou Cria arquivo B em/com 600 bytes
 Apaga arquivo D
 Cria arquivo H com 600 bytes
 Expande ou Cria arquivo A em/com 300 bytes
 Apaga arquivo B
 Expande ou Cria arquivo A em/com 300 bytes
 Expande ou Cria arquivo C em/com 400 bytes
 Cria arquivo E com 1000 bytes
 Cria arquivo F com 700 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1100 bytes
 Cria arquivo B com 700 bytes
 Cria arquivo C com 600 bytes
 Cria arquivo D com 100 bytes
 Expande ou Cria arquivo B em/com 600 bytes
 Expande ou Cria arquivo B em/com 700 bytes
 Expande ou Cria arquivo A em/com 1200 bytes
 Expande ou Cria arquivo D em/com 600 bytes
 Expande ou Cria arquivo C em/com 300 bytes
 Expande ou Cria arquivo A em/com 900 bytes
 Cria arquivo G com 1200 bytes
 Expande ou Cria arquivo A em/com 300 bytes
 Cria arquivo E com 1100 bytes
 Apaga arquivo D

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 500 bytes
 Cria arquivo B com 400 bytes
 Cria arquivo C com 400 bytes
 Cria arquivo D com 700 bytes
 Apaga arquivo C
 Expande ou Cria arquivo C em/com 1200 bytes
 Cria arquivo E com 1200 bytes
 Cria arquivo G com 800 bytes
 Expande ou Cria arquivo A em/com 1300 bytes
 Expande ou Cria arquivo B em/com 100 bytes
 Expande ou Cria arquivo A em/com 200 bytes
 Expande ou Cria arquivo B em/com 900 bytes
 Apaga arquivo B
 Expande ou Cria arquivo B em/com 1200 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl.inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1000 bytes
 Cria arquivo B com 100 bytes
 Cria arquivo C com 100 bytes
 Cria arquivo D com 200 bytes
 Apaga arquivo B
 Cria arquivo G com 100 bytes
 Apaga arquivo A
 Expande ou Cria arquivo A em/com 900 bytes
 Expande ou Cria arquivo A em/com 600 bytes
 Expande ou Cria arquivo B em/com 300 bytes
 Expande ou Cria arquivo C em/com 800 bytes
 Cria arquivo H com 500 bytes
 Cria arquivo E com 500 bytes
 Expande ou Cria arquivo C em/com 1100 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1000 bytes
 Cria arquivo B com 1300 bytes
 Cria arquivo C com 500 bytes
 Cria arquivo D com 100 bytes
 Apaga arquivo A
 Cria arquivo F com 900 bytes
 Expande ou Cria arquivo B em/com 500 bytes
 Expande ou Cria arquivo D em/com 400 bytes
 Cria arquivo E com 400 bytes
 Expande ou Cria arquivo C em/com 1300 bytes
 Apaga arquivo C
 Apaga arquivo B
 Cria arquivo H com 900 bytes
 Expande ou Cria arquivo B em/com 600 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar no próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1400 bytes
 Cria arquivo B com 200 bytes
 Cria arquivo C com 500 bytes
 Cria arquivo D com 500 bytes
 Expande ou Cria arquivo D em/com 900 bytes
 Cria arquivo F com 1000 bytes
 Cria arquivo H com 1400 bytes
 Expande ou Cria arquivo B em/com 600 bytes
 Expande ou Cria arquivo C em/com 1200 bytes
 Cria arquivo G com 1400 bytes
 Apaga arquivo C
 Apaga arquivo D
 Expande ou Cria arquivo A em/com 900 bytes
 Expande ou Cria arquivo A em/com 900 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 600 bytes
 Cria arquivo B com 300 bytes
 Cria arquivo C com 100 bytes
 Cria arquivo D com 400 bytes
 Expande ou Cria arquivo B em/com 1300 bytes
 Apaga arquivo A
 Expande ou Cria arquivo C em/com 300 bytes
 Expande ou Cria arquivo A em/com 1100 bytes
 Expande ou Cria arquivo B em/com 1300 bytes
 Expande ou Cria arquivo B em/com 1100 bytes
 Apaga arquivo D
 Apaga arquivo C
 Expande ou Cria arquivo A em/com 1300 bytes
 Cria arquivo G com 800 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar no próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS, ...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 400 bytes
 Cria arquivo B com 700 bytes
 Cria arquivo C com 400 bytes
 Cria arquivo D com 1200 bytes
 Expande ou Cria arquivo B em/com 300 bytes
 Expande ou Cria arquivo D em/com 800 bytes
 Expande ou Cria arquivo C em/com 500 bytes
 Cria arquivo E com 700 bytes
 Cria arquivo G com 200 bytes
 Apaga arquivo A
 Expande ou Cria arquivo B em/com 600 bytes
 Expande ou Cria arquivo B em/com 500 bytes
 Expande ou Cria arquivo A em/com 1400 bytes
 Apaga arquivo D

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 1100 bytes
 Cria arquivo B com 1300 bytes
 Cria arquivo C com 900 bytes
 Cria arquivo D com 200 bytes
 Expande ou Cria arquivo D em/com 900 bytes
 Apaga arquivo C
 Cria arquivo G com 1100 bytes
 Expande ou Cria arquivo C em/com 900 bytes
 Expande ou Cria arquivo B em/com 1100 bytes
 Expande ou Cria arquivo A em/com 1400 bytes
 Apaga arquivo B
 Expande ou Cria arquivo B em/com 900 bytes
 Expande ou Cria arquivo A em/com 900 bytes
 Cria arquivo H com 900 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encadeada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 600 bytes
 Cria arquivo B com 1400 bytes
 Cria arquivo C com 1300 bytes
 Cria arquivo D com 1100 bytes
 Apaga arquivo C
 Apaga arquivo A
 Cria arquivo G com 100 bytes
 Expande ou Cria arquivo B em/com 100 bytes
 Apaga arquivo D
 Expande ou Cria arquivo C em/com 1100 bytes
 Cria arquivo H com 1400 bytes
 Cria arquivo E com 1100 bytes
 Expande ou Cria arquivo A em/com 500 bytes
 Expande ou Cria arquivo D em/com 1300 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4
---	---	---	---

- Divide a área total em cluster (em português: grânulos) de tamanho fixo, que passarão a ser daqui para a frente as unidades de alocação.
- Constrói a infraestrutura de gerenciamento dos clusters. A propósito os clusters variam de tamanho entre 1 e 16KB).

Agora, sistema operacional e a memória secundária trabalham juntos para receber e atender solicitações de leitura e gravação de blocos ou setores (atenção: a unidade de alocação é o cluster, a unidade de transferência é o setor (ou bloco) e a unidade de armazenamento ainda é o byte. O bloco pode ser de 512 bytes (HDs tradicionais, 4K para HDs mais modernos e SSDs – pendrives, ou ainda mais do que isso). A escolha destes tamanhos leva em consideração que clusters e blocos maiores agilizam as transferências, mas tendem a desperdiçar partes não usadas.

DOS e depois Windows

O mecanismo de controle das alocações no disco/pendrive foi chamado FAT (*File Allocation Table*). Trata-se de uma tabela que controla onde um arquivo reside no disco. O tamanho original do elemento desta tabela era de 16 bits, mas hoje em dia é de no mínimo 32 bits (daí seu nome se converte em FAT32) podendo chegar a 64 bits no sistema exFAT. Tudo para se adaptar a discos/pendrives/SSDs cada vez maiores.

Embora não vejamos aqui, há todo um processamento para isolar clusters defeituosos, já que não tem sentido, num disco de milhões de clusters, desprezar o disco se apenas alguns poucos apresentarem algum defeito.

Embora cada encarnação desta tecnologia tenha adicionado camadas de proteção, segurança e aumento de desempenho, vamos descrever a idéia original, até porque, ainda presente.

Em área reservada do disco/pendrive há uma tabela de arquivos aí residentes. Esta tabela é rica, cheia de detalhes, mas o que nos interessa aqui:

- nome** do arquivo, dado pelo seu criador
- tamanho atual** quantidade de bytes que o arquivo realmente tem
- cluster inicial** o endereço do cluster onde o arquivo COMEÇA
- outros** direitos, data e hora de criação e atualização, etc.

Para entender melhor, vamos a uma simulação. Suponha que mandei o S.O. criar um arquivo chamado AAAA. O SO buscou um cluster livre (consultando a FAT) e me devolveu seu endereço, digamos 133. Comecei a usá-lo. Se o arquivo tiver um comprimento menor ou igual ao do cluster, acabou a intervenção da FAT e este fato ficará marcado com uma indicação de terminador (por hipótese -1) no item 133 da FAT.

Em resumo, na tabela de arquivos diz: AAAA começa no cluster 133 e na posição 133 da FAT aparece -1, indicando fim-de-arquivo.

As coisas começam a ficar interessantes se o arquivo AAAA for maior do que 1 cluster. Quando o cluster 133 ficar cheio, haverá uma requisição ao SO pedindo um novo cluster. Suponhamos que há poucos milésimos de segundo um cluster foi liberado no início do disco. Por hipótese o cluster 21 está livre. O SO decide que o arquivo que começou no 133 agora seguirá no 21. E, faz isso. Agora, a posição 133 da FAT conterá 21, e a posição 21 terá o terminador -1. Como se viu, é obrigatório alocar os clusters vazios de menor número (no início do disco).

Começa a ser construída uma estrutura encaixada como se descreveu no início, com a diferença de que o endereço do próximo não está agregado ao elemento e sim em uma tabela separada (a FAT no caso).

Uma simulação Seja a seguinte tabela de arquivos, com tamanhos de bloco de 100 bytes.

nome	tam.	cl. inic.
AAAA	150	3
BBBB	80	5
CCCC	400	9
DDDD	102	10

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
2	7	8	-1	-1	-1	-1	1	6	

Agora a área do disco onde os dados são gravados

1	2	3	4	5
C	C	A		B
6	7	8	9	10
D	C	A	C	D

Exemplo

Neste exemplo de FAT considere o cluster tendo 500 bytes.

Cria arquivo A com 900 bytes
 Cria arquivo B com 1000 bytes
 Cria arquivo C com 800 bytes
 Cria arquivo D com 200 bytes
 Expande o arquivo A em 800 bytes
 Expande o arquivo B em 700 bytes
 Expande o arquivo B em 100 bytes
 Expande o arquivo A em 1400 bytes
 Cria arquivo E com 800 bytes
 Apaga arquivo A
 Cria o arquivo A com 900 bytes
 Expande o arquivo D em 700 bytes
 Cria arquivo G com 800 bytes
 Expande o arquivo C em 900 bytes

Eis como ficará a FAT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	-1	4	10	6	13	8	-1	12	11	-1	-1	14	-1	16	-1

Se quiser refazer o exemplo, eis como ficará a tabela de arquivos AO FINAL do exemplo:

Arquivo	Tamanho	Cl. inic.	Clusters
B	1800	3	3,4,10,11
C	1700	5	5,6,13,14
D	900	7	7,8
E	800	15	15,16
A	900	1	1,2
G	800	9	9,12

Para você fazer

Suponha um sistema com cluster igual a 500 bytes. Eis as ordens a executar no disco:

Cria arquivo A com 300 bytes
 Cria arquivo B com 1200 bytes
 Cria arquivo C com 1100 bytes
 Cria arquivo D com 1200 bytes
 Cria arquivo E com 1100 bytes
 Expande ou Cria arquivo A em/com 1000 bytes
 Cria arquivo G com 400 bytes
 Expande ou Cria arquivo D em/com 700 bytes
 Apaga arquivo C
 Expande ou Cria arquivo A em/com 400 bytes
 Cria arquivo H com 700 bytes
 Expande ou Cria arquivo A em/com 1100 bytes
 Apaga arquivo B
 Expande ou Cria arquivo B em/com 500 bytes

Tabela de arquivos:

nome	tam.	cl. inic.

Agora, a FAT

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Agora a área do disco onde os dados são gravados

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

As 4 primeiras posições da FAT:

1	2	3	4

Alocação sequencial e encadeada

Imagine a memória do computador como sendo um enorme armário de gavetas. Cada gaveta corresponde a um byte e pode conter 256 valores (binários) diferentes. Cada gaveta tem um nome (um endereço) e por meio dele que se examina o que tem lá (leitura) ou se modifica esse conteúdo (gravação). Cada vez que você (seu programa) precisa armazenar algum dado ele pede memória ao sistema operacional. Há duas maneiras desse pedido ser feito: memória sequencial e memória encadeada. Na alocação sequencial, os endereços pedidos estão contíguos. Na alocação encadeada não existe contiguidade.

Na alocação sequencial: 1. Todos os dados estão juntos. Podem ser processados sequencialmente de maneira muito fácil. Podem também ser acessados individualmente muito rápido. 2. Não é fácil crescer a estrutura, já que o espaço ao final dela pode estar sendo já usado por outrém. Neste caso, todos os dados precisam ser copiados. 3. Inclusões e exclusões são mais demoradas, pois há necessidade de efetuar deslocamentos dos dados remanescentes. 4. Exemplos de alocação sequencial: uma fita magnética, um vetor ou matriz (genericamente chamados de tensores), listas, filas, arquivos sequenciais (do ponto de vista dos programas)...

Na alocação encadeada: 1. Os dados estão em qualquer lugar da memória. Para chegar ao próximo, o anterior precisa conter um apontador para o próximo. Isto é o encadeamento. 2. A estrutura pode crescer à vontade até o limite da memória. 3. Não é possível acessar um dado diretamente. A cadeia toda tem que ser percorrida. 4. Inclusões e exclusões (depois de localizado o local exato) são muito rápidas. 5. Exemplos de alocação encadeada: alocações na memória real, alocações de área em disco (do ponto de vista do sistema operacional), listas, filas, arquivos diretos, ...

A grande pergunta

Imagine que você é o sistema operacional (Windows, Linux, MacOS,...). Você tem um disco com talvez 1 TB de capacidade (1.024 GB ou 1.048.576 MB ou 1.073.741.824 KB ou 1.099.511.627.776 bytes), no qual há milhares de arquivos sendo criados, excluídos, aumentando ou diminuindo de tamanho de maneira dinâmica. É sua responsabilidade gravar e ler todos os dados recuperando erros, eventualmente comprimindo/descomprimindo dados, é uma trabalhadeira enorme da qual depende o sucesso do computador.

No ambiente de programação em camadas (modelo da cebola), se você é a camada do S.O., de um lado estão os programas aplicativos que recebem os dados de maneira provavelmente sequencial – imagine um arquivo .DOC sendo recebido pelo MS-WORD, ou um arquivo .HTML sendo recebido pelo browser da máquina – sem se preocupar onde os dados estão fisicamente gravados, e com todos os blocos de controle (verificação de erros, paridade, etc) retirados. Do outro lado, o S.O. enxerga o hardware (na verdade o controlador do hardware) que tem sob sua responsabilidade manter o fluxo de dados de maneira otimizada e tanto quanto possível ininterrupta. Cabe a você fazer as traduções de comandos/dados de um e de outro lado.

A pergunta acima citada é: como fazer para gerir uma área em disco enorme, dividida entre milhares de usuários (programas e processos) altamente dinâmica ou como se diz, em tempo-real ?

A formatação

Começemos do início: antes de usar qualquer dispositivo de memória secundária, há que formatá-lo. Trata-se de um processo que:

- Grava uma constante em todo o disco e faz a leitura imediata. O objetivo é garantir que o que foi gravado pode ser lido. Isto é: não há falhas na estrutura de gravação/leitura.