

Capítulo 21

Dispositivos de armazenamento

O armazenamento de dados é uma necessidade fundamental em sistemas de computação. Apesar de rápida, a memória RAM é relativamente pequena e perde seu conteúdo ao ficar sem energia, o que inviabiliza o armazenamento de grandes volumes de dados por longos períodos. Os dispositivos de armazenamento permitem guardar grandes volumes de dados de forma não-volátil, ou seja, preservando os dados mesmo na ausência de energia. Esses dados podem ser carregados na memória RAM e acessados pelo processador quando desejado, através dos mecanismos de entrada e saída apresentados nos capítulos precedentes.

Há diversas famílias de dispositivos de armazenamento de dados. No mundo da computação pessoal, os principais são os discos rígidos, os dispositivos de estado sólido (SSDs, pendrives, cartões SD) e os dispositivos óticos (CDs, DVDs). Este capítulo apresenta uma visão geral do funcionamento dos principais dispositivos de armazenamento: discos rígidos e dispositivos de estado sólido. São discutidos também aspectos de interface, escalonamento de entrada/saída e sistemas RAID.

21.1 Discos rígidos

Discos rígidos (HDD - *Hard-Disk Drives*) estão presentes na grande maioria dos computadores pessoais e servidores. Um disco rígido permite o armazenamento persistente (não-volátil) de grandes volumes de dados com baixo custo e tempos de acesso razoáveis. Além disso, a leitura e escrita de dados em um disco rígido é mais simples e flexível que em outros meios, como fitas magnéticas ou discos óticos (CDs, DVDs). Por essas razões, eles são intensivamente utilizados em computadores para o armazenamento de arquivos do sistema operacional, das aplicações e dos dados dos usuários. Os discos rígidos também são frequentemente usados como área de armazenamento de páginas em sistemas de paginação em disco (*swapping* e *paging*, vide Capítulo 17).

Esta seção apresenta alguns aspectos de hardware relacionados aos discos rígidos, como sua estrutura física e os principais padrões de interface entre o disco e sua controladora no computador. Em seguida, detalha aspectos de software que estão sob a responsabilidade direta do sistema operacional, como o escalonamento de operações de leitura/escrita no disco.

21.1.1 Estrutura física

Um disco rígido é composto por um ou mais discos metálicos que giram juntos em alta velocidade (usualmente entre 4.200 e 15.000 RPM), acionados por um motor elétrico. Para cada face de cada disco há uma cabeça de leitura móvel, responsável por ler e escrever dados através da magnetização de pequenas áreas da superfície metálica. Cada face é dividida logicamente em **trilhas** (ou **cilindros**) e **setores**; a interseção de uma trilha e um setor em uma face define um **bloco físico**¹, que é a unidade básica de armazenamento e transferência de dados no disco. Até 2010, os discos rígidos usavam blocos físicos de 512 bytes, mas o padrão da indústria migrou nos últimos anos para blocos de 4.096 bytes. A Figura 21.1 apresenta os principais elementos que compõem a estrutura de um disco rígido.

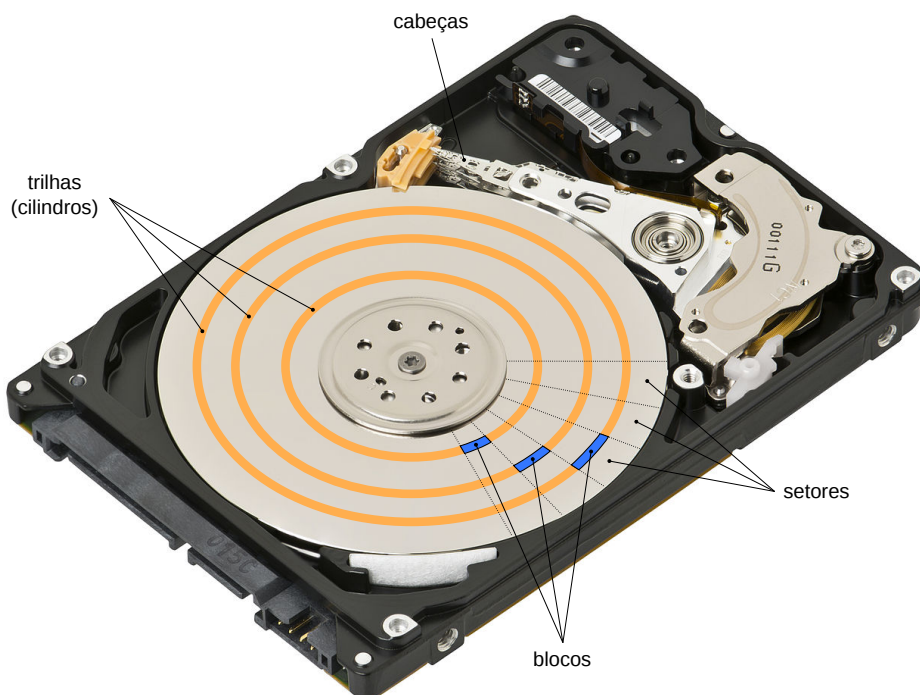


Figura 21.1: Elementos da estrutura de um disco rígido.

Um disco típico contém várias faces e milhares de trilhas e de setores por face [Patterson and Hennessy, 2005], resultando em milhões de blocos de dados disponíveis. Cada bloco pode ser individualmente acessado (lido ou escrito) através de seu *endereço*. Historicamente, o endereçamento dos blocos usava um padrão denominado CHS (*Cylinder-Head-Sector*): para acessar cada bloco, era necessário informar a cabeça (ou seja, a face), o cilindro (trilha) e o setor do disco onde se encontra o bloco. Esse sistema foi mais tarde substituído pelo padrão LBA (*Logical Block Addressing*), no qual os blocos são endereçados linearmente (0, 1, 2, 3, ...), o que é muito mais fácil de gerenciar pelo sistema operacional. Como a estrutura física do disco rígido continua a ter faces, trilhas e setores, uma conversão entre endereços LBA e CHS é feita pelo firmware do disco rígido, de forma transparente para o restante do sistema.

Por serem dispositivos eletromecânicos, os discos rígidos são extremamente lentos, se comparados à velocidade da memória ou do processador. Para cada bloco a

¹Alguns autores denominam essa intersecção como “setor de trilha” (*track sector*).

ser lido/escrito, a cabeça de leitura deve se posicionar na trilha desejada e aguardar o disco girar até encontrar o setor desejado. Esses dois passos definem o *tempo de busca* (t_s – *seek time*), que é o tempo necessário para a cabeça de leitura se posicionar sobre uma determinada trilha, e a *latência rotacional* (t_r – *rotation latency*), que é o tempo necessário para o disco girar até que o setor desejado esteja sob a cabeça de leitura. Valores médios típicos desses atrasos para discos de uso doméstico são $t_s \approx 10ms$ e $t_r \approx 5ms$. Juntos, esses dois atrasos podem ter um forte impacto no desempenho do acesso a disco.

21.1.2 Escalonamento de acessos

Em um sistema operacional multitarefas, várias aplicações e processos podem solicitar acessos ao disco simultaneamente, para escrita e leitura de dados. Devido à sua estrutura mecânica, um disco rígido só pode atender a uma requisição de acesso por vez, o que torna necessário manter uma fila de acessos pendentes. Cada nova requisição de acesso ao disco é colocada nessa fila e o processo solicitante é suspenso até seu pedido ser atendido. Sempre que o disco concluir um acesso, ele informa o sistema operacional, que deve buscar nessa fila a próxima requisição de acesso a ser atendida. A ordem de atendimento das requisições pendentes na fila de acesso ao disco é denominada **escalonamento de disco** e pode ter um grande impacto no desempenho do sistema operacional.

Na sequência do texto serão apresentados alguns algoritmos de escalonamento de disco clássicos. Para exemplificar seu funcionamento, será considerado um disco hipotético com 1.000 blocos (enumerados de 0 ao 999), cuja cabeça de leitura se encontra inicialmente sobre o bloco 500. A fila de pedidos de acesso pendentes contém pedidos de acesso aos seguintes blocos do disco, em sequência:

278, 914, 447, 71, 161, 659, 335

Todos esses pedidos são de processos distintos, portanto podem ser atendidos em qualquer ordem. Para simplificar, considera-se que nenhum pedido de acesso novo chegará à fila durante a execução do algoritmo de escalonamento.

FCFS (*First Come, First Served*): este algoritmo consiste em atender as requisições na ordem da fila, ou seja, na ordem em foram pedidas pelos processos. É a estratégia mais simples de implementar, mas raramente oferece um bom desempenho. Se os pedidos de acesso estiverem muito espalhados pelo disco, este irá perder muito tempo movendo a cabeça de leitura de um lado para o outro. A sequência de blocos percorridos pela cabeça de leitura é:

$500 \xrightarrow{222} 278 \xrightarrow{636} 914 \xrightarrow{467} 447 \xrightarrow{376} 71 \xrightarrow{90} 161 \xrightarrow{498} 659 \xrightarrow{324} 335$ (2.613 blocos)

Percebe-se que, para atender os pedidos de leitura na ordem indicada pelo algoritmo FCFS, a cabeça de leitura teve de deslocar-se por 2.613 blocos do disco ($222 + 636 + 467 + \dots$). A Figura 21.2 mostra os deslocamentos da cabeça de leitura para atender os pedidos de acesso da fila de exemplo.

SSTF (*Shortest Seek Time First – Menor Tempo de Busca Primeiro*): esta estratégia de escalonamento de disco consiste em sempre atender o pedido que está mais próximo da posição atual da cabeça de leitura (que é geralmente a posição do

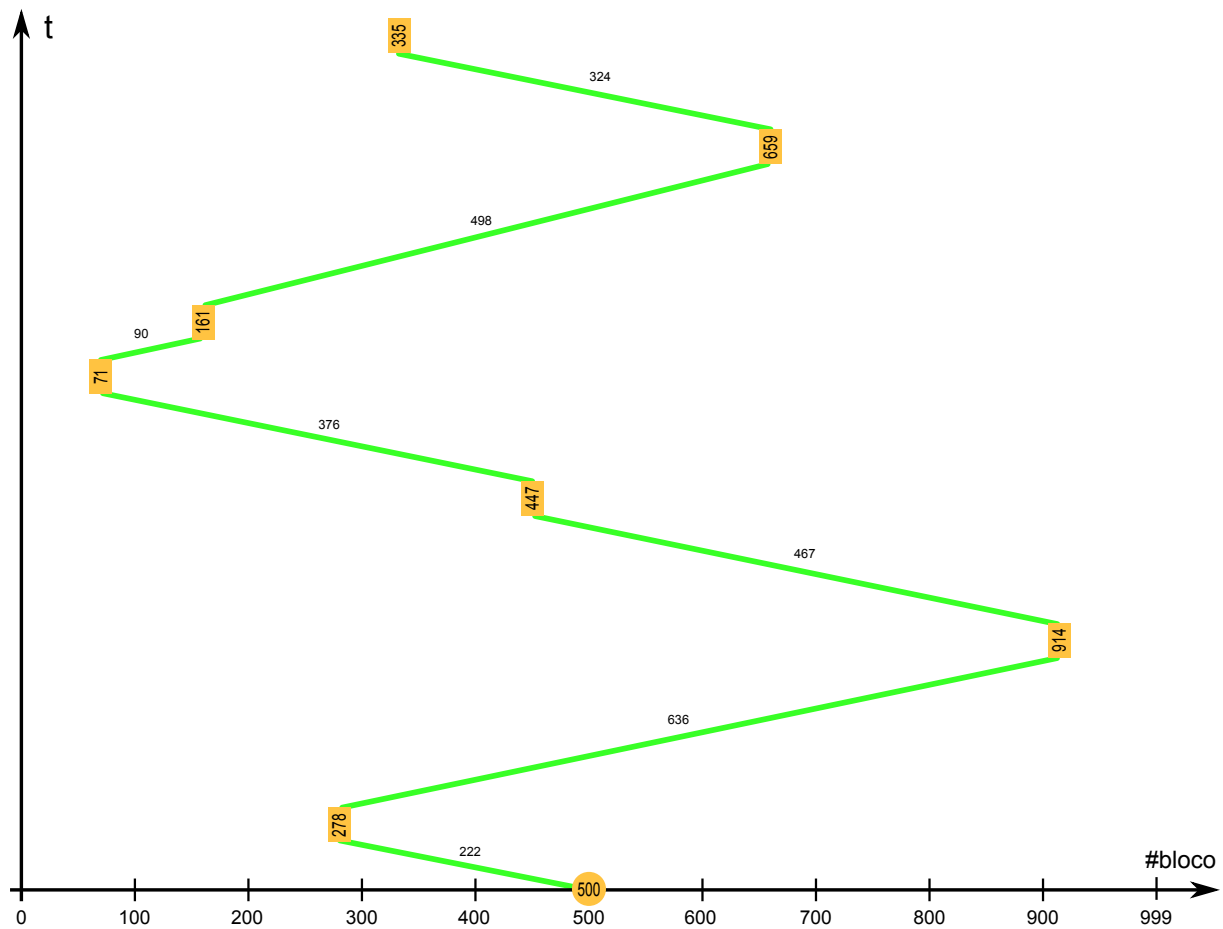


Figura 21.2: Escalonamento de disco FCFS.

último pedido atendido). Dessa forma, ela busca reduzir os movimentos da cabeça de leitura, e com isso o tempo perdido entre os acessos.

A sequência de acesso efetuadas pelo algoritmo SSTF está indicada a seguir e na Figura 21.3. Pode-se observar uma grande redução da movimentação da cabeça de leitura em relação à estratégia FCFS, que passou de 2.613 para 1.272 blocos percorridos. Contudo, a estratégia SSTF não garante obter sempre um percurso mínimo.

$$500 \xrightarrow{53} 447 \xrightarrow{112} 335 \xrightarrow{57} 278 \xrightarrow{117} 161 \xrightarrow{90} 71 \xrightarrow{588} 659 \xrightarrow{255} 914 \text{ (1.272 blocos)}$$

Apesar de oferecer um ótimo desempenho, a estratégia SSTF pode levar à inanição (*starvation*) de requisições de acesso: caso existam muitas requisições em uma determinada região do disco, pedidos de acesso a blocos distantes dessa região podem ficar esperando indefinidamente. Para resolver esse problema, torna-se necessário implementar uma estratégia de *envelhecimento* dos pedidos pendentes.

SCAN: neste algoritmo, a cabeça “varre” (*scan*) continuamente o disco, do início ao final, atendendo os pedidos que encontra pela frente; ao atingir o final do disco, ela inverte seu sentido de movimento e volta, atendendo os próximos pedidos. Apesar de ser mais lento que SSTF, este algoritmo atende os pedidos

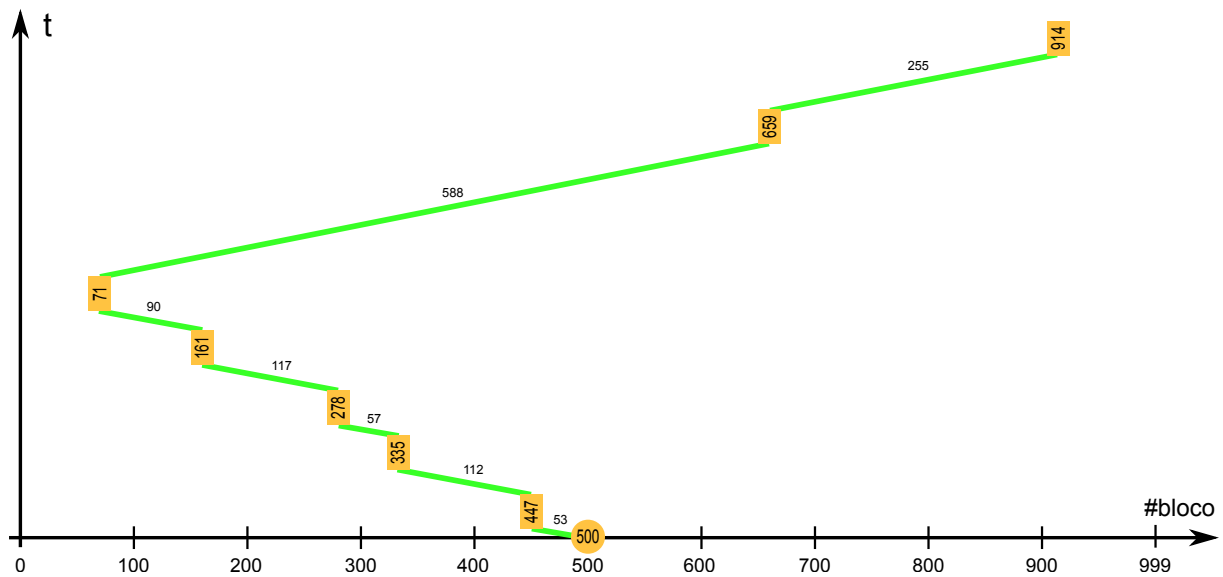


Figura 21.3: Escalonamento de disco SSTF.

de forma mais uniforme ao longo do disco, eliminando o risco de inanição de pedidos e mantendo um desempenho equilibrado para todos os processos. Ele é adequado para sistemas com muitos pedidos simultâneos de acesso a disco, como servidores de arquivos. O comportamento deste algoritmo para a sequência de requisições de exemplo está indicado a seguir e na Figura 21.4:

$$\begin{aligned}
 &500 \xrightarrow{159} 659 \xrightarrow{255} 914 \xrightarrow{85} 999 \xrightarrow{552} 447 \xrightarrow{112} \\
 &\rightarrow 335 \xrightarrow{57} 278 \xrightarrow{117} 161 \xrightarrow{90} 71 \text{ (1.427 blocos)}
 \end{aligned}$$

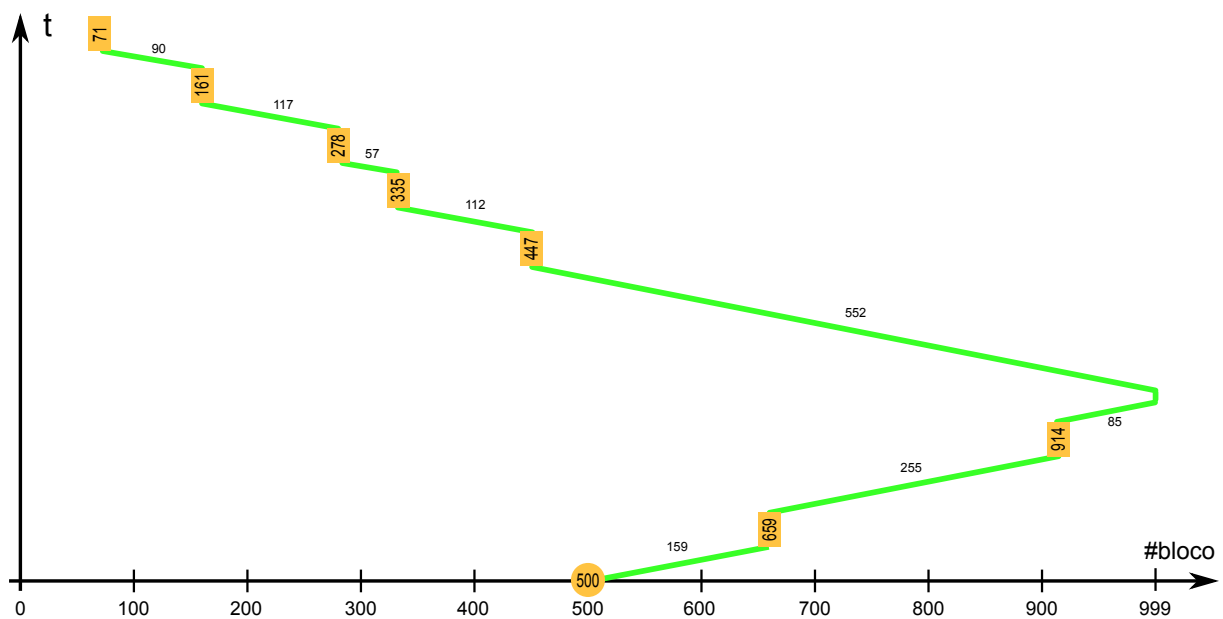


Figura 21.4: Escalonamento de disco com o algoritmo SCAN.

C-SCAN: (*Circular SCAN*) esta é uma variante “circular” do algoritmo SCAN, na qual a cabeça de leitura varre o disco somente em um sentido. Ao atingir o final

do disco, ela retorna diretamente ao início do disco, sem atender os pedidos intermediários, e recomeça a varredura. O nome “circular” é devido ao disco ser visto pelo algoritmo como uma lista circular de blocos. Sua vantagem em relação ao algoritmo SCAN é prover um tempo de espera mais homogêneo aos pedidos pendentes, o que é importante em servidores. O comportamento deste algoritmo para a sequência de requisições de exemplo está indicado a seguir e na Figura 21.5:

$$500 \xrightarrow{159} 659 \xrightarrow{255} 914 \xrightarrow{85} 999 \xrightarrow{999} 0 \xrightarrow{71} 71 \xrightarrow{90} 161 \xrightarrow{117} 278 \xrightarrow{57} 335 \xrightarrow{112} 447 \text{ (1.945 blocos)}$$

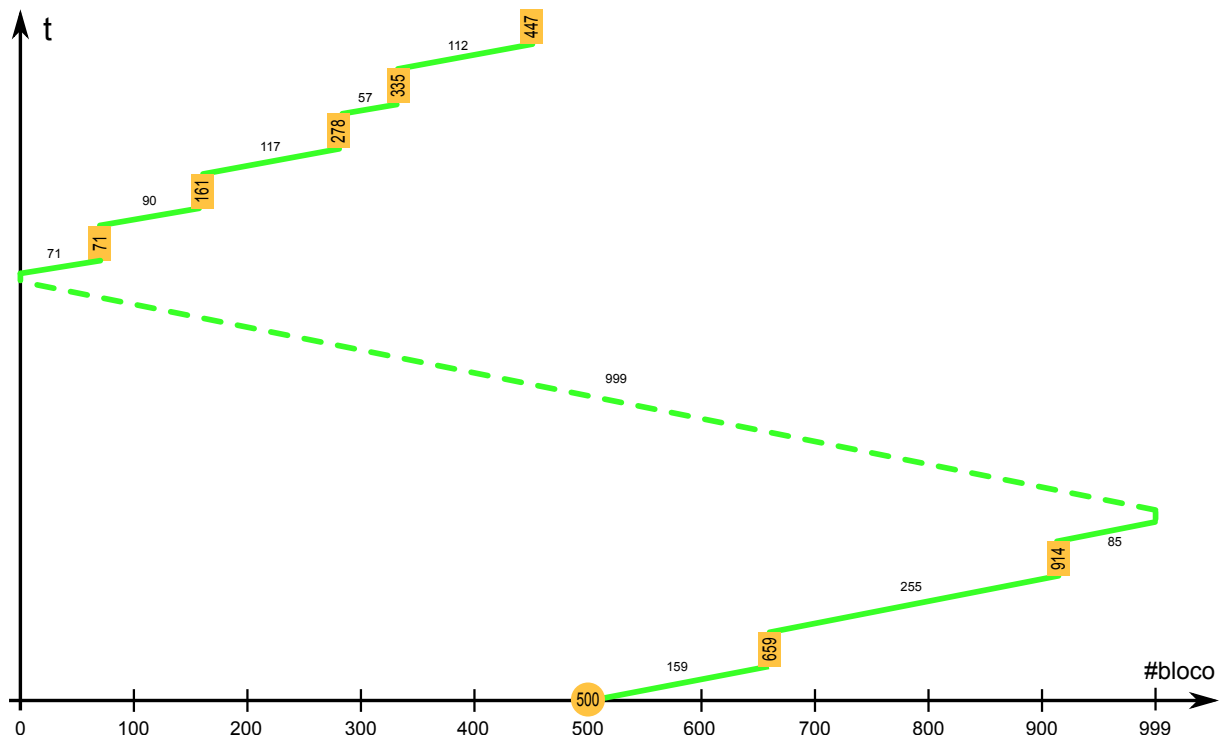


Figura 21.5: Escalonamento de disco com o algoritmo C-SCAN.

LOOK: é uma otimização do algoritmo SCAN, na qual a cabeça do disco não avança até o final do disco, mas inverte seu movimento assim que tiver tratado o último pedido em cada sentido do movimento:

$$500 \xrightarrow{159} 659 \xrightarrow{255} 914 \xrightarrow{467} 447 \xrightarrow{112} 335 \xrightarrow{57} 278 \xrightarrow{117} 161 \xrightarrow{90} 71 \text{ (1.257 blocos)}$$

C-LOOK: (*Circular LOOK*) idem, otimizando o algoritmo C-SCAN:

$$500 \xrightarrow{159} 659 \xrightarrow{255} 914 \xrightarrow{843} 71 \xrightarrow{90} 161 \xrightarrow{117} 278 \xrightarrow{57} 335 \xrightarrow{112} 447 \text{ (1.644 blocos)}$$

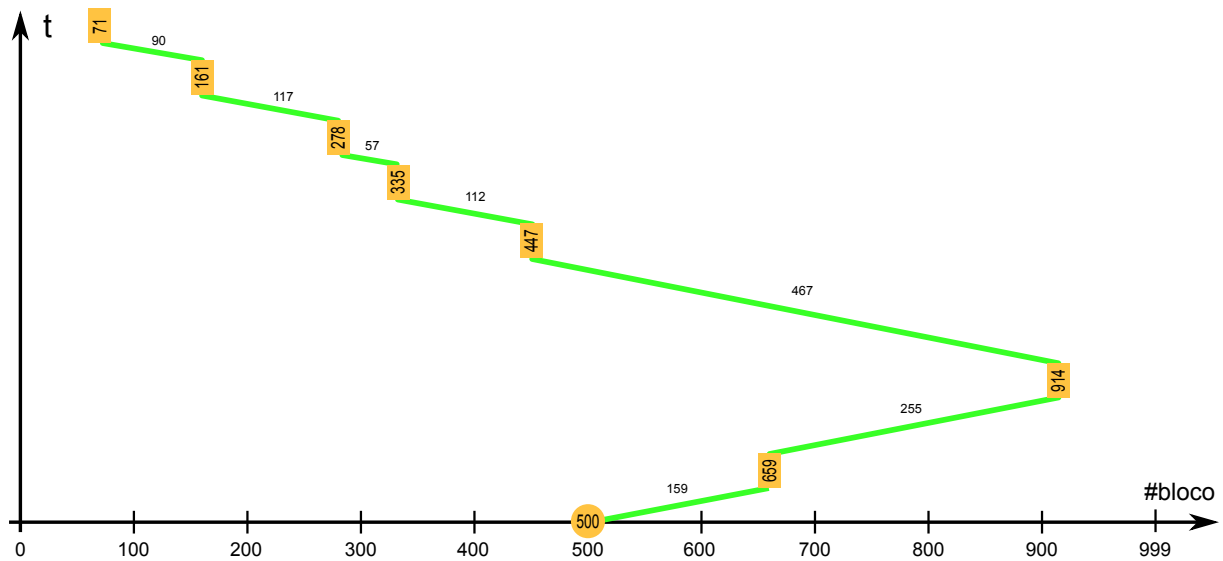


Figura 21.6: Escalonamento de disco com o algoritmo LOOK.

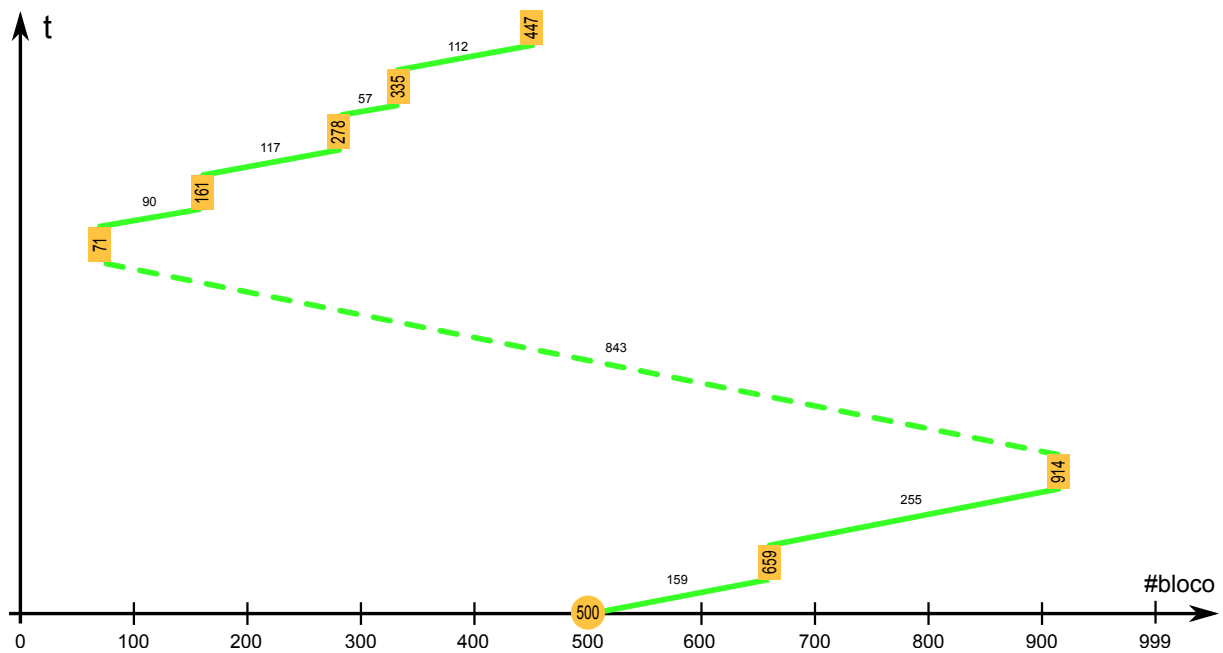


Figura 21.7: Escalonamento de disco com o algoritmo C-LOOK.

Os algoritmos SCAN, C-SCAN e suas variantes LOOK e C-LOOK são denominados coletivamente de *Algoritmo do Elevador*, pois seu comportamento reproduz o comportamento do elevador em um edifício: a cabeça de leitura avança em um sentido, atendendo as requisições dos usuários; ao chegar ao final, inverte seu sentido e retorna [Silberschatz et al., 2001].

Sistemas operacionais reais, como Solaris, Windows e Linux, utilizam escalonadores de disco bem mais sofisticados. No caso do sistema Linux, por exemplo, os seguintes escalonadores de disco estão presentes no núcleo, podendo ser configurados pelo administrador do sistema em função das características dos discos e da carga de trabalho do sistema [Love, 2010; Bovet and Cesati, 2005]:

Noop (*No-Operation*): é o escalonador mais simples, baseado em FCFS, que não reordena os pedidos de acesso, apenas agrupa os pedidos direcionados ao mesmo bloco ou a blocos adjacentes. Este escalonador é voltado para discos de estado sólido (SSD, vide Seção 21.3) ou sistemas de armazenamento que façam seu próprio escalonamento interno, como sistemas RAID (vide Seção 21.2).

Deadline: este escalonador é baseado no algoritmo do elevador circular (C-SCAN), mas associa um prazo (*deadline*) a cada requisição, para evitar problemas de inanição. Como os pedidos de leitura implicam no bloqueio dos processos solicitantes, eles recebem um prazo de 500 *ms*; pedidos de escrita podem ser executados de forma assíncrona, sem bloquear o processo solicitante, por isso recebem um prazo maior, de 5 segundos. O escalonador processa os pedidos usando o algoritmo do elevador, mas prioriza os pedidos cujos prazos estejam esgotando.

Anticipatory: este algoritmo é baseado no anterior (*deadline*), mas busca se antecipar às operações de leitura de dados feitas pelos processos. Como as operações de leitura são geralmente feitas de forma sequencial (em blocos contíguos ou próximos), a cada operação de leitura realizada o escalonador aguarda um certo tempo (por default 6 *ms*) por um novo pedido de leitura naquela mesma região do disco, que é imediatamente atendido. Caso não surja nenhum pedido novo, o escalonador volta a tratar a fila de pedidos pendentes normalmente. Essa espera por pedidos adjacentes melhora o desempenho das operações de leitura emitidas pelo mesmo processo.

CFQ (*Completely Fair Queuing*): os pedidos dos processos são divididos em várias filas (64 filas por default); cada fila recebe uma fatia de tempo para acesso ao disco, que varia de acordo com a prioridade de entrada/saída dos processos contidos na mesma. Este é o escalonador default do Linux na maioria das distribuições mais recentes.

21.2 Sistemas RAID

Apesar dos avanços dos sistemas de armazenamento em estado sólido (como os dispositivos baseados em memórias *flash*), os discos rígidos continuam a ser o principal meio de armazenamento não-volátil de grandes volumes de dados. Os discos atuais têm capacidades de armazenamento impressionantes: encontram-se facilmente no mercado discos rígidos com capacidade da ordem de terabytes para computadores domésticos.

Entretanto, o desempenho dos discos rígidos evolui a uma velocidade muito menor que a observada nos demais componentes dos computadores, como processadores, memórias e barramentos. Com isso, o acesso aos discos constitui um dos maiores gargalos de desempenhos nos sistemas de computação. Boa parte do baixo desempenho no acesso aos discos é devida aos aspectos mecânicos do disco, como a latência rotacional e o tempo de posicionamento da cabeça de leitura do disco (vide Seção 21.1.2) [Chen et al., 1994].

Outro problema relevante associado aos discos rígidos diz respeito à sua confiabilidade. Os componentes internos do disco podem falhar, levando à perda de dados. Essas falhas podem estar localizadas no meio magnético, ficando restritas

a alguns setores, ou podem estar nos componentes mecânicos/eletrônicos do disco, levando à corrupção ou mesmo à perda total dos dados armazenados.

Buscando soluções eficientes para os problemas de desempenho e confiabilidade dos discos rígidos, pesquisadores da Universidade de Berkeley, na Califórnia, propuseram em 1988 a construção de discos virtuais compostos por conjuntos de discos físicos, que eles denominaram RAID – *Redundant Array of Inexpensive Disks*² [Patterson et al., 1988], que em português pode ser traduzido como *Conjunto Redundante de Discos Econômicos*.

Um sistema RAID é constituído de dois ou mais discos rígidos que são vistos pelo sistema operacional e pelas aplicações como um único disco lógico, ou seja, um grande espaço contíguo de armazenamento de dados. O objetivo central de um sistema RAID é proporcionar mais desempenho nas operações de transferência de dados, através do paralelismo no acesso aos vários discos, e também mais confiabilidade no armazenamento, usando mecanismos de redundância dos dados armazenados nos discos, como cópias de dados ou códigos corretores de erros.

Um sistema RAID pode ser construído “por hardware”, usando uma placa controladora dedicada a esse fim, à qual estão conectados os discos rígidos. Essa placa controladora oferece a visão de um disco lógico único ao restante do computador. Também pode ser usada uma abordagem “por software”, na qual são usados *drivers* apropriados dentro do sistema operacional para combinar os discos rígidos conectados ao computador em um único disco lógico. Obviamente, a solução por software é mais flexível e econômica, por não exigir uma placa controladora dedicada, enquanto a solução por hardware é mais robusta e tem um desempenho melhor. É importante observar que os sistemas RAID operam abaixo dos sistemas de arquivos, ou seja, eles se preocupam apenas com o armazenamento e recuperação de blocos de dados.

Há várias formas de se organizar um conjunto de discos rígidos em RAID, cada uma com suas próprias características de desempenho e confiabilidade. Essas formas de organização são usualmente chamadas *Níveis RAID*. Os níveis RAID padronizados pela *Storage Networking Industry Association* são [SNIA]:

RAID 0 (linear): neste nível os discos físicos (ou partições) são simplesmente concatenados em sequência para construir um disco lógico. Essa abordagem, ilustrada na Figura 21.8, é denominada por alguns autores de *RAID 0 linear*, enquanto outros a denominam JBoD (*Just a Bunch of Disks* – apenas um punhado de discos). Na figura, $d_i.b_j$ indica o bloco j do disco físico i .

Em teoria, esta estratégia oferece maior velocidade de leitura e de escrita, pois acessos a blocos em discos físicos distintos podem ser feitos em paralelo. Entretanto, esse ganho pode ser pequeno caso os acessos se concentrem em uma área pequena do disco lógico, pois ela provavelmente estará mapeada em um mesmo disco físico (o acesso a cada disco é sempre sequencial). Além disso, alguns discos tendem a ser mais usados que outros.

Esta abordagem não oferece nenhuma redundância de dados, o que a torna suscetível a erros de disco: caso um disco falhe, todos os blocos lógicos mapeados nele serão perdidos. Como a probabilidade de falhas aumenta com o número de discos, esta abordagem acaba reduzindo a confiabilidade geral do sistema de discos.

²Mais recentemente alguns autores adotaram a expressão *Redundant Array of Independent Disks* para a sigla RAID, buscando evitar a subjetividade da palavra *Inexpensive* (econômico).

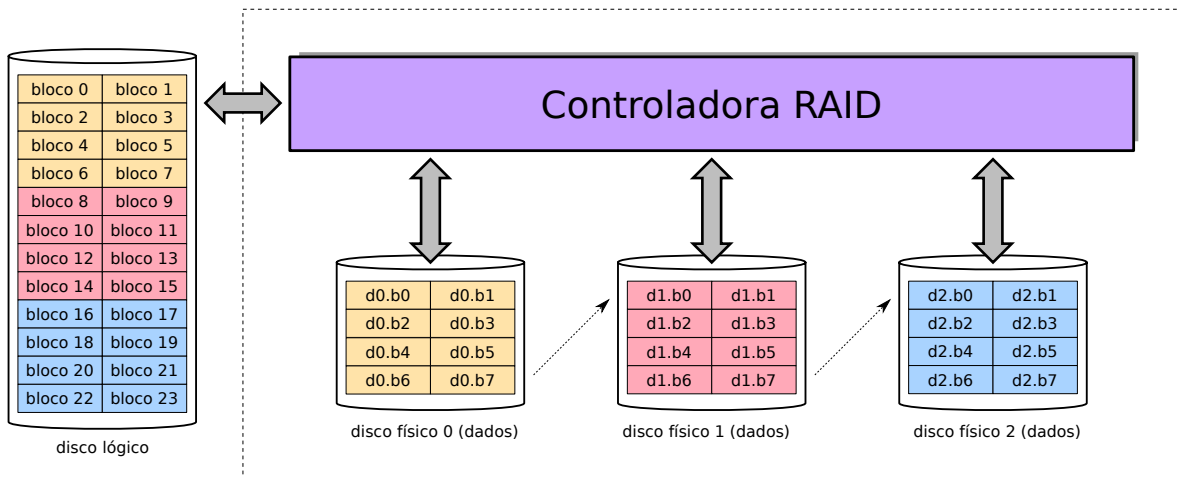


Figura 21.8: RAID nível 0 (*linear*).

RAID 0 (stripping): neste nível os discos físicos são divididos em áreas de tamanhos fixo chamadas *fatias* ou *faixas (stripes)*. Cada fatia de disco físico armazena um ou mais blocos do disco lógico (tipicamente são usadas faixas de 32, 64 ou 128 KBytes). As fatias são concatenadas usando uma estratégia *round-robin* para construir o disco lógico, como mostra a Figura 21.9.

O maior espalhamento dos blocos sobre os discos físicos contribui para distribuir melhor a carga de acessos entre eles e assim ter um melhor desempenho. Suas características de suporte a grande volume de dados e alto desempenho em leitura/escrita tornam esta abordagem adequada para ambientes que precisam processar grandes volumes de dados temporários, como os sistemas de computação científica [Chen et al., 1994]. Esta abordagem também não oferece redundância de dados.

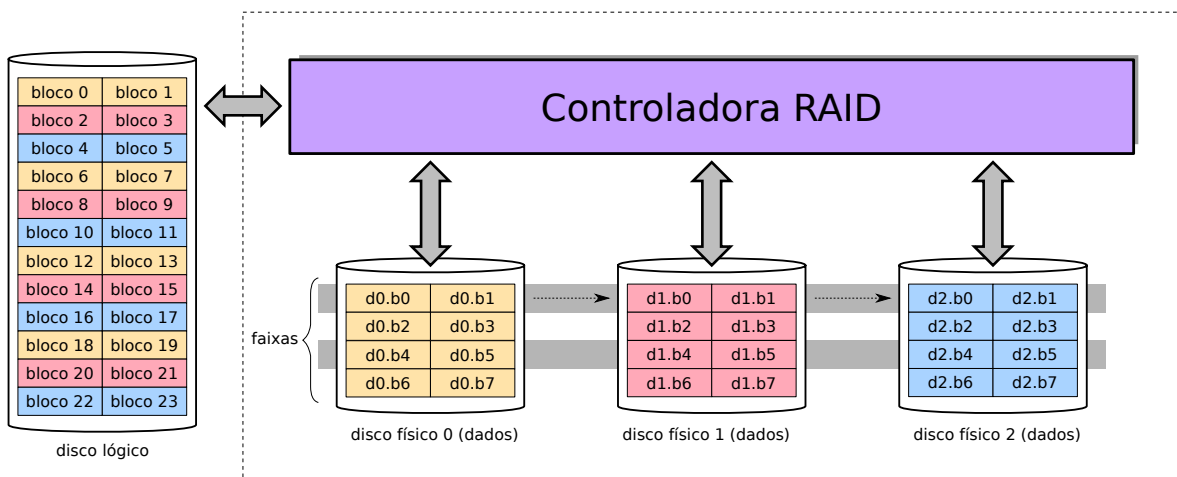


Figura 21.9: RAID nível 0 (*stripping*).

RAID 1: neste nível, o conteúdo é replicado em dois ou mais discos, sendo por isso comumente chamado de *espelhamento de discos*. A Figura 21.10 mostra uma configuração usual deste nível de RAID, com dois discos físicos (embora seja menos usual, o conteúdo pode ser replicado em $N > 2$ discos, para tolerar $N - 1$

discos falhos). O espelhamento também pode ser feito com conjuntos de discos em RAID 0, levando a configurações híbridas como RAID 0+1, RAID 1+0 ou RAID 1E.

Esta abordagem oferece uma excelente confiabilidade, pois cada bloco lógico está escrito em dois ou mais discos distintos; caso um deles falhe, os demais continuam acessíveis. O desempenho em leituras também é beneficiado, pois a controladora pode distribuir as leituras entre as cópias dos dados. Contudo, não há ganho de desempenho em escrita, pois cada operação de escrita deve ser replicada em todos os discos. Além disso, seu custo de implantação é elevado, pois os N discos físicos são vistos como apenas um disco lógico do mesmo tamanho.

Quando um disco físico de um sistema RAID 1 falha, ele é substituído e o conteúdo do novo disco é copiado de outro disco do conjunto, sem parar a operação do sistema. Essa operação, geralmente suportada pela própria controladora, é chamada *reconstrução do RAID (RAID rebuild)*.

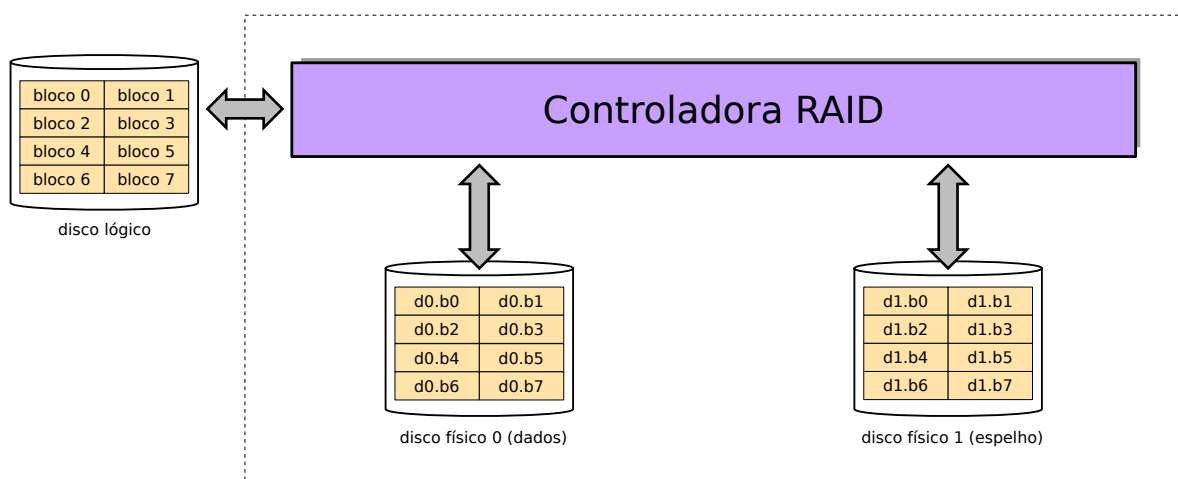


Figura 21.10: RAID nível 1 (espelhamento) com dois discos.

RAID 2: neste nível os dados são “fatiados” em bits individuais que são escritos nos discos físicos em sequência; discos adicionais são usados para armazenar códigos corretores de erros (*Hamming Codes*), em um arranjo similar ao usado nas memórias RAM. Esses códigos corretores de erros permitem resgatar dados no caso de falha em blocos ou discos de dados. Por ser pouco eficiente e complexo de implementar, este nível não é usado na prática.

RAID 3: de forma similar ao RAID 2, este nível fatia os dados em bytes escritos nos discos em sequência. Um disco adicional é usado para armazenar um byte com os bits de paridade dos bytes correspondentes em cada disco, sendo usado para a recuperação de erros nos demais discos. A cada escrita, os dados de paridade devem ser atualizados, o que transforma o disco de paridade em um gargalo de desempenho. Também não é usado na prática.

RAID 4: esta abordagem é similar ao RAID 3, com a diferença de que o fatiamento é feito em blocos ao invés de bytes, como mostra a Figura 21.11. Ela sofre dos

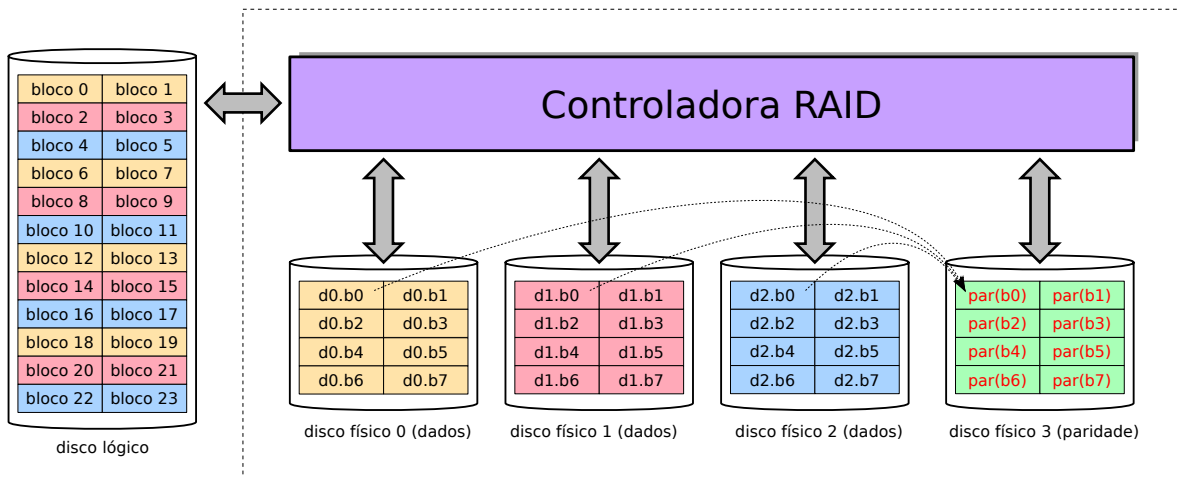


Figura 21.11: RAID nível 4 (disco de paridade).

mesmos problemas de desempenho que o RAID 3, sendo por isso pouco usada. Todavia, ela serve como base conceitual para o RAID 5.

A tabela 21.1 mostra de forma simplificada como a informação de paridade permite reconstruir o conteúdo de um disco falho. Neste exemplo, cada bloco par_i do disco de paridade contém o XOR (OU-exclusivo: \oplus) dos blocos correspondentes nos discos de dados: $par_i = d_i^1 \oplus d_i^2 \oplus d_i^3$. Se um disco falhar, seu conteúdo pode ser reconstruído com operações XOR sobre o conteúdo dos demais discos.

	disco 1	disco 2	disco 3	paridade	cálculo
Cálculo da paridade	3A22	A0FF	0747	-	$3A22 \oplus A0FF \oplus 0747 \rightarrow 9D9A$
Operação normal	3A22	A0FF	0747	9D9A	
Falha disco 1	-	A0FF	0747	9D9A	$A0FF \oplus 0747 \oplus 9D9A \rightarrow 3A22$
Falha disco 2	3A22	-	0747	9D9A	$3A22 \oplus 0747 \oplus 9D9A \rightarrow A0FF$
Falha disco 3	3A22	A0FF	-	9D9A	$3A22 \oplus A0FF \oplus 9D9A \rightarrow 0747$
Falha disco de paridade	3A22	A0FF	0747	-	não afeta os dados

Tabela 21.1: Exemplo de reconstrução de blocos perdidos usando a informação de paridade.

RAID 5: assim como o RAID 4, esta abordagem também armazena informações de paridade para tolerar falhas em blocos ou discos. Todavia, essas informações não ficam concentradas em um único disco físico, mas são distribuídas uniformemente entre eles, eliminando o gargalo de desempenho no acesso aos dados de paridade visto nos níveis RAID 3 e 4. A Figura 21.12 ilustra uma possibilidade de distribuição das informações de paridade.

Esta estratégia provê um bom desempenho em leitura, ao permitir acessar os discos em paralelo, mas o desempenho em escrita é prejudicado pela necessidade de atualizar a informação de paridade: a cada escrita de um bloco de dados, deve-se ler o bloco, ler o bloco de paridade, recalcular a paridade, escrever o novo bloco de dados e o novo bloco de paridade (4 operações em disco). O RAID 5 é bastante popular, por oferecer um bom desempenho e redundância de dados com menor custo que o espelhamento (RAID 1).

A reconstrução de um sistema RAID 5 envolve substituir o disco falho e escrever os blocos do novo disco, calculados a partir dos blocos de dados e de paridade presentes nos demais discos. Entretanto, durante a reconstrução o sistema RAID 5 fica vulnerável: dados serão perdidos se outro disco falhar antes do RAID ser completamente reconstruído.

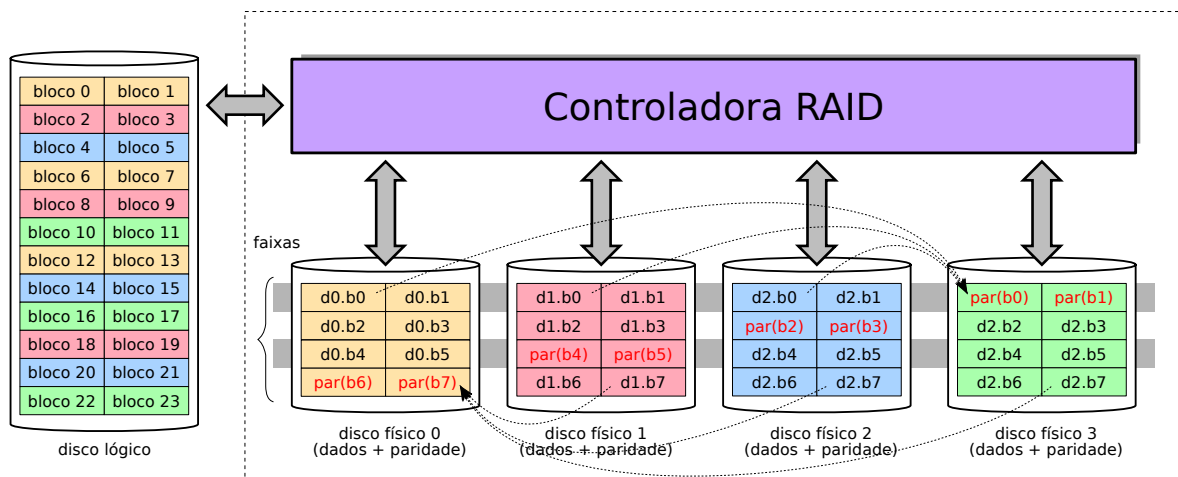


Figura 21.12: RAID nível 5 (paridade distribuída).

RAID 6: é uma extensão do nível RAID 5 que utiliza blocos com códigos corretores de erros de *Reed-Solomon*, além dos blocos de paridade. Esta redundância extra demanda dois discos adicionais, mas permite tolerar falhas simultâneas de até dois discos. Por outro lado, cada operação de escrita envolve atualizar dois blocos de paridade em discos distintos, reduzindo o desempenho em escrita. O uso do RAID 6 cresceu nos últimos anos, porque o tempo de reconstrução de um sistema RAID 5 com discos muito grandes pode ser bastante longo, aumentando o risco de perda de dados.

Além dos níveis padronizados, no mercado podem ser encontrados produtos oferecendo outros níveis RAID, como 1+0, 0+1, 1E, 10, 50, 100, etc., que muitas vezes implementam combinações dos níveis básicos ou soluções proprietárias. Outra observação importante é que os vários níveis de RAID não têm necessariamente uma relação hierárquica entre si, ou seja, um sistema RAID 5 não é necessariamente melhor que um sistema RAID 1, pois isso depende do contexto de aplicação do sistema. Uma descrição mais aprofundada dos vários níveis RAID, de suas variantes e características pode ser encontrada em [Chen et al., 1994] e [SNIA].

A Tabela 21.2 traz um comparativo entre as principais características das diversas estratégias de RAID apresentadas nesta seção. As velocidades máximas de leitura e de escrita são analisadas em relação a um disco isolado de mesmo tipo³, com velocidade de leitura L e de escrita E . A coluna *Espaço* indica a fração do espaço total dos N discos que está disponível para o armazenamento de dados, considerando discos de tamanho T e descontando as perdas com replicação ou paridade. A coluna *Falhas* indica o número máximo de discos que pode falhar sem causar perda de dados. A coluna *Discos* define o número mínimo de discos necessários para contruir a configuração.

³As expressões indicam as velocidades esperadas para leituras e escritas pequenas e aleatórias; para operações longas sequenciais os cálculos são diversos.

Estratégia	Vel. máx. leitura	Vel. máx. escrita	Espaço	Falhas	Discos
RAID 0 linear	$L \times N$ (pode ler os discos em paralelo)	$E \times N$ (pode escrever nos discos em paralelo)	$T \times N$	0	≥ 2
RAID 0 stripping	$L \times N$ (idem)	$E \times N$ (idem)	$T \times N$	0	≥ 2
RAID 1	$L \times N$ (idem)	E (deve atualizar todos os discos)	T	$N - 1$	≥ 2
RAID 4	$L \times (N - 1)$ (pode ler os discos de dados em paralelo)	E (deve atualizar o disco de paridade)	$T \times (N - 1)$	1	≥ 3
RAID 5	$L \times N$ (pode ler os discos em paralelo)	$E \times N/4$ (4 acessos a disco em cada escrita)	$T \times (N - 1)$	1	≥ 3
RAID 6	$L \times N$ (idem)	$E \times N/6$ (6 acessos a disco em cada escrita)	$T \times (N - 2)$	2	≥ 4

Tabela 21.2: Comparativo de estratégias de RAID, considerando arranjos com N discos iguais de tamanho T , velocidade de leitura L e de escrita E .

21.3 Dispositivos de estado sólido

Uma unidade ou disco de estado sólido (SSD, do inglês *Solid-State Device*), é um dispositivo de armazenamento de dados não-volátil construído com circuitos integrados de memória *flash*, sem partes mecânicas móveis. Apesar da menor capacidade de armazenamento e de seu preço por gigabyte ser mais elevado, um SSD é mais compacto, mais leve, mais rápido, mais silencioso, mais resistente a choques físicos e tem menor consumo de energia que um disco rígido equivalente. Por conta dessas vantagens, nos últimos anos, dispositivos de estado sólido têm conquistado grande espaço no mercado, sobretudo em computadores pessoais e aparelhos móveis [Hutchinson, 2012; Micheloni et al., 2018].

Os primeiros SSDs foram criados na década de 1970; devido ao custo elevado e baixa capacidade, até os anos 2000 seu uso era restrito a supercomputadores e em aplicações espaciais/militares. Com o barateamento da tecnologia de memórias flash, nos últimos anos eles vêm ocupando o espaço dos discos rígidos na computação pessoal. Esta seção apresenta uma visão geral da tecnologia de memória usada nos SSDs, pendrives e cartões de memória, contemplando sua organização interna, seu funcionamento e suas interfaces de acesso.

21.3.1 Memória flash

Em um SSD, os dados são armazenados em células de memória flash. Em sua forma mais simples, uma célula de memória flash é uma estrutura semicondutora capaz de armazenar um bit (0 ou 1) de forma não-volátil, ou seja, cujo conteúdo se preserva na ausência de energia (ao contrário das memórias RAM, que perdem seu conteúdo ao ficar sem energia) [Larrivee, 2016]. Células flash de nível único (SLC - *Single Level Cell*) armazenam 1 bit de informação, enquanto células multinível (MLC - *Multi-Level Cell*) podem armazenar 2 ou mais bits por célula. Células multinível permitem maior capacidade de armazenamento nos chips de memória, sendo portanto mais baratas e populares.

A estrutura interna de uma célula flash pode ser NAND ou NOR, por sua similaridade com as portas lógicas de mesmo nome, e ambas têm características elétricas e aplicações distintas. Para o armazenamento maciço de dados em unidades SSD, pendrives ou cartões de memória geralmente são usadas células NAND.

Para seu uso efetivo, as células flash são agrupadas inicialmente em **strings** (com 32 a 128 células ligadas em série), que são agrupadas em **páginas** (entre 4 e 8 KBytes cada). Por sua vez, páginas são agrupadas em blocos (de 1 a 128 MBytes cada), que são agrupados em planos para formar um chip de memória. A figura 21.13 ilustra a estrutura típica de um chip de memória flash.

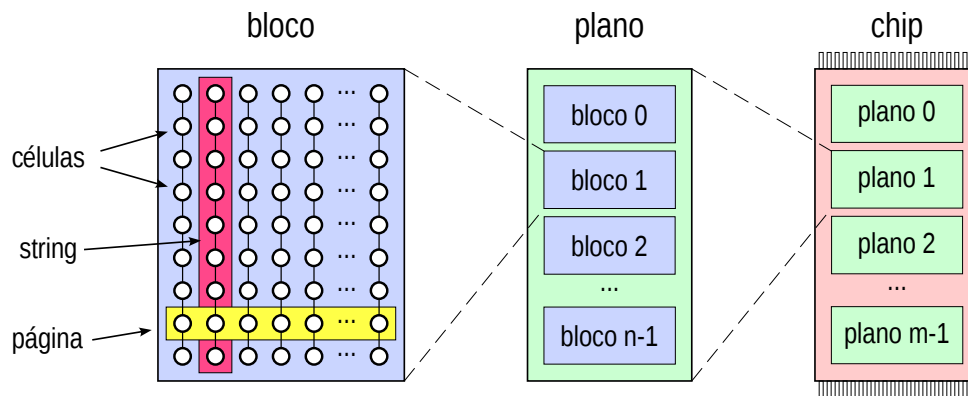


Figura 21.13: Estrutura de um chip de memória flash.

Uma célula flash pode ser escrita ou apagada pela aplicação de tensões elevadas (da ordem de 15V a 20V) em seus terminais. Em consequência, escrever ou apagar uma célula flash é um processo que causa desgaste em sua estrutura interna, o que limita a vida útil da mesma. Células flash SLC geralmente suportam cerca de 100.000 apagamentos, enquanto células MLC suportam 10.000 apagamentos ou menos.

Por suas características operacionais e construtivas, uma memória flash deve ser acessada (lida ou escrita) em páginas, similarmente aos discos rígidos, onde os dados são lidos ou escritos em blocos. Entretanto, dados somente podem ser escritos em uma página de memória flash se esta tiver sido previamente apagada. Além disso, páginas não podem ser apagadas individualmente: o bloco inteiro que contém a página deve ser apagado na mesma operação (um bloco de páginas contém dezenas ou centenas de páginas). Por fim, o apagamento é uma operação lenta: enquanto o tempo médio de leitura em um SSD varia de 25 a 75 μs e o de escrita varia de 200 a 1.350 μs , o tempo médio de apagamento de um bloco é bem maior, de 1,5 a 4,5 ms (1.500 a 4.500 μs).

Todas essas restrições exigem o desenvolvimento de algoritmos específicos para o gerenciamento das escritas e apagamentos em uma memória flash, a fim de preservar seu desempenho e maximizar sua vida útil. Esses algoritmos são geralmente implementados por um controlador interno ao SSD, discutido na próxima seção.

21.3.2 Estrutura de um SSD

Em termos de hardware, um dispositivo de estado sólido (SSD - *Solid State Device*) é constituído por um conjunto de chips de memória flash, um microprocessador dedicado denominado *controlador* e uma interface para conexão com o computador. Essa estrutura é similar em SSDs, cartões de memória e pendrives e está representada na figura 21.14.

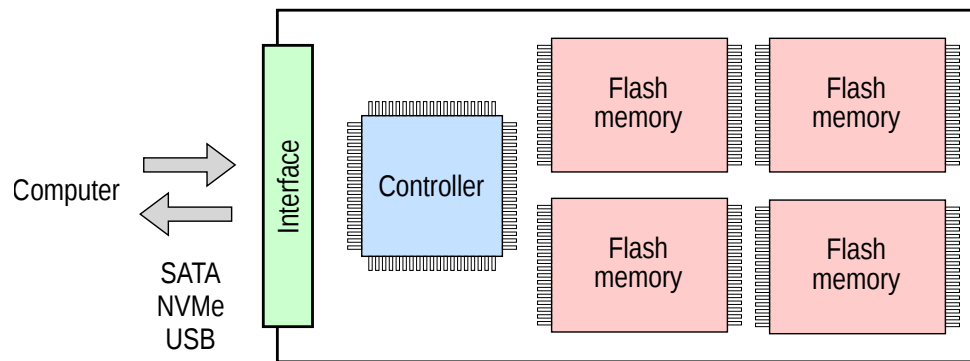


Figura 21.14: Estrutura básica de um dispositivo de estado sólido (SSD).

O controlador do SSD implementa muitas funcionalidades, dentre as quais se destacam [Michelsoni et al., 2018]:

Interface : o controlador recebe comandos do processador para ler e escrever dados no dispositivo e para ações de gerenciamento; a conexão com o restante do computador geralmente é feita por interfaces padronizadas como SATA, USB ou NVMe (vide seção 21.4).

Tradução de endereços : o controlador deve traduzir os endereços dos pedidos de leitura e escrita enviados pelo SO para as posições efetivas onde os dados se encontram nos chips de memória flash (seção 21.3.3).

Gerência de escrita e apagamento : o controlador gerencia onde novos dados devem ser escritos na memória flash e que blocos podem/devem ser apagados para poderem ser utilizados novamente (seção 21.3.4).

Nivelamento de desgaste : as células flash se desgastam ao ser apagadas, então o controlador deve distribuir essas operações no conjunto de células disponíveis, para nivelar uniformemente o desgaste e prolongar a vida útil do dispositivo (seção 21.3.5).

Buffer/cache : o controlador mantém uma área de memória RAM interna como cache de dados e operações na memória flash, a fim de aumentar o desempenho do acessos.

Correção de erros : eventuais erros na leitura ou escrita de células flash podem ocorrer, então o controlador usa códigos corretores de erro (ECC - *Error Correcting Codes*) para detectá-los e corrigi-los.

Gestão de defeitos : o processo de fabricação de memórias flash pode produzir células com defeito. Além disso, mais células podem apresentar falhas ao longo da vida útil do dispositivo. Cabe ao controlador identificá-las e gerenciá-las, mantendo tabelas de páginas ou blocos com defeito.

Além das indicadas acima, o controlador do SSD também pode assumir funcionalidades de criptografia, gestão de energia, etc, dependendo do tipo e complexidade do dispositivo.

21.3.3 Camada de tradução flash

Para permitir um gerenciamento flexível do armazenamento, os endereços dos dados referenciados pelo sistema operacional em um SSD não correspondem às posições físicas dos mesmos na memória flash. O sistema operacional “vê” o SSD como um grande vetor de blocos ou setores lógicos, numerados de 0 (zero) até um valor máximo. Por outro lado, os dados são fisicamente armazenados nas memórias flash em páginas, geralmente com 4 ou 8 KBytes cada, contidas em blocos dentro dos chips.

O controlador do SSD gerencia a conversão entre os endereços dos setores lógicos e as páginas físicas correspondentes através da *Camada de Tradução Flash* (FTL, do inglês *Flash Translation Layer*), apresentada na Figura 21.15. Além de implementar a tradução de endereços, essa camada de software também mantém tabelas com outras informações relevantes, como o estado atual de cada página física (livre, em uso, inválida, defeituosa) e o número de vezes em que cada bloco foi apagado, por exemplo. A FTL também implementa os algoritmos de coleta de lixo e nivelamento de desgaste, apresentados a seguir.

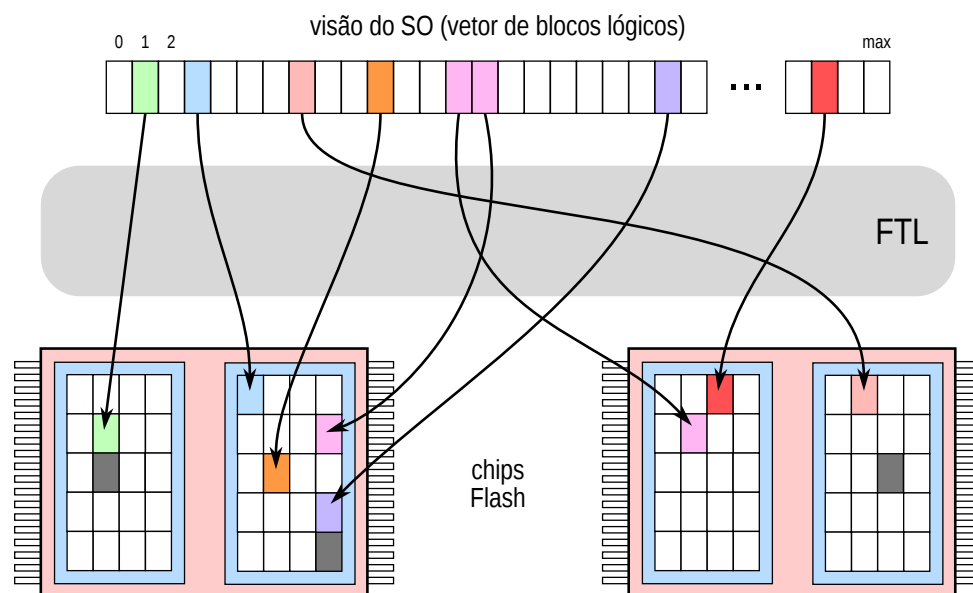


Figura 21.15: Camada de Tradução Flash (FTL).

21.3.4 Estratégia de escrita e apagamento

Conforme apresentado nas seções anteriores, a memória flash usada nos SSDs impõe algumas restrições de operação:

- Dados devem ser lidos e escritos em páginas;
- Páginas podem ser lidas sem restrições;
- Cada página deve ser apagada antes de ser escrita;
- Páginas só podem ser apagadas em blocos;
- Células flash têm um número limitado de apagamentos.

Para conseguir usar a memória flash de forma eficiente e duradoura, o controlador do dispositivo deve implementar uma estratégia de escrita e apagamento. Essa estratégia pode ser complexa e varia conforme o fabricante, então será apresentada neste texto uma visão abstrata e genérica de como ela funciona.

Seja considerado um dispositivo hipotético com 2 blocos de 32 páginas de 4 KBytes cada (resultando em um espaço total de minúsculos 256 KBytes), mostrado na figura 21.16. Inicialmente, todas as páginas físicas do dispositivo estão livres (ou seja, foram previamente apagadas). Na figura, uma página em branco está apagada (livre para uso), uma página numerada indica o número da página lógica que ela armazena (conforme vista pelo SO) e uma página marcada com X está inválida e deve ser apagada pelo controlador antes de ser usada novamente.

A figura 21.16 mostra os cinco passos básicos das operações de escrita em um dispositivo flash:

1. O sistema operacional solicita a escritas das páginas lógicas 0 a 9 no SSD; o controlador as aloca nas páginas físicas iniciais livres do bloco 0.
2. Em seguida, o SO solicita reescrever as páginas lógicas 0 a 9, alterando seus conteúdos. Uma página física da memória flash não pode ser reescrita sem antes apagar o bloco inteiro onde ela se encontra. Para evitar isso, o controlador escreve os conteúdos alterados em novas páginas físicas livres do bloco 0 e marca as páginas anteriores como inválidas, para posteriormente apagá-las.
3. O SO escreve novas páginas lógicas (10 a 25); o controlador as aloca no restante do bloco 0 e em parte do bloco 1. Neste momento o bloco 0 está completamente ocupado e não pode mais receber escritas, até ser apagado.
4. Ao detectar que o bloco 0 está cheio e com páginas inválidas, o controlador pode acionar um mecanismo de “coleta de lixo” (*garbage collection*) para reorganizar a memória e resgatar as páginas inválidas. Isso é feito inicialmente migrando (copiando) o conteúdo das páginas válidas do bloco 0 para o bloco 1 e marcando as páginas anteriores como inválidas.
5. Agora o bloco 0 só tem páginas inválidas e pode ser apagado, liberando todas as suas páginas para novas escritas.

Neste exemplo simples, o mecanismo de coleta de lixo foi usado para liberar apenas um bloco ocupado. Em um situação real, a coleta de lixo usa algoritmos de otimização combinatória para reorganizar a memória e apagar/liberar de forma otimizada diversos blocos.

21.3.5 Nivelamento de desgaste

Uma restrição significativa na operação de dispositivos baseados em memória flash é o limite no número de vezes em que uma célula flash pode ser apagada, também chamado de *ciclos de apagamento* (do inglês *erasure cycles*). Ao atingir esse limite, a célula e a página onde ela se encontra não podem mais ser apagadas com segurança e perdem seu uso. Blocos contendo dados alterados com frequência, como tabelas de diretórios e arquivos de logs, têm de ser apagados com frequência, exaurindo rapidamente a vida útil de suas páginas.

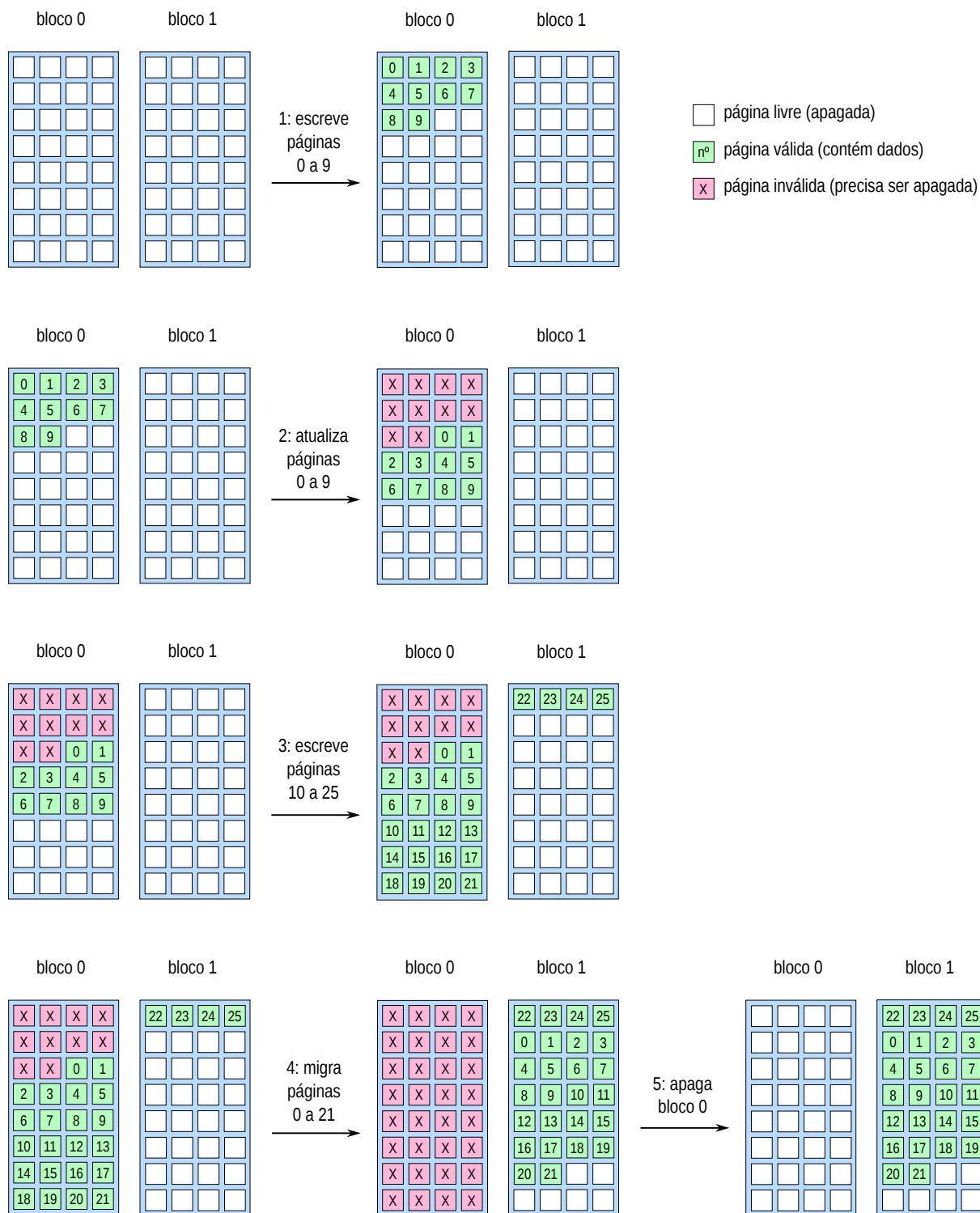


Figura 21.16: Operações de escrita em um dispositivo flash.

Ao escolher páginas físicas para escrever dados e ao realizar migrações de páginas para a coleta de lixo, o controlador deve levar em conta o número de vezes que cada bloco de páginas foi apagado, ou seja, o seu *nível de desgaste*, e usar páginas livres nos blocos com o menor desgaste. Essa estratégia se chama *nivelamento de desgaste* (do inglês *wear leveling*) e visa distribuir melhor o desgaste entre todos os blocos do SSD.

Em um sistema de arquivos, há muitos arquivos que não são alterados com frequência, como arquivos de configuração, binários, bibliotecas e aplicativos. Esses arquivos usualmente só são alterados em atualizações de software e, portanto, os blocos em que estão armazenados sofrem bem menos apagamentos que os blocos contendo dados. Para melhorar o nivelamento de desgaste, os controladores SSD atuais podem mover as páginas desses blocos com pouco desgaste para blocos mais desgastados, a fim de aproveitar seus ciclos de apagamento e maximizar a vida útil do dispositivo. Essa abordagem é denominada **nivelamento estático**, por envolver páginas já alocadas (estáticas), em oposição ao **nivelamento dinâmico**, que envolve somente as páginas físicas livres.

Os algoritmos de coleta de lixo e nivelamento de desgaste implementados pela FTL são genéricos, transparentes para o sistema operacional e funcionam adequadamente em computadores de uso geral. Entretanto, eles podem ser inadequados para sistemas com cargas de trabalho muito específicas, como servidores de bancos de dados, nos quais a quantidade de reescritas em alguns blocos pode ser muito grande. Para esses casos existem os SSDs de canal aberto (*Open-Channel SSDs*), nos quais não há FTL: neles, o próprio sistema operacional assume a responsabilidade pelas tarefas de tradução de endereços, gerência de escritas, apagamentos, coleta de lixo e nivelamento de desgaste, podendo adequá-las para cargas de trabalho específicas [Bjørling et al., 2017].

21.3.6 O comando TRIM

O sistema operacional continuamente cria, altera e apaga arquivos no disco, ao longo de seu funcionamento. Ao apagar um arquivo, a entrada correspondente ao mesmo na tabela de diretórios é removida e todos os blocos lógicos que ele ocupa no disco são liberados para uso em outros arquivos. Em um sistema de arquivos com disco rígido, geralmente o SO somente ajusta a tabela de diretórios e as tabelas que registram quais blocos do disco estão livres e quais estão ocupados; não há necessidade de reescrever o conteúdo dos blocos de disco ocupados pelo arquivo.

Essa forma de apagamento de arquivos não é adequada para dispositivos baseados em memória flash. Como o SO somente atualiza a tabela de diretório, a FTL não é informada sobre a liberação dos blocos de dados ocupados pelo arquivo apagado e continuará a tratar as páginas físicas correspondentes como válidas, até que elas sejam reescritas em outras operações. Essas páginas físicas consumirão espaço útil e irão participar dos algoritmos de coleta de lixo e nivelamento de desgaste sem necessidade, impactando o desempenho do dispositivo.

Para resolver esse problema, a FTL implementa um comando denominado *TRIM*⁴ (ou UNMAP) que deve ser invocado pelo SO para informar a FTL sobre a liberação de blocos lógicos. Dessa forma, o controlador pode marcar as páginas físicas correspondentes aos blocos liberados como inválidas e disponibilizá-las para a coleta de lixo. O comando TRIM precisa ser implementado pela FTL e deve ser acionado periodicamente pelo sistema operacional; ele é suportado pela maioria dos SSDs e SOs atuais [Michelsoni et al., 2018].

A Figura 21.17 ilustra o funcionamento do comando TRIM: ela mostra a visão lógica de um arquivo como uma entrada na tabela de diretório, um vetor de blocos lógicos com o conteúdo do arquivo e o armazenamento dessa informação nas páginas do SSD. Ao apagar o arquivo, o SO atualiza a tabela de diretório no disco e então invoca

⁴Do inglês “to trim” - aparar, desbastar.

o comando TRIM para invalidar os blocos do arquivo, que estão sendo liberados. As páginas físicas ocupadas pela tabela de diretório estão indicadas pela letra “d” e as ocupadas pelo conteúdo do arquivo estão indicadas pelos números dos blocos lógicos do arquivo.

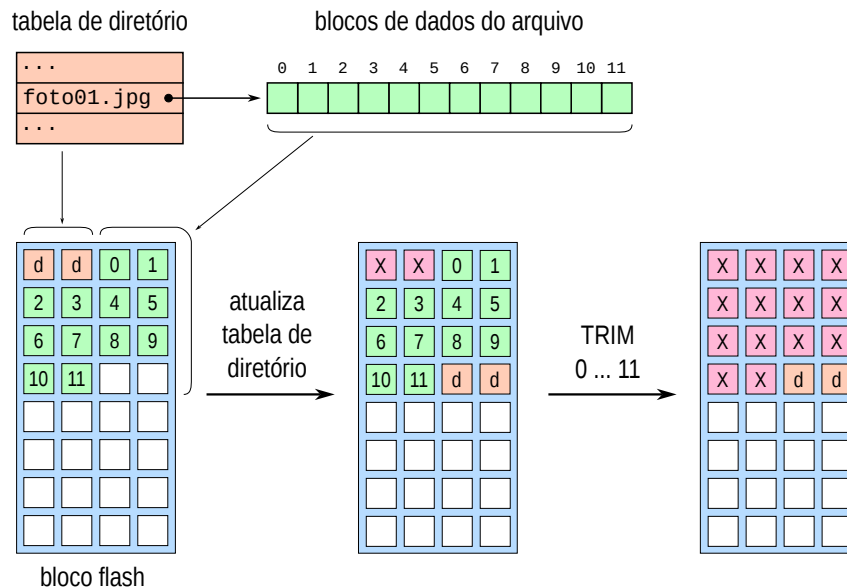


Figura 21.17: Funcionamento do comando TRIM.

21.3.7 Amplificação de escritas

Na operação dos SSDs, muitas vezes uma única solicitação de escrita de um bloco lógico pelo SO pode resultar em várias operações de escrita nos chips de memória flash. Isso ocorre por várias razões:

- Se o bloco lógico a ser escrito já estiver armazenado no SSD, a versão anterior dele precisa ser invalidada;
- A coleta de lixo pode ser necessária para liberar páginas físicas para a escrita;
- As tabelas da FTL que mantêm informações sobre as páginas e blocos físicos precisam ser atualizadas.

Esse aumento do número de escritas físicas em relação às escritas solicitadas pelo SO é denominado *amplificação de escritas*. O *Fator de Amplificação de Escrita* (WAF - *Write Amplification Factor*) é definido pela seguinte razão:

$$WAF = \frac{\text{bytes escritos na memória flash}}{\text{bytes escritos pelo SO}} \quad (21.1)$$

Idealmente, o fator de amplificação de escritas seria 1,0; fabricantes de SSDs comerciais costumam anunciar valores de WAF entre 1,2 e 1,5, mas esse número é fortemente dependente da carga de trabalho e do nível de preenchimento do disco. Em alguns casos, controladores que implementam compressão de dados dentro do SSD podem obter um fator de amplificação inferior a 1, ou seja, escrever menos dados na memória flash que a quantidade enviada pelo SO.

21.3.8 Provisionamento em excesso

O comportamento dos algoritmos de coleta de lixo e de nivelamento de desgaste é influenciado pela quantidade de espaço livre disponível no dispositivo: discos com muitas operações de escrita e pouco espaço livre são forçados a coletar lixo com mais frequência para liberar blocos, o que leva à queda no desempenho e desgaste prematuro dos blocos. Além disso, a própria FTL precisa de armazenamento flash para suas tabelas e estruturas de controle, que também serão frequentemente atualizadas.

Para amenizar esses problemas, os fabricantes de dispositivos de armazenamento em estado sólido provisionam uma área adicional de memória flash, acessível somente ao controlador do SSD, com as seguintes finalidades:

- Prover espaço adicional para melhorar o desempenho dos algoritmos de coleta de lixo e de nivelamento de desgaste;
- Prover espaço para as tabelas e estruturas necessárias à FTL;
- Substituir páginas e blocos com defeito de fabricação ou com falhas detectadas na operação.

Esse provisionamento em excesso (do inglês *Over Provisioning*) costuma variar entre 0% e 28%, sendo maior em SSDs de maior qualidade, com perfil voltado à operação intensiva em escrita, o que garante uma vida útil mais longa. A Tabela 21.3 traz alguns valores típicos observados em SSDs de mercado:

Cap. física real	Cap. anunciada	Prov. em excesso	Aplicação
128 GB	128 GB	0%	SSD básico
128 GB	120 GB	7%	leitura intensiva
128 GB	100 GB	28%	escrita intensiva

Tabela 21.3: Provisionamento em excesso em alguns SSDs de mercado.

Além da diferença entre a capacidade real e a anunciada do dispositivo, outra fonte de provisionamento em excesso em um dispositivo é a diferença entre sua capacidade anunciada em *Gigabytes* (1 GB = 1000³ bytes) e a capacidade real em *Gibibytes* (1 GiB = 1024³ bytes), da ordem de 7,4% para um SSD de 500 GB.

21.4 Interface de acesso

Como visto na seção 20.2, o sistema operacional deve dispor de *drivers* para interagir com cada controlador de dispositivo e solicitar operações de escrita e leitura de dados. Dispositivos de armazenamento são geralmente orientados a blocos, então cada operação de leitura/escrita é feita sobre blocos físicos individuais ou *clusters* (grupos de 2ⁿ blocos físicos). O acesso ao dispositivo normalmente é feito usando interação por eventos (Seção 20.5.2): o *driver* solicita uma operação de E/S ao controlador do dispositivo e suspende o fluxo de execução solicitante; quando a operação é completada, o controlador gera uma interrupção que é encaminhada ao *driver*, para retomar a execução e/ou solicitar novas operações. A estratégia de acesso direto à memória

(DMA, Seção 20.5.3) também é frequentemente usada, para aumentar o desempenho de transferência de dados.

O *controlador* de dispositivo normalmente é conectado a um barramento do computador. Por sua vez, os dispositivos são conectados ao controlador através de uma interface de conexão que pode usar diversas tecnologias. As mais comuns estão descritas a seguir:

IDE: *Integrated Drive Electronics*, padrão também conhecido como PATA (*Parallel ATA - Advanced Technology Attachment*); surgiu nos anos 1980 e durante muito tempo foi o padrão de interface de discos rígidos mais usado em computadores pessoais. Suporta velocidades de até 1 Gbit/s, através de cabos paralelos de 40 ou 80 vias. Cada barramento IDE suporta até dois discos, em uma configuração mestre/escravo.

SCSI: *Small Computer System Interface*, padrão de interface desenvolvida nos anos 1980, foi muito usada em servidores e estações de trabalho de alto desempenho. Um barramento SCSI suporta até 16 dispositivos e atinge taxas de transferência de até 2,5 Gbit/s (divididos entre os dispositivos conectados no mesmo barramento).

SATA: *Serial ATA* é o padrão de interface de discos em *desktops* e *notebooks* atuais. A transmissão dos dados entre o disco e o controlador é serial, atingindo taxas de transferência de 6 Gbit/s através de cabos com 7 vias.

SAS: *Serial Attached SCSI*, evolução do padrão SCSI, permitindo atingir taxas de transferência de até 12 Gbit/s em cada dispositivo conectado ao controlador. É usado em equipamentos de alto desempenho, como servidores.

NVMe: *Non-Volatile Memory Express*, padrão de interface mais recente, desenvolvido especificamente para dispositivos de estado sólido com memória flash (SSDs). Ele está conectado diretamente ao barramento de entrada/saída PCI Express e atinge velocidades de até 30 Gbits/s em dispositivos atuais.

USB MSC: *USB Mass Storage Class*, padrão de interface para dispositivos de armazenamento conectados ao computador através de uma porta USB (*Universal Serial Bus*); tem velocidade menor e é usado sobretudo com mídias removíveis, como discos externos, pendrives e cartões de memória SD.

É importante observar que esses padrões de interface não são de uso exclusivo em discos, muito pelo contrário. Há vários tipos de dispositivos que podem se conectar ao computador através dessas interfaces, como dispositivos de estado sólido (SSDs), leitores óticos (CD, DVD), unidades de fita magnética, *scanners*, etc.

Exercícios

1. Considere um escalonador de disco com os seguintes pedidos de leitura de blocos em sua fila, nessa ordem: 95, 164, 36, 68, 17 e 115. Determine todos os deslocamentos da cabeça de leitura do disco para atender esses pedidos e o número total de blocos percorridos, para as políticas FCFS, SSTF, SCAN, C-SCAN, LOOK e C-LOOK. O disco tem 200 setores, numerados de 0 a 199, e a cabeça de leitura acabou de atender um pedido para o bloco 50 e está subindo.

2. Você possui 4 discos rígidos de 8 TBytes cada. Para os arranjos desses discos em RAID 0 *Striping*, RAID 1 e RAID 5, apresente o espaço útil disponível, o número máximo de discos com falha e as velocidades máximas de leitura/escrita em relação a um disco isolado (por exemplo: 1×, 5×, ...).
3. Você tem 4 discos rígidos de 8 TB cada, que pode organizar de diversas formas. Indique os arranjos RAID que escolheria para obter:
 - (a) O maior espaço útil de disco.
 - (b) A maior tolerância a falhas de disco.
 - (c) A maior velocidade média de leitura.
 - (d) A maior velocidade média de escrita.
 - (e) Equilíbrio entre espaço útil, velocidades e tolerância a falhas.

Justifique/explice suas respostas.

Atividades

1. Construa um simulador de escalonamento de disco. O simulador deve receber como entrada o tamanho do disco (em blocos) e a sequência de números de blocos a acessar. Ele deve gerar como saída a sequência de blocos acessados e o deslocamento total da cabeça do disco (em blocos), para os algoritmos apresentados neste capítulo.

Referências

- M. Bjørling, J. González, and P. Bonnet. LightNVM: the Linux open-channel SSD subsystem. In *15th Usenix Conference on File and Storage Technologies, FAST'17*, page 359–373. USENIX Association, 2017. ISBN 9781931971362.
- D. Bovet and M. Cesati. *Understanding the Linux Kernel, 3rd edition*. O'Reilly Media, Inc, 2005.
- P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, 26:145–185, June 1994.
- L. Hutchinson. Solid-state revolution: in-depth on how SSDs really work. Technical report, Ars Technica, 2012.
- S. Larrivee. Solid state drives 101. Technical report, Cactus Technologies Inc, 2016.
- R. Love. *Linux Kernel Development, Third Edition*. Addison-Wesley, 2010.
- R. Micheloni, A. Marelli, and K. Eshghi. *Inside Solid State Drives (SSDs), 2nd edition*. Springer, 2018. ISBN 978-981-13-0598-6.
- D. Patterson and J. Henessy. *Organização e Projeto de Computadores*. Campus, 2005.

- D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *ACM SIGMOD International Conference on Management of Data*, pages 109–116. ACM, 1988.
- A. Silberschatz, P. Galvin, and G. Gagne. *Sistemas Operacionais – Conceitos e Aplicações*. Campus, 2001.
- SNIA. *Common RAID Disk Data Format Specification*. SNIA – Storage Networking Industry Association, March 2009. Version 2.0 Revision 19.