GERÊNCIA DE ENTRADA E SAÍDA

1. INTRODUÇÃO

Uma das principais funções dos sistemas operacionais é controlar todos os dispositivos de entrada/saída de um computador. Ele precisa emitir comandos para os dispositivos, capturar interrupções e manipular erros. Ele também fornece uma interface entre os dispositivos e o resto do sistema, interface esta que deve ser simples e fácil de usar. Tanto quanto possível, ela deve ser a mesma para todos os dispositivos, ou seja, deve independer do dispositivo. O código para entrada/saída representa uma parte significativa de todo o sistema operacional.

Sua implementação é estruturada através de camadas em um modelo semelhante ao apresentado para o sistema operacional como um todo. As camadas de mais baixo nível escondem características dos dispositivos das camadas superiores, oferecendo uma interface simples e confiável ao usuário e suas aplicações, conforme a figura 1.

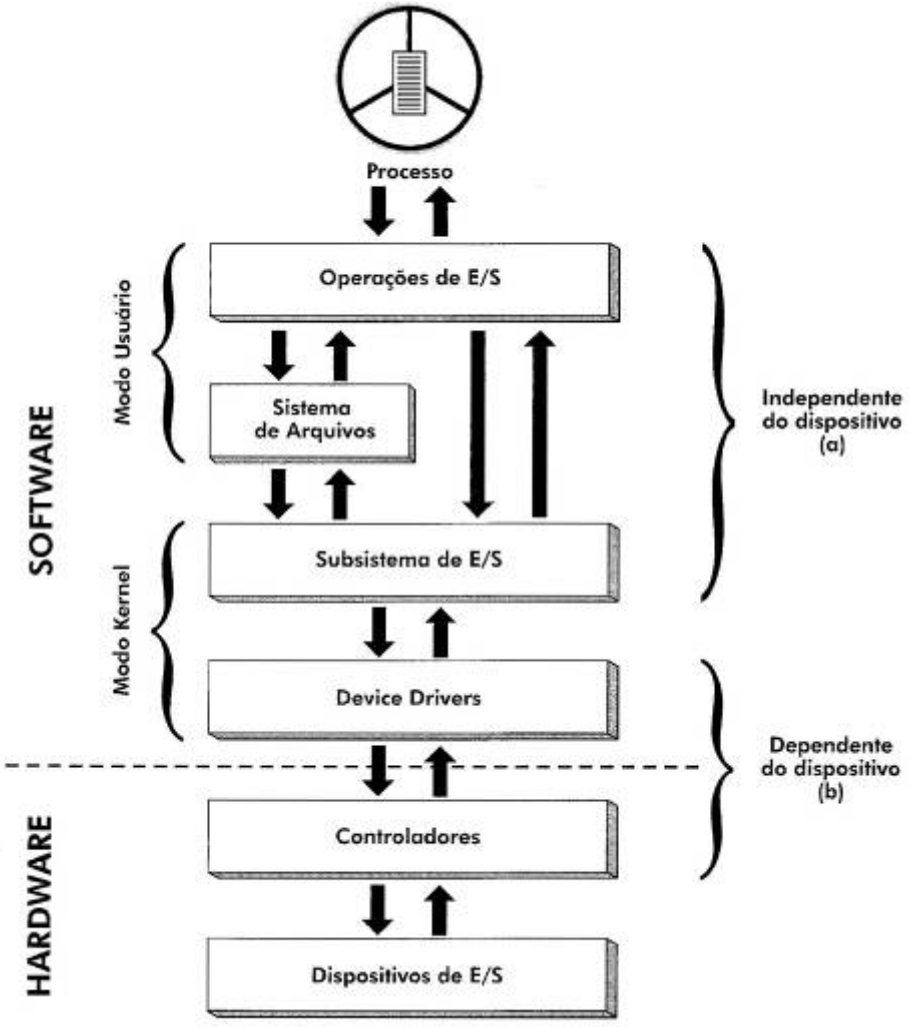

Figura 1: Gerência de dispositivos

A diversidade de dispositivos de E/S exige que o sistema operacional implemente uma camada, chamada subsistema de E/S, com a função de isolar a complexidade dos dispositivos da camada de sistemas de arquivo e da aplicação. Dessa forma, é possível ao sistema operacional ser flexível, permitindo a comunicação dos processos com qualquer tipo de periférico. Aspectos como velocidade de operação, unidade de transferência, representação dos dados, tipos de operações e demais detalhes de cada periférico são tratados pela camada de device driver, oferecendo uma interface uniforme entre o subsistema de E/S e todos os dispositivos.

As camadas são divididas em dois grupos, onde o primeiro grupo visualiza os diversos tipos de dispositivos do sistema de um modo único (figura 1a), enquanto o segundo é específico para cada dispositivo (figura 1b). A maior parte das camadas trabalha de forma independente do dispositivo.

2. PRINCÍPIOS DO HARDWARE DE ENTRADA/SAÍDA

Pessoas diferentes consideram o hardware de entrada/saída de formas diferentes. Para os engenheiros eletrônicos, ele é visto em termos de chips, fios, fontes de alimentação e demais componentes. Aos programadores interessa a interface com o software – os comandos que o hardware aceita, as funções que tais comandos executam e os erros que devem ser reportados nas diversas situações. Vale observar que a programação de diversos dispositivos de entrada/saída está intimamente ligada ao seu funcionamento interno.

2.1.Dispositivos de Entrada/Saída

Os dispositivos de entrada/saída podem ser grosseiramente divididos em duas categorias: **dispositivos de bloco e dispositivos de caractere**. Um **dispositivo de bloco** é aquele que armazena informações em blocos de tamanho fixo, cada um deles com o seu próprio endereço. Tamanhos comuns para tais blocos estão na faixa de 128 bytes até 1.024 bytes. A propriedade fundamental de todo dispositivo de bloco é a possibilidade de se ler ou escrever em qualquer dos blocos. Os discos são dispositivos de bloco.

O outro tipo de dispositivo de entrada/saída é o dispositivo de **caractere**. Tal dispositivo libera ou aceita um conjunto de caracteres sem respeitar nenhuma estrutura de bloco. Ele não é endereçável, e não aceita nenhuma operação de *seek*. Os terminais, as impressoras, as interfaces de rede, mouses, etc, que não têm as características de um disco podem ser vistos como dispositivos de caractere.

2.2.Controladoras de Dispositivos

As unidades de entrada/saída tipicamente são constituídas de duas partes distintas, uma mecânica e outra eletrônica. A parte eletrônica é chamada de **controlador de dispositivo** ou **adaptadora**. Nos mini e nos microcomputadores, a controladora é uma placa de circuito impresso que pode ser inserida na máquina. A parte mecânica é o dispositivo propriamente dito.

A placa controladora tem usualmente um conector no qual deve ser ligado um cabo vindo do dispositivo. Se a interface entre a controladora é uma interface padrão, ou uma das oficiais tal como as ANSI, IEEE ou ISSO, então diversas companhias podem fabricar controladoras e dispositivos seguindo as regras ditadas por tais interfaces.

É importante enfatizar a diferença entre controladora e dispositivo, pois os sistemas operacionais quase sempre tratam com as controladoras e não com os dispositivos. Os mini e microcomputadores usam geralmente o modelo de barramento único, mostrado na figura 2, para implementar a comunicação entre o processador e as controladoras de dispositivos. Os mainframes usam um esquema diferente, com vários barramentos e processadores para realizar somente operações de entrada/saída, denominados **canais de entrada/saída**, processadores esses que absorvem parte do trabalho do processador central.

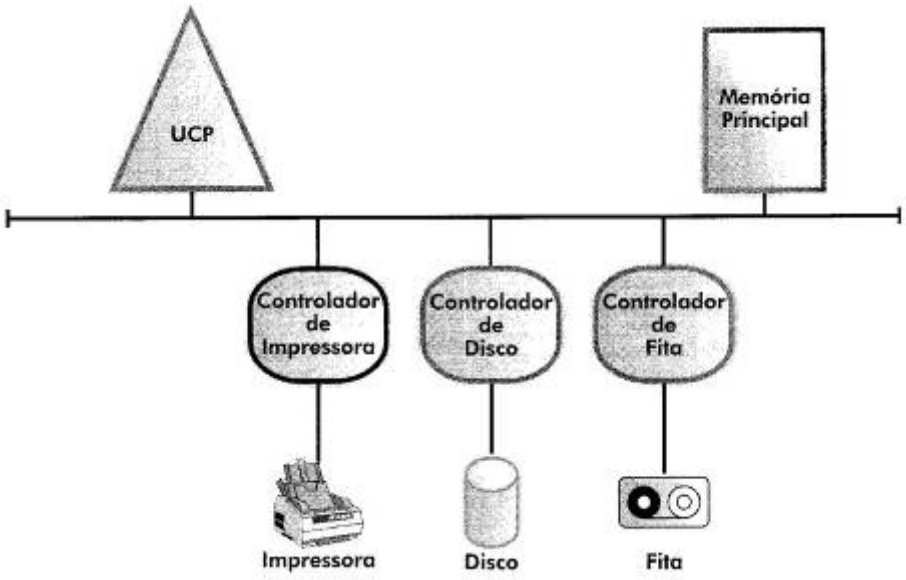

Figura 2: Um modelo de conexão do processador, memória, controladoras e dispositivos de entrada/saída

A interface entre a controladora e o dispositivo é muitas vezes uma interface de baixíssimo nível. Um disco, por exemplo, pode ser formatado com oito setores de 512 bytes por trilha. O que, na realidade, chega na unidade de disco é um fluxo serial de bits, começando por um preâmbulo, seguido pelos 4.096 bits de um setor, e de um checksum,

ou código de correção de erros (ECC). O preâmbulo é escrito quando da formatação do disco, e contém os números do cilindro e do setor, o tamanho do setor, além de alguns outros dados.

O trabalho da controladora é converter o fluxo serial de dados em um bloco de bytes (as instruções enviadas pelo device driver), realizando as correções de erros que se fizerem necessárias. Em operações de leitura, o controlador deve armazenar em seu buffer interno uma sequência de bits proveniente do dispositivo até formar um bloco de bytes que é montado bit a bit. Após o seu checksum ser verificado, e o bloco ter sido declarado correto, ele poderá ser copiado para um buffer de E/S na memória principal. A transferência do bloco do buffer interno do controlador para o buffer de E/S da memória principal pode ser realizada pela CPU ou por um controlador de DMA.

A controladora para um terminal de vídeo também trabalha como um dispositivo serial, num nível bem baixo. Ela lê da memória os bytes contendo os caracteres a serem colocados no vídeo, e gera os sinais necessários a modular o feixe de raios catódicos, que resultarão na escrita do caractere no vídeo. A controladora também gera os sinais necessários ao retraço horizontal após a varredura de uma linha, bem como aqueles necessários ao retraço vertical, após a tela inteira ter sido varrida. Se não existisse a controladora de vídeo, o programador do sistema operacional seria obrigado a programar explicitamente a varredura analógica do tubo. Com a controladora, ela apenas a inicializa com uns poucos parâmetros, tais como o número de caracteres por linha, e o número de linhas por tela, e deixa que ela cuide da programação do feixe eletrônico.

Cada controladora tem uns poucos registradores que são usados na comunicação com o processador. Em alguns computadores, tais registradores são parte do espaço de endereçamento da memória. Tal esquema é denominado **entrada/saída mapeada na memória**. A família 680x0, por exemplo, usa esse método. Outros computadores usam um espaço de endereçamento especial para entrada/saída, com cada controladora alocada a uma certa parte desse espaço. A figura 3 mostra os endereços de entrada/saída e os vetores de interrupção alocados a algumas das controladoras do IBM PC. A atribuição de endereços de entrada/saída a dispositivos é feita através de uma lógica de decodificação associada à controladora.

Controladora de entrada/saída	Endereço de entrada/saída	Vetor de interrupção
Clock	040 - 043	8
Teclado	060 - 063	9
RS-232 secundário	2F8 - 2FF	11
Disco rígido	320 - 32F	13
Impressora	378 - 37F	15
Monitor monocromático	380 - 3BF	—
Monitor colorido	3D0 - 3DF	—
Disquete	3F0 - 3F7	14
RS-232 primário	3F8 - 3FF	12

Figura 3: Exemplos de controladoras do IBM PC, seus endereços de entrada/saída e seus vetores de interrupção

O sistema operacional faz entrada/saída, escrevendo comandos nos registros das controladoras. A controladora da unidade de disquete do IBM PC, por exemplo, aceita 15 comandos diferentes, como READ, SEEK, WRITE, FORMATE e RECALIBRATE. Muitos desses comandos têm parâmetros que também precisam ser carregados nos registradores da controladora. Quando um comando é aceito, o processador pode deixar a controladora trabalhando sozinha, e ir tratar de outra coisa. Quando a execução do comando termina, a controladora gera uma interrupção, de forma a permitir que o sistema operacional ganhe o processador, e verifique o resultado da operação. O processador obtém o resultado e o estado do dispositivo, lendo um ou mais bytes de informação contidos nos registradores da controladora.

2.3.Acesso Direto à Memória (DMA)

Muitas controladoras, especialmente aquelas desenvolvidas para dispositivos de blocos, suportam **operações de acesso direto à memória** ou **DMA**. Para explicar o funcionamento do DMA, vamos primeiro examinar a leitura de disco quando tal técnica não é usada. Primeiro, a controladora lê o bloco do drive serialmente, bit a bit, até que todo o bloco esteja em seu buffer interno. A seguir ela calcula o ckecksum para verificar a ocorrência de erros. Depois, a controladora gera uma interrupção. Quando o sistema operacional começa a rodar, ele pode ler o bloco de disco escrito no buffer da controladora, um bit por vez, executando um loop, com cada interação lendo um byte do registrador da controladora, e armazenando tal informação na memória.

Naturalmente que tal loop gasta muito tempo do processador. A DMA foi inventada justamente para tirar do processador este trabalho de baixo nível. Quando ela é usada, o processador fornece à controladora dois itens de informação, além do endereço do bloco do disco: o endereço de memória para onde o bloco deve ir e o número de bytes a serem transferidos, conforme mostrado na figura 4.

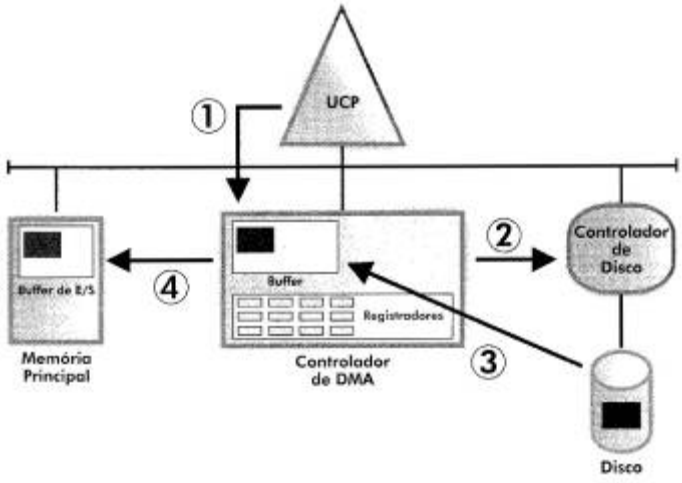

Figura 4: uma transferência DMA é completamente realizada pela controladora

Após a controladora ter lido todo o bloco do dispositivo, tê-lo colocado em seu buffer e ter verificado o ckecksum, ela copia o primeiro byte para o endereço especificado no registrador de memória da controladora DMA. A seguir ela incrementa o valor de tal endereço, e decrementa o do contador de bytes do número que acabou de ser transferido. Este processo continua até que o valor do contador de bytes chegue a zero, momento em que a controladora gera uma interrupção. Quando o sistema operacional começa a sua execução para tratar da interrupção, ele não vai precisar ler o bloco para a memória, pois ele já está lá.

Por que a controladora não armazena os bytes na memória tão logo ela os obtém do disco, ou seja, por que existe o buffer interno? A razão deste buffer é o fato de que, quando o disco começa sua transferência, os bits vão chegando do disco numa taxa constante, esteja ou não a controladora pronta para recebê-los. Se ela tentar escrever os dados diretamente na memória, seria obrigada a acessar o barramento a cada palavra transferida. Se o barramento estiver ocupado, sendo usado por algum outro dispositivo, a controladora teria que esperar. Se a próxima palavra chegar do disco antes que a anterior tenha sido escrita, a controladora deverá armazenar esta última em algum lugar determinado. Se o barramento estiver sendo muito usado, ela pode acabar armazenando muitas palavras, tendo então muito trabalho burocrático para tomar conta de todo o processo. Quando o bloco vai para um buffer interno, o barramento não é usado até que a DMA comece, de forma que o projeto da controladora fica muito mais simples, pois a transferência de informação para a memória usando DMA não é uma operação crítica no que diz respeito ao tempo.

O processo de bufferização de dois passos listado acima tem implicações importantes na performance do sistema operacional. Enquanto um dado está sendo transferido de uma controladora para a memória ou vice-versa, o próximo setor vai estar passando debaixo da cabeça, e os bits vão estar chegando na controladora. Observe que ambos os eventos, setor passando debaixo da cabeça e dados chegando na controladora, acontecem ao mesmo tempo. Como as controladoras mais simples não podem fazer entrada e saída ao mesmo tempo, enquanto está se realizando uma transferência de memória, o setor que estiver passando debaixo da cabeça do disco é perdido.

Desta forma, só estará apta a ler blocos salteados, de forma que a leitura de uma trilha completa vai demorar duas rotações completas do disco, uma para leitura dos blocos pares e outra para leitura dos blocos ímpares. Se o tempo para transferir um bloco da controladora para a memória através do barramento é maior do que o tempo de leitura de um bloco do disco, será então necessário ler um bloco e então saltar dois blocos.

O salto de blocos para dar tempo à controladora de transferir dados para a memória é denominado **intercalação**. Quando o disco é formatado, os blocos são numerados de maneira a levar em conta o fator de intercalação. Na figura 5a aparece um disco com oito blocos por trilha, sem intercalação. Na figura 5b aparece o mesmo disco com uma única intercalação e na figura 5c, é mostrada a intercalação dupla.

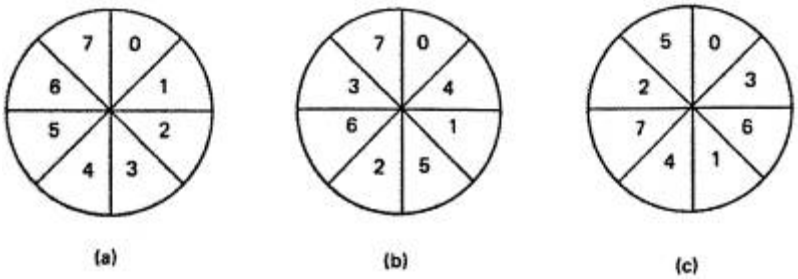

Figura 5: (a) sem intercalação, (b) intercalação simples, (c) intercalação dupla

A idéia de numerar os blocos desta forma é para permitir que o sistema operacional leia blocos consecutivos, e mesmo assim consiga explorar o máximo de velocidade que o hardware permite. Se os blocos fossem numerados na forma da figura 5a, mas a controladora só pudesse ler blocos alternados, um sistema operacional que alocasse um arquivo de oito blocos em blocos consecutivos do disco precisaria gastar oito rotações do disco para que os blocos de 0 a 7 pudessem ser lidos.

2.4.Device Drivers

O **device driver**, ou somente driver, tem como função implementar a comunicação do subsistema de E/S com os dispositivos, através de controladores. Enquanto o subsistema de E/S trata de funções ligadas a todos os dispositivos, os drivers tratam apenas dos seus aspectos particulares.

Os drivers têm como função receber comandos gerais sobre acessos aos dispositivos e traduzi-los para comandos específicos, que poderão ser executados pelos controladores (figura 6). Cada driver manipula somente um tipo de dispositivo ou grupo de dispositivos semelhantes. Normalmente, um sistema possui diferentes drivers, como drivers para disco, fita magnética, rede e vídeo.

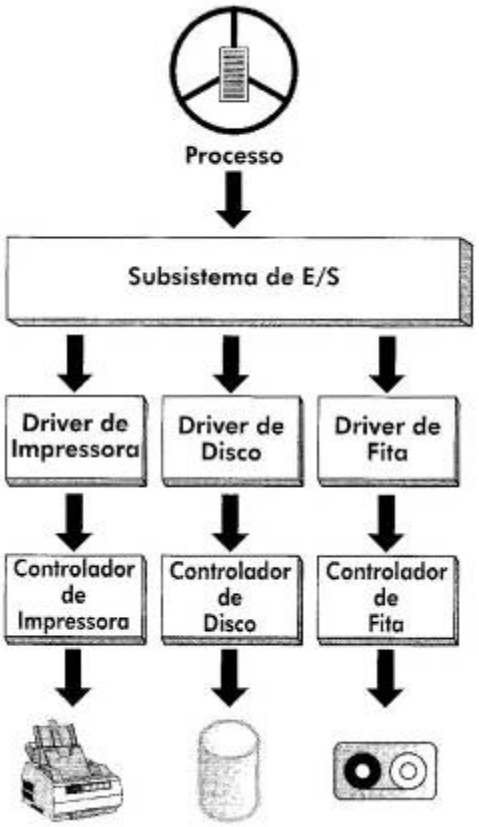

Figura 6: Device drivers

O device driver está integrado diretamente às funções do controlador, sendo o componente do sistema que reconhece as características particulares do funcionamento de cada dispositivo de E/S, como número de registradores do controlador, funcionamento e comando do dispositivo. Sua função principal é receber comandos abstratos do subsistema de E/S e traduzi-los para comandos que o controlador possa entender e executar. Além disso, o driver pode realizar outras funções, como a inicialização do dispositivo e seu gerenciamento.

Por exemplo, na leitura síncrona de um dado em disco, o driver recebe a solicitação de leitura de um determinado bloco e informa ao controlador o disco, cilindro, trilha e setor que o bloco se localiza, iniciando, dessa forma, a operação (figura 7). Enquanto se realiza a leitura, o processo que solicitou a operação é colocado no estado de espera até que o controlador avise a CPU do término da operação através de uma interrupção que, por sua vez, ativa novamente o device driver. Após verificar a inexistência de erros, o device driver transfere as informações para a camada superior. Com os dados disponíveis, o processo pode ser retirado do estado de espera e retornar ao estado de pronto para continuar seu processamento.

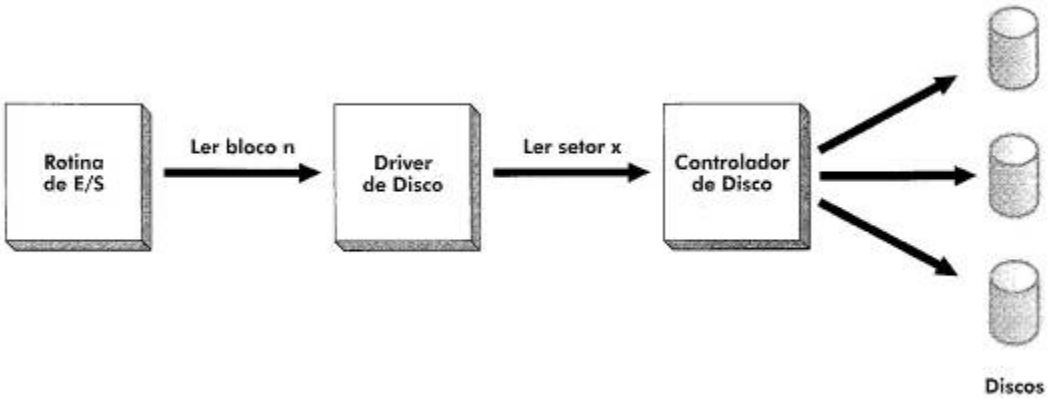

Figura 7: Driver de disco

Os device drivers fazem parte do núcleo do sistema operacional, sendo escritos geralmente em assembly. Como os drivers são códigos reentrantes que executam em modo kernel, qualquer erro de programação pode comprometer por completo o funcionamento do sistema. Por isso, um driver deve ser cuidadosamente desenvolvido e testado.

Devido ao alto grau de dependência entre os drivers e o restante do sistema, os fabricantes desenvolvem, para um mesmo dispositivo, diferentes drivers, um para cada sistema operacional. Sempre que um novo dispositivo é instalado, o driver do dispositivo deve ser adicionado ao núcleo do sistema.

2.5.Subsistema de Entrada e Saída

O **subsistema de entrada e saída** é responsável por realizar as funções comuns a todos os tipos de dispositivos, ficando os aspectos específicos de cada periférico como responsabilidade do device drivers. Dessa forma, o subsistema de E/S é a parte do sistema operacional que oferece uma interface uniforme com as camadas superiores.

Cada dispositivo trabalha com unidades de informação de tamanho diferentes, como caracteres ou blocos. O subsistema de E/S é responsável por criar uma unidade lógica de transferência independente do dispositivo e repassá-lo para os níveis superiores, sem o conhecimento do conteúdo da informação.

Normalmente, o tratamento de erros nas operações de E/S é realizado pelas camadas mais próximas ao hardware. Existem, porém, certos erros que podem ser tratados e reportados de maneira uniforme pelo sistema de arquivos, independente do dispositivo. Erros como a gravação em dispositivos de entrada, leitura em dispositivos de saída e operações em dispositivos inexistentes podem ser tratados neste nível.

Todos os dispositivos de E/S são controlados, com o objetivo de obter o maior compartilhamento possível entre os diversos usuários de forma segura e confiável. Alguns dispositivos, como os discos, podem ser compartilhados, simultaneamente, entre diversos usuários, sendo o sistema operacional responsável pela integridade dos dados

acessados. Outros, como as impressoras, devem ter acesso exclusivo, e o sistema deve controlar seu compartilhamento de forma organizada. O subsistema de E/S é responsável também por implementar todo um mecanismo de proteção de acesso aos dispositivos. No momento que o usuário realiza uma operação de E/S, é verificado se o seu processo possui permissão para realizar a operação.

A *bufferização* é outra tarefa realizada por esse subsistema. Essa técnica permite reduzir o número de operações de E/S, utilizando uma área de memória intermediária chamada de buffer. Por exemplo, quando um dado é lido do disco, o sistema traz para a área de buffer não só o dado solicitado, mas um bloco de dados. Caso haja uma solicitação de leitura de um novo dado que pertença ao bloco anteriormente lido, não existe a necessidade de uma nova operação de E/S, melhorando desta forma a eficiência do sistema.

Uma das principais funções do subsistema de E/S é criar uma interface padronizada com os device drivers. Sempre que um novo dispositivo é instalado no computador, é necessário que um novo driver seja adicionado ao sistema. O subsistema de E/S deve oferecer uma interface padronizada que permita a inclusão de novos drivers sem a necessidade de alteração da camada de subsistema de E/S.

3. PRINCÍPIOS DE SOFTWARE DE ENTRADA/SAÍDA

Os objetivos gerais do software de entrada/saída são muito simples de ser colocados. A idéia básica é organizá-lo como uma série de níveis, sendo o mais baixo deles voltado a esconder do usuário as peculiaridades do hardware e os demais responsáveis por apresentar-lhes uma interface boa e simples de usar.

3.1. Objetivos do Software de Entrada/Saída

Um conceito-chave no projeto do software de entrada/saída é a **independência dos dispositivos**. De acordo com tal conceito deve ser possível escrever programas que possam ser usados em arquivos armazenados tanto numa unidade de disquete quanto numa de disco rígido, sem ser preciso fazer qualquer mudança no programa para adequá-lo ao tipo de dispositivo que está sendo usado. Em resumo, deve ser possível digitar o comando e ter certeza de que ele funciona tanto com a entrada e a saída em disquete quanto em disco rígido, ou mesmo indo ou vindo de um terminal. É tarefa do sistema operacional resolver os problemas que decorrem do fato de estes dispositivos serem muito diferentes uns dos outros, e, por conta disso, requerem drivers para gerenciá-los.

Um outro conceito intimamente ligado à independência dos dispositivos é o da **uniformidade de identificação**. O nome de um arquivo ou de um dispositivo deve ser

simplesmente um string de caracteres ou um número inteiro, e não deve de forma alguma depender do dispositivo.

Outro ponto muito importante relacionado ao software de entrada e saída é a **manipulação de erros**. Em geral, os erros devem ser tratados o mais próximo possível do hardware. Se a controladora descobrir um erro de leitura, ela própria deve tentar corrigir o erro. Se não for possível, então o driver do dispositivo deve tratá-lo, muito provavelmente tentando apenas reler o bloco. Muitos erros são transientes, tais como os causados por partículas de poeira na cabeça de leitura/gravação, e não vão persistir se a operação for repetida. Os níveis mais altos do software só devem tomar conhecimento de um erro se os mais baixos não puderem resolvê-los. Na maioria dos casos, a recuperação de erros deve ser realizada de forma transparente pelos níveis mais baixos do software ou até mesmo pelo hardware.

Um quarto aspecto diz respeito ao emprego da **transferência** síncrona (bloqueada) ou da assíncrona (dirigida por interrupções). A operação de entrada/saída é essencialmente assíncrona, ou seja, o processador inicia a transferência e vai fazer alguma outra coisa, até que chegue a interrupção proveniente do dispositivo em questão. Já os programas de usuário são muito mais simples de escrever se as operações de entrada/saída forem bloqueadas – após um comando READ, o programa é automaticamente suspenso até que o dado esteja disponível no buffer. É função do sistema operacional fazer com que operações sejam fisicamente dirigidas por interrupções parecerem bloqueadas aos programas de usuário.

O último conceito a ser abordado aqui é o que se refere aos **dispositivos compartilhados e os dedicados**. Alguns dispositivos de entrada/saída, como os discos, podem ser usados por vários usuários ao mesmo tempo. Não há nenhum problema que decorra do fato de muitos usuários terem arquivos abertos no mesmo disco, ao mesmo tempo. Outros dispositivos tais como as impressoras, devem ser dedicados a um único usuário, até que este venha a terminar o seu trabalho. A introdução de dispositivos dedicados leva a uma série de problemas. Novamente, o sistema operacional precisa manipular tanto os dispositivos dedicados quanto os compartilháveis de maneira a fazer com que o usuário tenha a impressão de estar interagindo com um dispositivo muito simples e versátil.

Tais objetivos podem ser alcançados de uma forma didática e eficiente, estruturando o software de entrada/saída em quatro níveis:

- Manipuladores de interrupção
- Drivers de dispositivo
- Software do sistema operacional independente
- Software do nível do usuário

3.2.Manipuladores de Interrupção

As interrupções devem ser colocadas nos níveis mais baixos do sistema operacional, de forma que o mínimo possível de partes do sistema tenham que lidar com ela. A melhor maneira de escondê-las é fazer com que todo o processo que inicie uma operação de entrada/saída fique bloqueado até o término de tal operação, fato este que é sinalizado através de uma interrupção. O processo pode autobloquear-se executando, por exemplo, um DOWN em um semáforo ou um WAIT em uma variável de condição ou um RECEIVE em uma mensagem.

Quando a interrupção ocorrer, a rotina de tratamento de interrupção faz tudo o que for preciso para desbloquear o processo que iniciou a operação. Em alguns sistemas, basta executar um UP no semáforo. Em outros, um SIGNAL em uma variável de condição do monitor. Ainda em outros, deve ser enviada uma mensagem ao processo bloqueado. Em todos os casos citados, o efeito da interrupção será fazer com que um processo anteriormente bloqueado esteja de novo apto a executar.

3.3.Drivers de Dispositivos

Todo o código que depende do dispositivo está no driver de tal dispositivo. Cada driver manipula um tipo de dispositivo. As controladoras, conforme já visto, devem ter um ou mais registradores de dispositivo, usados para mandar os comandos a serem executados pela controladora. O driver de dispositivo emite tais comandos e verifica se eles foram corretamente executados. Desta forma, o driver de disco é a única parte do sistema operacional que conhece quantos registradores tem a controladora do disco, e para que eles são usados. Somente trata com setores, trilhas, cilindros, movimentos do braço, cabeça, fator de intercalação, acionamento de motores, e outros aspectos mecânicos que fazem com que o disco venha a funcionar corretamente.

Em termos gerais, o trabalho de um driver de dispositivo é o de aceitar requisições abstratas por parte do software dependente do dispositivo, que roda acima dele, e fazer com que tal requisição seja executada.

O primeiro passo na realização de uma requisição de entrada/saída é traduzi-la de uma forma abstrata para uma mais concreta. Para um driver de disco, isto significa saber onde no disco está fisicamente localizado um bloco, verificar se um motor da unidade está ligado, determinar se a cabeça está posicionada sobre o cilindro correto, etc. Em resumo, ele deve determinar quais as operações da controladora são necessárias e em que ordem.

Uma vez decididos os comandos a serem enviados à controladora, o driver começa a escrevê-los nos registradores da mesma. Algumas delas só podem tratar de um comando por vez. Outras estão preparadas para aceitar uma lista ligada de comandos, que elas podem executar sozinhas, sem nenhuma ajuda do sistema operacional.

Após a emissão de um comando, podem acontecer uma de duas situações. Em alguns casos, o driver do dispositivo deve esperar até que a controladora tenha feito alguma coisa, de forma que ele se autobloqueie até a chegada da interrupção que o desbloqueia. Em outros casos, no entanto, a operação termina sem nenhum retardo, de modo que o driver não precisa se bloquear.

No primeiro caso, o driver bloqueado vai ser acordado por uma interrupção. No último, ele nunca dorme. Em ambos os casos, após o término da operação, ele deve verificar se houve a ocorrência de algum erro. Se tudo estiver bem, o driver deve ter dados a serem passados para o software, independente do dispositivo. Por fim, ele retorna algumas informações de estado, que em alguns casos, podem ser a identificação de erros ocorridos no decorrer da operação. Se outras requisições estiverem aguardando na fila, uma delas deve ser escolhida e iniciada. Se não houver ninguém na fila, o driver fica bloqueado esperando pela próxima requisição.

3.4.Software de Entrada/Saída Independente do Dispositivo

Apesar de parte do software de entrada/saída ser específico para cada dispositivo, uma grande parte dele é independente. O limite exato entre os drivers e o software independente varia com o sistema operacional, pois algumas funções que podem ser realizadas de forma independente do dispositivo podem ser também realizadas no driver por motivos de eficiência, ou por alguma outra razão, a critério do projetista. As funções mostradas na figura 8 são típicas de serem realizadas pelo software independente do dispositivo.

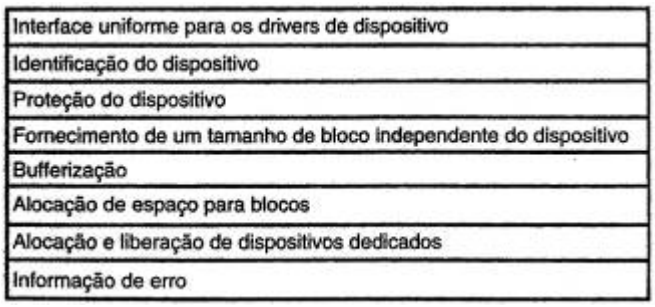

Figura 8: Funções do software de entrada/saída independente do dispositivo

O objetivo básico deste software é a realização de funções de entrada/saída que são comuns a todos os dispositivos, e fornecer uma interface uniforme para o nível do software do usuário.

Uma questão importante é a forma como os objetos, como arquivos e dispositivos e entrada/saída, são identificados. O software independente do dispositivo toma conta do processo de mapeamento do nome simbólico do dispositivo, para seu driver específico.

Intimamente relacionado com a identificação, está o problema da segurança. Como pode o sistema operacional evitar que usuários não autorizados acessem os dispositivos? Em alguns sistemas, tal como o MS-DOS, não há nenhum tipo de proteção. Qualquer processo pode fazer tudo o que bem entenda. Já nos sistemas para máquinas de maior porte, o acesso a dispositivos de entrada/saída por parte de processos de usuários é terminantemente proibido.

Discos diferentes podem ter tamanhos de setor também diferentes. É função do software independente esconder este fato, fornecendo um tamanho de bloco uniforme para os níveis mais altos, por exemplo, tratando vários setores como um único bloco lógico. Desta forma, os níveis mais altos só tratam com dispositivos abstratos, usando o mesmo tamanho lógico de bloco, independente do tamanho físico do setor. Da mesma forma, alguns dispositivos de caractere liberam seus dados um byte por vez, enquanto outras os liberam em unidades de informação maiores. Tais diferenças também não podem aparecer.

Outro aspecto importante é a bufferização, tanto para dispositivos de bloco quanto para os de caractere. Para os dispositivos de bloco, o hardware normalmente lê/escreve blocos inteiros de dados de uma vez, mas os processos de usuários lêem tais informações em unidades arbitrárias. Se um processo escrever metade de um bloco, o sistema operacional deve manter tais dados num buffer interno, até que o resto dos dados seja escrito, e o bloco completo possa ser transferido para o disco. Para dispositivos de caractere, os usuários podem escrever dados no sistema mais rapidamente que a velocidade com que eles possam ser colocados na saída, necessitando assim de bufferização. A entrada via teclado também pode chegar antes de poder ser tratada, também precisando ser colocadas em um buffer.

Quando um arquivo é criado, e preenchido com dados, novos blocos de disco devem ser alocados a tal arquivo. Para realizar esta alocação, o sistema operacional precisa de uma lista ou um mapa de bits contendo os blocos livres do disco, mas o algoritmo para alocar um bloco livre é independente do dispositivo, e pode ser executado acima do nível do driver.

A manipulação de erros é realizada em sua quase totalidade pelos drivers. A maioria dos erros é altamente dependente do dispositivo, de forma que apenas o driver sabe o que fazer. Um erro típico é o causado por um bloco de disco que foi danificado e não pode mais ser lido. Após o driver tentar lê-lo um certo número de vezes, ele desiste e informa ao software independente. A forma de tratar o erro daí para frente não mais depende do dispositivo. Se o erro ocorreu quando da leitura de um arquivo de usuário, deve ser suficiente reportar o erro a tal usuário. No entanto, se ele ocorreu durante a leitura de uma estrutura de dados crítica ao sistema, tal como bloco que contém um mapa

de bits com os blocos livres, o sistema operacional não tem escolha, a não ser imprimir uma mensagem de erro e terminar a execução.

3.5.Software de Entrada/Saída no Espaço do Usuário

Apesar da maior parte do software de entrada/saída estar dentro do sistema operacional, uma pequena porção dele consiste em rotinas de biblioteca, ligadas junto com os programas de usuário, e mesmo em programas inteiros rodando fora do kernel. As chamadas de sistema, incluindo as chamadas de entrada/saída normalmente são executadas por procedimentos da biblioteca. Quando um programa em C executa a chamada

count = write (fd, buffer, nbytes);

o procedimento *write* será ligado com o programa, estando contido no código binário presente na memória para ser executado. A coleção de todas as rotinas é parte do sistema de entrada/saída.

Enquanto tais procedimentos não fazem nada mais que colocar seus parâmetros na posição apropriada para a chamada de sistema, existem outras rotinas da biblioteca que fazem um trabalho efetivo. Em particular, a formação de entradas e saídas é feita por procedimentos da biblioteca. Um exemplo extraído da linguagem C é a rotina *printf*, que tem como entrada um string e algumas variáveis, e constrói a partir destas entradas um string equivalente em ASCII, chamando então WRITE para colocá-lo na saída.

Nem todo software de entrada/saída no nível dos usuários pertence às bibliotecas de procedimentos. Outra categoria importante é a dos sistemas de spooling. O **spooling** é uma forma de se tratar com dispositivos dedicados que fazem parte de um sistema multiprogramado. Considere um dispositivo típico desta categoria: a impressora de linha. Apesar de ser tecnicamente fácil deixar que qualquer processo de usuário abra o arquivo especial da impressora, suponha que um processo o abra e não faça nada por horas e horas. Nenhum outro processo vai poder imprimir nada, enquanto este arquivo não for fechado.

Em vez disso, é criado um processo especial, denominado **daemon**, e um diretório também especial, **diretório de spooling**. Para imprimir um arquivo, o processo primeiro gera todo o arquivo a ser impresso e o coloca no diretório de spooling. É tarefa do daemon, que é o único processo que pode abrir o arquivo especial da impressora, imprimir os arquivos constantes deste diretório. O problema de algum processo manter este arquivo aberto desnecessariamente é resolvido protegendo-se do acesso de usuários comuns.

O spooling não é usado apenas para impressoras, sendo também útil em outras situações. Por exemplo, a transferência de arquivos em uma rede usa o conceito de daemon. Para enviar um arquivo para algum ponto da rede, o usuário deve colocá-lo no

diretório de spooling da rede. Mais tarde o daemon pega este arquivo e o transmite para o destino correto.

A figura 9 resume o sistema de entrada/saída, mostrando todos os níveis e as principais funções de cada um deles. As setas na figura mostram o sentido do fluxo de controle.

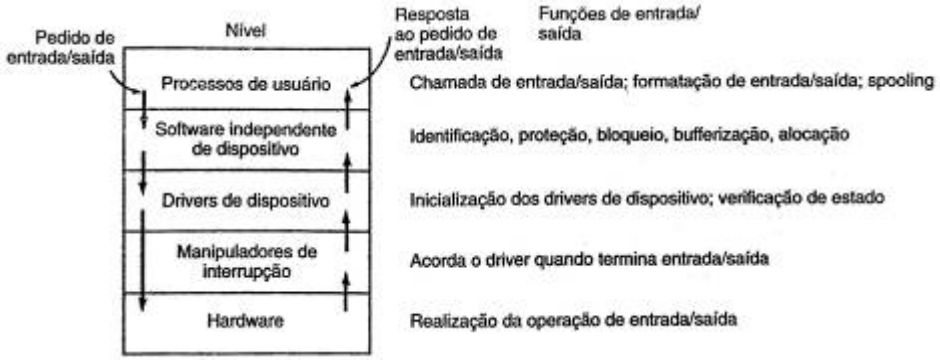

Figura 9: Níveis do sistema de entrada/saída

4. DISCOS

Quase todos os computadores possuem discos para armazenamento de informação. Armazenar a informação em disco tem as seguintes vantagens sobre guardá-las na memória principal:

1. A capacidade de armazenamento disponível é muito grande
2. O preço por bit armazenado é muito menor
3. A informação não se perde quando se desliga a máquina

4.1.Hardware de Disco

Um disco magnético é constituído por vários discos sobrepostos, unidos por um mesmo eixo vertical, girando a uma velocidade constante. Cada disco é composto por trilhas concêntricas, que por sua vez são divididas em setores; apesar de os setores situados na parte mais externa do disco serem fisicamente maiores que os mais internos, o espaço extra não é utilizado. As trilhas dos diferentes discos que ocupam a mesma posição vertical formam um cilindro. Para a superfície de cada disco existe um mecanismo de leitura/gravação. Todos os mecanismos de leitura/gravação são conectados a um braço que se movimenta entre os vários cilindros dos discos no sentido radial (figura 10).

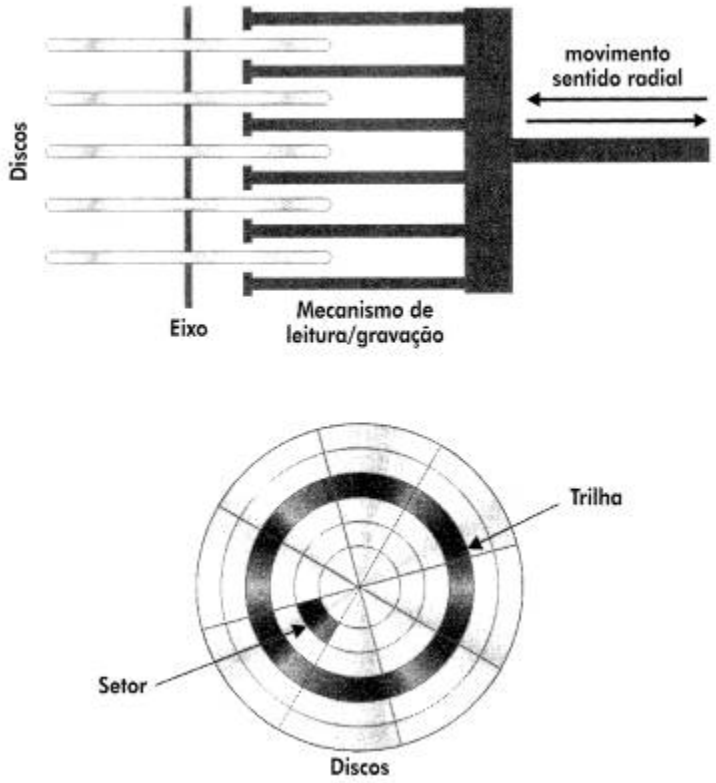

Figura 10: Estrutura de um disco magnético

Uma característica desse dispositivo que é muito útil ao driver de disco é a possibilidade da controladora fazer *seeks* em duas ou mais unidades ao mesmo tempo. Isto é conhecido como **seeks sobrepostos**. Enquanto a controladora e o software estiverem esperando que o seek de uma unidade se complete, a controladora pode iniciar um seek de outra unidade. Muitas controladoras também podem ler ou escrever em uma unidade enquanto estão em processo vários seeks em outras unidades, mas nenhuma pode ler ou escrever em mais de uma unidade ao mesmo tempo. A possibilidade de se realizar dois ou mais seeks ao mesmo tempo reduz consideravelmente o tempo médio de acesso.

4.2.Algoritmos para Escalonamento do Braço do Disco

O tempo para ler ou escrever um bloco do disco é determinado por três fatores: o tempo de seek (o tempo que o braço leva para se mover até o cilindro desejado), a latência rotacional (o tempo para que o setor procurado se coloque embaixo da cabeça), e o tempo de transferência propriamente dito. Para a grande maioria dos discos o tempo dominante é o tempo de seek, de forma que a redução do tempo médio do seek pode melhorar consideravelmente a performance do sistema.

Se o driver de disco só aceitar uma requisição por vez, executando-as em ordem de chegada, ou seja, de acordo com a política do Primeiro-a-chegar, Primeiro-a-ser-servido (FCFS), muito pouco pode ser feito para otimizar o tempo de seek. Entretanto existe uma

outra estratégia que pode ser usada quando o disco está muito carregado. Tal estratégia funciona fazendo com que, enquanto o braço estiver buscando o cilindro referente a uma requisição de um determinado processo, outras requisições possam ser geradas por outros processos. Muitos drivers de disco mantêm uma tabela indexada pelo número do cilindro, com todas as requisições pendentes para cada cilindro, encadeadas juntas, numa lista ligada encabeçada pelas entradas da tabela.

Dado este tipo de estrutura, podemos melhorar o algoritmo FCFS. Para ver como, considere um disco de 40 cilindros. Em primeiro lugar chega um pedido de leitura de um bloco cilindro 11. Enquanto se dá o seek para tal cilindro, novos pedidos chegam para os cilindros 1, 36, 16, 34, 9 e 12, nesta ordem. Eles entram em uma tabela de requisições pendentes, como uma lista separada para cada cilindro. Tais requisições aparecem na figura 11.

Quando terminar o acesso corrente (para o cilindro 11), o driver tem que escolher quais das requisições vai ser atendida a seguir. Usando o algoritmo FCFS, a próxima a ser atendida será a do cilindro 1, depois a do 36, e assim por diante. Observe que este algoritmo requer movimentos do braço por 10, 35, 20, 18, 25 e 3 cilindros, respectivamente, perfazendo um total de 111 cilindros percorridos.

Alternativamente o driver poderá atender sempre a requisição que precisar do menor deslocamento do braço de forma a minimizar o tempo de seek. Dados os pedidos da figura 11, esta seqüência será 12, 9, 16, 1, 34 e 36, conforme mostra a linha sinuosa da figura 11. Seguindo esta seqüência, o braço desloca-se por 1, 3, 7, 15, 33 e 2 cilindros, num total de 61 cilindros. Estes algoritmos, chamado do **menor seek primeiro** (SSF), reduz quase metade do movimento do braço, quando comparado com o FCFS.

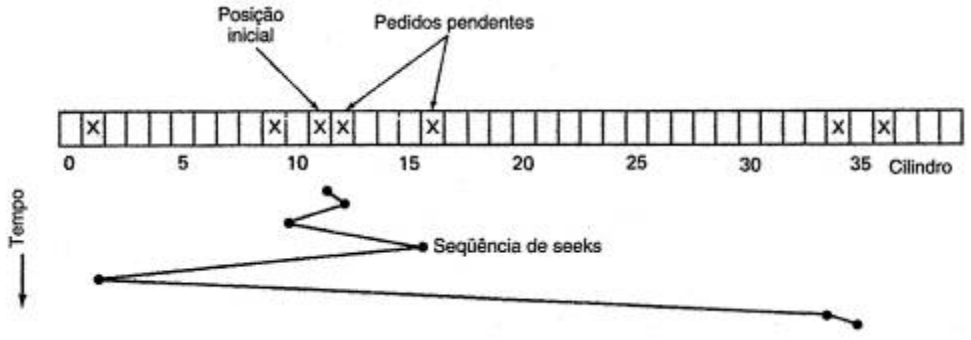

Figura 11: O algoritmo de escalonamento de disco SSF

Infelizmente, o SSF tem um problema. Suponha que cheguem novas requisições enquanto as da lista da figura 11 estão sendo atendidas. Por exemplo, se, após ir para o cilindro 16, ocorrer um novo pedido para o 8, tal requisição será atendida antes da do cilindro 1. Se chegar uma para o 13, esta também será atendida antes da do cilindro 1. Com um disco altamente carregado, o braço vai tender a estar sempre no meio do disco,

de forma que requisições em qualquer dos extremos terão que esperar até que uma flutuação estatística do carregamento do disco faça com que não haja nenhuma requisição próxima ao meio. Mesmo assim, os pedidos de acesso a cilindros longe do meio serão precariamente atendidos. Os objetivos de menor tempo de resposta e justiça igual para todos estão em conflito neste algoritmo.

Os edifícios muito altos têm que lidar com este dilema. O problema de otimizar o uso de elevadores em edifícios deste tipo é similar ao braço do disco. Os pedidos chegam continuamente, pedindo que o elevador pare em diversos andares (cilindros) aleatoriamente. Os sistemas de controle de elevadores procuram manter o elevador seguindo uma determinada direção, até que não haja nenhum pedido pendente nesta direção. Neste caso, ele muda de direção. Este algoritmo, conhecido como **algoritmo do elevador (look)**, requer que o software atualize sempre um bit, que controla a direção do elevador, UP ou DOWN. Quando uma requisição termina de ser atendida, o driver do disco olha para este bit. Se ele indicar UP, o braço move-se para cima, de forma a atender a próxima requisição pendente, se houver alguma. Se nenhum pedido estiver pendente na direção corrente, o bit de direção é invertido. Quando este bit estiver DOWN, o próximo pedido a ser atendido é o seguinte para baixo, se houver algum.

A figura 12 mostra o algoritmo do elevador usando os mesmos pedidos da figura 11, assumindo que o bit de direção está inicialmente UP. A ordem em que os cilindros são servidos é 12, 16, 34, 36, 9 e 1, que faz com que o braço percorra 1, 4, 18, 2, 27 e 8 cilindros, perfazendo um total de 60. Neste caso, o algoritmo do elevador é ligeiramente melhor que o SSF, apesar de ele constantemente apresentar um desempenho pior que o deste último. Uma boa propriedade que o algoritmo do elevador tem é que, dado qualquer conjunto de pedidos, o limite máximo de movimentação do disco é sempre fixo, e igual a duas vezes o número de cilindros a serem percorridos.

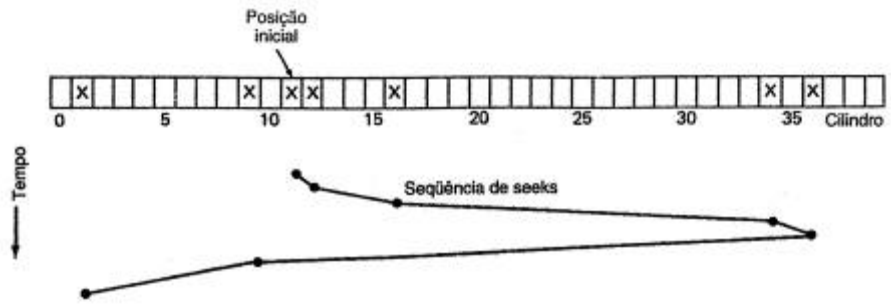

Figura 12: O algoritmo do elevador (look) para escalonamento dos pedidos de acesso ao disco

Quando várias unidades estiverem presentes no sistema, deve ser mantida uma tabela de requisições pendentes para cada uma das unidades separadamente. Sempre que qualquer das unidades estiver livre, deve ser emitido um seek para mover seu braço

para um cilindro onde ele será necessário a seguir. Quando terminar a transferência corrente, é feita uma verificação para ver se qualquer das unidades está com sua cabeça posicionada sobre o cilindro correto. Se uma ou mais estiver, a próxima transferência iniciar-se-á em uma unidade cuja cabeça já está na posição correta. Se nenhum dos braços estiver na posição correta, o driver deve emitir um novo comando de seek para a unidade que acabou de completar a transferência, e esperar até que a próxima interrupção indique qual das unidades atingiu sua posição em primeiro lugar.

Outra tendência é a de ter vários discos trabalhando juntos, sobretudo nos sistemas mais sofisticados. Tal sistema é chamado de RAID (Redundant Array of Inexpensive Disks).

4.3.RAID

A melhoria no desempenho de memórias secundárias tem sido consideravelmente menor do que a de processadores e da memória principal. Essa diferença fez dos sistemas de armazenamento em discos o principal foco de preocupação para melhoria do desempenho global de sistemas de computação.

Assim como em outras áreas, os projetistas de armazenamento de disco sabem que, se um dispositivo pode ser melhorado apenas até certo ponto, ganhos adicionais de desempenho podem ser obtidos utilizando vários componentes em paralelo. No caso de armazenamento de disco, essa idéia levou ao desenvolvimento de um agrupamento de discos que operam independentemente e em paralelo. Com diversos discos, diferentes requisições de E/S podem ser processadas em paralelo, desde que os dados requeridos residam em discos separados. Mais do que isso, uma única requisição de E/S poderá também ser executada em paralelo, se o bloco de dados a ser acessado for distribuído em vários discos.

Com o uso de múltiplos discos, os dados podem ser organizados de diversas maneiras, podendo ser empregada alguma redundância para melhorar a confiabilidade. A possibilidade de organizar os dados de vários modos poderia tornar difícil o desenvolvimento de bancos de dados compatíveis com diferentes plataformas e sistemas operacionais. Felizmente, a indústria decidiu adotar um padrão para o projeto de bancos de dados de vários discos, conhecido como RAID (agrupamento redundante de discos independentes). O esquema RAID consiste em sete níveis, de zero a seis. Esses níveis não implicam em uma relação hierárquica, mas designam diferentes arquiteturas de projeto que compartilham três características comuns:

a. O RAID consiste em um agrupamento de unidades de discos físicos, visto pelo sistema operacional como uma única unidade de disco lógico.
b. Os dados são distribuídos pelas unidades de discos físicos do agrupamento.

c. A capacidade de armazenamento redundante é utilizada para armazenar informação de paridade, garantindo a recuperação dos dados em caso de falha em algum disco.

Os detalhes da segunda e da terceira características diferem para os diferentes níveis de RAID. O RAID 0 não oferece a terceira característica.

O termo RAID foi originalmente empregado em um artigo de um grupo de pesquisadores da Universidade da Califórnia, em Berkeley, em 1988. Esse artigo esboçava várias configurações e aplicações de RAID e introduzia as definições dos níveis de RAID usadas até hoje. A estratégia RAID substitui as unidades de disco de grande capacidade por várias unidades de capacidade menor, distribuindo os dados para possibilitar acessos simultâneos nas várias unidades e, desse modo, melhorar o desempenho de E/S e facilitar o aumento significativo de capacidade da memória secundária.

Uma característica única da tecnologia RAID se refere ao uso eficaz de redundância de dados. Embora o uso simultâneo de vários cabeçotes e discos possibilite obter taxas de transferência de E/S mais altas, isso aumenta também a probabilidade de falhas. Para compensar essa diminuição de confiabilidade, o RAID usa a informação de paridade armazenada que possibilita a recuperação dos dados perdidos devido a uma falha de disco.

As características de cada um dos níveis de RAID são examinadas a seguir. A tabela abaixo resume as características dos sete níveis. Os níveis 2 e 4 ainda não estão disponíveis no mercado e existe pequena probabilidade de serem aceitos industrialmente.

Categoria	Nível	Descrição	Taxa de requisição de E/S	Taxa de transferência de dados	Aplicação típica
Intercalação de dados (striping)	0	Não-redundante	Tiras grandes: excelente	Tiras pequenas: excelente	Aplicações que requerem alto desempenho para dados não-críticos
Espelhamento	1	Espelhado	Bom / razoável	Razoável / razoável	Unidade de disco de sistema; arquivos críticos
Acesso paralelo	2	Redundante via código de Hamming	Pobre	Excelente	
	3	Paridade de bit intercalada	Pobre	Excelente	Aplicações com grandes requisições de E/S, como processamento de imagens e CAD
Acesso independente	4	Paridade de bloco intercalada	Excelente / razoável	Razoável / pobre	
	5	Paridade de bloco intercalada e distribuída	Excelente / razoável	Razoável / pobre	Buscas de dados, com altas taxas de requisição e grande volume de leituras
	6	Paridade de bloco dupla intercalada e distribuída	Excelente / pobre	Razoável / pobre	Aplicações que requerem grande disponibilidade de dados

As figuras 13 a/b/c e 13 d/e/f/g mostram um exemplo de uso dos sete esquemas de RAID para implementar uma capacidade de dados que requer quatro discos, excluindo a redundância. As figuras enfatizam a organização dos dados de usuários e dos dados redundantes e indicam os requisitos de armazenamento dos vários níveis.

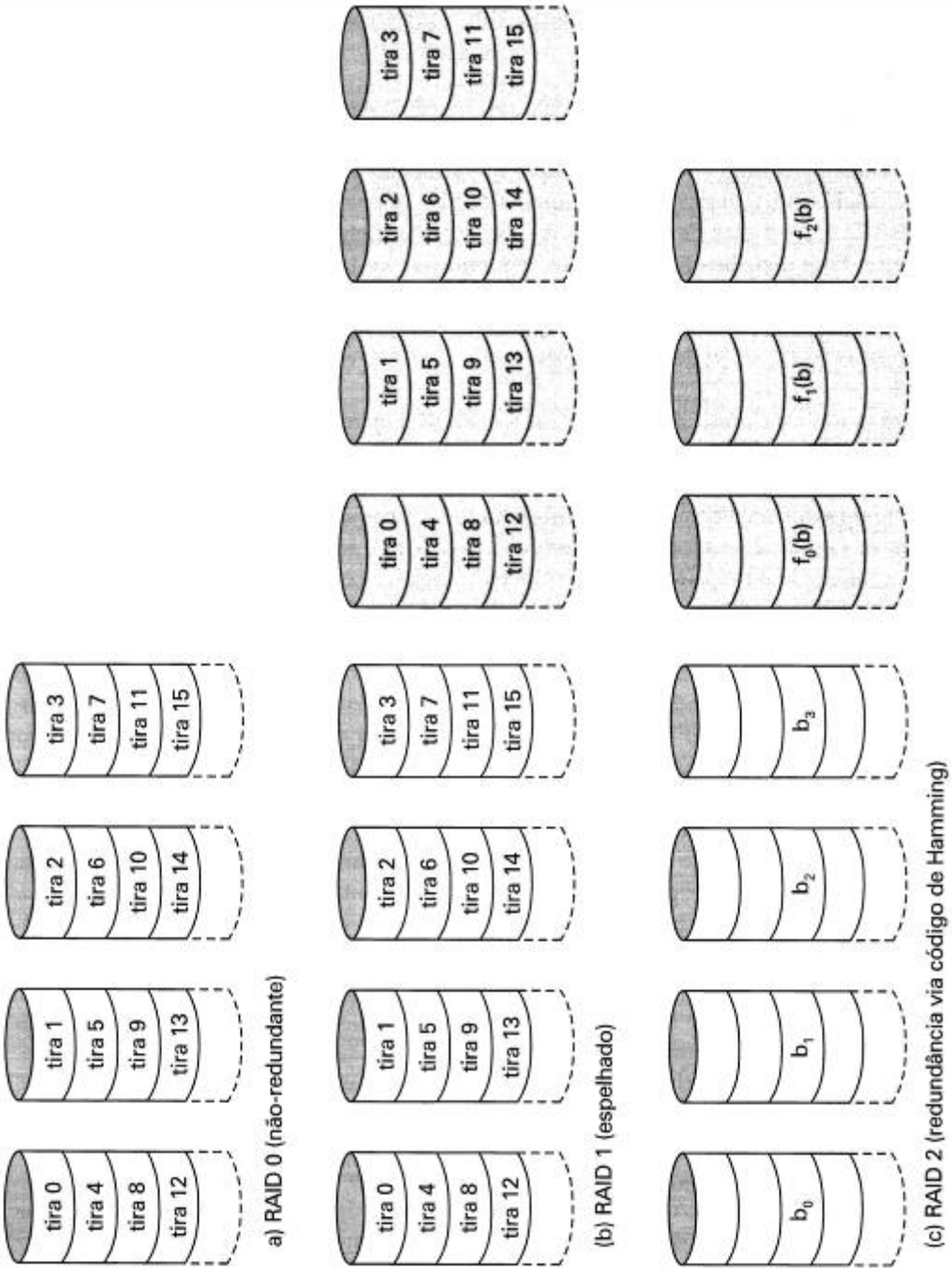

Figura 13 a/b/c: Níveis de RAID

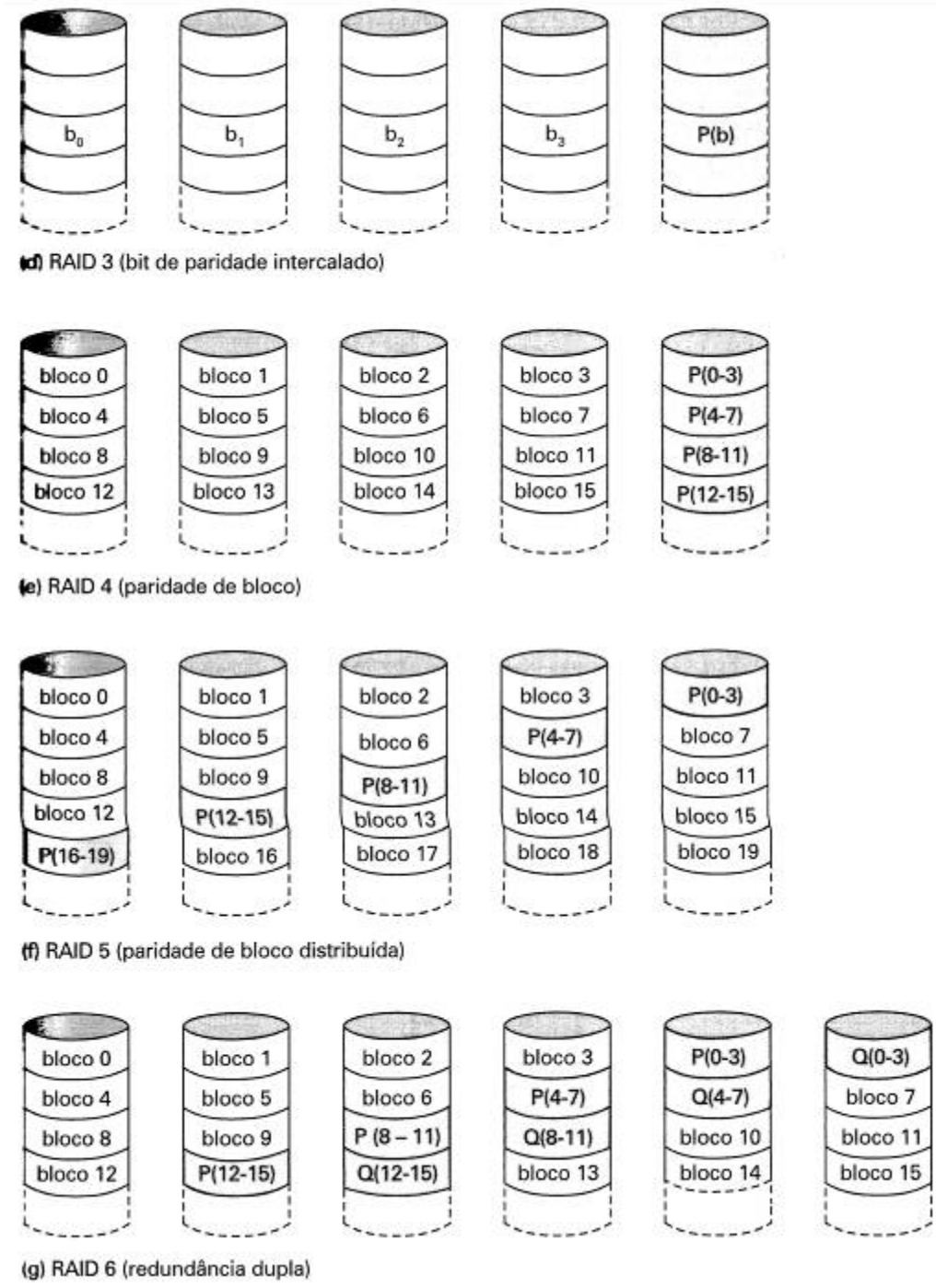

Figura 13 d/e/f/g: Níveis de RAID

RAID DE NÍVEL 0

O RAID de nível 0 não constitui de fato um membro da família RAID, uma vez que não inclui redundância para a melhora do desempenho. Contudo, ele é utilizado em poucas aplicações, como em supercomputadores, nos quais o desempenho e a capacidade constituem requisitos primordiais e o baixo custo é mais importante do que maior confiabilidade.

No RAID 0, os dados de usuário e de sistema são distribuídos em todos os discos do agrupamento. Essa distribuição dos dados tem enorme vantagem em

relação ao uso de um único disco grande: se existirem duas requisições de E/S pendentes para dois blocos de dados distintos, haverá grande probabilidade de que esses blocos estejam em discos diferentes. Portanto, as duas requisições podem ser atendidas em paralelo, reduzindo o tempo de espera na fila de E/S.

Assim como os demais níveis de RAID, o RAID 0 não se limita a distribuir os dados pelo agrupamento de discos. Os dados são *intercalados em tiras* (*strips*) por meio dos discos disponíveis. Isso pode ser bem compreendido observando-se a figura 14. Os dados de usuários e de sistema são vistos como sendo armazenados em um disco lógico. Esse disco é dividido em tiras; as tiras podem ser constituídas de blocos físicos, setores ou alguma outra unidade de armazenamento. As tiras são mapeadas sobre membros consecutivos do agrupamento de forma circular, cujo modo de distribuição é mais conhecido como *round robin*. Um conjunto de tiras logicamente consecutivas, cada qual mapeada em um dos membros do agrupamento, é denominada *faixa* (*stripe*). Em um agrupamento de *n* discos, as *n* primeiras tiras lógicas são fisicamente armazenadas como a primeira tira de cada um dos *n* discos, formando a primeira faixa; as *n* tiras seguintes são distribuídas como a segunda tira de cada disco, e assim por diante. A vantagem desse esquema é que, em grandes requisições de E/S, compostas de múltiplas tiras logicamente contíguas, até *n* tiras podem ser manipuladas em paralelo, reduzindo bastante o tempo de transferência de E/S.

A figura 14 indica o uso de um software para gerenciamento do agrupamento, que faz o mapeamento entre o disco lógico e os discos físicos. Esse software pode ser executado tanto no subsistema de disco como no computador principal.

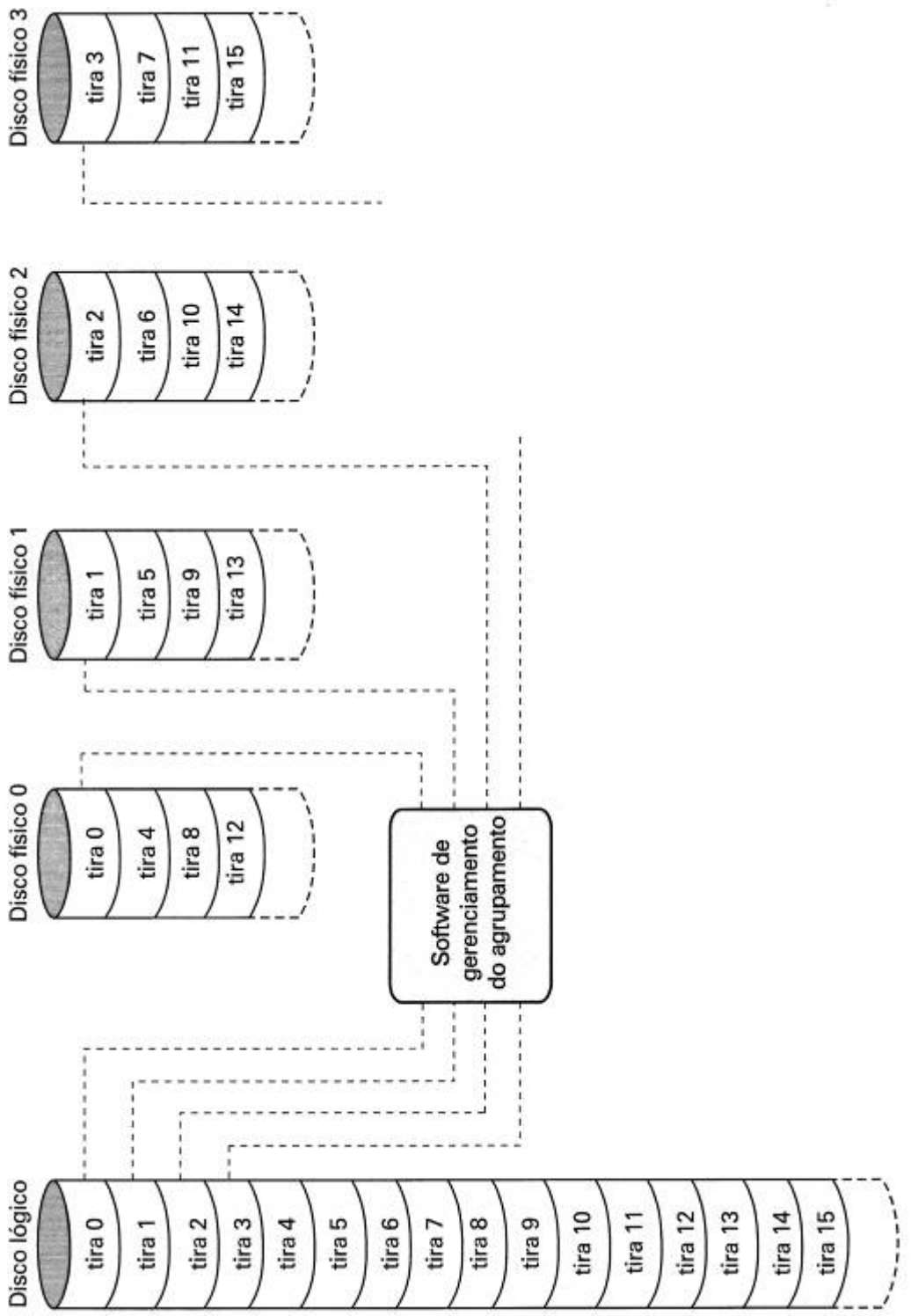

Figura 14: Mapeamento de dados para um agrupamento RAID de nível 0

RAID DE NÍVEL 1

No RAID 1, a redundância é obtida pela simples duplicação dos dados. Como mostra a figura 13b, os dados são intercalados em tiras, como no RAID 0. Entretanto, nesse caso, cada tira lógica é mapeada em dois discos físicos separados, de modo que cada disco do agrupamento tenha como espelho um outro disco que contém os mesmos dados.

A organização do RAID 1 tem diversos aspectos positivos:

a. Uma requisição de leitura pode ser servida por qualquer dos dois discos que contenha os dados requeridos, preferencialmente por aquele no qual o tempo de busca somado à latência rotacional seja menor.
b. Uma requisição de escrita requer a atualização das duas tiras correspondentes, mas isso pode ser feito em paralelo. Dessa maneira, o desempenho da requisição de escrita é determinado pela operação de escrita mais lenta. Não existe qualquer “penalidade de escrita” no RAID 1. Os RAIDs de níveis 2 a 6 envolvem o uso de bits de paridade. Por isso, quando uma tira é atualizada, o software de gerenciamento do agrupamento deve calcular e atualizar os bits de paridade, além de atualizar a tira em questão.
c. A recuperação de uma falha é simples. Quando ocorre uma falha em uma unidade de disco, os dados ainda podem ser obtidos a partir da segunda unidade.

A principal desvantagem do RAID 1 é o custo; ele requer um espaço de disco físico igual a duas vezes o do disco lógico. Por isso, uma configuração RAID 1 geralmente é utilizada apenas em unidades de disco que armazenam software e dados do sistema e outros arquivos altamente críticos. Nesses casos, o RAID 1 oferece uma cópia de segurança de todos os dados em tempo real, de modo que, mesmo ocorrendo uma falha em um disco, todos os dados críticos permaneçam disponíveis.

Em um ambiente orientado para transações, o RAID 1 pode alcançar uma alta taxa de execução de requisições de E/S, caso a maioria das requisições seja de leitura. Nesse caso, o desempenho do RAID 1 pode se aproximar do dobro do desempenho do RAID 0. Entretanto, se uma fração substancial das requisições de E/S for de escrita, pode não haver um ganho significativo de desempenho em relação ao RAID 0. O RAID 1 pode também apresentar um desempenho melhor que o RAID 0 para aplicações com transferência de dados intensiva e com alta porcentagem de leituras. Essa melhoria de desempenho ocorre somente se a aplicação puder dividir cada requisição de leitura, de modo que os dois discos membros possam ser utilizados.

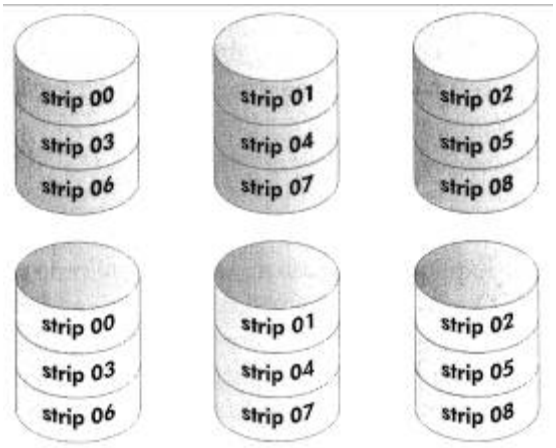

Figura 15: RAID 1

RAID DE NÍVEL 5

O RAID 5 usa a técnica de acesso independente. Em um agrupamento com acesso independente, cada disco opera independentemente, permitindo que requisições de E/S distintas possam ser satisfeitas em paralelo. Por isso, agrupamentos com acesso independente são mais adequados para aplicações que requerem altas taxas de requisições de E/S, não sendo apropriadas para aplicações que necessitam de altas taxas de transferências de dados.

Como nos demais esquemas de RAID, é usada a intercalação de dados em tiras. No RAID de níveis 5, as tiras são relativamente grandes e distribuídas por todos os discos. Uma alocação típica consiste em um esquema circular, como mostrado na figura 13f. Para um agrupamento de *n* discos, a tira de paridade das *n* primeiras tiras de dados é armazenada em um disco diferente e esse padrão então se repete.

A distribuição das tiras de paridade em todos os discos evita a possibilidade de formação de gargalos no desempenho do sistema.

Figura 16: RAID 5

4.4.Tratamento de Erros

Os discos são dispositivos sujeitos a uma grande variedade de erros. Alguns dos mais comuns são:

1. Erros de programação

2. Erro transiente no checksum
3. Erro permanente no checksum
4. Erro de seek
5. Erro da controladora

É da responsabilidade do driver de disco tratar cada um destes tipos de erros da melhor maneira possível.

Os erros de programação ocorrem quando o programador manda a controladora buscar informação num setor que não existe, ou usar uma cabeça inexistente, ou fazer uma transferência que envolva um endereço de memória que não existe. A maioria das controladoras verifica a exatidão dos parâmetros que lhe são fornecidos e reclama se eles não forem válidos. A única coisa a fazer é abortar a requisição corrente que está apresentando erro e torcer para que tal erro não ocorra com muita freqüência.

Os erros transientes no checksum são causados por partículas de poeira existentes no ar, e que passam entre a cabeça e a superfície do disco. Na maioria das vezes, esse tipo de erro pode ser eliminado simplesmente repetindo a operação umas poucas vezes. Se o erro persistir, o bloco deve ser marcado como **ruim** e seu uso posterior evitado por software. Alternativamente, algumas controladoras de disco mantêm algumas trilhas não acessíveis aos programas de usuários. Quando um disco de determinada unidade é formatado, a controladora determina automaticamente os setores ruins, e os substitui pelos reservas. A tabela que mapeia os setores ruins nos reservas é mantida na memória interna da controladora e no próprio disco. Esta substituição é transparente para o driver.

Erros de seek são causados por problemas mecânicos no braço. A controladora determina internamente a posição do braço. Para realizar um seek, ela envia uma série de pulsos para o motor do braço, um pulso por cilindro, para mover o braço em direção ao novo cilindro desejado. Quando o braço teoricamente chega ao seu destino, a controladora lê o número do cilindro. Se o braço não estiver na posição correta, acabou de ocorrer um erro de seek. Algumas controladoras corrigem automaticamente o erro de seek, nas outras simplesmente setam um bit e deixam o problema com um driver. O driver trata este erro emitindo um comando RECALIBRATE, para mover a cabeça até onde ela deveria ter ido inicialmente, e leva o valor do cilindro corrente da controladora para 0. Em geral, isto resolve o problema. Se não resolver, a unidade precisa ser reparada.

Conforme visto anteriormente, a controladora é, na verdade, um pequeno computador especializado, com software, variáveis, buffers e, ocasionalmente, bugs. Algumas vezes uma sequência de eventos não-usual, como a interrupção em uma unidade ocorrendo ao mesmo tempo de um comando RECALIBRATE para outra unidade, pode dar início a um bug, fazendo com que a controladora entre em loop ou venha a perder a trilha onde estava trabalhando. Os projetistas destas máquinas usualmente se

planejam para que o pior possa acontecer, de forma que quase sempre existe um pino no chip ou na placa que, quando no nível alto, força a controladora a abandonar o que estava fazendo e resetar a si própria. Se tudo o mais falhar, o driver de disco pode setar este bit e resetar a controladora. Se isto também não ajudar, tudo o que o driver tem a fazer é mandar uma mensagem de erro e desistir.

4.5.Cache em Disco

Nas unidades de disco disponíveis atualmente, o tempo necessário para buscar um novo cilindro é maior que o de rotação e o de transferência, Uma vez que o driver tenha movido o braço para determinada posição, ele pode fazer alguma coisa, enquanto o setor ou a trilha estiverem sendo lidos ou escritos. Esse efeito é especialmente verdadeiro, se a controladora for dotada de um sensor rotacional, de maneira a permitir que o driver saiba qual setor está atualmente embaixo da cabeça, e mandar uma requisição para o próximo setor, tornando possível a leitura de uma trilha no tempo correspondente a uma rotação.

Alguns drivers de disco tiram vantagens desta propriedade, mantendo uma cache secreta, desconhecida do software independente do dispositivo. Se o setor necessário estiver na cache, não há necessidade de se efetuar uma transferência do disco. A desvantagem da cache de disco, além da complexidade do software e do espaço do buffer necessário, é que a transferência da cache para o programa que deve ser feita pelo processador, usando um loop programado, em vez de ser feito pelo hardware de DMA.

Algumas controladoras levam esse processo um pouco mais além, e mantém a cache em sua própria memória interna, independente do driver, de forma que a transferência entre a controladora e a memória pode fazer uso do hardware de DMA. Observe que, neste caso, tanto a controladora quanto o driver estão numa excelente situação para ler e escrever trilhas inteiras em um único comando, coisa que o software independente do dispositivo não pode fazer, por considerar o disco uma seqüência linear de blocos, sem levar em conta sua divisão em trilhas e cilindros.