GERÊNCIA DE ENTRADA E SAÍDA

Uma das principais funções de um SO é controlar os dispositivos de entrada e saída (E/S), fornecendo uma interface de uso adequada. Esse capítulo mostra como o SO interage com os dispositivos de E/S, enviando comandos e capturando suas interrupções. A gerência de E/S está intimamente relacionada com os aspectos de hardware de um computador.

5.1 Princípios básicos de hardware

Um **periférico** (ou **dispositivo de E/S**) é qualquer dispositivo conectado a um computador de forma a possibilitar a interação do computador com o mundo externo. Atualmente, existe uma grande variedade de dispositivos, desde aqueles desenvolvidos para permitir a comunicação do homem com o computador (teclado, mouse, monitor de vídeo, etc) até dispositivos que possibilitam a comunicação entre computadores (modems, placas de redes, etc), ou ainda aqueles destinados ao armazenamento de informações (unidades de fita, disquetes, disco rígido, CD-ROM, etc). Apesar dessa diversidade, existe uma razoável padronização na forma de acessar (acionar) esses periféricos.

De acordo com o sentido do fluxo de dados entre o computador e o dispositivo, esses podem ser divididos em *periféricos de entrada*, *periféricos de saída*, ou ainda *periféricos de entrada e saída*.

Um periférico é conectado ao computador através de um componente de hardware denominado **interface**. Essa, por sua vez, é interconectada aos barramentos internos do computador. Para tratar a diversidade, a complexidade, e as diferentes formas de operações de cada periférico, as interfaces empregam no seu projeto um outro componente, o **controlador**, que nada mais é que um processador projetado especificamente para realizar uma função, como, por exemplo, controlar um disco rígido.

A UCP se comunica com o controlador através de um conjunto de registradores situados na interface. Tipicamente, são 3 registradores: **registrador de dado**, **registrador de status** e **registrador de comando**. Por exemplo, para escrever um dado em um dispositivo de saída, seriam realizadas as seguintes duas operações: (1) UCP coloca o dado no registrador de dados, (2) UCP coloca comando *write* no registrador de comando. A partir daí, o controlador comanda as ações do periférico, de forma independente da UCP. No final das ações, o controlador coloca o resultado da operação (sucesso ou falha) no registrador de status e gera uma interrupção. O caso de não haver mecanismo de interrupção é considerado adiante.

A função básica de um controlador é implementar um conjunto de comandos para o seu dispositivo. O controlador vai traduzir cada ordem colocada no registrador de comando numa seqüência específica de acionamentos eletrônicos, elétricos e mecânicos que irão realizar a operação solicitada. Evidentemente, cada tipo de periférico necessita de um controlador diferente. Contudo, apesar da grande variedade de dispositivos, não dá para negar que existe uma boa padronização na forma de conectá-los ao computador e na forma de acioná-los através da UCP.

Os dispositivos de E/S, dependendo de sua interconexão fisica às interfaces, podem ser do tipo **serial** ou **paralelo**. Uma **interface serial** é aquela em que existe apenas uma linha para transmitir dados. Nesse caso, um byte vai ser transferido como

uma seqüência de bits. Os modems, alguns tipos de *mouses* e impressoras são exemplos de dispositivos seriais. Uma **interface paralela** possui várias linhas para os dados, permitindo assim que vários bits sejam transferidos simultaneamente (em paralelo) entre o dispositivo de E/S e a interface. O número de linhas corresponde normalmente ao número de bits que compõem um byte (*8 bits*) ou palavra (*n x 8 bits*). As impressoras paralelas são exemplos típicos desse tipo de dispositivo.

- **Acesso aos registradores dos periféricos**

Conforme visto anteriormente, a UCP "enxerga" um periférico através de um conjunto de registradores, os quais registram ordens, fornecem o estado de uma operação, e permitem a leitura (escrita) de dados do (no) periférico. Esses registradores residem fisicamente no hardware da interface e podem ser acessados pela UCP através de duas técnicas distintas: **entrada e saída mapeada em espaço de E/S** ou **entrada e saída mapeada em memória**. A opção por uma das técnica é feita em tempo de projeto do computador.

Na primeira (**mapeamento em espaço de E/S**), o processador possui instruções de máquina próprias para operações de E/S e estas instruções especificam registradores de periféricos. Nesse tipo de processador existem dois espaços de endereçamento distintos: o espaço normal, que corresponde à memória principal e o espaço de E/S, que corresponde aos registradores dos periféricos. Um conjunto de instruções (do estilo *load*, *store*, *move*) acessa a "memória normal" e outro conjunto (as instruções de E/S, do estilo *in* e *out*) acessa a "memória de E/S". Enquanto a instrução *move end,dado* representa escrever *dado* na posição de memória *end*, a instrução *out end,dado* representa escrever *dado* no registrador de algum dispositivo. Observe que, numericamente, o valor de *end* pode ser o mesmo nos dois casos, o que faz a diferença é a instrução empregada.

No **mapeamento em memória**, o espaço de endereçamento é único (só existe uma memória), mas alguns endereços representam registradores de (controladores de) periféricos. Em tempo de projeto do computador, são reservados alguns endereços da memória principal para representar registradores de periféricos. Esses endereços não corresponderão a palavras de memória, mas sim a registradores de dispositivos. Com essa técnica, qualquer periférico pode ser programado através de instruções de acesso à memória do estilo *mov end,dado*. Se *end* é um endereço correspondente a um controlador de dispositivo, o dado será escrito no registrador do dispositivo (o valor de *dado* será interpretado segundo a função específica desse registrador).

- **Interação entre a UCP e os controladores de periféricos**

Independentemente do tipo de mapeamento utilizado, a interação entre a UCP e a interface (controlador), para realizar operações de E/S, pode acontecer de três maneiras diferentes: "E/S programada", "via interrupções" e "acesso direto à memória (DMA)".

E/S programada

Essa técnica é usada quando não há sistema de interrupção (nos computadores antigos era assim, hoje a técnica só é usada em máquinas simples). Com esta técnica, toda interação entre a UCP e o controlador é de responsabilidade do programador. O ciclo de funcionamento é baseado no envio de um comando ao controlador e na espera de sua

realização. Por exemplo, o processador envia um comando de leitura ao controlador e, em seguida, fica testando continuamente (em *busy loop*) o registrador de estado para verificar se o dado solicitado já está disponível. Em caso afirmativo, o processador efetua a leitura.

O problema desse método é que as operações de E/S são muito lentas em comparação com as operações de cálculo. Utilizar continuamente o processador para verificar o andamento de uma operação de E/S representa um desperdício muito grande de tempo de cálculo. Uma solução paliativa é inserir operações de cálculo entre as verificações sucessivas do estado da operação de E/S. Esse procedimento de verificar periodicamente o estado de uma operação de E/S é denominado de ***polling***.

O emprego de *polling* reduz o desperdício de tempo do processador, mas introduz um outro problema: determinar a freqüência de sua realização. Se a freqüência é muito alta, há desperdício de tempo, principalmente se o dispositivo é realmente lento. Por outro lado, efetuar o polling com uma freqüência baixa pode acarretar esperas desnecessárias por parte do dispositivo de E/S, ou ainda, o que é pior, perda de informações (este problema existe nas interfaces de comunicação que possuem *buffers* de tamanho limitado para a recepção de dados: a não leitura dos dados recebidos em um certo período de tempo pode acarretar perdas por "sobre-escrita"). O ideal seria que o dispositivo de E/S sinalizasse o processador logo que o dado estivesse disponível.

Comunicação via interrupção

O uso do mecanismo de interrupção elimina a necessidade de *polling* em operações de E/S. Nesse caso tem-se E/S via interrupções (*interrupt driven I/O*).

Nessa situação, o processador é responsável – via software – apenas por iniciar uma operação de E/S enviando comandos à interface (controlador). Após, o processador passa a executar outra tarefa, e o controlador, a operação de E/S. Quando a operação de E/S termina, o controlador interrompe o processador, provocando a execução do tratador de interrupção, o qual irá acionar o driver do dispositivo.

Acesso Direto à Memória

O emprego de interrupções resolve o problema de determinar o momento exato em que um dispositivo de E/S necessita da atenção do processador, entretanto não auxilia na execução de outra tarefa bastante importante em operações de E/S: a transferência de dados.

Para a maioria dos periféricos, o processador é responsável pela leitura de dados na interface e por sua transferência à memória, ou pela leitura na memória e transferência para a interface. Quando o volume de dados é importante, esse procedimento torna-se ineficiente, pois, além de envolver o processador, essa transferência representa dois movimentos de dados: um da interface ao processador, e outro do processador à memória. Uma solução para isso é transferir diretamente os dados da interface para a memória, o que é conseguido por um mecanismo conhecido como **DMA** (*Direct Memory Access*).

A técnica de DMA baseia-se no emprego de um hardware especial, o controlador de DMA, para realizar a transferência de dados entre um dispositivo e a memória. Para tanto, o controlador de DMA possui a capacidade de acessar diretamente a memória,

sendo então conectado fisicamente ao barramento de dados e de endereços do computador. O controlador de DMA possui internamente uma série de registradores utilizados pela UCP para programar a transferência de dados. Tipicamente, tem um par de registradores para o armazenamento dos endereços fonte e destino da transferência, um registrador que determina quantos bytes devem ser transferidos, um registrador de comando e um de estado. Após acionar o DMA, o processador pode se dedicar a outra tarefa. No término da transferência, o controlador de DMA sinaliza o processador através de uma interrupção de hardware.

A técnica de DMA é mais eficiente que as discutidas anteriormente quando a operação de E/S envolve a leitura (ou escrita) de muitos dados, como, por exemplo, uma leitura de disco ou a recepção de uma mensagem em uma rede local. É importante observar que tanto o controlador de DMA quanto o processador competem para acessar à memória (não esqueça que o processador está executando outras tarefas, o que envolve sem dúvida acessos à memória). Essa disputa pelo acesso à memória é coordenada pelo que se denomina **arbitramento do barramento**. Como o mecanismo de acesso à memória passa a ser compartilhado, o processador (durante o funcionamento do DMA) irá executar a uma velocidade menor que a normal. Mesmo assim, esse método será ainda mais eficiente do que usar diretamente o processador para realizar a transferência via software.

5.2 Princípios básicos de software de entrada e saída

O subsistema de E/S de um SO é um software bastante complexo devido principalmente à diversidade de periféricos que ele deve tratar. Mesmo dentro de uma mesma "classe de dispositivos", existe uma grande variedade, como por exemplo, placas de rede com protocolos, tecnologias e velocidades diferentes, ou ainda discos magnéticos do tipo SCSI, IDE, EIDE, ZIP, CD-ROM, etc.

O objetivo primeiro do subsistema de E/S é padronizar ao máximo a forma de acesso aos periféricos. Para atingir esse objetivo, o subsistema de E/S é normalmente organizado em uma estrutura de quatro camadas, onde cada camada fornece funções (serviços) à camada superior. A Figura 5.1 apresenta essa estrutura.

A camada mais inferior é composta pelo hardware dos dispositivos de E/S. A interface que ela apresenta à camada imediatamente superior (*drivers* de dispositivos) é composta pelos mecanismos apresentados na Seção 5.1. Em resumo, o hardware interage com os *drivers* através das interfaces físicas (paralelas ou seriais) e seus controladores e pelo emprego de interrupções e DMA. A seguir, estudaremos o software de E/S.

- **Drivers de dispositivo**

A camada inferior de software ─ *driver*s de dispositivos (*device drivers*) ─ é composta por um conjunto de módulos, cada um responsável por implementar o acesso a um dispositivo específico. O principal objetivo dos *driver*s é "esconder" as diferenças entre os vários dispositivos de E/S, fornecendo à camada superior uma "visão uniforme" desses dispositivos através de uma interface de programação única.

A camada de *driver*s é responsável por implementar as rotinas necessárias ao acesso e à gerência de cada dispositivo. É nesse nível que o software de E/S realiza a programação de registradores internos de controladores que compõem a interface física

dos dispositivos e implementa os respectivos tratadores de interrupção. Assim, cada tipo de dispositivo requer um *driver* apropriado. Essa camada fornece uma abstração a mais genérica possível para a camada superior, a de E/S independente do dispositivo.

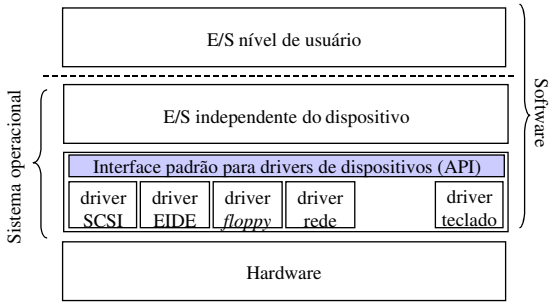

Figura 5.1 – Estrutura em camadas do subsistema de E/S.

Para fornecer esta "visão uniforme", os dispositivos de E/S são classificados em duas grandes categorias, de acordo com a unidade de transferência de dados, que pode ser bloco ou caractere (byte). Em um **dispositivo orientado a bloco** (*block devices*), o armazenamento de dados e a transferência são realizados através de blocos de tamanho fixo. Tipicamente, o tamanho de um bloco varia entre 512 bytes e 32 kbytes. As unidades de disco são o exemplo mais comum de dispositivos orientados a bloco. Os **dispositivos orientados a caractere** (*character devices*) realizam as transferências byte a byte, a partir de um fluxo de caracteres, sem necessidade de considerar uma estrutura qualquer. As portas seriais são exemplos de dispositivos orientados a caractere. Essa classificação, entretanto, não é completamente adequada, pois nem todos os dispositivos de E/S podem ser enquadrados em um desses dois grupos. Os temporizadores (relógios) e monitores de vídeo são exemplos de dispositivos que não se enquadram em nenhuma dessas categorias.

Existe ainda um outro tipo de dispositivo denominado **pseudo-dispositivo** (*pseudo-devices*) que na realidade não corresponde a nenhum periférico físico. Ele é apenas uma abstração empregada para adicionar funcionalidades ao SO, explorando a interface padronizada já existente para o tratamento de dispositivos. É dessa forma, por exemplo, que o sistema UNIX permite o acesso à memória principal como se fosse um disco (*ramdisk*), ou ainda o emprego do dispositivo nulo (*/dev/null*) para descartar dados.

- **E/S independente do dispositivo**

Pode-se dizer que a camada dos *drivers* é a camada "dependente dos dispositivos", já que nela estão as funções diretamente ligadas ao hardware dos dispositivos. A camada acima usa apenas a interface abstrata ou virtual (padronizada e mais amigável) provida pelos *drivers* e, como tal, pode ser considerada "independente de dispositivo".

A camada independente de dispositivo implementa funções genéricas (no sentido de valer para qualquer dispositivo) e serviços gerais de E/S, importantes para o

funcionamento do sistema como um todo. As funções genéricas são implementadas através de estruturas que representam "classes" de dispositivos e operações associadas. Internamente, essas estruturas possuem ponteiros para descritores que especializam as operações. Por exemplo, pode-se ter uma função genérica **read** cujo primeiro argumento indica o dispositivo a ser usado. Essa operação genérica vai ser mapeada para uma seqüência de operações compatíveis com o dispositivo em questão, pois ler de um teclado é diferente de ler de um disco.

Alguns serviços sob responsabilidade desta camada são:

- **Escalonamento de E/S:** Usado em dispositivos compartilhados por vários processos (por exemplo, discos magnéticos) para melhorar o desempenho dos mesmos.
- **Buferização:** Um exemplo típico de buferização ocorre em protocolos de comunicação; o usuário pode desejar enviar, digamos, 64 Kbytes, mas a interface de rede pode enviar apenas seqüências máximas de 4 Kbytes. Nesse caso, é necessário armazenar a requisição do usuário e enviá-la em blocos de 4 kbytes.
- ***Cache* de dados:** Consiste em armazenar na memória os blocos de dados freqüentemente acessados. Um exemplo são as *caches* de disco (esse mecanismo será apresentado quando estudarmos sistemas de arquivos).
- **Alocação de dispositivo:** Muitos dispositivos admitem, no máximo, um usuário por vez. Esse controle é normalmente efetuado através da técnica de ***spooling***, que consiste em seqüencializar os pedidos de acesso e atendê-los um a um. Os pedidos são registrados em uma fila especial (*spool*), a qual é acessada por um processo especial do SO (*daemon*), o qual atende as requisições de E/S. A gerência de impressora é um exemplo clássico do emprego de *spool*.
- **Direitos de acesso:** Nem todos os usuários podem acessar os dispositivos da mesma forma e cabe ao SO garantir essa proteção. O sistema UNIX, por exemplo, emprega os bits de permissão de acesso a arquivos (*rwx* - leitura, escrita e execução) para selecionar o tipo de operação que um determinado usuário, ou grupo de usuários, pode efetuar sobre um dispositivo particular.
- **Tratamento de erros:** O software de E/S deve ser capaz de tratar erros, informando à camada superior o sucesso ou o fracasso de uma operação. Erros transientes – como impossibilidade de transmissão por *overflow* em *buffers* – podem implicar apenas uma nova tentativa e não no término do processo requisitante.

- **Entrada e saída à nível de usuário**

O usuário "vê" os periféricos através dos aplicativos e das linguagens de programação que ele utiliza. No caso de um editor de textos, por exemplo, as operações de leitura e escrita dos registros do arquivo editado ficam invisíveis, sendo feitas automaticamente pelo aplicativo. No caso de uma linguagem de programação, o usuário escreve comandos de E/S de alto nível (por exemplo, a função printf() da linguagem C para realizar saída formatada de dados), os quais são traduzidos para um código que contém chamadas para

rotinas de uma biblioteca de E/S. O fabricante do compilador é responsável por fornecer uma biblioteca de E/S para cada sistema em que seu compilador vá ser utilizado. As funções dessa biblioteca contém as chamadas para o SO. As bibliotecas são fornecidas na forma de um módulo objeto (relocável), o qual é ligado com o programa do usuário para compor o executável. É importante notar que as bibliotecas de E/S não fazem parte do SO, pois elas são associadas às linguagens de programação e/ou aos aplicativos de desenvolvimento.

5.3 Dispositivos periféricos típicos

Existem atualmente uma grande gama de dispositivos periféricos que possibilitam a comunicação do homem com o computador e entre computadores. A seguir serão abordados apenas os dispositivos mais importantes e/ou comuns em computadores (disco, vídeo, teclado e rede).

Na perspectiva do SO, o periférico mais importante é o disco magnético, principalmente o disco rígido. Ele desempenha um papel fundamental em diversos aspectos do SO, servindo para desde o simples armazenamento de dados até a implementação de mecanismos complexos como a memória virtual.

- **Discos rígidos**

Os discos rígidos são dispositivos capazes de armazenar grande volume de dados. O atual estado tecnológico permite a construção de discos magnéticos de vários gigabytes a um baixo custo, sendo encontrados em qualquer computador de uso pessoal.

A unidade de disco (ver figura 5.2) contém um conjunto de discos metálicos (aço ou alumínio) superpostos, que giram em torno de um eixo central (*spindle*). A tecnologia atual permite superpor até 8 discos. As duas superfícies de cada disco são recobertas por uma película magnética na qual os dados são gravados. O eixo gira a uma rotação constante (3600 rpm, por exemplo). Os cabeçotes de leitura/gravação (um para cada superfície de disco) realizam movimentos de vai-e-vem e podem se deslocar desde a borda do disco até o centro. Graças ao movimento de rotação dos discos e ao movimento retilíneo dos cabeçotes, toda a superfície de cada disco pode ser alcançada por seu respectivo cabeçote.

Cada superfície é dividida em circunferências concêntricas denominadas **trilhas (*tracks*)**, as quais são divididas radialmente em unidades chamadas **setores (*sectors*)**. Todos os setores tem o mesmo tamanho, o qual varia entre 512 a 4096 bytes (sendo mais comum 512). O setor constitui a unidade mínima de leitura e gravação em um disco. O conjunto de trilhas de todas as superfícies que ficam exatamente à mesma distância do eixo central forma o **cilindro (*cylinder*)**.

A definição das trilhas e setores de um disco chama-se **formatação física** e é um procedimento realizado pelo fabricante. É fácil observar que, sendo divisões radiais do disco, os setores correspondentes às trilhas mais externas são mais longos que os setores de trilhas mais internas. Como, em princípio, os setores armazenam a mesma quantidade de dados, a densidade de gravação nos setores externos é menor que no setores internos.

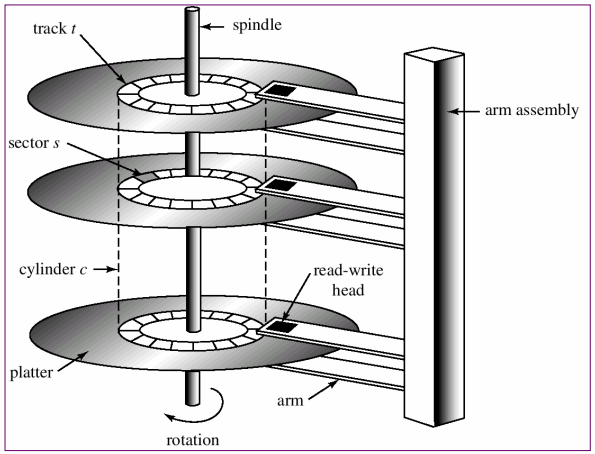

Figura 5.2 - Organização física do disco magnético.

Para acessar dados no disco, é necessário informar, ao controlador, o cilindro, a superfície e o setor a ser acessado. Esse método de acesso é denominado CHS (*Cylinder, Head, Sector*). Outra maneira consiste em "enxergar" o disco como um conjunto de blocos, formados por um ou mais setores. Neste caso, é informado o número do bloco a acessar e este é convertido no endereço físico CHS (cilindro, superfície, setor) por um procedimento que se denomina de LBA (*Linear Block Addressing*).

Outros termos bastante comuns associados a discos rígidos são **formatação lógica** e **partições**. Ambos os conceitos estão mais relacionados com o sistema de arquivos do que com o disco propriamente dito. A formatação lógica consiste em gravar informações no disco de forma que arquivos possam ser escritos, lidos e localizados pelo SO. Por sua vez, o conceito de partição está associado à capacidade de dividir logicamente um disco em vários outros discos.

Tempo de acesso

Para realizar um acesso a um disco rígido, é necessário posicionar o cabeçote de leitura/escrita no setor onde o dado será lido ou escrito. Considerando a organização de um disco, esse procedimento de posicionamento implica um certo tempo: **o tempo de acesso**. Esse tempo é formado por três parcelas:

$$t_{access} = t_{seek} + t_{latency} + t_{transfer}$$

- ***Seek* time (t_{seek}):** é o tempo necessário para deslocar os cabeçotes até o cilindro onde está a trilha a ser acessada; a seleção da trilha (cabeçote) é feita eletrônicamente, em tempo zero.
- **Latency time ($t_{latency}$):** é o tempo necessário para o cabeçote se posicionar no início do setor a ser lido ou escrito. Esse tempo também é denominado de atraso rotacional (*rotational delay*).

- **Transfer time ($t_{transfer}$):** é o tempo necessário para realizar a transferência dos dados (leitura ou a escrita dos dados).

O tempo predominante é o tempo de *seek*. Por essa razão, alguns discos antigos possuíam um sistema com cabeçotes fixos, um para cada trilha do disco.

Entrelaçamento

Outro fator relacionado com a redução do tempo de acesso a um disco é o **entrelaçamento** (*interleaving*). É muito comum o acesso a vários setores contíguos em uma trilha do disco. Suponha que se deseja ler os setores 4 e 5 de uma determinada trilha. O SO envia ao controlador de disco o comando para ler o setor 4. Após o *seek* apropriado, o cabeçote passa sobre o setor 4, e a transferência ocorre. Quando o cabeçote sai do setor 4, os dados são transferidos do *buffer* do controlador para a memória, provocando uma interrupção no processador para informar o término da leitura do setor 4. Nesse momento, o processador (via SO) envia um novo comando de leitura, dessa vez para o setor 5. Um novo *seek* não será necessário, pois o cabeçote já se encontra sobre o cilindro desejado. Entretanto, devido à rotação do disco, o cabeçote provavelmente não se encontra mais no início do setor 5. Será necessário esperar que o disco dê uma volta completa (tempo de latência) para então efetuar a leitura do setor 5.

A forma usual para evitar esse problema é realizar um entrelaçamento dos setores (*interleaving*). Essa técnica numera os setores não mais de forma contígua mas sim com um espaço entre eles. O disco 2 da Figura 5.3 possui um fator de entrelaçamento igual a 2. Isso significa que entre o setor *k* e o setor *k+1,* existem dois outros setores. Podemos considerar que o disco 1 nessa figura possui um fator de entrelaçamento igual a zero.

Voltando ao exemplo no qual os setores 4 e 5 são lidos, mas agora considerando um entrelaçamento de 2, após o cabeçote sair do setor 4, ele passa sobre os setores 15 e 10 antes de chegar no início do setor 5. Desta forma, quando o processador mandar o comando de leitura do setor 5, este não terá passado ainda sob o cabeçote. O melhor fator de entrelaçamento para uma determinado disco depende da velocidade do processador, do barramento, do controlador e da velocidade de rotação do disco.

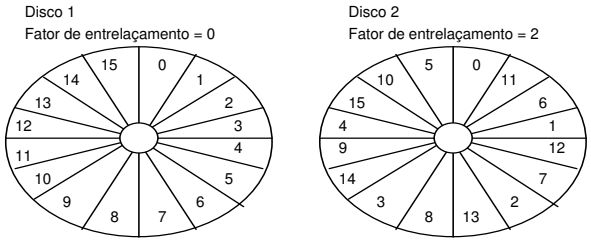

Figura 5.3 - Trilha com 16 setores e diferentes fatores de entrelaçamento.

Escalonamento de disco

Como se sabe, uma das principais funções do SO é gerenciar os recursos do sistema de forma eficaz. Por outro lado, o disco rígido é um dos principais recursos de um sistema

multiprogramado. Uma operação de E/S em disco envolve uma chamada de sistema e a especificação de uma série de parâmetros: tipo de operação (leitura ou escrita), número de bytes a serem transferidos, endereço de memória (*buffer*) e endereço no disco. No caso de um sistema multiprogramado, pode-se ter vários processos realizando "simultaneamente" pedidos desse tipo e sendo bloqueados até que a operação seja realizada. O problema do escalonador do disco consiste em ordenar e atender os pedidos de E/S, de forma a realizar um bom atendimento, buscando minimizar o tempo em que processos permanecem bloqueados.

O tempo necessário a uma operação de E/S em disco tem como principal componente o tempo de *seek*. Nos algoritmos de escalonamento apresentados a seguir, todos eles, com exceção do FCFS (ou FIFO), se preocupam em atender os pedidos de forma a minimizar o tempo médio de *seek*.

- **FCFS (*First Come First Served*):** É o algoritmo de escalonamento mais simples. As solicitações de acesso são atendidas na ordem em que os pedidos são feitos.
- **SSTF (*Shortest Seek Time First*):** O próximo pedido a ser atendido é aquele que se refere ao cilindro mais próximo do cilindro atual (isto é, aquele que envolve a menor movimentação do braço do disco). Novos pedidos são ordenados em relação ao cilindro atual, usando uma lista duplamente encadeada. Como sempre é selecionado o pedido correspondente ao cilindro mais próximo, pode ocorrer postergação indefinida (*starvation*) de um pedido que refere um cilindro distante.
- **SCAN:** Esse algoritmo é uma variação do SSTF. Ele se diferencia por estipular um sentido preferencial para o movimento do cabeçote, como por exemplo, do cilindro mais externo para o mais interno. O algoritmo SCAN opera da seguinte maneira: enquanto restam pedidos no sentido corrente, o braço do disco continua se movendo nesse sentido, atendendo os pedidos (correspondentes aos cilindros) mais próximos; se não há mais pedido no sentido corrente (possivelmente porque o fim da superfície foi atingido), então o disco inverte o sentido do braço e se inicia uma varredura no sentido inverso. O comportamento do algoritmo SCAN é similar ao de um elevador (por isso ele também é denominado **algoritmo do elevador**). Sua vantagem é eliminar a possibilidade de *starvation.*
- **C-SCAN (*Circular SCAN*)**: Neste algoritmo os pedidos são atendidos em um só sentido da varredura, por exemplo, do cilindro mais externo para o mais interno. O C-SCAN elimina o seguinte defeito do algoritmo SCAN: imediatamente após inverter a varredura, o SCAN privilegia os pedidos correspondentes aos cilindros recém servidos e, por conseqüência, os pedidos dos cilindros do outro extremo da varredura – mais antigos – devem esperar.

Muitos fatores influem no desempenho de um algoritmo de escalonamento de disco, entre eles o número de pedidos (carga), a organização dos arquivos e a estrutura de diretórios do disco. Por essa razão, para facilitar a adequação do algoritmo de escalonamento a um sistema específico, este componente é um módulo a parte do SO. Para discos que não apresentam carga de trabalho elevada (por exemplo, *CD-ROM* e *ZIP disk*), o algoritmo de escalonamento é do tipo FCFS.

- **Vídeo**

O funcionamento desse dispositivo pode ser melhor entendido considerando as unidades de vídeo antigas. Os primeiros terminais de vídeo eram formados por um teclado e um monitor cuja tela era organizada em uma matriz composta por 40 linhas e 80 colunas, na qual cada elemento correspondia a um caractere. Os caracteres, por sua vez, eram armazenados em uma memória de vídeo, cujas posições correspondiam a elementos dessa matriz. Na realidade, a cada elemento da matriz eram associados dois bytes: um que correspondia ao código ASCII do caractere, e outro ao atributo (fundo invertido, piscando, etc). Devido a essa característica, esse tipo de terminal é denominado de "memória mapeada". Alguns desses terminais ainda eram ditos inteligentes, pois eram capazes de aceitar seqüências de comandos como movimentar e posicionar o cursor, limpar a tela, executar rolamento de tela (*scroll*). Internamente, o terminal possuia um controlador de vídeo e um controlador de comunicação serial.

Embora hoje em dia a realidade seja outra, o princípio continua o mesmo. A diferença dos vídeos atuais está em dois pontos. O primeiro é que o controlador de vídeo é integrado na placa mãe do computador, ou interconectado diretamente no barramento do computador através de uma interface de vídeo. Isto aumenta consideravelmente a interação entre vídeo, processador e outros dispositivos de E/S. O segundo é a adição de capacidades gráficas aos vídeos. Os vídeos continuam com a mesma estrutura de seus ancestrais, isto é, o vídeo ainda é visto como uma matriz de linhas e de colunas, entretanto essa matriz pode ser acessada de dois modos diferentes: texto ou gráfico. No modo texto, o vídeo é organizado em uma matriz de caracteres e as posições da memória de video correspondem a caracteres e seus atributos. No modo gráfico, a matriz é organizada em *pixels*; cada *pixel* tem associada uma série de posições de memória correspondentes a suas cores. A memória de vídeo determina a resolução (número de linhas e colunas da matriz) e o número de cores de cada *pixel*. A capacidade de resolução e número de cores determina o tipo do controlador (EGA, VGA, SVGA, etc).

O *driver* de terminal (vídeo) continua com a mesma função básica do início da era dos terminais: programar o controlador de vídeo. Essa programação consiste essencialmente em determinar a memória de vídeo disponível, o modo de operação (texto ou gráfico), freqüências para a varredura horizontal e vertical, etc. Os ambientes gráficos X-windows, gnome, KDE, etc., constituem uma aplicação que intercepta e processa as informações provenientes do *driver* de teclado e do mouse. Esses ambientes constróem imagens na memória de vídeo e enviam comandos para o controlador de vídeo de forma a termos a realimentação visual que estamos acostumados a ver.

As placas aceleradoras e 3D, bastante em moda atualmente, também seguem esse mesmo princípio. A diferença está no fato de que o controlador de vídeo é mais potente e realiza por hardware uma série de funções bastante comuns em efeitos de animações e de desenho. No caso específico de uma placa 3D, existe um potente processador gráfico capaz de efetuar operações de visualização como perspectiva, iluminação, etc., bastando apenas, para isso que a aplicação forneça um comando.

- **Teclado**

O teclado é o principal periférico de entrada, utilizado na interação direta dos usuários com o computador. O princípio de operação do teclado é bastante simples: gerar um símbolo para cada tecla pressionada. Mecanicamente, um teclado pode ser visto como

uma matriz de *i* linhas e *j* colunas as quais entram em contato quando uma tecla é pressionada. A cada elemento *i,j* da matriz corresponde um caractere (tecla).

Quando uma tecla é pressionada, o teclado identifica a linha e a coluna associadas a essa tecla e gera um código denominado *scan code* (código de varredura), o qual é colocado no registrador de dados da interface do teclado. Além disso, gera uma interrupção o que ocasiona a execução do tratador de interrupções do teclado. O tratador lê o registrador de dados para recuperar o *scan code* da tecla pressionada. Com base no *scan code*, o tratador consulta uma tabela interna, substituindo o *scan code* pelo código ASCII correspondente à tecla pressionada. O código ASCII, em seguida, é armazenado em uma região especial da memória (*buffer* de teclado) de onde poderá ser recuperado por chamadas do SO. A leitura de caracteres é disponibilizada ao usuário final através de rotinas de biblioteca do tipo getc(), da linguagem C.

Alguns detalhes foram omitidos na descrição acima, pois nosso objetivo é dar apenas uma visão geral do funcionamento do teclado. Na verdade, para cada tecla, são geradas duas interrupções (com *scan code* diferentes), uma para indicar o pressionar da tecla e outra para sinalizar que a tecla foi liberada. Lembre que a tecla *shift*, por exemplo, fica pressionada enquanto se digita uma letra maiúscula.

Nós, usuários, queremos visualizar os caracteres que digitamos na tela de nosso computador. Esse procedimento de ler dados do teclado e escrevê-los na tela denomina-se **ecoamento**. Outra característica bastante comum na utilização de computadores é a possibilidade de "abrir" várias janelas. Nesse caso, os caracteres digitados devem ser direcionados à janela correta. Esse "direcionamento" é feito através do conceito de "janela ativa" e da interação entre o *driver* de teclado e o *driver* de vídeo sobre o *buffer* de teclado.

- **Rede**

Uma rede pode ser genericamente definida como um conjunto de computadores interconectados de forma a compartilhar recursos comuns: discos, impressoras, arquivos, etc. A forma mais simples de rede é uma rede ponto a ponto, na qual dois computadores podem ser interligados diretamente através de suas interfaces paralelas e seriais padrões. Entretanto, o que se convenciona denominar de rede supõe o emprego de um hardware especial – a interface de rede – que provê a interconexão de várias máquinas segundo uma tecnologia específica. A tecnologia empregada pelo hardware da interface, ethernet ou ATM, por exemplo, determina como fisicamente os dados serão transmitidos, a velocidade de transmissão, a capacidade de transmitir e receber ao mesmo tempo (*full duplex*), entre outras. Independente da tecnologia e dos aspectos físicos ligados à transmissão/recepção, o software de gerenciamento deve tratar de problemas similares.

Uma placa de rede é constituída por um hardware que transforma os sinais digitais em sinais analógicos segundo sua tecnologia. Ela também possui internamente uma capacidade de memória (*buffers*) na qual os dados a serem transmitidos, ou recebidos, são armazenados. Algumas placas de rede possuem um processador dedicado a essa função (o controlador de rede). A placa de rede trabalha sob um certo aspecto orientada a eventos. Um evento é o final da transmissão. A ocorrência desse evento é interpretada como disponibilidade para nova transmissão. Outro evento é a recepção de uma mensagem. Nesse caso, a mensagem deve ser lida. Essa descrição, entretanto, é extremamente simplista. O software de rede é bastante complexo e envolve muitas

abstrações; por isso, normalmente ele é organizado em camadas (você já deve ter ouvido falar no modelo OSI/ISO).

Para apresentar um pouco o grau de dificuldade e os problemas a serem gerenciados por um software de rede, vamos examinar o caminho seguido por uma mensagem. Vamos considerar que um processo A quer enviar uma mensagem para um processo B em uma máquina remota. A quantidade de bytes dessa mensagem pode ser maior que a quantidade máxima permitida fisicamente para transmissão. Essa quantidade depende da tecnologia de rede empregada. Cada tipo de rede suporta o que se denomina de MTU (*Maximum Transfer Unit*). Se nossa mensagem é maior que o MTU, ela deve ser dividida em pedaços (pacotes) de tamanho máximo igual a MTU. A primeira tarefa do software do lado remetente é realizar essa "quebra" da mensagem original. Esse processo denomina-se **fragmentação**. Ao final da transmissão de cada pacote, uma interrupção é gerada pela placa de rede para sinalizar esse evento e indicar que ela está pronta para enviar outro pacote.

No outro extremo, a máquina do processo B, recebe os pacotes. Para cada pacote recebido é gerada uma interrupção. Esta interrupção tem por objetivo transferir este pacote para um *buffer* maior e reconstruir com os pacotes a mensagem original. Aqui surge outro problema. Como o *driver* de rede identifica que estes dados são para o processo B e não para um outro processo C? Pior ainda, como ele não mistura estes pacotes? Uma solução é criar canais lógicos entre os processos. Desta forma o software de rede antes de começar a enviar/receber dados deve estabelecer uma conexão (canal lógico) entre os dois processos que desejam comunicar. Este canal lógico identificará então os pares. O *driver* de rede – estando consciente dos canais – poderá redirecionar os pacotes recebidos para um, ou para outro canal.

Esse cenário, entretanto, é simplificado, pois ele não considera a ocorrência de erros. Os pacotes podem ser perdidos, ou por erros de transmissão no meio físico, ou por falta de espaço no *buffer* do destinatário. Pacotes podem ainda ser invertidos, isto é, o pacote *i+1* pode chegar antes do pacote *i*. Isto pode ocorrer quando, devido a um erro, um pacote *i* é retransmitido, enquanto o *i+1* foi recebido corretamente, ou ainda quando o pacote *i+1* utiliza um caminho mais curto que o pacote *i* (roteamento). O destinatário deve então ser responsável por contar e por sinalizar ao remetente a perda de pacotes. Ele deve ainda ser capaz de ordenar os pacotes recebidos.

Todos esses procedimentos e muitos outros são gerenciados pelo que se denomina de protocolo. Um exemplo bastante conhecido de protocolos é a família TCP/IP, pois toda a Internet está baseada nessa família. Os problemas mencionados acima, entre outros, são tratados pela implementação de protocolos, os quais são organizados em níveis (camadas). No nível mais baixo, estão as placas de rede; no mais alto, a aplicação do usuário.

Exercícios

1) Mostre como é a técnica de *interleaving* (entrelaçamento) utilizada em discos e para que ela serve.

2) Descreva o funcionamento da otimização de *seek* em acesso a disco quando o algoritmo SSTF (*shortest seek time first* - menor *seek* primeiro) é utilizado.

4) Descreva o funcionamento da otimização de *seek* em acesso a disco quando o algoritmo SCAN (elevador) é utilizado.

5) Considere que um disco tenha 100 cilindros (numerados de 0 a 99), que o braço do disco esteja posicionado no cilindro 0 (mais externo), que o tempo atual seja 0, que uma operação de E/S demore (**10+d**) u.t. (unidades de tempo), onde **d** é a distância percorrida pelo braço[1] e que ocorra a seqüência de pedidos de acesso mostrada abaixo. (a) Preencha a segunda coluna com o tempo no qual será atendido cada pedido, considerando o algoritmo SSTF; (b) Preencha a terceira coluna com o tempo no qual será atendido cada pedido, considerando o algoritmo SCAN.

Pedido		Atendimento SSTF	Atendimento SCAN
tempo	cilindro		
0	30	0 – 40	0 – 40
10	20		
15	50		
25	40		
30	35		
35	30	40 – 50	
45	10		
55	70		

[1] A distância entre os cilindros a e b é dada por |b-a|.