Embora a ideia de combinar a telefonia e a computação em um dispositivo semelhante a um telefone exista desde a década de 1970 também, o primeiro smartphone de verdade não foi inventado até meados de 1990, quando a Nokia lançou o N9000, que literalmente combinava dois dispositivos mormente separados: um telefone e um **PDA** (**Personal Digital Assistant** — assistente digital pessoal). Em 1997, a Ericsson cunhou o termo *smartphone* para o seu "Penelope" GS88.

Agora que os smartphones tornaram-se onipresentes, a competição entre os vários sistemas operacionais tornou-se feroz e o desfecho é mais incerto ainda que no mundo dos PCs. No momento em que escrevo este livro, o Android da Google é o sistema operacional dominante, com o iOS da Apple sozinho em segundo lugar, mas esse nem sempre foi o caso e tudo pode estar diferente de novo em apenas alguns anos. Se algo está claro no mundo dos smartphones é que não é fácil manter-se no topo por muito tempo.

Afinal de contas, a maioria dos smartphones na primeira década após sua criação era executada em **Symbian** OS. Era o sistema operacional escolhido para as marcas populares como Samsung, Sony Ericsson, Motorola e especialmente Nokia. No entanto, outros sistemas operacionais como o Blackberry OS da **RIM** (introduzido para smartphones em 2002) e o iOS da Apple (lançado para o primeiro **iPhone** em 2007) começaram a ganhar mercado do Symbian. Muitos esperavam que o RIM dominasse o mercado de negócios, enquanto o iOS seria o rei dos dispositivos de consumo. A participação de mercado do Symbian desabou. Em 2011, a Nokia abandonou o Symbian e anunciou que se concentraria no Windows Phone como sua principal plataforma. Por algum tempo, a Apple e o RIM eram festejados por todos (embora não tão dominantes quanto o Symbian tinha sido), mas não levou muito tempo para o Android, um sistema operacional baseado no Linux lançado pelo Google em 2008, dominar os seus rivais.

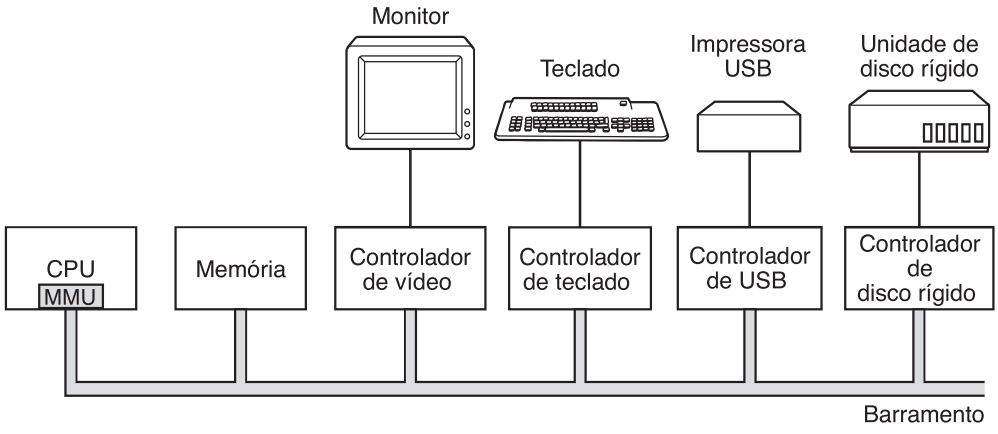
Para os fabricantes de telefone, o Android tinha a vantagem de ser um sistema aberto e disponível sob uma licença permissiva. Como resultado, podiam mexer nele e adaptá-lo a seu hardware com facilidade. Ele também tem uma enorme comunidade de desenvolvedores escrevendo aplicativos, a maior parte na popular linguagem de programação Java. Mesmo assim, os últimos anos mostraram que o domínio talvez não dure, e os competidores do Android estão ansiosos para retomar parte da sua participação de mercado. Examinaremos o Android detalhadamente na Seção 10.8.

1.3 Revisão sobre hardware de computadores

Um sistema operacional está intimamente ligado ao hardware do computador no qual ele é executado. Ele estende o conjunto de instruções do computador e gerencia seus recursos. Para funcionar, ele deve conhecer profundamente o hardware, pelo menos como aparece para o programador. Por esta razão, vamos revisar brevemente o hardware de computadores como encontrado nos computadores pessoais modernos. Depois, podemos começar a entrar nos detalhes do que os sistemas operacionais fazem e como eles funcionam.

Conceitualmente, um computador pessoal simples pode ser abstraído em um modelo que lembra a Figura 1.6. A CPU, memória e dispositivos de E/S estão todos conectados por um sistema de barramento e comunicam-se uns com os outros sobre ele. Computadores pessoais modernos têm uma estrutura mais complicada,

FIGURA 1.6 Alguns dos componentes de um computador pessoal simples.



envolvendo múltiplos barramentos, os quais examinaremos mais tarde. Por ora, este modelo será suficiente. Nas seções seguintes, revisaremos brevemente esses componentes e examinaremos algumas das questões de hardware que interessam aos projetistas de sistemas operacionais. Desnecessário dizer que este será um resumo bastante compacto. Muitos livros foram escritos sobre o tema hardware e organização de computadores. Dois títulos bem conhecidos foram escritos por Tanenbaum e Austin (2012) e Patterson e Hennessy (2013).

1.3.1 Processadores

O "cérebro" do computador é a CPU. Ela busca instruções da memória e as executa. O ciclo básico de toda CPU é buscar a primeira instrução da memória, decodificá-la para determinar o seu tipo e operandos, executá-la, e então buscar, decodificar e executar as instruções subsequentes. O ciclo é repetido até o programa terminar. É dessa maneira que os programas são executados.

Cada CPU tem um conjunto específico de instruções que ela consegue executar. Desse modo, um processador x86 não pode executar programas ARM e um processador ARM não consegue executar programas x86. Como o tempo para acessar a memória para buscar uma instrução ou palavra dos operandos é muito maior do que o tempo para executar uma instrução, todas as CPUs têm alguns registradores internos para armazenamento de variáveis e resultados temporários. Desse modo, o conjunto de instruções geralmente contém instruções para carregar uma palavra da memória para um registrador e armazenar uma palavra de um registrador para a memória. Outras instruções combinam dois operandos provenientes de registradores, da memória, ou ambos, para produzir um resultado como adicionar duas palavras e armazenar o resultado em um registrador ou na memória.

Além dos registradores gerais usados para armazenar variáveis e resultados temporários, a maioria dos computadores tem vários registradores especiais que são visíveis para o programador. Um desses é o **contador de programa**, que contém o endereço de memória da próxima instrução a ser buscada. Após essa instrução ter sido buscada, o contador de programa é atualizado para apontar a próxima instrução.

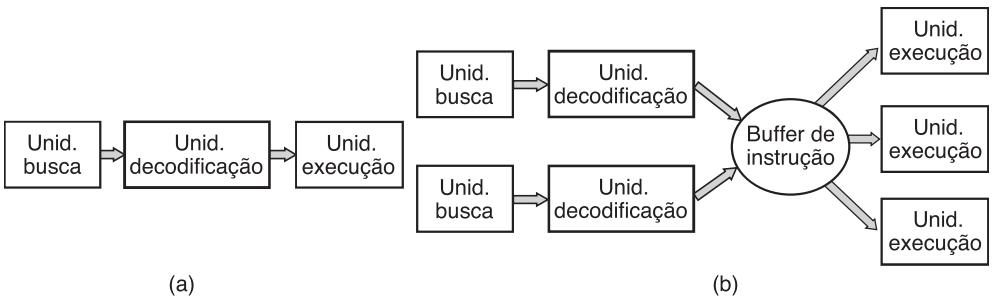
Outro registrador é o **ponteiro de pilha**, que aponta para o topo da pilha atual na memória. A pilha contém uma estrutura para cada rotina que foi chamada, mas ainda não encerrada. Uma estrutura de pilha de rotina armazena aqueles parâmetros de entrada, variáveis locais e variáveis temporárias que não são mantidas em registradores.

Outro registrador ainda é o **PSW (Program Status Word** — palavra de estado do programa). Esse registrador contém os bits do código de condições, que são estabelecidos por instruções de comparação, a prioridade da CPU, o modo de execução (usuário ou núcleo) e vários outros bits de controle. Programas de usuários normalmente podem ler todo o PSW, mas em geral podem escrever somente parte dos seus campos. O PSW tem um papel importante nas chamadas de sistema e em E/S.

O sistema operacional deve estar absolutamente ciente de todos os registros. Quando realizando a multiplexação de tempo da CPU, ele muitas vezes vai interromper o programa em execução para (re)começar outro. Toda vez que ele para um programa em execução, o sistema operacional tem de salvar todos os registradores de maneira que eles possam ser restaurados quando o programa for executado mais tarde.

Para melhorar o desempenho, os projetistas de CPU há muito tempo abandonaram o modelo simples de buscar, decodificar e executar uma instrução de cada vez. Muitas CPUs modernas têm recursos para executar mais de uma instrução ao mesmo tempo. Por exemplo, uma CPU pode ter unidades de busca, decodificação e execução separadas, assim enquanto ela está executando a instrução n, poderia também estar decodificando a instrução $n + 1$ e buscando a instrução $n + 2$. Uma organização com essas características é chamada de **pipeline** e é ilustrada na Figura 1.7(a) para um pipeline

FIGURA 1.7 (a) Um pipeline com três estágios. (b) Uma CPU superescalar.



com três estágios. Pipelines mais longos são comuns. Na maioria desses projetos, uma vez que a instrução tenha sido levada para o pipeline, ela deve ser executada, mesmo que a instrução anterior tenha sido um desvio condicional tomado. Pipelines provocam grandes dores de cabeça nos projetistas de compiladores e de sistemas operacionais, pois expõem as complexidades da máquina subjacente e eles têm de lidar com elas.

Ainda mais avançada que um projeto de pipeline é uma CPU **superescalar**, mostrada na Figura 1.7(b). Nesse projeto, unidades múltiplas de execução estão presentes. Uma unidade para aritmética de números inteiros, por exemplo, uma unidade para aritmética de ponto flutuante e uma para operações booleanas. Duas ou mais instruções são buscadas ao mesmo tempo, decodificadas e jogadas em um buffer de instrução até que possam ser executadas. Tão logo uma unidade de execução fica disponível, ela procura no buffer de instrução para ver se há uma instrução que ela pode executar e, se assim for, ela remove a instrução do buffer e a executa. Uma implicação desse projeto é que as instruções do programa são muitas vezes executadas fora de ordem. Em geral, cabe ao hardware certificar-se de que o resultado produzido é o mesmo que uma implementação sequencial conseguiria, mas como veremos adiante, uma quantidade incômoda de tarefas complexas é empurrada para o sistema operacional.

A maioria das CPUs — exceto aquelas muito simples usadas em sistemas embarcados, tem dois modos, núcleo e usuário, como mencionado anteriormente. Em geral, um bit no PSW controla o modo. Quando operando em modo núcleo, a CPU pode executar todas as instruções em seu conjunto de instruções e usar todos os recursos do hardware. Em computadores de mesa e servidores, o sistema operacional normalmente opera em modo núcleo, dando a ele acesso a todo o hardware. Na maioria dos sistemas embarcados, uma parte pequena opera em modo núcleo, com o resto do sistema operacional operando em modo usuário.

Programas de usuários sempre são executados em modo usuário, o que permite que apenas um subconjunto das instruções possa ser executado e um subconjunto dos recursos possa ser acessado. Geralmente, todas as instruções envolvendo E/S e proteção de memória são inacessíveis no modo usuário. Alterar o bit de modo PSW para modo núcleo também é proibido, claro.

Para obter serviços do sistema operacional, um programa de usuário deve fazer uma **chamada de sistema**, que, por meio de uma instrução TRAP, chaveia do modo usuário para o modo núcleo e passa o controle para o sistema operacional. Quando o trabalho é finalizado, o controle retorna para o programa do usuário na instrução posterior à chamada de sistema. Explicaremos os detalhes do mecanismo de chamada de sistema posteriormente neste capítulo. Por ora, pense nele como um tipo especial de procedimento de instrução de chamada que tem a propriedade adicional de chavear do modo usuário para o modo núcleo. Como nota a respeito da tipografia, usaremos a fonte Helvética com letras minúsculas para indicar chamadas de sistema ao longo do texto, como: read.

Vale a pena observar que os computadores têm outras armadilhas ("traps") além da instrução para executar uma chamada de sistema. A maioria das outras armadilhas é causada pelo hardware para advertir sobre uma situação excepcional como uma tentativa de divisão por 0 ou um *underflow* (incapacidade de representação de um número muito pequeno) em ponto flutuante. Em todos os casos o sistema operacional assume o controle e tem de decidir o que fazer. Às vezes, o programa precisa ser encerrado por um erro. Outras vezes, o erro pode ser ignorado (a um número com underflow pode-se atribuir o valor 0). Por fim, quando o programa anunciou com antecedência que ele quer lidar com determinados tipos de condições, o controle pode ser passado de volta ao programa para deixá-lo cuidar do problema.

Chips multithread e multinúcleo

A lei de Moore afirma que o número de transistores em um chip dobra a cada 18 meses. Tal "lei" não é nenhum tipo de lei da física, como a conservação do momento, mas é uma observação do cofundador da Intel, Gordon Moore, de quão rápido os engenheiros de processo nas empresas de semicondutores são capazes de reduzir o tamanho dos seus transistores. A lei de Moore se mantém há mais de três décadas até agora e espera-se que se mantenha por pelo menos mais uma. Após isso, o número de átomos por transistor tornar-se-á pequeno demais e a mecânica quântica começará a ter um papel maior, evitando uma redução ainda maior dos tamanhos dos transistores.

A abundância de transistores está levando a um problema: o que fazer com todos eles? Vimos uma abordagem acima: arquiteturas superescalares, com múltiplas unidades funcionais. Mas à medida que o número de transistores aumenta, mais ainda é possível. Algo óbvio a ser feito é colocar memórias cache maiores no chip da CPU. Isso de fato está acontecendo, mas finalmente o ponto de ganhos decrescentes será alcançado.

O próximo passo óbvio é replicar não apenas as unidades funcionais, mas também parte da lógica de controle.

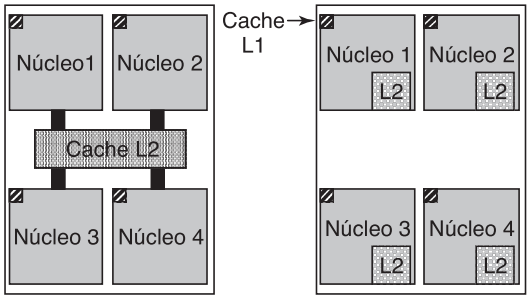
O Pentium 4 da Intel introduziu essa propriedade, chamada **multithreading** ou **hyperthreading** (o nome da Intel para ela), ao processador x86 e vários outros chips de CPU também o têm — incluindo o SPARC, o Power5, o Intel Xeon e a família Intel Core. Para uma primeira aproximação, o que ela faz é permitir que a CPU mantenha o estado de dois threads diferentes e então faça o chaveamento entre um e outro em uma escala de tempo de nanossegundos. (Um thread é um tipo de processo leve, o qual, por sua vez, é um programa de execução; entraremos nos detalhes no Capítulo 2.) Por exemplo, se um dos processos precisa ler uma palavra da memória (o que leva muitos ciclos de relógio), uma CPU multithread pode simplesmente fazer o chaveamento para outro thread. O multithreading não proporciona paralelismo real. Apenas um processo de cada vez é executado, mas o tempo de chaveamento de thread é reduzido para a ordem de um nanossegundo.

O multithreading tem implicações para o sistema operacional, pois cada thread aparece para o sistema operacional como uma CPU em separado. Considere um sistema com duas CPUs efetivas, cada uma com dois threads. O sistema operacional verá isso como quatro CPUs. Se há apenas trabalho suficiente para manter duas CPUs ocupadas em um determinado momento no tempo, ele pode escalonar inadvertidamente dois threads para a mesma CPU, com a outra completamente ociosa. Essa escolha é muito menos eficiente do que usar um thread para cada CPU.

Além do multithreading, muitos chips de CPU têm agora quatro, oito ou mais processadores completos ou **núcleos** neles. Os chips multinúcleo da Figura 1.8 efetivamente trazem quatro minichips, cada um com sua CPU independente. (As caches serão explicadas a seguir.) Alguns processadores, como o Intel Xeon Phi e o Tilera TilePro, já apresentam mais de 60 núcleos em um único chip. Fazer uso de um chip com múltiplos núcleos como esse definitivamente exigirá um sistema operacional de multiprocessador.

Incidentalmente, em termos de números absolutos, nada bate uma **GPU (Graphics Processing Unit** — unidade de processamento gráfico) moderna. Uma GPU é um processador com, literalmente, milhares de núcleos minúsculos. Eles são muito bons para realizar muitos pequenos cálculos feitos em paralelo, como reproduzir polígonos em aplicações gráficas. Não são tão bons em tarefas em série. Eles também são difíceis de programar. Embora GPUs possam ser úteis para sistemas operacionais (por exemplo, codificação ou processamento de tráfego de rede), não é provável que grande parte do sistema operacional em si vá ser executada nas GPUs.

FIGURA 1.8 (a) Chip quad-core com uma cache L2 compartilhada. (b) Um chip quad-core com caches L2 separadas.



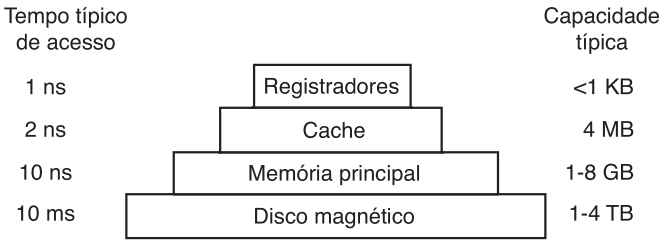
1.3.2 Memória

O segundo principal componente em qualquer computador é a memória. Idealmente, uma memória deve ser rápida ao extremo (mais rápida do que executar uma instrução, de maneira que a CPU não seja atrasada pela memória), abundantemente grande e muito barata. Nenhuma tecnologia atual satisfaz todas essas metas, assim uma abordagem diferente é tomada. O sistema de memória é construído como uma hierarquia de camadas, como mostrado na Figura 1.9. As camadas superiores têm uma velocidade mais alta, capacidade menor e um custo maior por bit do que as inferiores, muitas vezes por fatores de um bilhão ou mais.

A camada superior consiste em registradores internos à CPU. Eles são feitos do mesmo material que a CPU e são, desse modo, tão rápidos quanto ela. Em consequência, não há um atraso ao acessá-los. A capacidade de armazenamento disponível neles é tipicamente 32 × 32 bits em uma CPU de 32 bits e 64 × 64 bits em uma CPU de 64 bits. Menos de 1 KB em ambos os casos. Os programas devem gerenciar os próprios registradores (isto é, decidir o que manter neles) no software.

Em seguida, vem a memória cache, que é controlada principalmente pelo hardware. A memória principal é dividida em **linhas de cache**, tipicamente 64 bytes, com endereços 0 a 63 na linha de cache 0, 64 a 127 na linha de cache 1 e assim por diante. As linhas de cache mais

FIGURA 1.9 Uma hierarquia de memória típica. Os números são apenas aproximações.



utilizadas são mantidas em uma cache de alta velocidade localizada dentro ou muito próximo da CPU. Quando o programa precisa ler uma palavra de memória, o hardware de cache confere se a linha requisitada está na cache. Se ela estiver **presente na cache** (*cache hit*), a requisição é atendida e nenhuma requisição de memória é feita para a memória principal sobre o barramento. *Cache hits* costumam levar em torno de dois ciclos de CPU. Se a linha requisitada estiver ausente da cache (*cache miss*), uma requisição adicional é feita à memória, com uma penalidade de tempo substancial. A memória da cache é limitada em tamanho por causa do alto custo. Algumas máquinas têm dois ou três níveis de cache, cada um mais lento e maior do que o antecedente.

O conceito de *caching* exerce um papel importante em muitas áreas da ciência de computadores, não apenas na colocação de linhas de RAM na cache. Sempre que um recurso pode ser dividido em partes, algumas das quais são usadas com muito mais frequência que as outras, o *caching* é muitas vezes utilizado para melhorar o desempenho. Sistemas operacionais o utilizam seguidamente. Por exemplo, a maioria dos sistemas operacionais mantém (partes de) arquivos muito usados na memória principal para evitar ter de buscá-los do disco de modo repetido. Similarmente, os resultados da conversão de nomes de rota longa como

/home/ast/projects/minix3/src/kernel/clock.c

no endereço de disco onde o arquivo está localizado podem ser registrados em cache para evitar buscas repetidas. Por fim, quando o endereço de uma página da web (URL) é convertido em um endereço de rede (endereço IP), o resultado pode ser armazenado em cache para uso futuro. Há muitos outros usos.

Em qualquer sistema de cache, muitas perguntas surgem relativamente rápido, incluindo:

1. Quando colocar um novo item na cache.
2. Em qual linha de cache colocar o novo item.
3. Qual item remover da cache quando for preciso espaço.
4. Onde colocar um item recentemente desalojado na memória maior.

Nem toda pergunta é relevante para toda situação de cache. Para linhas de cache da memória principal na cache da CPU, um novo item geralmente será inserido em cada ausência de cache. A linha de cache a ser usada em geral é calculada usando alguns dos bits de alta ordem do endereço de memória mencionado. Por exemplo, com 4.096 linhas de cache de 64 bytes e endereços de 32 bits, os bits 6 a 17 podem ser usados para especificar a linha de cache, com os bits de 0 a 5 especificando os bytes dentro da linha de cache. Aqui, o item a ser removido é o mesmo de onde os novos dados são inseridos, mas em outros sistemas este pode não ser o caso. Por fim, quando uma linha de cache é reescrita para a memória principal (se ela tiver sido modificada desde que foi colocada na cache), o lugar na memória para reescrevê-la é determinado unicamente pelo endereço em questão.

Caches são uma ideia tão boa que as CPUs modernas têm duas delas. O primeiro nível, ou **cache L1**, está sempre dentro da CPU e normalmente alimenta instruções decodificadas no mecanismo de execução da CPU. A maioria dos chips tem uma segunda cache L1 para palavras de dados usadas com muita intensidade. As caches L1 são em geral de 16 KB cada. Além disso, há muitas vezes uma segunda cache, chamada de **cache L2**, que armazena vários megabytes de palavras de memória recentemente usadas. A diferença entre as caches L1 e L2 encontra-se na sincronização. O acesso à cache L1 é feito sem atraso algum, enquanto o acesso à cache L2 envolve um atraso de um ou dois ciclos de relógio.

Em chips de multinúcleo, os projetistas têm de decidir onde colocar as caches. Na Figura 1.8(a), uma única cache L2 é compartilhada por todos os núcleos. Essa abordagem é usada em chips de multinúcleo da Intel. Em comparação, na Figura 1.8(b), cada núcleo tem sua própria cache L2. Essa abordagem é usada pela AMD. Cada estratégia tem seus prós e contras. Por exemplo, a cache L2 compartilhada da Intel exige um controlador de cache mais complicado, mas o método AMD torna mais difícil manter a consistência entre as caches L2.

A memória principal vem a seguir na hierarquia da Figura 1.9. Trata-se da locomotiva do sistema de memória. A memória principal é normalmente chamada de **RAM** (**Random Access Memory** — memória de acesso aleatório). Os mais antigos às vezes a chamam de **memória de núcleo** (**core memory**), pois os computadores nas décadas de 1950 e 1960 usavam minúsculos núcleos de ferrite magnetizáveis como memória principal. Hoje, as memórias têm centenas de megabytes a vários gigabytes e vêm crescendo rapidamente. Todas as requisições da CPU que não podem ser atendidas pela cache vão para a memória principal.

Além da memória principal, muitos computadores têm uma pequena memória de acesso aleatório não volátil. Diferentemente da RAM, a memória não volátil não perde o seu conteúdo quando a energia é desligada. A **ROM** (**Read Only Memory** — memória somente de leitura) é programada na fábrica e não pode ser modificada depois. Ela é rápida e barata. Em alguns computadores, o carregador (*bootstrap loader*) usado para

inicializar o computador está contido na ROM. Também algumas placas de E/S vêm com a ROM para lidar com o controle de dispositivos de baixo nível.

A **EEPROM** (**Electrically Erasable PROM** — ROM eletricamente apagável) e a **memória flash** também são não voláteis, mas, diferentemente da ROM, podem ser apagadas e reescritas. No entanto, escrevê-las leva muito mais tempo do que escrever em RAM, então elas são usadas da mesma maneira que a ROM, apenas com a característica adicional de que é possível agora corrigir erros nos programas que elas armazenam mediante sua regravação.

A memória flash também é bastante usada como um meio de armazenamento em dispositivos eletrônicos portáteis. Ela serve como um filme em câmeras digitais e como disco em reprodutores de música portáteis, apenas como exemplo. A memória flash é intermediária em velocidade entre a RAM e o disco. Também, diferentemente da memória de disco, ela se desgasta quando apagada muitas vezes.

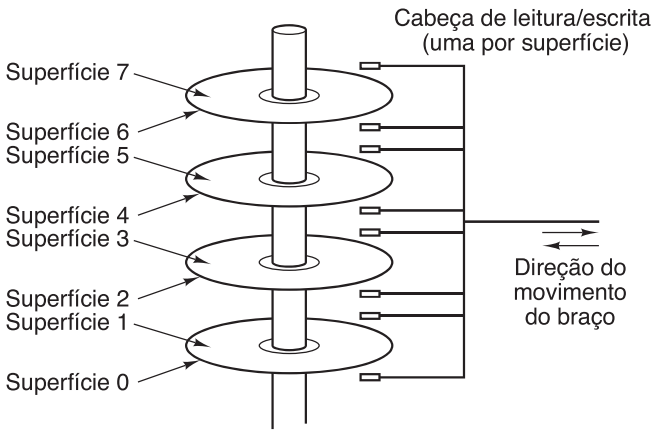
Outro tipo ainda de memória é a CMOS, que é volátil. Muitos computadores usam a memória CMOS para armazenar a hora e a data atualizadas. A memória CMOS e o circuito de relógio que incrementa o tempo registrado nela são alimentados por uma bateria pequena, então a hora é atualizada corretamente, mesmo quando o computador estiver desligado. A memória CMOS também pode conter os parâmetros de configuração, como de qual disco deve se carregar o sistema. A CMOS é usada porque consome tão pouca energia que a bateria original instalada na fábrica muitas vezes dura por vários anos. No entanto, quando ela começa a falhar, o computador pode parecer ter a doença de Alzheimer, esquecendo coisas que ele sabia há anos, como de qual disco rígido carregar o sistema operacional.

1.3.3 Discos

Em seguida na hierarquia está o disco magnético (disco rígido). O armazenamento de disco é duas ordens de magnitude mais barato, por bit, que o da RAM e frequentemente duas ordens de magnitude maior também. O único problema é que o tempo para acessar aleatoriamente os dados é próximo de três ordens de magnitude mais lento. Isso ocorre porque o disco é um dispositivo mecânico, como mostrado na Figura 1.10.

Um disco consiste em um ou mais pratos metálicos que rodam a 5.400, 7.200, 10.800 RPM, ou mais. Um braço mecânico move-se sobre esses pratos a partir da lateral, como o braço de toca-discos de um velho fonógrafo de 33 RPM para tocar discos de vinil. A

FIGURA 1.10 Estrutura de uma unidade de disco.



informação é escrita no disco em uma série de círculos concêntricos. Em qualquer posição do braço, cada uma das cabeças pode ler uma região circular chamada de **trilha**. Juntas, todas as trilhas de uma dada posição do braço formam um **cilindro**.

Cada trilha é dividida em um determinado número de setores, com tipicamente 512 bytes por setor. Em discos modernos, os cilindros externos contêm mais setores do que os internos. Mover o braço de um cilindro para o próximo leva em torno de 1 ms. Movê-lo para um cilindro aleatório costuma levar de 5 a 10 ms, dependendo do dispositivo acionador. Uma vez que o braço esteja na trilha correta, o dispositivo acionador tem de esperar até que o setor desejado gire sob a cabeça, um atraso adicional de 5 a 10 ms, dependendo da RPM do dispositivo acionador. Assim que o setor estiver sob a cabeça, a leitura ou escrita ocorre a uma taxa de 50 MB/s em discos de baixo desempenho até 160 MB/s em discos mais rápidos.

Às vezes você ouvirá as pessoas falando sobre discos que não são discos de maneira alguma, como os **SSDs** (**Solid State Disks** — discos em estado sólido). SSDs não têm partes móveis, não contêm placas na forma de discos e armazenam dados na memória (flash). A única maneira pela qual lembram discos é que eles também armazenam uma quantidade grande de dados que não é perdida quando a energia é desligada.

Muitos computadores dão suporte a um esquema conhecido como **memória virtual**, que discutiremos de maneira mais aprofundada no Capítulo 3. Esse esquema torna possível executar programas maiores que a memória física colocando-os no disco e usando a memória principal como um tipo de cache para as partes mais intensivamente executadas. Esse esquema exige o remapeamento dos endereços de memória rapidamente para converter o endereço que o programa gerou para

o endereço físico em RAM onde a palavra está localizada. Esse mapeamento é feito por uma parte da CPU chamada **MMU** (**Memory Management Unit** — unidade de gerenciamento de memória), como mostrado na Figura 1.6.

A presença da cache e da MMU pode ter um impacto importante sobre o desempenho. Em um sistema de multiprogramação, quando há o chaveamento de um programa para outro, às vezes chamado de um **chaveamento de contexto**, pode ser necessário limpar todos os blocos modificados da cache e mudar os registros de mapeamento na MMU. Ambas são operações caras, e os programadores fazem o que podem para evitá-las. Veremos algumas das implicações de suas táticas mais tarde.

1.3.4 Dispositivos de E/S

A CPU e a memória não são os únicos recursos que o sistema operacional tem de gerenciar. Dispositivos de E/S também interagem intensamente com o sistema operacional. Como vimos na Figura 1.6, dispositivos de E/S consistem em geral em duas partes: um controlador e o dispositivo em si. O controlador é um chip ou um conjunto de chips que controla fisicamente o dispositivo. Ele aceita comandos do sistema operacional, por exemplo, para ler dados do dispositivo, e os executa.

Em muitos casos, o controle real do dispositivo é complicado e detalhado, então faz parte do trabalho do controlador apresentar uma interface mais simples (mas mesmo assim muito complexa) para o sistema operacional. Por exemplo, um controlador de disco pode aceitar um comando para ler o setor 11.206 do disco 2. O controlador tem então de converter esse número do setor linear para um cilindro, setor e cabeça. Essa conversão pode ser complicada porque os cilindros exteriores têm mais setores do que os interiores, e alguns setores danificados foram remapeados para outros. Então o controlador tem de determinar em qual cilindro está o braço do disco e dar a ele um comando correspondente à distância em número de cilindros. Ele deve aguardar até que o setor apropriado tenha girado sob a cabeça e então começar a ler e a armazenar os bits à medida que eles saem do acionador, removendo o cabeçalho e conferindo a soma de verificação (*checksum*). Por fim, ele tem de montar os bits que chegam em palavras e armazená-las na memória. Para fazer todo esse trabalho, os controladores muitas vezes contêm pequenos computadores embutidos que são programados para realizar o seu trabalho.

A outra parte é o dispositivo real em si. Os dispositivos possuem interfaces relativamente simples, tanto porque eles não podem fazer muito, como para padronizá-los. A padronização é necessária para que qualquer controlador de disco SATA possa controlar qualquer disco SATA, por exemplo. **SATA** é a sigla para **Serial ATA**, e **ATA** por sua vez é a sigla para **AT Attachment**. Caso você esteja curioso para saber o significado de AT, esta foi a segunda geração da "Personal Computer Advanced Technology" (tecnologia avançada de computadores pessoais) da IBM, produzida em torno do então extremamente potente processador 80286 de 6 MHz que a empresa introduziu em 1984. O que aprendemos disso é que a indústria de computadores tem o hábito de incrementar continuamente os acrônimos existentes com novos prefixos e sufixos. Também aprendemos que um adjetivo como "avançado" deve ser usado com grande cuidado, ou você passará ridículo daqui a trinta anos.

O SATA é atualmente o tipo de disco padrão em muitos computadores. Dado que a interface do dispositivo real está escondida atrás do controlador, tudo o que o sistema operacional vê é a interface para o controlador, o que pode ser bastante diferente da interface para o dispositivo.

Como cada tipo de controlador é diferente, diversos softwares são necessários para controlar cada um. O software que conversa com um controlador, dando a ele comandos e aceitando respostas, é chamado de **driver de dispositivo**. Cada fabricante de controladores tem de fornecer um driver para cada sistema operacional a que dá suporte. Assim, um digitalizador de imagens pode vir com drivers para OS X, Windows 7, Windows 8 e Linux, por exemplo.

Para ser usado, o driver tem de ser colocado dentro do sistema operacional de maneira que ele possa ser executado em modo núcleo. Na realidade, drivers podem ser executados fora do núcleo, e sistemas operacionais como Linux e Windows hoje em dia oferecem algum suporte para isso. A vasta maioria dos drivers ainda opera abaixo do nível do núcleo. Apenas muito poucos sistemas atuais, como o MINIX 3, operam todos os drivers em espaço do usuário. Drivers no espaço do usuário precisam ter permissão de acesso ao dispositivo de uma maneira controlada, o que não é algo trivial.

Há três maneiras pelas quais o driver pode ser colocado no núcleo. A primeira é religar o núcleo com o novo driver e então reinicializar o sistema. Muitos sistemas UNIX mais antigos funcionam assim. A segunda maneira é adicionar uma entrada em um arquivo do sistema operacional dizendo-lhe que ele precisa do driver e então reinicializar o sistema. No momento da inicialização, o

sistema operacional vai e encontra os drivers que ele precisa e os carrega. O Windows funciona dessa maneira. A terceira maneira é capacitar o sistema operacional a aceitar novos drivers enquanto estiver sendo executado e instalá-los rapidamente sem a necessidade da reinicialização. Essa maneira costumava ser rara, mas está se tornando muito mais comum hoje. Dispositivos do tipo *hot-pluggable* (acoplados a quente), como dispositivos USB e IEEE 1394 (discutidos a seguir), sempre precisam de drivers carregados dinamicamente.

Todo controlador tem um pequeno número de registradores que são usados para comunicar-se com ele. Por exemplo, um controlador de discos mínimo pode ter registradores para especificar o endereço de disco, endereço de memória, contador de setores e direção (leitura ou escrita). Para ativar o controlador, o driver recebe um comando do sistema operacional, então o traduz para os valores apropriados a serem escritos nos registradores dos dispositivos. A reunião de todos esses registradores de dispositivos forma o **espaço de portas de E/S**, um assunto que retomaremos no Capítulo 5.

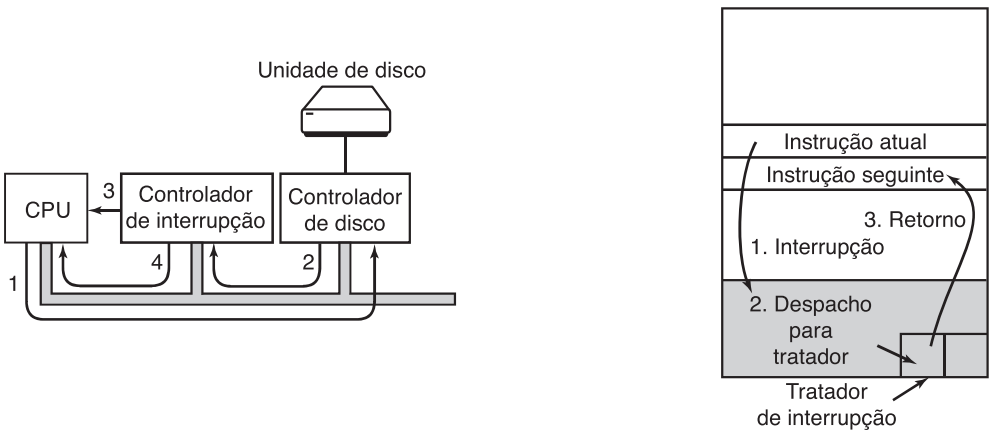
Em alguns computadores, os registradores dos dispositivos estão mapeados no espaço do endereço do sistema operacional (os endereços que ele pode usar), portanto podem ser lidos e escritos como palavras de memória comuns. Nesses computadores, não são necessárias instruções de E/S especiais e os programas de usuários podem ser mantidos distantes do hardware deixando esses endereços de memória fora de seu alcance (por exemplo, pelo uso de registradores-base e limite). Em outros computadores, os registradores dos dispositivos são colocados em um espaço de porta E/S especial, com cada registrador tendo um endereço de porta. Nessas máquinas, instruções especiais IN e OUT estão disponíveis em modo núcleo para permitir que os drivers leiam e escrevam nos registradores. O primeiro esquema elimina a necessidade para instruções de E/S especiais, mas consome parte do espaço do endereço. O segundo esquema não utiliza espaço do endereço, mas exige instruções especiais. Ambos os sistemas são amplamente usados.

A entrada e a saída podem ser realizadas de três maneiras diferentes. No método mais simples, um programa de usuário emite uma chamada de sistema, que o núcleo traduz em uma chamada de rotina para o driver apropriado. O driver então inicia a E/S e aguarda usando um laço curto, inquirindo continuamente o dispositivo para ver se ele terminou a operação (em geral há algum bit que indica que o dispositivo ainda está ocupado). Quando a operação de E/S termina, o driver coloca os dados (se algum) onde eles são necessários e retorna. O sistema operacional então retorna o controle a quem o chamou. Esse método é chamado de **espera ocupada** e tem a desvantagem de manter a CPU ocupada interrogando o dispositivo até o término da operação de E/S.

No segundo método, o driver inicia o dispositivo e pede a ele que o interrompa quando tiver terminado. Nesse ponto, o driver retorna. O sistema operacional bloqueia então o programa que o chamou, se necessário, e procura por mais trabalho para fazer. Quando o controlador detecta o fim da transferência, ele gera uma **interrupção** para sinalizar o término.

Interrupções são muito importantes nos sistemas operacionais, então vamos examinar a ideia mais de perto. Na Figura 1.11(a), vemos um processo de quatro passos para a E/S. No passo 1, o driver diz para o controlador o que fazer escrevendo nos seus registradores de dispositivo. O controlador então inicia o dispositivo. Quando o controlador termina de ler ou escrever o

FIGURA 1.11 (a) Os passos para iniciar um dispositivo de E/S e obter uma interrupção. (b) O processamento de interrupção envolve obter a interrupção, executar o tratador de interrupção e retornar ao programa do usuário.



número de bytes que lhe disseram para transferir, ele sinaliza o chip controlador de interrupção usando determinadas linhas de barramento no passo 2. Se o controlador de interrupção está pronto para aceitar a interrupção (o que ele talvez não esteja, se estiver ocupado lidando com uma interrupção de maior prioridade), ele sinaliza isso à CPU no passo 3. No passo 4, o controlador de interrupção insere o número do dispositivo no barramento de maneira que a CPU possa lê-lo e saber qual dispositivo acabou de terminar (muitos dispositivos podem estar sendo executados ao mesmo tempo).

Uma vez que a CPU tenha decidido aceitar a interrupção, o contador de programa (PC) e a palavra de estado do programa (PSW) normalmente são empilhados na pilha atual e a CPU chaveada para o modo núcleo. O número do dispositivo pode ser usado como um índice para parte da memória para encontrar o endereço do tratador de interrupção (*interrupt handler*) para esse dispositivo. Essa parte da memória é chamada de **vetor de interrupção**. Uma vez que o tratador de interrupção (parte do driver para o dispositivo de interrupção) tenha iniciado, ele remove o contador de programa e PSW empilhados e os salva, e então indaga o dispositivo para saber como está a sua situação. Assim que o tratador de interrupção tenha sido encerrado, ele retorna para o programa do usuário previamente executado para a primeira instrução que ainda não tenha sido executada. Esses passos são mostrados na Figura 1.11 (b).

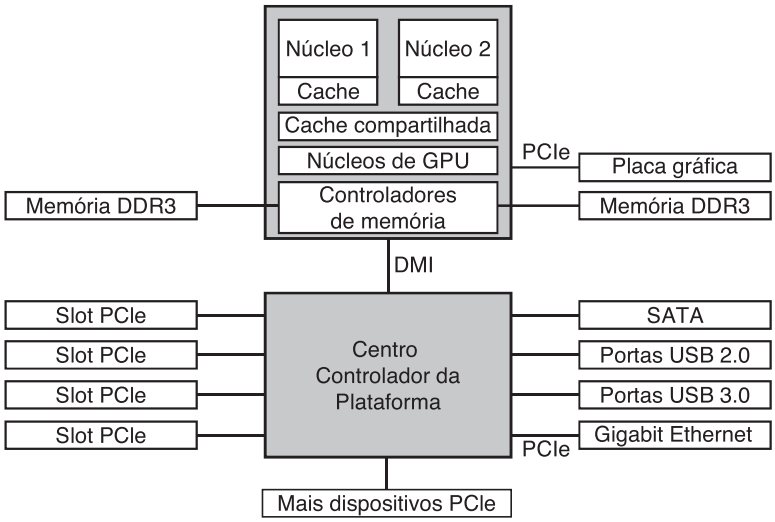
O terceiro método para implementar E/S faz uso de um hardware especial: um chip **DMA (Direct Memory Access** — acesso direto à memória) que pode controlar o fluxo de bits entre a memória e algum controlador sem a intervenção da CPU constante. A CPU configura o chip DMA, dizendo a ele quantos bytes transferir, o dispositivo e endereços de memória envolvidos, e a direção, e então o deixa executar. Quando o chip de DMA tiver finalizado a sua tarefa, ele causa uma interrupção, que é tratada como já descrito. Os hardwares de DMA e E/S em geral serão discutidos mais detalhadamente no Capítulo 5.

Interrupções podem (e muitas vezes isso ocorre) acontecer em momentos altamente inconvenientes, por exemplo, enquanto outro tratador de interrupção estiver em execução. Por essa razão, a CPU tem uma maneira para desabilitar interrupções e então reabilitá-las depois. Enquanto as interrupções estiverem desabilitadas, quaisquer dispositivos que terminem suas atividades continuam a emitir sinais de interrupção, mas a CPU não é interrompida até que as interrupções sejam habilitadas novamente. Se múltiplos dispositivos finalizarem enquanto as interrupções estiverem desabilitadas, o controlador de interrupção decide qual deixar passar primeiro, normalmente baseado em prioridades estáticas designadas para cada dispositivo. O dispositivo de maior prioridade vence e é servido primeiro. Os outros precisam esperar.

1.3.5 Barramentos

A organização da Figura 1.6 foi usada em microcomputadores por anos e também no IBM original. No entanto, à medida que os processadores e as memórias foram ficando mais rápidos, a capacidade de um único barramento (e certamente o barramento do PC IBM) de lidar com todo o tráfego foi exigida até o limite. Algo tinha de ceder. Como resultado, barramentos adicionais foram acrescentados, tanto para dispositivos de E/S mais rápidos quanto para o tráfego CPU para memória. Como consequência dessa evolução, um sistema x86 grande atualmente se parece com algo como a Figura 1.12.

FIGURA 1.12 A estrutura de um sistema x86 grande.



Este sistema tem muitos barramentos (por exemplo, cache, memória, PCIe, PCI, USB, SATA e DMI), cada um com uma taxa de transferência e função diferentes. O sistema operacional precisa ter ciência de todos eles para configuração e gerenciamento. O barramento principal é o **PCIe** (**Peripheral Component Interconnect Express** — interconexão expressa de componentes periféricos).

O PCIe foi inventado pela Intel como um sucessor para o barramento **PCI** mais antigo, que por sua vez foi uma substituição para o barramento **ISA** (**Industry Standard Architecture** — arquitetura padrão industrial). Capaz de transferir dezenas de gigabits por segundo, o PCIe é muito mais rápido que os seus predecessores. Ele também é muito diferente em sua natureza. Uma **arquitetura de barramento compartilhado** significa que múltiplos dispositivos usam os mesmos fios para transferir dados. Assim, quando múltiplos dispositivos têm dados para enviar, você precisa de um árbitro para determinar quem pode utilizar o barramento. Em comparação, o PCIe faz uso de conexões dedicadas de ponto a ponto. Uma **arquitetura de barramento paralela** como usada no PCI tradicional significa que você pode enviar uma palavra de dados através de múltiplos fios. Por exemplo, em barramentos PCI regulares, um único número de 32 bits é enviado através de 32 fios paralelos. Em comparação com isso, o PCIe usa uma **arquitetura de barramento serial** e envia todos os bits em uma mensagem através de uma única conexão, chamada faixa, de maneira muito semelhante a um pacote de rede. Isso é muito mais simples, pois você não tem de assegurar que todos os 32 bits cheguem ao destino exatamente ao mesmo tempo. O paralelismo ainda é usado, pois você pode ter múltiplas faixas em paralelo. Por exemplo, podemos usar 32 faixas para carregar 32 mensagens em paralelo. À medida que a velocidade de dispositivos periféricos como cartões de rede e adaptadores de gráficos aumenta rapidamente, o padrão PCIe é atualizado a cada 3-5 anos. Por exemplo, 16 faixas de PCIe 2.0 oferecem 64 gigabits por segundo. Atualizar para PCIe 3.0 dará a você duas vezes aquela velocidade e o PCIe 4.0 dobrará isso novamente.

Enquanto isso, ainda temos muitos dispositivos de legado do padrão PCI mais antigo. Como vemos na Figura 1.12, esses dispositivos estão ligados a um centro processador em separado. No futuro, quando virmos o PCI não mais como meramente *velho*, mas *ancestral*, é possível que todos os dispositivos PCI vão se ligar a mais um centro ainda que, por sua vez, vai conectá-los ao centro principal, criando uma árvore de barramentos.

Nessa configuração, a CPU se comunica com a memória por meio de um barramento DDR3 rápido, com um dispositivo gráfico externo através do PCIe e com todos os outros dispositivos via um centro controlador usando um barramento **DMI** (**Direct Media Interface** — interface de mídia direta). O centro por sua vez conecta-se com todos os outros dispositivos, usando o Barramento Serial Universal para conversar com os dispositivos USB, o barramento SATA para interagir com discos rígidos e acionadores de DVD, e o PCIe para transferir quadros (frames) Ethernet. Já mencionamos os dispositivos PCI que usam um barramento PCI tradicional.

Além disso, cada um dos núcleos tem uma cache dedicada e uma muito maior que é compartilhada entre eles. Cada uma dessas caches introduz outro barramento.

O **USB** (**Universal Serial Bus** — barramento serial universal) foi inventado para conectar todos os dispositivos de E/S lentos, como o teclado e o mouse, ao computador. No entanto, chamar um dispositivo USB 3.0 zunindo a 5 Gbps de "lento" pode não soar natural para a geração que cresceu com o ISA de 8 Mbps como o barramento principal nos primeiros PCs da IBM. O USB usa um pequeno conector com quatro a onze fios (dependendo da versão), alguns dos quais fornecem energia elétrica para os dispositivos USB ou conectam-se com o terra. O USB é um barramento centralizado no qual um dispositivo-raiz interroga todos os dispositivos de E/S a cada 1 ms para ver se eles têm algum tráfego. O USB 1.0 pode lidar com uma carga agregada de 12 Mbps, o USB 2.0 aumentou a velocidade para 480 Mbps e o USB 3.0 chega a não menos que 5 Gbps. Qualquer dispositivo USB pode ser conectado a um computador e ele funcionará imediatamente, sem exigir uma reinicialização, algo que os dispositivos pré-USB exigiam para a consternação de uma geração de usuários frustrados.

O barramento **SCSI** (**Small Computer System Interface** — interface pequena de sistema computacional) é um barramento de alto desempenho voltado para discos rápidos, digitalizadores de imagens e outros dispositivos que precisam de uma considerável largura de banda. Hoje em dia, eles são encontrados na maior parte das vezes em servidores e estações de trabalho, e podem operar a até 640 MB/s.

Para trabalhar em um ambiente como o da Figura 1.12, o sistema operacional tem de saber quais dispositivos periféricos estão conectados ao computador e configurá-los. Essa exigência levou a Intel e a Microsoft a projetar um sistema para o PC chamado de **plug and play**, baseado em um conceito similar primeiro

implementado no Apple Macintosh. Antes do plug and play, cada placa de E/S tinha um nível fixo de requisição de interrupção e endereços específicos para seus registradores de E/S. Por exemplo, o teclado era interrupção 1 e usava endereços 0x60 a 0x64, o controlador de disco flexível era a interrupção 6 e usava endereços de E/S 0x3F0 a 0x3F7, e a impressora era a interrupção 7 e usava os endereços de E/S 0x378 a 0x37A, e assim por diante.

Até aqui, tudo bem. O problema começava quando o usuário trazia uma placa de som e uma placa de modem e ocorria de ambas usarem, digamos, a interrupção 4. Elas entravam em conflito e não funcionavam juntas. A solução era incluir chaves DIP ou jumpers em todas as placas de E/S e instruir o usuário a ter o cuidado de configurá-las para selecionar o nível de interrupção e endereços dos dispositivos de E/S que não entrassem em conflito com quaisquer outros no sistema do usuário. Adolescentes que devotaram a vida às complexidades do hardware do PC podiam fazê-lo às vezes sem cometer erros. Infelizmente, ninguém mais conseguia, levando ao caos.

O plug and play faz o sistema coletar automaticamente informações sobre os dispositivos de E/S, atribuir centralmente níveis de interrupção e endereços desses dispositivos e, então, informar a cada placa quais são os seus números. Esse trabalho está relacionado de perto à inicialização do computador, então vamos examinar essa questão. Ela não é completamente trivial.

1.3.6 Inicializando o computador

De modo bem resumido, o processo de inicialização funciona da seguinte maneira: todo PC contém uma parentboard (placa-pais) (que era chamada de placa-mãe antes da onda politicamente correta atingir a indústria de computadores). Na placa-pais há um programa chamado de sistema **BIOS** (**Basic Input Output System** — sistema básico de entrada e saída). O BIOS conta com rotinas de E/S de baixo nível, incluindo procedimentos para ler o teclado, escrever na tela e realizar a E/S no disco, entre outras coisas. Hoje, ele fica em um flash RAM, que é não volátil, mas que pode ser atualizado pelo sistema operacional quando erros são encontrados no BIOS.

Quando o computador é inicializado, o BIOS começa a executar. Primeiro ele confere para ver quanta RAM está instalada e se o teclado e os outros dispositivos básicos estão instalados e respondendo corretamente. Ele segue varrendo os barramentos PCIe e PCI para detectar todos os dispositivos ligados a ele. Se os dispositivos presentes forem diferentes de quando o sistema foi inicializado pela última vez, os novos dispositivos são configurados.

O BIOS então determina o dispositivo de inicialização tentando uma lista de dispositivos armazenados na memória CMOS. O usuário pode mudar essa lista entrando em um programa de configuração do BIOS logo após a inicialização. Tipicamente, é feita uma tentativa para inicializar a partir de uma unidade de CD-ROM (ou às vezes USB), se houver uma. Se isso não der certo, o sistema inicializa a partir do disco rígido. O primeiro setor do dispositivo de inicialização é lido na memória e executado. Ele contém um programa que normalmente examina a tabela de partições no final do setor de inicialização para determinar qual partição está ativa. Então um carregador de inicialização secundário é lido daquela partição. Esse carregador lê o sistema operacional da partição ativa e, então, o inicia.

O sistema operacional consulta então o BIOS para conseguir as informações de configuração. Para cada dispositivo, ele confere para ver se possui o driver do dispositivo. Se não possuir, pede para o usuário inserir um CD-ROM contendo o driver (fornecido pelo fabricante do dispositivo) ou para baixá-lo da internet. Assim que todos os drivers dos dispositivos estiverem disponíveis, o sistema operacional os carrega no núcleo. Então ele inicializa suas tabelas, cria os processos de segundo plano necessários e inicia um programa de identificação (*login*) ou uma interface gráfica GUI.

1.4 O zoológico dos sistemas operacionais

Os sistemas operacionais existem há mais de meio século. Durante esse tempo, uma variedade bastante significativa deles foi desenvolvida, nem todos bastante conhecidos. Nesta seção abordaremos brevemente nove deles. Voltaremos a alguns desses tipos diferentes de sistemas mais tarde no livro.

1.4.1 Sistemas operacionais de computadores de grande porte

No topo estão os sistemas operacionais para computadores de grande porte (*mainframes*), aquelas máquinas do tamanho de uma sala ainda encontradas nos centros de processamento de dados de grandes corporações. Esses computadores diferem dos computadores pessoais em termos de sua capacidade de E/S.