Introdução

Um sistema operacional é um programa que gerencia o hardware de um computador. Ele também fornece uma base para os programas aplicativos e atua como intermediário entre o usuário e o hardware do computador. Um aspecto interessante dos sistemas operacionais é o quanto eles assumem diferentes abordagens ao cumprir essas tarefas. Os sistemas operacionais de mainframe são projetados basicamente para otimizar a utilização do hardware. Os sistemas operacionais dos computadores pessoais (PCs) suportam jogos complexos, aplicações comerciais e tudo o mais entre estes. Os sistemas operacionais de computadores móveis fornecem um ambiente no qual o usuário pode interagir facilmente com o computador para executar programas. Assim, alguns sistemas operacionais são projetados para serem ***convenientes***, outros para serem ***eficientes***, e outros para atenderem a alguma combinação de ambos os aspectos.

Antes de podermos explorar os detalhes de operação do sistema de computação, precisamos saber algo sobre a estrutura do sistema. Assim, discutimos as funções básicas de inicialização do sistema, o I/O e o armazenamento no início deste capítulo. Também descrevemos a arquitetura básica do computador que torna possível criar um sistema operacional funcional.

Já que um sistema operacional é grande e complexo, ele deve ser criado módulo a módulo. Cada um desses módulos deve ser uma parte bem delineada do sistema, com entradas, saídas e funções bem definidas. Neste capítulo, fornecemos uma visão geral dos principais componentes de um sistema de computação contemporâneo e das funções oferecidas pelo sistema operacional. Além disso, abordamos vários outros tópicos para ajudar a preparar o terreno para o restante do texto: estruturas de dados usadas em sistemas operacionais, ambientes de computação e sistemas operacionais de fonte aberta.

OBJETIVOS DO CAPÍTULO

- Descrever a organização básica dos sistemas de computação.
- Oferecer um giro completo pelos principais componentes dos sistemas operacionais.
- Fornecer uma visão geral dos vários tipos de ambientes de computação.
- Examinar diversos sistemas operacionais de fonte aberta.

1.1 O que Fazem os Sistemas Operacionais

Iniciamos nossa discussão examinando o papel do sistema operacional no sistema de computação como um todo. Um sistema de computação pode ser grosseiramente dividido em quatro componentes: o hardware, o sistema operacional, os programas aplicativos e os usuários (Figura 1.1).

O **hardware** — a **unidade central de processamento** (**CPU** — ***central processing unit***), a **memória** e os **dispositivos de entrada/saída** (**I/O** — ***input/output***) — fornece os recursos básicos de computação do sistema. Os **programas aplicativos** — como processadores de texto, planilhas, compiladores e navegadores da web — definem as formas pelas quais esses recursos são utilizados para resolver os problemas computacionais dos usuários. O sistema operacional controla o hardware e coordena seu uso pelos diversos programas aplicativos de vários usuários.

Também podemos considerar um sistema de computação como composto de hardware, software e dados. O sistema operacional fornece os meios para a utilização apropriada desses recursos durante a operação do sistema de computação. Um sistema operacional é semelhante ao governo. Tal como o governo, ele não desempenha funções

úteis para si mesmo. Ele simplesmente proporciona um ambiente no qual outros programas possam desempenhar tarefas úteis.

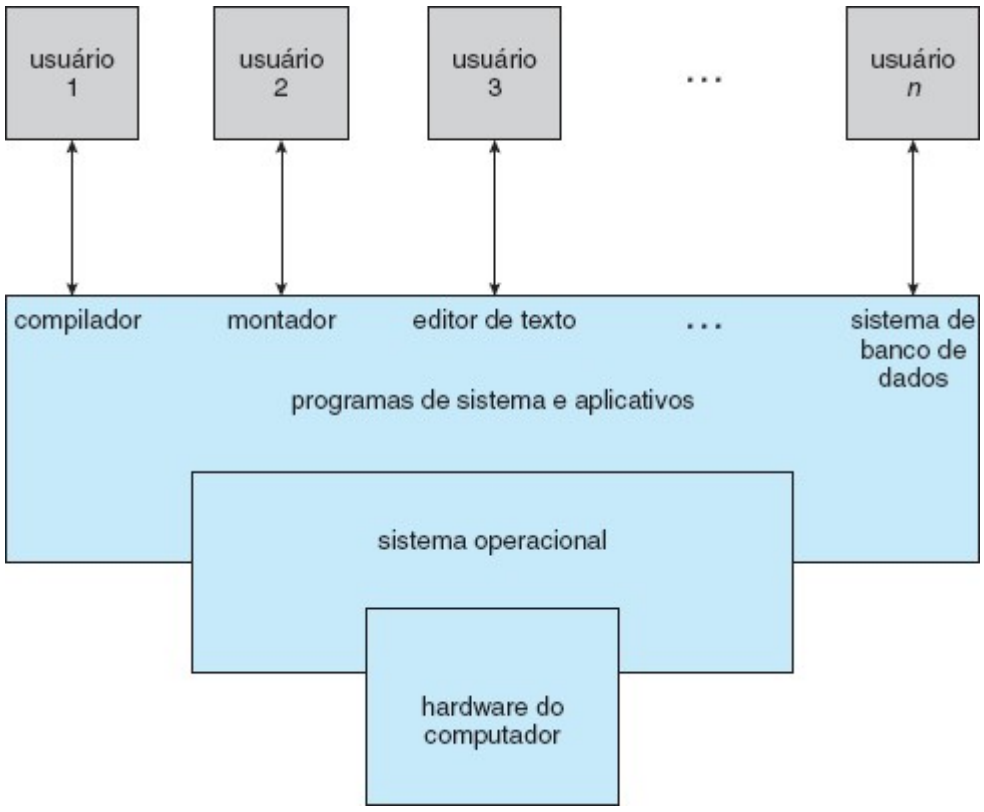

Figura 1.1 Visão abstrata dos componentes de um sistema de computação.

Para entender melhor o papel do sistema operacional, examinamos a seguir os sistemas operacionais a partir de dois pontos de vista: o do usuário e o do sistema.

1.1.1O Ponto de Vista do Usuário

A perspectiva do usuário em relação ao computador varia, dependendo da interface que estiver sendo utilizada. A maioria dos usuários de computador senta-se diante de um PC, constituído de um monitor, um teclado, um mouse e a unidade do sistema. Tal sistema é projetado para que um único usuário monopolize seus recursos. O objetivo é maximizar o trabalho (ou jogo) que o usuário esteja executando. Nesse caso, o sistema operacional é projetado principalmente para **facilidade de uso**, com alguma atenção dada ao desempenho e nenhuma à **utilização dos recursos** — a forma como vários recursos de hardware e software são compartilhados. É claro que o desempenho é importante para o usuário, mas esses sistemas são otimizados para a experiência de um único usuário e não para atender a vários usuários.

Em outros casos, o usuário senta-se diante de um terminal conectado a um **mainframe** ou a um **minicomputador**. Outros usuários acessam o mesmo computador por intermédio de outros terminais. Esses usuários compartilham recursos e podem trocar informações. Em tais casos, o sistema operacional é projetado para maximizar a utilização dos recursos — assegurando que todo o tempo de CPU, memória e I/O disponíveis, seja utilizado eficientemente e que nenhum usuário individual ocupe mais do que sua cota.

Há ainda outros casos em que os usuários ocupam **estações de trabalho** conectadas a redes de outras estações de trabalho e **servidores**. Esses usuários possuem recursos dedicados à sua disposição, mas também compartilham recursos, tais como a rede e os servidores, incluindo servidores de arquivo, de computação e de impressão. Portanto, seu sistema operacional é projetado para estabelecer um compromisso entre usabilidade individual e utilização dos recursos.

Ultimamente, estão na moda muitas variedades de computadores móveis, como smartphones e tablets. A maioria dos computadores móveis é de unidades autônomas para usuários individuais. Quase sempre, eles são conectados a redes por celulares ou outras tecnologias sem fio. Cada vez mais, esses dispositivos móveis estão substituindo os computadores desktop e laptop para pessoas que estão interessadas, principalmente, em usar computadores para enviar e-mails e navegar na web. Geralmente, a interface de usuário dos computadores móveis tem uma **tela sensível ao toque**, em que o usuário interage com o sistema pressionando e batendo com os dedos na tela em vez de usar um teclado ou mouse físico.

Alguns computadores são pouco ou nada visíveis ao usuário. Por exemplo, computadores embutidos em dispositivos domésticos e em automóveis podem ter um mostrador numérico e podem acender ou apagar luzes indicativas para mostrar um estado, mas basicamente eles e seus sistemas operacionais são projetados para operar sem intervenção do usuário.

1.1.2 O Ponto de Vista do Sistema

Do ponto de vista do computador, o sistema operacional é o programa mais intimamente envolvido com o hardware. Nesse contexto, podemos considerar um sistema operacional como um **alocador de recursos**. Um sistema de computação tem muitos recursos que podem ser necessários à resolução de um problema: tempo de CPU, espaço de memória, espaço de armazenamento em arquivo, dispositivos de I/O etc. O sistema operacional atua como o gerenciador desses recursos. Ao lidar com solicitações de recursos numerosas e possivelmente conflitantes, o sistema operacional precisa decidir como alocá-los a programas e usuários específicos de modo a poder operar o sistema de computação de maneira eficiente e justa. Como vimos, a alocação de recursos é particularmente importante quando muitos usuários acessam o mesmo mainframe ou minicomputador.

Uma visão ligeiramente diferente de um sistema operacional enfatiza a necessidade de controle dos diversos dispositivos de I/O e programas de usuário. Um sistema operacional é um programa de controle. Um **programa de controle** gerencia a execução de programas de usuário para evitar erros e o uso impróprio do computador. Ele se preocupa principalmente com a operação e o controle de dispositivos de I/O.

1.1.3 Definindo Sistemas Operacionais

A essa altura, você já deve ter notado que o termo ***sistema operacional*** abrange muitos papéis e funções. Isso ocorre, pelo menos em parte, em razão dos inúmeros designs e usos dos computadores. Os computadores estão presentes dentro de torradeiras, carros, navios, espaçonaves, casas e empresas. Eles são a base das máquinas de jogos, reprodutores de música, sintonizadores de TV a cabo e sistemas de controle industrial. Embora os computadores tenham uma história relativamente curta, eles evoluíram rapidamente. A computação começou como um experimento para determinar o que poderia ser feito e migrou rapidamente para sistemas de finalidade específica de uso militar, como decifração de códigos e plotagem de trajetórias, e uso governamental, como no cálculo do censo. Esses computadores iniciais evoluíram para mainframes multifuncionais de uso geral, e foi então que nasceram os sistemas operacionais. Nos anos 1960, a Lei de Moore previu que o número de transistores de um circuito integrado dobraria a cada dezoito meses, e essa previsão se confirmou. Os computadores aumentaram em funcionalidade e diminuíram em tamanho, levando a um vasto número de usos e a um vasto número e variedade de sistemas operacionais. (Consulte o Capítulo 20 para conhecer mais detalhes sobre a história dos sistemas operacionais.)

Como, então, podemos definir o que é um sistema operacional? Em geral, não possuímos uma definição totalmente adequada de um sistema operacional. Sistemas operacionais existem porque oferecem uma forma razoável de resolver o problema de criar um sistema de computação utilizável. O objetivo fundamental dos sistemas de computação é executar os programas dos usuários e facilitar a resolução dos seus problemas. É com esse objetivo que o hardware do computador é construído. Os programas aplicativos são desenvolvidos, tendo em vista que o hardware puro, sozinho, não é particularmente fácil de ser utilizado. Esses programas requerem determinadas operações comuns, como as que controlam os dispositivos de I/O. As funções comuns de controle e alocação de recursos são, então, reunidas em um único software: o sistema operacional.

Além disso, não temos uma definição universalmente aceita sobre o que compõe o sistema operacional. Um ponto de vista simplista é o de que esse sistema inclui tudo que um fornecedor oferece quando você encomenda "o sistema operacional". Entretanto, os recursos incluídos variam muito entre os sistemas. Alguns sistemas ocupam menos do que um megabyte de espaço e não têm nem mesmo um editor de tela inteira, enquanto outros requerem gigabytes de espaço e são inteiramente baseados em sistemas de janelas gráficas. Uma definição mais comum, que é a que costumamos seguir, é que o sistema operacional é o único programa que permanece em execução no computador durante todo o tempo — chamado, em geral, de **kernel**. (Além do kernel, há dois outros tipos de programas: os **programas do sistema** que estão associados ao sistema operacional, mas não necessariamente fazem parte do kernel, e os **programas aplicativos** que incluem todos os programas não associados à operação do sistema.)

O problema do que constitui um sistema operacional foi se tornando cada vez mais importante, à medida que os computadores pessoais se popularizaram e os sistemas operacionais foram ficando mais sofisticados. Em 1998, o Departamento de Justiça dos Estados Unidos fez uma representação contra a Microsoft, alegando em essência que ela incluiu funcionalidades demais em seus sistemas operacionais impedindo a competição por parte dos vendedores de aplicativos. (Por exemplo, um navegador da web era parte integrante dos sistemas operacionais.) Como resultado, a Microsoft foi declarada culpada de usar seu monopólio na área de sistemas operacionais para limitar a competição.

Atualmente, no entanto, se olharmos os sistemas operacionais de dispositivos móveis, vamos ver que, mais uma vez, o número de recursos que compõem o sistema operacional está aumentando. Com frequência, os sistemas operacionais de dispositivos móveis incluem não só um kernel básico, mas também **middleware** — um conjunto de estruturas de

software que fornece serviços adicionais para desenvolvedores de aplicativos. Por exemplo, os dois sistemas operacionais de dispositivos móveis predominantes — o iOS da Apple e o Android da Google — têm um kernel básico e o middleware que suporta bancos de dados, ambientes multimídia e elementos gráficos (para citar apenas alguns recursos).

1.2 Organização do Sistema de Computação

Antes que possamos explorar os detalhes de como os sistemas de computação funcionam, precisamos de um conhecimento geral da estrutura de um sistema de computação. Nesta seção, examinamos várias partes dessa estrutura. A seção é principalmente dedicada à organização do sistema de computação, portanto, você pode olhá-la superficialmente, ou saltá-la se já conhece os conceitos.

1.2.1 Operação do Sistema de Computação

Um moderno sistema de computação de uso geral é composto por uma ou mais CPUs e vários controladores de dispositivos conectados por intermédio de um bus comum que dá acesso à memória compartilhada (Figura 1.2). Cada controlador de dispositivos é responsável por um tipo específico de dispositivo (por exemplo, drives de disco, dispositivos de áudio ou exibidores de vídeo). A CPU e os controladores de dispositivos podem operar concorrentemente, competindo por ciclos de memória. Um controlador de memória sincroniza o acesso à memória compartilhada para assegurar um acesso ordenado.

Para que um computador comece a operar — por exemplo, quando é ligado ou reiniciado — ele precisa ter um programa inicial para executar. Esse programa inicial, ou **programa bootstrap**, tende a ser simples. Normalmente, ele é armazenado dentro do hardware do computador em memória somente de leitura (**ROM**) ou em memória somente de leitura eletricamente apagável e programável (**EEPROM — *electrically erasable programmable read-only memory***), conhecida pelo termo geral **firmware**. Ele inicializa todos os aspectos do sistema, dos registradores da CPU aos controladores de dispositivos e conteúdos da memória. O programa bootstrap precisa saber como carregar o sistema operacional e iniciar sua execução. Para alcançar esse objetivo, o programa tem que localizar e carregar na memória o kernel do sistema operacional.

Assim que o kernel é carregado e está em execução, ele pode começar a fornecer serviços para o sistema e seus usuários. Alguns serviços são fornecidos fora do kernel por programas do sistema que são carregados na memória em tempo de inicialização para se tornarem **processos do sistema**, ou **daemons do sistema**, que são executados durante todo o tempo em que o kernel é executado. No UNIX, o primeiro processo do sistema é "init", e ele inicia muitos outros daemons. Quando essa fase é concluída, o sistema está totalmente inicializado e aguarda que algum evento ocorra.

Geralmente, a ocorrência de um evento é indicada por uma **interrupção** proveniente do hardware ou do software. O hardware pode disparar uma interrupção a qualquer momento enviando um sinal à CPU, normalmente pelo bus do sistema. O software pode disparar uma interrupção executando uma operação especial denominada **chamada de sistema** (também conhecida como **chamada de monitor**).

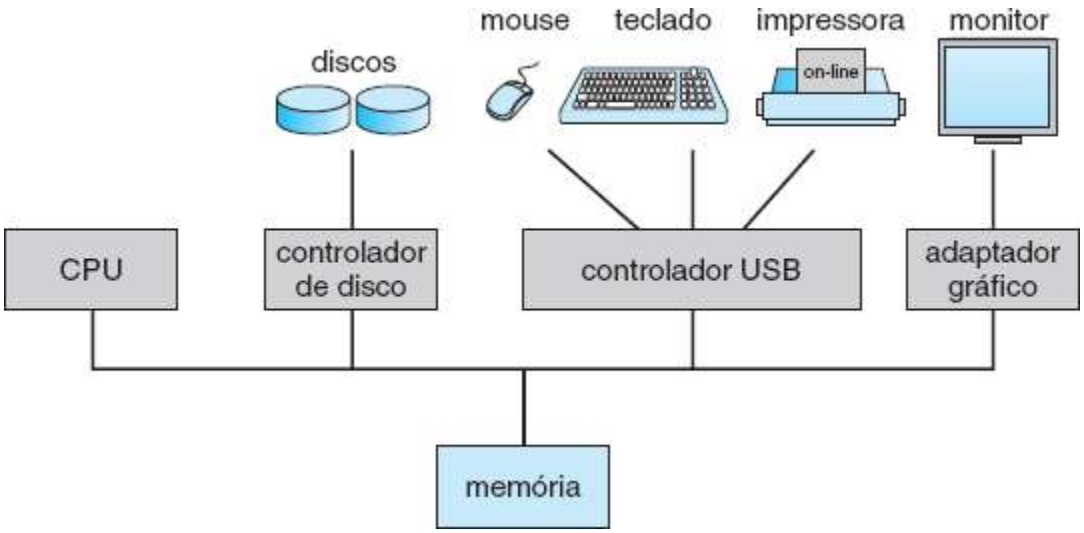

Figura 1.3 Linha de tempo de interrupções para um processo individual gerando saídas.

Quando a CPU é interrompida, ela para o que está fazendo e transfere imediatamente a execução para uma localização fixa. Normalmente, essa localização fixa contém o endereço inicial no qual se encontra a rotina de serviço

da interrupção. A rotina de serviço da interrupção entra em operação; ao completar a execução, a CPU retoma a computação interrompida. Uma linha de tempo dessa operação é mostrada na Figura 1.3.

As interrupções são uma parte importante da arquitetura do computador. Cada projeto de computador tem seu próprio mecanismo de interrupção, mas diversas funções são comuns a todos. A interrupção deve transferir o controle para a rotina de serviço de interrupção apropriada. O método mais simples para a manipulação dessa transferência seria invocar uma rotina genérica que examinasse a informação de interrupção. A rotina, por sua vez, chamaria o manipulador de interrupções específico. Entretanto, as interrupções precisam ser manipuladas rapidamente. Já que só é permitida uma quantidade predeterminada de interrupções, uma tabela de ponteiros para rotinas de interrupção pode ser usada como alternativa para fornecer a velocidade necessária. A rotina de interrupção é chamada indiretamente pela tabela, sem necessidade de rotina intermediária. Geralmente, a tabela de ponteiros é armazenada na memória baixa (mais ou menos as 100 primeiras localizações). Essas localizações mantêm os endereços das rotinas de serviço de interrupção dos diversos dispositivos. Esse array de endereços, ou **vetor de interrupções**, é então indexado por um único número de dispositivo, fornecido com a solicitação de interrupção, de modo a que seja obtido o endereço da rotina de serviço de interrupção do dispositivo que a está causando. Sistemas operacionais tão diferentes como o Windows e o UNIX despacham as interrupções dessa maneira.

A arquitetura de interrupções também deve salvar o endereço da instrução interrompida. Muitos projetos antigos simplesmente armazenavam o endereço interrompido em uma localização fixa ou em uma localização indexada pelo número do dispositivo. Arquiteturas mais recentes armazenam o endereço de retorno na pilha do sistema. Se a rotina de interrupção precisar modificar o estado do processador — por exemplo, modificando os valores dos registradores — ela deve salvar explicitamente o estado corrente e, então, restaurar esse estado antes de retornar. Após a interrupção ser atendida, o endereço de retorno salvo é carregado no contador do programa, e a computação interrompida é retomada como se a interrupção não tivesse ocorrido.

1.2.2 Estrutura de Armazenamento

A CPU só pode carregar instruções a partir da memória; portanto, para serem executados, os programas devem estar armazenados na memória. Computadores de uso geral executam a maioria de seus programas a partir de memória regravável, ou seja, a memória principal (também chamada de **memória de acesso randômico** ou **RAM**). Normalmente, a memória principal é implementada em uma tecnologia de semicondutor denominada **memória de acesso randômico dinâmica** (**DRAM** — ***dynamic random-access memory***).

DEFINIÇÕES E NOTAÇÃO DE ARMAZENAMENTO

A unidade básica de armazenamento nos computadores é o **bit**. Um bit pode conter um entre dois valores, 0 e 1. Todos os outros tipos de armazenamento em um computador são baseados em conjuntos de bits. Quando há bits suficientes, é espantoso quantas coisas um computador pode representar: números, letras, imagens, filmes, sons, documentos e programas, para citar apenas algumas. Um **byte** é composto por 8 bits e, na maioria dos computadores, é o menor bloco de armazenamento conveniente. Por exemplo, a maioria dos computadores não tem uma instrução para mover um bit e sim para mover um byte. Um termo menos comum é **palavra** que é uma unidade de dados nativa de determinada arquitetura de computador. Uma palavra é composta por um ou mais bytes. Por exemplo, um computador que tem registradores de 64 bits e endereçamento de memória de 64 bits, tem, normalmente, palavras de 64 bits (8 bytes). Um computador executa muitas operações em seu tamanho de palavra nativo, em vez de usar um byte de cada vez.

O espaço de armazenamento do computador, junto com a maior parte do seu throughput, é geralmente medido e manipulado em bytes e conjuntos de bytes. Um **quilobyte** ou **KB** é igual a 1.024 bytes; um **megabyte** ou **MB** equivale a 1.024^2 bytes; um **gigabyte** ou **GB** é o mesmo que 1.024^3 bytes; um **terabyte** ou **TB** corresponde a 1.024^4 bytes; e um **pentabyte** ou **PB** é composto por 1.024^5 bytes. Os fabricantes de computador costumam arredondar esses números e dizem que um megabyte corresponde a 1 milhão de bytes, e um gigabyte a 1 bilhão de bytes. As medidas aplicadas às redes são uma exceção a essa regra geral; elas são fornecidas em bits (porque as redes movem os dados bit a bit).

Os computadores também usam outros tipos de memória. Já mencionamos a memória somente de leitura (ROM) e a memória somente de leitura eletricamente apagável e programável (EEPROM). Como a ROM não pode ser alterada, somente programas estáticos, como o programa bootstrap descrito anteriormente, são nela armazenados. A imutabilidade da ROM é útil em cartuchos de jogos. A EEPROM pode ser alterada, mas não com frequência e, portanto, contém programas estáticos em sua maior parte. Por exemplo, os smartphones têm EEPROM para armazenar seus programas instalados de fábrica.

Todos os tipos de memória fornecem um array de bytes. Cada byte tem seu próprio endereço. A interação é alcançada por intermédio de uma sequência de instruções `load` ou `store` para endereços de memória específicos. A instrução `load` move um byte ou palavra da memória principal para um registrador interno da CPU enquanto a instrução `store` move o conteúdo de um registrador para a memória principal. Além das cargas e armazenamentos explícitos, a CPU carrega automaticamente para execução instruções a partir da memória principal.

Um típico ciclo instrução-execução, conforme realizado em um sistema com **arquitetura von Neumann**, traz primeiro uma instrução da memória e a armazena no **registrador de instruções**. A instrução é, então, decodificada e pode provocar a busca de operandos na memória e o seu armazenamento em algum registrador interno. Após a execução da instrução sobre os operandos, o resultado pode ser armazenado novamente na memória. Observe que a unidade de memória enxerga apenas um fluxo de endereços de memória. Ela não sabe como eles são gerados (pelo contador de instruções, por indexação, indiretamente, como endereços literais ou por algum outro meio) ou para que servem (instruções ou dados). Da mesma forma, podemos ignorar *como* um endereço de memória é gerado por um programa. Só estamos interessados na sequência de endereços de memória gerados pelo programa em execução.

Idealmente, gostaríamos que os programas e dados residissem na memória principal de modo permanente. Esse esquema, geralmente, não é possível pelas duas razões a seguir:

1. A memória principal costuma ser muito pequena para armazenar permanentemente todos os programas e dados necessários.
2. A memória principal é um dispositivo de armazenamento **volátil** que perde seus conteúdos quando a energia é desligada ou quando ela falta por outro motivo.

Portanto, a maioria dos sistemas de computação fornece **memória secundária** como uma extensão da memória principal. O principal requisito da memória secundária é que ela seja capaz de armazenar grandes quantidades de dados permanentemente.

O dispositivo de memória secundária mais comum é o **disco magnético**, que fornece armazenamento tanto para programas quanto para dados. A maioria dos programas (de sistema e de aplicação) é armazenada em um disco até que seja carregada na memória. Por isso, muitos programas utilizam o disco tanto como fonte quanto como destino de seu processamento. Logo, o gerenciamento apropriado do armazenamento em disco é de importância capital para um sistema de computação, como discutimos no Capítulo 10.

De modo geral, no entanto, a estrutura de armazenamento que descrevemos — constituída de registradores, memória principal e discos magnéticos — é apenas um dos muitos sistemas de armazenamento possíveis. Há ainda a memória cache, o CD-ROM, as fitas magnéticas etc. Cada sistema de armazenamento fornece as funções básicas de armazenamento e de manutenção de dados até que sejam recuperados mais tarde. As principais diferenças entre os vários sistemas de armazenamento residem na velocidade, no custo, no tamanho e na volatilidade.

A ampla variedade de sistemas de armazenamento pode ser organizada em uma hierarquia (Figura 1.4), de acordo com a velocidade e o custo. Os níveis mais altos são caros, porém velozes. À medida que descemos na hierarquia, o custo por bit geralmente decresce, enquanto o tempo de acesso em geral aumenta. Essa desvantagem é razoável; se determinado sistema de armazenamento fosse ao mesmo tempo mais rápido e menos caro que outro — sendo as demais propriedades idênticas —, então não haveria razão para utilizar a memória mais lenta e mais dispendiosa. Na verdade, muitos dispositivos de armazenamento antigos, inclusive fita de papel e memórias de núcleo, foram relegados aos museus depois que a fita magnética e a **memória semicondutora** tornaram-se mais rápidas e mais baratas. Os quatro níveis mais altos de memória na Figura 1.4 podem ser construídos com o uso de memória semicondutora.

Além de diferirem na velocidade e no custo, os diversos sistemas de armazenamento podem ser voláteis ou não voláteis. Como mencionado anteriormente, a **memória volátil** perde seus conteúdos quando a energia para o dispositivo é removida. Na ausência de dispendiosos sistemas de backup por bateria ou gerador, os dados devem ser gravados em **memória não volátil**, por segurança. Na hierarquia mostrada na Figura 1.4, os sistemas de armazenamento situados acima do disco de estado sólido são voláteis enquanto os que o incluem e estão abaixo dele são não voláteis.

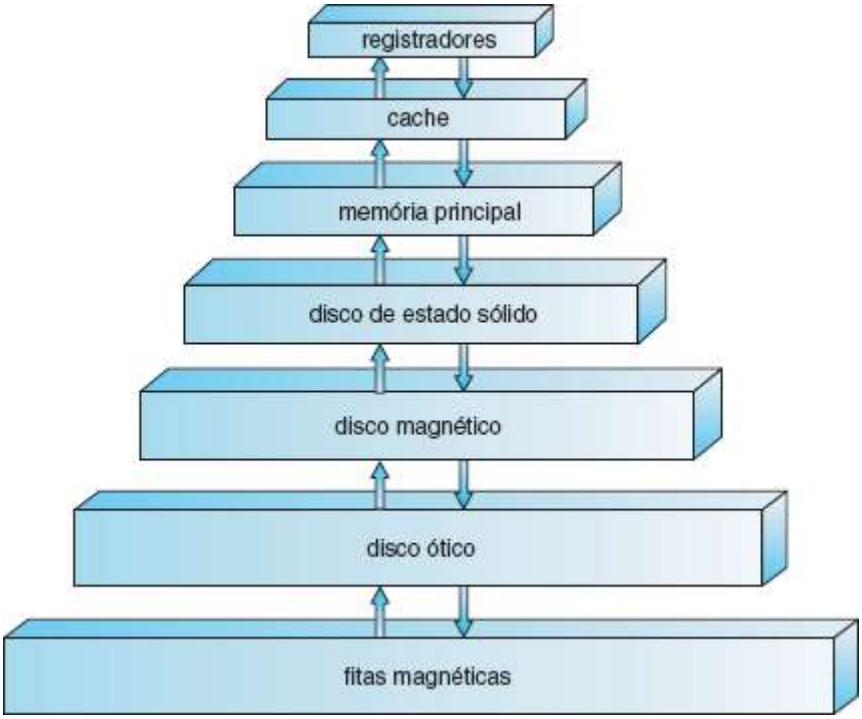

Figura 1.4 Hierarquia dos dispositivos de armazenamento.

Há muitos tipos de **discos de estado sólido**, mas em geral eles são mais rápidos do que os discos magnéticos e são não voláteis. Um tipo de disco de estado sólido armazena dados em um extenso array DRAM durante a operação normal, mas também contém um disco rígido magnético oculto e uma bateria como energia de backup. Quando há interrupção da energia externa, o controlador do disco de estado sólido copia os dados da RAM no disco magnético. Quando a energia externa é restaurada, o controlador copia os dados novamente na RAM. Outro tipo de disco de estado sólido é a memória flash que é popular em câmeras e **assistentes digitais pessoais** (**PDAs — *personal digital assistants***), em robôs e cada vez mais para armazenamento em computadores de uso geral. A memória flash é mais lenta que a DRAM, mas não precisa de energia para reter seus conteúdos. Mais um tipo de armazenamento não volátil é a **NVRAM**, que é a DRAM com energia de backup por bateria. Essa memória pode ser tão rápida quanto a DRAM e (enquanto a bateria durar) é não volátil.

O projeto de um sistema de memória completo deve balancear todos os fatores que acabamos de discutir: só deve usar memória cara quando necessário e fornecer o máximo possível de memória barata e não volátil. Caches podem ser instalados para melhorar o desempenho quando existir grande disparidade no tempo de acesso ou na taxa de transferência entre dois componentes.

1.2.3 Estrutura de I/O

O armazenamento é apenas um dos muitos tipos de dispositivos de I/O de um computador. Grande parte do código do sistema operacional é dedicada ao gerenciamento de I/O, tanto por causa de sua importância para a confiabilidade e o desempenho de um sistema quanto em razão da natureza variada dos dispositivos. A seguir, fornecemos uma visão geral do I/O.

Um sistema de computação de uso geral é composto por CPUs e vários controladores de dispositivos conectados por um bus comum. Cada controlador de dispositivos é responsável por um tipo específico de dispositivo. Dependendo do controlador, pode haver mais de um dispositivo conectado. Por exemplo, sete ou mais dispositivos podem ser conectados ao controlador da **interface de pequenos sistemas de computação** (**SCSI — *small computer-systems interface***). Um controlador de dispositivos mantém algum armazenamento em buffer local e um conjunto de registradores de uso específico. O controlador de dispositivos é responsável por movimentar os dados entre os dispositivos periféricos que controla e seu buffer de armazenamento local. Normalmente, os sistemas operacionais têm um **driver de dispositivo** para cada controlador de dispositivos. Esse driver entende o controlador de dispositivos e fornece uma interface uniforme entre o dispositivo e o resto do sistema operacional.

Para iniciar uma operação de I/O, o driver do dispositivo carrega os registradores apropriados dentro do controlador do dispositivo. Este, por sua vez, examina o conteúdo dos registradores para determinar que ação tomar (tal como "ler um caractere a partir do teclado"). O controlador inicia a transferência de dados do dispositivo para o seu buffer local. Quando a transferência de dados é concluída, o controlador do dispositivo informa ao driver do dispositivo, por uma interrupção, que terminou sua operação. O driver do dispositivo retorna então o controle para o sistema operacional, possivelmente retornando os dados ou um ponteiro para os dados se a operação foi uma leitura. Para outras operações, o driver do dispositivo retorna informações de *status*.

Esse tipo de I/O dirigido por interrupção é adequado para a movimentação de pequenas quantidades de dados, mas pode produzir um overhead alto quando usado na movimentação de dados de massa como no I/O de disco. Para resolver esse problema, é utilizado o **acesso direto à memória** (**DMA**). Após estabelecer os buffers, ponteiros e contadores para o dispositivo de I/O, o controlador do dispositivo transfere um bloco inteiro de dados diretamente da memória para o seu próprio buffer ou a partir dele para a memória, sem intervenção da CPU. Somente é gerada uma interrupção por bloco para informar ao driver do dispositivo que a operação foi concluída, em vez de uma interrupção por byte gerada para dispositivos de baixa velocidade. Enquanto o controlador do dispositivo está executando essas operações, a CPU está disponível para cumprir outras tarefas.

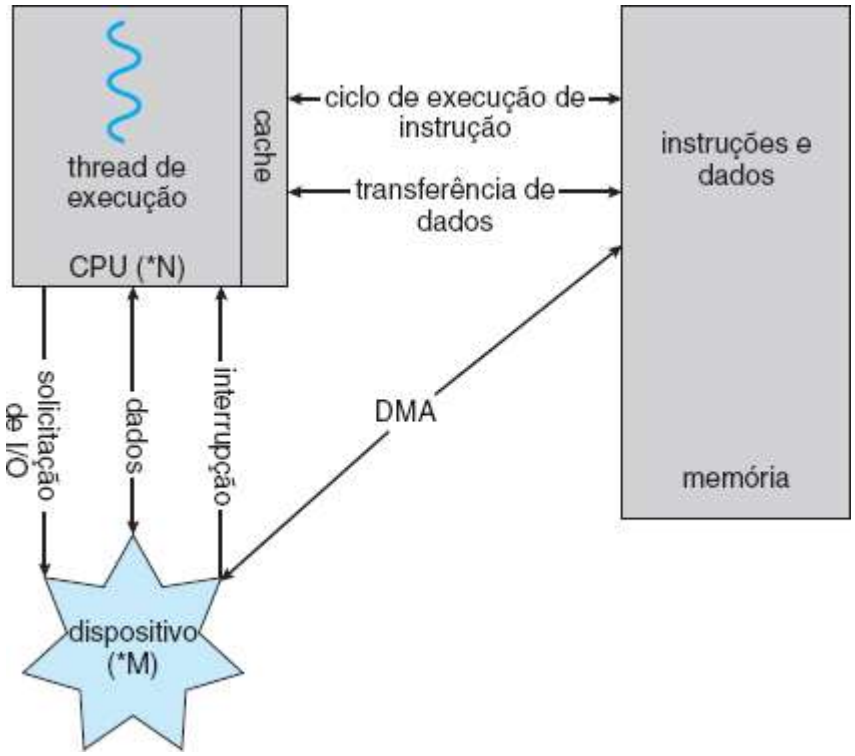

Figura 1.5 Como um moderno sistema de computação funciona.

Alguns sistemas de topo de linha usam arquitetura baseada em switch e não em bus. Nesses sistemas, vários componentes podem conversar com outros componentes concorrentemente, em vez de competirem por ciclos em um bus compartilhado. Nesse caso, o DMA é ainda mais eficaz. A Figura 1.5 mostra a interação de todos os componentes de um sistema de computação.

1.3 Arquitetura dos Sistemas de Computação

Na Seção 1.2, introduzimos a estrutura geral de um sistema de computação típico. Um sistema de computação pode ser organizado de várias maneiras diferentes que, grosso modo, podemos categorizar de acordo com o número de processadores de uso geral utilizados.

1.3.1 Sistemas Uniprocessadores

Até recentemente, a maioria dos sistemas de computação usava um único processador. Em um sistema de processador único, há uma CPU principal capaz de executar um conjunto de instruções de uso geral, incluindo instruções provenientes de processos de usuário. Quase todos os sistemas de processador único também têm outros processadores de uso específico. Eles podem estar na forma de processadores de dispositivos específicos, como controladores de disco, teclado e monitor; ou, nos mainframes, na forma de processadores de uso mais genérico, como os processadores de I/O que movem dados rapidamente entre os componentes do sistema.

Todos esses processadores de uso específico executam um conjunto limitado de instruções e não executam processos de usuário. Às vezes são gerenciados pelo sistema operacional, caso em que o sistema operacional lhes envia informações sobre sua próxima tarefa e monitora seu *status*. Por exemplo, o microprocessador de um controlador de disco recebe uma sequência de solicitações da CPU principal e implementa sua própria fila no disco e o algoritmo de scheduling do disco. Esse esquema libera a CPU principal do overhead do scheduling de disco. Os PCs contêm um microprocessador no teclado para converter os toques de teclas em códigos a serem enviados para a CPU. Em outros sistemas ou circunstâncias, os processadores de uso específico são componentes de baixo nível embutidos no hardware.

O sistema operacional não pode se comunicar com esses processadores; eles executam suas tarefas autonomamente. O uso de microprocessadores de uso específico é comum e não transforma um sistema uniprocessador em um multiprocessador. Quando só existe uma CPU de uso geral, o sistema é uniprocessador.

1.3.2 Sistemas Multiprocessadores

Nos últimos anos, os **sistemas multiprocessadores** (também conhecidos como **sistemas paralelos** ou **sistemas multicore**) começaram a dominar o cenário da computação. Tais sistemas possuem dois ou mais processadores em estreita comunicação, compartilhando o bus do computador e algumas vezes o relógio, a memória e os dispositivos periféricos. Os sistemas multiprocessadores apareceram pela primeira vez e principalmente em servidores e, desde então, migraram para sistemas desktop e laptop. Recentemente, múltiplos processadores têm aparecido em dispositivos móveis como os smartphones e tablets.

Os sistemas multiprocessadores têm três vantagens principais:

1. **Aumento do throughput**. Com o aumento do número de processadores, espera-se que mais trabalho seja executado em menos tempo. No entanto, a taxa de aumento de velocidade com *N* processadores não é *N*; é menor do que *N*. Quando vários processadores cooperam em uma tarefa, observa-se algum overhead para manter todas as partes trabalhando corretamente. Esse overhead, mais a concorrência por recursos compartilhados, diminui o ganho esperado dos processadores adicionais. Do mesmo modo, *N* programadores trabalhando em cooperação não produzem *N* vezes o montante de trabalho que um único programador produziria.

2. **Economia de escala.** Sistemas multiprocessadores podem custar menos do que múltiplos sistemas equivalentes de processador único, já que eles podem compartilhar periféricos, memória de massa e suprimentos de energia. Quando vários programas trabalham sobre o mesmo conjunto de dados, é mais barato armazenar esses dados em um disco, compartilhando-os com todos os processadores, do que usar muitos computadores com discos locais e muitas cópias dos dados.

3. **Aumento da confiabilidade.** Se as funções puderem ser distribuídas apropriadamente entre vários processadores, a falha de um processador não interromperá o sistema, tornando-o apenas mais lento. Se tivermos dez processadores e um falhar, cada um dos nove processadores remanescentes poderá ficar com uma parcela do trabalho do processador que falhou. Assim, o sistema inteiro operará somente 10% mais devagar, em vez de falhar completamente.

O aumento da confiabilidade de um sistema de computação é crucial em muitas aplicações. A capacidade de continuar fornecendo serviço proporcionalmente ao nível do hardware remanescente é chamada de **degradação limpa**. Alguns sistemas vão além da degradação limpa e são chamados de **tolerantes a falhas**, porque podem sofrer falha em qualquer um dos componentes e continuar a operar. A tolerância a falhas requer um mecanismo para permitir que a falha seja detectada, diagnosticada e, se possível, corrigida. O sistema HP NonStop (anteriormente chamado de Tandem) usa a duplicação tanto do hardware como do software para assegurar a operação continuada, apesar das falhas. O sistema consiste em múltiplos pares de CPUs, funcionando em lockstep. Os dois processadores do par executam cada instrução e comparam os resultados. Quando o resultado difere, é porque uma CPU do par está falhando e as duas são interrompidas. O processo que estava sendo executado é, então, transferido para outro par de CPUs e a instrução que falhou é reiniciada. Essa solução é cara, já que envolve hardware especial e considerável duplicação de hardware.

Os sistemas multiprocessadores em uso hoje em dia são de dois tipos. Alguns sistemas utilizam **multiprocessamento assimétrico**, no qual a cada processador é designada uma tarefa específica. Um processador *mestre* controla o sistema; os demais processadores consultam o mestre para instruções ou possuem tarefas predefinidas. Esse esquema define um relacionamento mestre-escravo. O processador mestre organiza e aloca o trabalho para os escravos.

Os sistemas mais comuns utilizam **multiprocessamento simétrico** (**SMP — *symmetric multiprocessing***) no qual cada processador executa todas as tarefas do sistema operacional. SMP significa que todos os processadores são pares; não existe um relacionamento mestre-escravo entre eles. A Figura 1.6 ilustra uma arquitetura SMP típica. Observe que cada processador tem seu próprio conjunto de registradores, assim como um cache privado — ou local. No entanto, todos os processadores compartilham memória física. Um exemplo de um sistema SMP é o AIX, uma versão comercial do UNIX projetada pela IBM. Um sistema AIX pode ser configurado de modo a empregar dúzias de processadores. O benefício desse modelo é que muitos processos podem ser executados simultaneamente — *N* processos podem ser executados se houver *N* CPUs — sem causar uma deterioração significativa de desempenho. Entretanto, devemos controlar cuidadosamente o I/O para assegurar que os dados alcancem o processador apropriado. Além disso, como as CPUs são separadas, uma pode ficar ociosa enquanto outra está sobrecarregada, resultando em ineficiências. Essas ineficiências podem ser evitadas, se os processadores compartilharem determinadas estruturas de dados. Um sistema multiprocessador desse tipo permitirá que processos e recursos — como a memória — sejam compartilhados dinamicamente entre os vários processadores e pode diminuir a diferença entre eles. Tal sistema deve ser escrito

cuidadosamente, como veremos no Capítulo 5. Praticamente todos os sistemas operacionais modernos — incluindo o Windows, o Mac OS X e o Linux — já suportam o SMP.

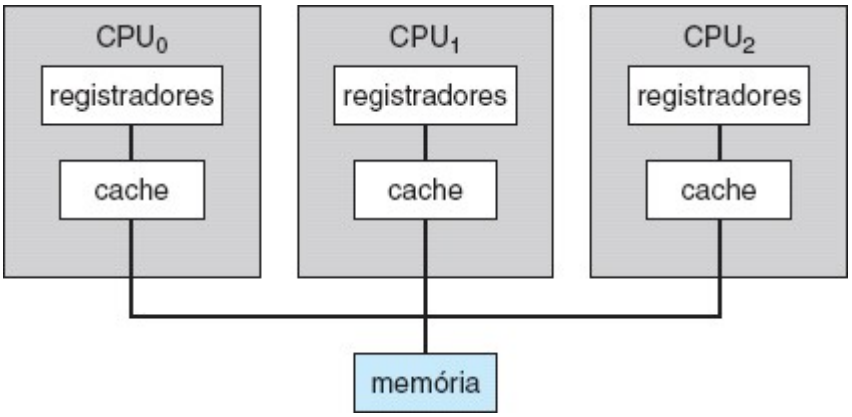

Figura 1.6 Arquitetura de multiprocessamento simétrico.

A diferença entre multiprocessamento simétrico e assimétrico pode resultar tanto do hardware quanto do software. Um hardware especial pode diferenciar os processadores múltiplos, ou o software pode ser escrito para permitir somente um mestre e múltiplos escravos. Por exemplo, o sistema operacional SunOS Versão 4 da Sun Microsystems fornecia multiprocessamento assimétrico, enquanto a Versão 5 (Solaris) é simétrica no mesmo hardware.

O multiprocessamento adiciona CPUs para aumentar o poder de computação. Se a CPU tem um controlador de memória integrado, o acréscimo de CPUs também pode aumentar o montante de memória endereçável no sistema. De qualquer forma, o multiprocessamento pode fazer com que um sistema altere seu modelo de acesso à memória, do acesso uniforme (**UMA — *uniforme memory access***) ao acesso não uniforme (**NUMA — *non-uniform memory access***). O UMA é definido como a situação em que o acesso a qualquer RAM a partir de qualquer CPU leva o mesmo período de tempo. No NUMA, algumas partes da memória podem demorar mais para serem acessadas do que outras, gerando perda de desempenho. Os sistemas operacionais podem minimizar a perda gerada pelo NUMA por meio do gerenciamento de recursos, como discutido na Seção 9.5.4.

Uma tendência recente no projeto de CPUs é a inclusão de múltiplos **núcleos** (**cores**) de computação em um único chip. Tais sistemas multiprocessadores são denominados **multicore**. Eles podem ser mais eficientes do que vários chips de núcleo único porque a comunicação *on-chip* (dentro do chip) é mais veloz do que a comunicação *between-chip* (entre chips). Além disso, um chip com vários núcleos usa bem menos energia do que vários chips de núcleo único.

É importante observar que, enquanto os sistemas multicore são multiprocessadores, nem todos os sistemas multiprocessadores são multicore, como veremos na Seção 1.3.3. Em nossa abordagem dos sistemas multiprocessadores no decorrer deste texto, a menos que mencionado o contrário, usamos o termo mais contemporâneo ***multicore***, que exclui alguns sistemas multiprocessadores.

Na Figura 1.7, mostramos um projeto dual-core com dois núcleos no mesmo chip. Nesse projeto, cada núcleo tem seu próprio conjunto de registradores, assim como seu próprio cache local. Outros projetos podem usar um cache compartilhado ou uma combinação de caches local e compartilhado. Além das considerações relacionadas com a arquitetura, como a disputa por cache, memória e bus, essas CPUs multicore aparecem para o sistema operacional como *N* processadores-padrão. É uma característica que pressiona os projetistas de sistemas operacionais — e programadores de aplicações — a fazer uso desses núcleos de processamento.

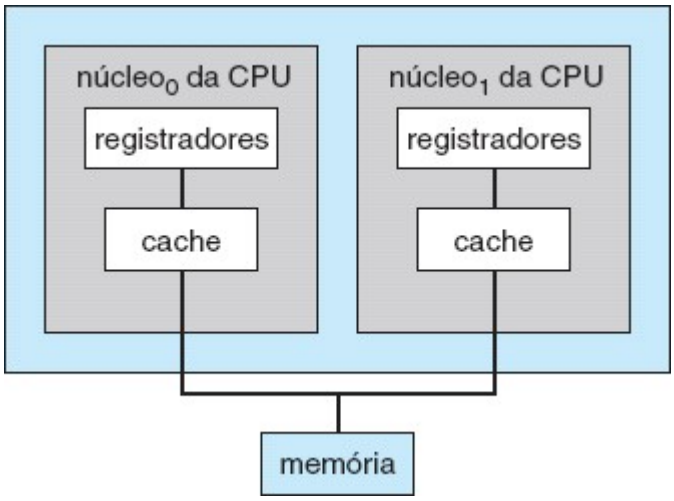

Figura 1.7 Um projeto dual-core com dois núcleos inseridos no mesmo chip.

Para concluir, os **servidores blade** são um desenvolvimento relativamente recente em que várias placas de processador, placas de I/O e placas de rede são inseridas no mesmo chassi. A diferença entre esses sistemas e os sistemas multiprocessadores tradicionais é que cada placa de processador blade é inicializada independentemente e executa seu próprio sistema operacional. Algumas placas de servidor blade também são multiprocessadoras, o que torna difícil diferenciar tipos de computadores. Em essência, esses servidores são compostos por vários sistemas multiprocessadores independentes.

1.3.3 Sistemas Agrupados (Clusters)

Outro tipo de sistema multiprocessador é o **sistema em cluster** que reúne várias CPUs. Esses sistemas diferem dos sistemas multiprocessadores descritos na Seção 1.3.2 por serem compostos de dois ou mais sistemas individuais — ou nós — acoplados. Tais sistemas são considerados **fracamente acoplados**. Cada nó pode ser um sistema uniprocessador ou um sistema multicore. É preciso ressaltar que a definição de *cluster* não é unânime; muitos pacotes comerciais divergem quanto à definição de um sistema em cluster e por que uma forma é melhor do que a outra. A definição geralmente aceita é a de que computadores em cluster compartilham memória e são estreitamente conectados por uma rede local LAN (como descrito no Capítulo 17) ou de uma interconexão mais veloz, como a InfiniBand.

O cluster costuma ser usado para fornecer serviço de **alta disponibilidade** — isto é, o serviço continuará mesmo se um ou mais sistemas do cluster falhar. Geralmente, a alta disponibilidade é obtida pela adição de um nível de redundância ao sistema. Uma camada de software de cluster opera sobre os nós do cluster. Cada nó pode monitorar um ou mais dos outros nós (por meio da LAN). Se a máquina monitorada falhar, a máquina de monitoramento pode apropriar-se da sua memória e reiniciar as aplicações que estavam sendo executadas na máquina que falhou. Os usuários e clientes das aplicações percebem somente uma breve interrupção do serviço.

O cluster pode ser estruturado assimétrica ou simetricamente. No **cluster assimétrico**, uma das máquinas permanece em **modalidade de alerta máximo**, enquanto a outra executa as aplicações. A máquina hospedeira em alerta nada faz além de monitorar o servidor ativo. Se esse servidor falhar, o hospedeiro em alerta torna-se o servidor ativo. No **cluster simétrico**, dois ou mais hospedeiros executam aplicações e se monitoram uns aos outros. É claro que essa estrutura é mais eficiente, já que usa todo o hardware disponível. No entanto, isso requer que mais de uma aplicação esteja disponível para execução.

Já que um cluster é composto por vários sistemas de computação conectados por uma rede, os clusters também podem ser usados para fornecer ambientes de **computação de alto desempenho**. Esses sistemas podem fornecer um poder de computação significativamente maior do que sistemas de um único processador, ou até mesmo sistemas SMP, porque podem executar uma aplicação concorrentemente em todos os computadores do cluster. No entanto, a aplicação deve ter sido escrita especificamente para se beneficiar do cluster. Isso envolve uma técnica conhecida como **paralelização** que divide um programa em componentes separados executados em paralelo em computadores individuais do cluster. Normalmente, essas aplicações são projetadas para que, após cada nó de computação do cluster ter resolvido sua parte do problema, os resultados de todos os nós sejam combinados em uma solução final.

Outras formas de clusters incluem os clusters paralelos e o cluster através de uma rede de longa distância (WAN) (como descrito no Capítulo 17). Os clusters paralelos permitem que múltiplos hospedeiros acessem os mesmos dados na memória compartilhada. Tendo em vista que a maioria dos sistemas operacionais não permite que múltiplos hospedeiros acessem simultaneamente os dados, geralmente os clusters paralelos demandam o uso de versões especiais de software e releases de aplicações especiais. Por exemplo, o Oracle Real Application Cluster é uma versão do banco de dados Oracle projetada para operar em clusters paralelos. Cada uma das máquinas executa o Oracle, e uma camada de software conduz o acesso ao disco compartilhado. Cada máquina tem acesso total a todos os dados do banco de dados. Para fornecer esse acesso compartilhado, o sistema também deve oferecer controle de acesso e trancamento (*locking*) para assegurar que não ocorram operações conflitantes. Essa função, normalmente conhecida como **gerenciador de lock distribuído** (**DLM — *distributed lock manager***), é incluída em algumas tecnologias de cluster.

A tecnologia de cluster está mudando rapidamente. Alguns produtos de cluster suportam vários sistemas em um cluster, assim como nós de cluster separados por quilômetros. Muitas dessas melhorias se tornaram possíveis, graças às **redes de área de armazenamento** (**SANs — *storage-area networks***), como descrito na Seção 10.3.3, que permitem que vários sistemas sejam conectados a um pool de armazenamento. Se as aplicações e seus dados são armazenados na SAN, o software de cluster pode designar a execução da aplicação a qualquer hospedeiro que esteja conectado à SAN. Se o hospedeiro falhar, qualquer outro hospedeiro poderá assumir a tarefa. Em um cluster de banco de dados, vários hospedeiros podem compartilhar o mesmo banco de dados, melhorando muito o desempenho e a confiabilidade. A Figura 1.8 mostra a estrutura geral de um sistema cluster.

CLUSTERS BEOWULF

Os clusters Beowulf são projetados para resolver tarefas de computação de alto desempenho. Um cluster Beowulf é composto por equipamentos de hardware disponíveis no comércio — como os computadores pessoais — conectados por uma simples rede local. Não é necessário um pacote de software específico para construir um cluster. Em vez disso, os nós usam um conjunto de bibliotecas de softwares de código-fonte aberto para se comunicarem entre si. Portanto, existem várias abordagens para a construção de um cluster Beowulf. Normalmente, no entanto, os nós de computação Beowulf executam o sistema operacional Linux. Já que os clusters Beowulf não requerem hardware especial e funcionam usando software de código-fonte aberto disponível gratuitamente, eles oferecem uma estratégia de baixo custo para a construção de um cluster de computação de alto desempenho. Na verdade, alguns clusters Beowulf construídos a partir de computadores pessoais descartados usam centenas de nós para resolver problemas dispendiosos de computação científica.

1.4 Estrutura do Sistema Operacional

Agora que já discutimos a organização e a arquitetura básicas do sistema de computação, estamos prontos para conversar sobre sistemas operacionais. Um sistema operacional fornece o ambiente dentro do qual os programas são executados. Internamente, os sistemas operacionais variam bastante em sua composição, já que estão organizados em muitas linhas diferentes. No entanto, há muitas semelhanças, que consideramos nesta seção.

Um dos aspectos mais importantes dos sistemas operacionais é sua capacidade de multiprogramar. Geralmente, um único programa não pode manter a CPU ou os dispositivos de I/O ocupados o tempo todo. Usuários individuais costumam ter vários programas em execução. A **multiprogramação** aumenta a utilização da CPU organizando os jobs (código e dados) de modo que a CPU tenha sempre um para executar.

A ideia é a seguinte: O sistema operacional mantém vários jobs na memória simultaneamente (Figura 1.9). Já que a memória principal costuma ser muito pequena para acomodar todos os jobs, inicialmente eles são mantidos em disco no **pool de jobs**. Esse pool é composto de todos os processos residentes em disco aguardando alocação da memória principal.

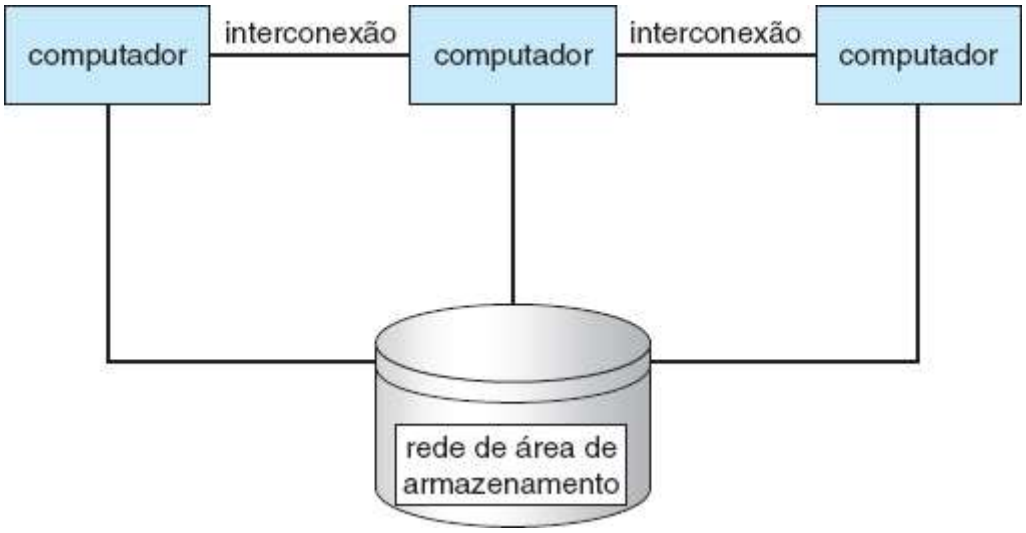

Figura 1.8 Estrutura geral de um sistema em cluster.

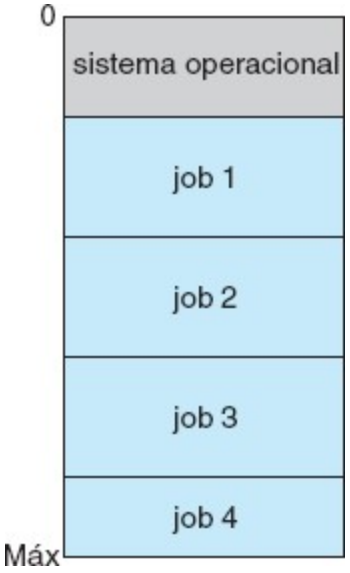

Figura 1.9 Layout da memória de um sistema de multiprogramação.

O conjunto de jobs na memória pode ser um subconjunto dos jobs mantidos no pool de jobs. O sistema operacional seleciona e começa a executar um dos jobs da memória. Em dado momento, o job pode ter de aguardar que alguma tarefa, como uma operação de I/O, seja concluída. Em um sistema sem multiprogramação, a CPU permaneceria ociosa. Em um sistema multiprogramado, o sistema operacional simplesmente passa para um novo job e o executa. Quando *este* job tem de aguardar, a CPU é redirecionada para *outro* job, e assim por diante. Por fim, o primeiro job sai da espera e retoma a CPU. Contanto que pelo menos um job tenha de ser executado, a CPU nunca fica ociosa.

Essa ideia é comum em outras situações da vida. Um advogado não trabalha para apenas um cliente de cada vez, por exemplo. Enquanto um caso está aguardando julgamento ou esperando a preparação de documentos, ele pode trabalhar em outro caso. Se tiver clientes suficientes, nunca ficará ocioso por falta de trabalho. (Advogados ociosos tendem a se tornar políticos e, portanto, existe certo valor social em manter os advogados ocupados.)

Os sistemas multiprogramados fornecem um ambiente em que os diversos recursos do sistema (por exemplo, CPU, memória e dispositivos periféricos) são utilizados eficientemente, mas não possibilitam a interação do usuário com o sistema de computação. O **tempo compartilhado** (ou **multitarefa**) é uma extensão lógica da multiprogramação. Em sistemas de tempo compartilhado, a CPU executa vários jobs alternando-se entre eles, mas as mudanças de job ocorrem com tanta frequência que os usuários podem interagir com cada programa enquanto ele está em execução.

O tempo compartilhado requer um sistema de computação **interativo** que forneça comunicação direta entre o usuário e o sistema. O usuário fornece instruções ao sistema operacional ou a um programa, diretamente, usando um dispositivo de entrada, como teclado, mouse, touch pad, ou touch screen, e espera os resultados imediatos em um dispositivo de saída. Em contrapartida, o **tempo de resposta** deve ser curto — normalmente, menos de um segundo.

Um sistema operacional de tempo compartilhado permite que muitos usuários compartilhem o computador simultaneamente. Como cada ação ou comando em um sistema de tempo compartilhado tende a ser breve, apenas um pequeno tempo de CPU é necessário para cada usuário. Já que o sistema muda rapidamente de um usuário para outro, cada usuário tem a impressão de que o sistema de computação inteiro está sendo dedicado ao seu uso, mesmo que esteja sendo compartilhado entre muitos usuários.

O sistema operacional de tempo compartilhado usa a multiprogramação e o scheduling da CPU para disponibilizar, a cada usuário, uma pequena porção de um computador de tempo compartilhado. Cada usuário tem, pelo menos, um programa separado na memória. Um programa carregado na memória e em execução é chamado um **processo**. Quando um processo entra em execução, normalmente opera apenas por um breve período antes que termine ou tenha de executar I/O. O I/O pode ser interativo, isto é, a saída é exibida para o usuário e a entrada vem do teclado, do mouse ou de outro dispositivo do usuário. Considerando que o I/O interativo costuma ser executado na "velocidade do usuário", pode levar um longo tempo para ser concluído. A entrada, por exemplo, pode ser limitada pela velocidade de digitação do usuário; sete caracteres por segundo é rápido para as pessoas, mas incrivelmente lento para os computadores. Em vez de deixar a CPU ociosa enquanto essa entrada interativa acontece, o sistema operacional direciona rapidamente a CPU ao programa de algum outro usuário.

O tempo compartilhado e a multiprogramação requerem que vários jobs sejam mantidos simultaneamente na memória. Se vários jobs estão prontos para serem conduzidos à memória e se não houver espaço suficiente para todos, o sistema deve selecionar um entre eles. Essa tomada de decisão envolve o **scheduling de jobs**, que discutimos no Capítulo 6. Quando o sistema operacional seleciona um job no pool de jobs, ele carrega esse job na memória para execução. A existência de vários programas na memória ao mesmo tempo requer algum tipo de gerenciamento da memória, o que abordamos nos Capítulos 8 e 9. Além disso, se vários jobs estão prontos para execução ao mesmo tempo,

o sistema deve selecionar que job será executado primeiro. Essa tomada de decisão é o **scheduling da CPU** que também é discutido no Capítulo 6. Para concluir, a execução de múltiplos jobs concorrentemente requer que suas capacidades de afetar uns aos outros sejam limitadas em todas as fases do sistema operacional, incluindo o scheduling de processos, o armazenamento em disco e o gerenciamento da memória. Discutimos essas considerações ao longo do texto.

Em um sistema de tempo compartilhado, o sistema operacional deve assegurar um tempo de resposta razoável. Às vezes, esse objetivo é alcançado pelo **swapping**, em que os processos são alternados entre a memória principal e o disco. Um método comum para garantir um tempo de resposta razoável é a **memória virtual**, uma técnica que permite a execução de um processo que não se encontra totalmente na memória (Capítulo 9). A principal vantagem do esquema de memória virtual é que ele permite que os usuários executem programas maiores do que a **memória física** real. Além disso, ele resume a memória principal a um grande e uniforme array de armazenamento, separando a memória física da **memória lógica** visualizada pelo usuário. Esse esquema libera os programadores da preocupação com limitações de armazenamento na memória.

Um sistema de tempo compartilhado também deve fornecer um sistema de arquivos (Capítulos 11 e 12). O sistema de arquivos reside em um conjunto de discos; portanto, o gerenciamento de discos deve estar disponível (Capítulo 10). Além disso, um sistema de tempo compartilhado fornece um mecanismo para a proteção dos recursos contra uso inapropriado (Capítulo 14). Para garantir execução ordenada, o sistema deve fornecer mecanismos de sincronização e comunicação entre jobs (Capítulo 5) e deve assegurar que os jobs não fiquem presos em um deadlock, esperando eternamente uns pelos outros (Capítulo 7).

Operações do Sistema Operacional

Como mencionado anteriormente, os sistemas operacionais modernos são **dirigidos por interrupções**. Se não existem processos para executar, dispositivos de I/O para servir e usuários a quem responder, então um sistema operacional permanece inativo esperando que algo aconteça. Os eventos são quase sempre sinalizados pela ocorrência de uma interrupção ou de uma exceção (*trap*). Uma **exceção** é uma interrupção gerada por software causada por um erro (por exemplo, divisão por zero ou acesso inválido à memória) ou por uma solicitação específica proveniente de um programa de usuário para que um serviço do sistema operacional seja executado. A natureza de ser dirigido por interrupções de um sistema operacional define a estrutura geral do sistema. Para cada tipo de interrupção, segmentos separados de código do sistema operacional determinam qual ação deve ser executada. É fornecida uma rotina de serviço de interrupções para lidar com a interrupção.

Já que o sistema operacional e os usuários compartilham os recursos de hardware e software do sistema de computação, precisamos assegurar que um erro em um programa de usuário, cause problemas somente para o programa em execução. Com o compartilhamento, muitos processos poderiam ser afetados adversamente por um bug em um programa. Por exemplo, se um processo fica travado em um loop infinito, esse loop poderia impedir a operação correta de muitos outros processos. Erros mais sutis podem ocorrer em um sistema com multiprogramação, em que um programa defeituoso pode modificar outro programa, os dados de outro programa ou até mesmo o próprio sistema operacional.

Sem proteção contra esses tipos de erros, o computador deve executar apenas um processo de cada vez, ou qualquer saída será suspeita. Um sistema operacional projetado adequadamente deve assegurar que um programa incorreto (ou malicioso) não possa causar a execução incorreta de outros programas.

1.5.1 Operação em Modalidade Dual e Multimodalidade

Para garantir a execução apropriada do sistema operacional, temos que ser capazes de distinguir entre a execução de código do sistema operacional e de código definido pelo usuário. A abordagem adotada pela maioria dos sistemas de computação é o fornecimento de suporte de hardware que permite diferenciar as diversas modalidades de execução.

Precisamos de, pelo menos, duas ***modalidades*** de operação separadas: a **modalidade de usuário** e a **modalidade de kernel** (também chamada de **modalidade de supervisor**, **modalidade de sistema** ou **modalidade privilegiada**). Um bit, chamado **bit de modalidade**, é adicionado ao hardware do computador para indicar a modalidade corrente: kernel (0) ou usuário (1). Com o bit de modalidade, podemos distinguir entre uma tarefa que é executada em nome do sistema operacional e a que é executada em nome do usuário. Quando o sistema de computação está operando em nome de uma aplicação de usuário, ele está em modalidade de usuário. Mas, quando a aplicação de usuário solicita um serviço do sistema operacional (por meio de uma chamada de sistema), o sistema deve passar da modalidade de usuário para a de kernel de modo a atender a solicitação. Isso é mostrado na Figura 1.10. Como veremos, essa melhoria de arquitetura também é útil em muitos outros aspectos de operação do sistema.

Em tempo de inicialização do sistema, o hardware começa a operar em modalidade de kernel. O sistema operacional é, então, carregado e inicia aplicações de usuário em modalidade de usuário. Sempre que uma exceção ou interrupção ocorre, o hardware passa da modalidade de usuário para a modalidade de kernel (isto é, muda para 0 o estado do bit de modalidade). Portanto, sempre que o sistema operacional obtém o controle do computador, ele está em modalidade de kernel. O sistema sempre passa para a modalidade de usuário (posicionando o bit de modalidade como 1) antes de passar o controle para um programa de usuário.

A modalidade dual de operação fornece os meios para a proteção do sistema operacional contra usuários errantes — e a proteção desses contra eles mesmos. Obtemos essa proteção designando como **instruções privilegiadas** algumas das instruções de máquina que podem causar erro. O hardware permite que as instruções privilegiadas sejam executadas somente em modalidade de kernel. Se é feita alguma tentativa de executar uma instrução privilegiada em modalidade de usuário, o hardware não executa a instrução, tratando-a como ilegal e interceptando-a (por uma exceção) para o sistema operacional.

A instrução de passagem para a modalidade de kernel é um exemplo de instrução privilegiada. Alguns outros exemplos incluem o controle de I/O, o gerenciamento do timer e o gerenciamento de interrupções. Como veremos no decorrer do texto, há muitas outras instruções privilegiadas.

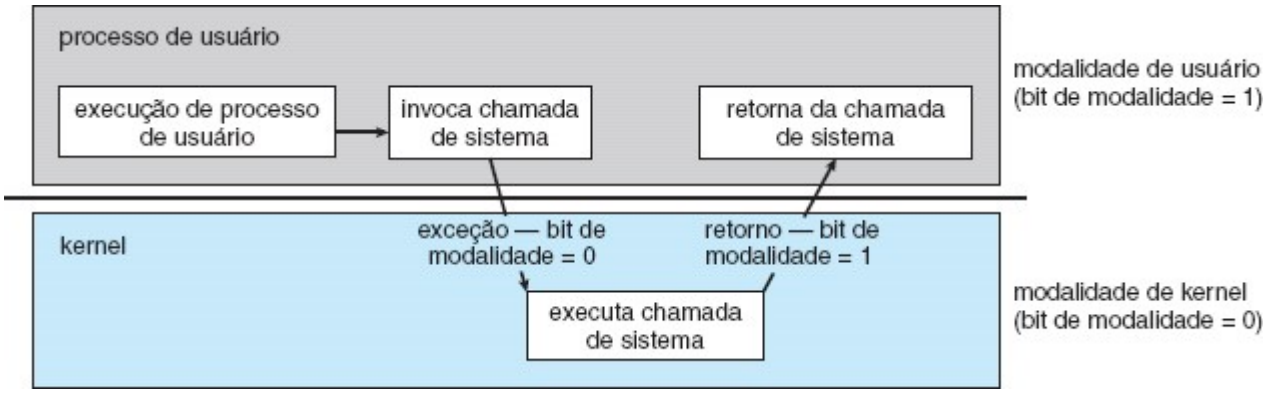

Figura 1.10 Transição da modalidade de usuário para a modalidade de kernel.

O conceito de modalidades pode ser estendido para além de duas modalidades (caso em que a CPU utiliza mais de um bit para estabelecer e testar a modalidade). Com frequência, CPUs que suportam virtualização (Seção 16.1) têm uma modalidade separada para indicar quando o **gerenciador de máquinas virtuais** (**VMM — *virtual machine manager***) — e o software de gerenciamento de virtualização — estão no controle do sistema. Nessa modalidade, o VMM tem mais privilégios do que os processos de usuário e menos do que o kernel. Ele precisa desse nível de privilégio para poder criar e gerenciar máquinas virtuais, alterando o estado da CPU para fazer isso. Às vezes, também, modalidades diferentes são usadas por vários componentes do kernel. Observemos que, como uma alternativa às modalidades, o projetista da CPU pode usar outros métodos para diferenciar privilégios operacionais. A família de CPUs Intel 64 suporta quatro *níveis de privilégios*, por exemplo, e suporta a virtualização, mas não tem uma modalidade separada para ela.

Podemos ver, agora, o ciclo de vida da execução de instruções em um sistema de computação. O controle inicial reside no sistema operacional, no qual instruções são executadas em modalidade de kernel. Quando o controle é passado para uma aplicação de usuário, a modalidade é posicionada como modalidade de usuário. O controle acaba sendo devolvido ao sistema operacional por uma interrupção, uma exceção ou uma chamada de sistema.

As chamadas de sistema fornecem o meio para que um programa de usuário solicite ao sistema operacional que execute tarefas reservadas a ele em nome do programa de usuário. Uma chamada de sistema pode ser invocada de várias maneiras, dependendo da funcionalidade fornecida pelo processador subjacente. De qualquer forma, é o método usado por um processo para solicitar uma ação ao sistema operacional. Geralmente, uma chamada de sistema assume a forma de uma exceção para uma localização específica no vetor de interrupções. Essa exceção pode ser executada por uma instrução `trap` genérica, embora alguns sistemas (como o MIPS) tenham uma instrução `syscall` específica para invocar uma chamada de sistema.

Quando uma chamada de sistema é executada, ela é tipicamente tratada pelo hardware como uma interrupção de software. O controle passa, por intermédio do vetor de interrupções, para uma rotina de serviço no sistema operacional, e o bit de modalidade é posicionado como modalidade de kernel. A rotina de serviço da chamada de sistema faz parte do sistema operacional. O kernel examina a instrução de interrupção para determinar qual chamada de sistema ocorreu; um parâmetro indica que tipo de serviço o programa do usuário está solicitando. Informações adicionais exigidas pela solicitação podem ser passadas em registradores, na pilha ou na memória (com ponteiros para as locações na memória passados em registradores). O kernel verifica se os parâmetros estão corretos e legais, executa a solicitação e retorna o controle para a instrução seguinte à chamada de sistema. Descrevemos as chamadas de sistema com mais detalhes na Seção 2.3.

A ausência de uma modalidade dual suportada por hardware pode causar falhas graves em um sistema operacional. Por exemplo, o MS-DOS foi escrito para a arquitetura Intel 8088 que não possui bit de modalidade e, portanto, não tem modalidade dual. Um programa de usuário operando incorretamente pode tirar do ar o sistema operacional gravando dados sobre ele; e vários programas poderiam gravar ao mesmo tempo em um dispositivo, com resultados potencialmente desastrosos. Versões modernas da CPU Intel fornecem operação em modalidade dual. Como resultado,

a maioria dos sistemas operacionais contemporâneos — como o Microsoft Windows 7 e, também, o Unix e o Linux — se beneficia desse recurso e fornece maior proteção ao sistema operacional.

Uma vez que a proteção de hardware esteja ativa, ela detecta erros que violam as modalidades. Normalmente, esses erros são manipulados pelo sistema operacional. Se um programa de usuário falha de alguma forma — como ao tentar executar uma instrução ilegal ou acessar memória que não faça parte do espaço de endereçamento do usuário —, o hardware gera uma exceção para o sistema operacional. A exceção transfere o controle, por meio do vetor de interrupções, para o sistema operacional, da mesma forma que a interrupção faz. Quando ocorre um erro no programa, o sistema operacional deve encerrá-lo anormalmente. Essa situação é manipulada pelo mesmo código de um encerramento anormal solicitado pelo usuário. Uma mensagem de erro apropriada é fornecida e a memória do programa pode ser despejada. Normalmente, o despejo da memória é gravado em um arquivo para que o usuário ou o programador possa examiná-lo, talvez corrigi-lo e reiniciar o programa.

1.5.2 Timer

Devemos assegurar que o sistema operacional mantenha o controle sobre a CPU. Não podemos permitir que um programa de usuário fique travado em um loop infinito ou não chame os serviços do sistema e nunca retorne o controle ao sistema operacional. Para alcançar esse objetivo, podemos utilizar um **timer**. Um timer pode ser configurado para interromper o computador após um período especificado. O período pode ser fixo (por exemplo, 1/60 segundos) ou variável (por exemplo, de 1 milissegundo a 1 segundo). Geralmente, um **timer variável** é implementado por um relógio de marcação fixa e um contador. O sistema operacional posiciona o contador. Cada vez que o relógio marca, o contador é decrementado. Quando o contador atinge 0, ocorre uma interrupção. Por exemplo, um contador de 10 bits com um relógio de 1 milissegundo permite interrupções a intervalos de 1 a 1024 milissegundos em passos de 1 milissegundo.

Antes de retornar o controle ao usuário, o sistema operacional assegura que o timer seja configurado para causar uma interrupção. Quando o timer causa a interrupção, o controle é transferido automaticamente para o sistema operacional que pode tratar a interrupção como um erro fatal ou dar mais tempo ao programa. É claro que as instruções que modificam o conteúdo do timer são privilegiadas.

Podemos usar o timer para impedir que um programa de usuário seja executado por muito tempo. Uma técnica simples é a inicialização de um contador com o período de tempo em que um programa pode ser executado. Um programa com um limite de tempo de 7 minutos, por exemplo, teria seu contador inicializado em 420. A cada segundo, o timer causaria uma interrupção e o contador seria decrementado de uma unidade. Enquanto o contador é positivo, o controle é retornado ao programa do usuário. Quando o contador se torna negativo, o sistema operacional encerra o programa por exceder o limite de tempo designado.

1.6 Gerenciamento de Processos

Um programa nada faz, a menos que suas instruções sejam executadas por uma CPU. Um programa em execução, como mencionado, é um processo. Um programa de usuário de tempo compartilhado, como um compilador, é um processo. Um programa de processamento de texto, sendo executado por um usuário individual em um PC, é um processo. Uma tarefa do sistema, como o envio de saída para uma impressora, também pode ser um processo (ou pelo menos parte de um processo). Por enquanto, você pode considerar um processo como um job ou um programa de tempo compartilhado, mas posteriormente aprenderá que o conceito é mais genérico. Como veremos no Capítulo 3, é possível fornecer chamadas de sistema que possibilitem aos processos criarem subprocessos a serem executados concorrentemente.

Um processo precisa de certos recursos — incluindo tempo de CPU, memória, arquivos e dispositivos de I/O — para cumprir sua tarefa. Esses recursos são fornecidos ao processo quando ele é criado, ou são alocados a ele durante sua execução. Além dos diversos recursos físicos e lógicos que um processo obtém quando é criado, vários dados (entradas) de inicialização também podem ser passados. Por exemplo, considere um processo cuja função seja exibir o *status* de um arquivo na tela de um terminal. O processo receberá como entrada o nome do arquivo e executará as instruções e chamadas de sistema apropriadas para obter e exibir as informações desejadas no terminal. Quando o processo terminar, o sistema operacional reclamará os recursos reutilizáveis.

Enfatizamos que um programa por si só não é um processo. Um programa é uma entidade *passiva*, como os conteúdos de um arquivo armazenado em disco, enquanto um processo é uma entidade *ativa*. Um processo de um único thread tem um **contador de programa** especificando a próxima instrução a ser executada. (Os threads são abordados no Capítulo 4.) A execução de tal processo deve ser sequencial. A CPU executa uma instrução do processo após a outra, até o processo ser concluído. Além disso, a qualquer momento, no máximo uma instrução é executada em nome do processo. Portanto, embora dois processos possam estar associados ao mesmo programa, eles são considerados como duas sequências de execução separadas. Um processo com vários threads tem múltiplos contadores de programa, cada um apontando para a próxima instrução a ser executada para determinado thread.

Um processo é a unidade de trabalho de um sistema. Um sistema consiste em um conjunto de processos, alguns dos quais são processos do sistema operacional (os que executam código do sistema), e o resto são processos de usuário (aqueles que executam código de usuário). Todos esses processos podem ser executados concorrentemente — pela multiplexação em uma única CPU, por exemplo.

O sistema operacional é responsável pelas seguintes atividades relacionadas com o gerenciamento de processos:

- Executar o scheduling de processos e threads nas CPUs
- Criar e excluir processos de usuário e do sistema
- Suspender e retomar processos
- Fornecer mecanismos de sincronização de processos
- Fornecer mecanismos de comunicação entre processos

Discutimos as técnicas de gerenciamento de processos nos Capítulos 3 a 5.

1.7 Gerenciamento da Memória

Como discutimos na Seção 1.2.2, a memória principal é essencial para a operação de um sistema de computação moderno. A memória principal é um grande array de bytes que variam em tamanho de centenas de milhares a bilhões. Cada byte tem seu próprio endereço. A memória principal é um repositório de dados rapidamente acessáveis compartilhado pela CPU e dispositivos de I/O. O processador central lê instruções na memória principal, durante o ciclo de busca de instruções, e lê e grava dados a partir da memória principal, durante o ciclo de busca de dados (em uma arquitetura von Neumann). Como mencionado anteriormente, a memória principal costuma ser o único grande dispositivo de armazenamento que a CPU consegue endereçar e acessar diretamente. Por exemplo, para a CPU processar dados do disco, primeiro esses dados devem ser transferidos para a memória principal por chamadas de I/O geradas pela CPU. Da mesma forma, as instruções devem estar na memória para a CPU executá-las.

Para que um programa seja executado, ele deve ser mapeado para endereços absolutos e carregado na memória. Quando o programa é executado, ele acessa suas instruções e dados na memória gerando esses endereços absolutos. Eventualmente, o programa é encerrado, seu espaço de memória é declarado disponível e o próximo programa pode ser carregado e executado.

Para melhorar tanto a utilização da CPU quanto a velocidade de resposta do computador para seus usuários, computadores de uso geral devem manter vários programas na memória, criando a necessidade de gerenciamento da memória. Muitos esquemas diferentes de gerenciamento da memória são usados. Esses esquemas refletem diversas abordagens, e a eficácia de determinado algoritmo depende da situação. Ao selecionar um esquema de gerenciamento da memória para um sistema específico, devemos levar em consideração muitos fatores — principalmente o projeto de *hardware* do sistema. Cada algoritmo requer seu próprio suporte de hardware.

O sistema operacional é responsável pelas seguintes atividades relacionadas com o gerenciamento da memória:

- Controlar quais partes da memória estão sendo correntemente utilizadas e quem as está utilizando
- Decidir que processos (ou partes de processos) e dados devem ser transferidos para dentro e para fora da memória
- Alocar e desalocar espaço de memória conforme necessário

As técnicas de gerenciamento da memória são discutidas nos Capítulos 8 e 9.

1.8 Gerenciamento do Armazenamento

Para tornar o sistema de computação conveniente para os usuários, o sistema operacional fornece uma visão uniforme e lógica do armazenamento de informações. O sistema operacional abstrai, das propriedades físicas dos seus dispositivos de armazenamento, a definição de uma unidade lógica de armazenamento, o **arquivo**. O sistema operacional mapeia arquivos para mídia física e acessa esses arquivos por meio dos dispositivos de armazenamento.

1.8.1 Gerenciamento do Sistema de Arquivos

O gerenciamento de arquivos é um dos componentes mais visíveis de um sistema operacional. Os computadores podem armazenar informações em vários tipos diferentes de mídia física. Disco magnético, disco ótico e fita magnética são os mais comuns. Todas essas mídias têm suas próprias características e organização física. Cada mídia é controlada por um dispositivo, tal como um drive de disco ou de fita, que também tem características próprias. Essas propriedades incluem velocidade de acesso, capacidade, taxa de transferência de dados e método de acesso (sequencial ou randômico).

Um arquivo é um conjunto de informações relacionadas definido por seu criador. Normalmente, os arquivos representam

programas (em ambas as formas, fonte e objeto) e dados. Arquivos de dados podem ser numéricos, alfabéticos, alfanuméricos ou binários. Os arquivos podem ter formato livre (por exemplo, arquivos de texto) ou podem ser formatados rigidamente (no caso de campos fixos). É claro que o conceito de arquivo é extremamente genérico.

O sistema operacional implementa o conceito abstrato de arquivo gerenciando mídias de armazenamento de massa, como fitas e discos, e os dispositivos que as controlam. Além disso, normalmente, os arquivos são organizados em diretórios para facilitar seu uso. Para concluir, quando vários usuários têm acesso aos arquivos, pode ser desejável controlar que usuário pode acessar um arquivo e como esse usuário pode acessá-lo (por exemplo, para ler, gravar, anexar).

O sistema operacional é responsável pelas seguintes atividades relacionadas com o gerenciamento de arquivos:

- Criar e apagar arquivos
- Criar e apagar diretórios para organizar arquivos

- Suportar primitivos para a manipulação de arquivos e diretórios
- Mapear arquivos em memória secundária
- Fazer backup de arquivos em mídias de armazenamento estáveis (não voláteis)

As técnicas de gerenciamento de arquivos são discutidas nos Capítulos 11 e 12.

1.8.2 Gerenciamento de Armazenamento de Massa

Como vimos, já que a memória principal é pequena demais para acomodar todos os dados e programas, e já que os dados que ela armazena são perdidos quando não há energia, o sistema de computação deve fornecer memória secundária como backup da memória principal. A maioria dos sistemas de computação modernos usa discos como o principal meio de armazenamento on-line tanto para programas quanto para dados. Grande parte dos programas — incluindo compiladores, montadores, processadores de texto, editores e formatadores — é armazenada em um disco até ser carregada na memória. Eles, então, usam o disco como fonte e destino de seu processamento. Portanto, o gerenciamento apropriado do armazenamento em disco é de importância primordial para um sistema de computação. O sistema operacional é responsável pelas seguintes atividades relacionadas com o gerenciamento de disco:

- Gerenciar o espaço livre
- Alocar espaço de armazenamento
- Fazer o scheduling de disco

Já que a memória secundária é utilizada com frequência, ela deve ser usada de maneira eficiente. A velocidade total de operação de um computador pode depender das velocidades do subsistema de disco e dos algoritmos que manipulam esse subsistema.

No entanto, há muitas utilidades para uma memória mais lenta e mais barata (e às vezes de maior capacidade) do que a memória secundária. Backups de dados de disco, armazenamento de dados pouco usados e armazenamento de longo prazo são alguns exemplos. Os drives de fita magnética e suas fitas e os drives de CD e DVD e discos são típicos dispositivos de **memória terciária**. A mídia (fitas e discos óticos) varia entre os formatos **WORM** (grava uma vez, lê várias vezes) e **RW** (lê e grava).

A memória terciária não é crucial para o desempenho do sistema, mas mesmo assim deve ser gerenciada. Alguns sistemas operacionais assumem essa tarefa, enquanto outros deixam o gerenciamento da memória terciária para programas aplicativos. Algumas das funções que os sistemas operacionais podem fornecer incluem a montagem e desmontagem da mídia em dispositivos, a alocação e liberação dos dispositivos para uso exclusivo pelos processos e a migração de dados da memória secundária para a terciária.

As técnicas de gerenciamento da memória secundária e terciária são discutidas no Capítulo 10.

1.8.3 Armazenamento em Cache (*Caching*)

Caching é um princípio importante dos sistemas de computação. Vejamos como funciona. Normalmente, as informações são mantidas em algum sistema de armazenamento (como a memória principal). Na medida em que são utilizadas, elas são copiadas temporariamente em um sistema de armazenamento mais veloz — o cache. Quando precisamos de uma informação específica, primeiro verificamos se ela está no cache. Se estiver, usamos a informação diretamente a partir do cache. Se não estiver, usamos a informação a partir da fonte, inserindo uma cópia no cache supondo que, em breve, ela possa ser necessária novamente.

Além disso, registradores internos programáveis, como os registradores índice, fornecem um cache de alta velocidade para a memória principal. O programador (ou compilador) implementa os algoritmos de alocação e realocação de registradores para decidir quais informações devem ser mantidas em registradores e quais devem ficar na memória principal.

Outros caches são implementados totalmente em hardware. Por exemplo, a maioria dos sistemas tem um cache de instruções para armazenar as instruções cuja execução é esperada em seguida. Sem esse cache, a CPU teria de esperar vários ciclos enquanto uma instrução é buscada na memória principal. Por motivos semelhantes, grande parte dos sistemas tem um ou mais caches de dados de alta velocidade na hierarquia da memória. Não estamos interessados nesses caches de hardware neste texto, já que eles estão fora do controle do sistema operacional.

Como os caches têm tamanho limitado, o **gerenciamento do cache** é um importante problema de projeto. A seleção cuidadosa do tamanho do cache e de uma política de realocação pode resultar em um desempenho muito melhor. A Figura 1.11 compara o desempenho do armazenamento em grandes estações de trabalho e pequenos servidores. Vários algoritmos de realocação para caches controlados por software são discutidos no Capítulo 9.

A memória principal pode ser vista como um cache veloz da memória secundária, já que os dados da memória secundária devem ser copiados na memória principal para uso e devem estar na memória principal antes de serem movidos para a memória secundária por segurança. Os dados do sistema de arquivos, que residem permanentemente na memória secundária, podem aparecer em vários níveis na hierarquia de armazenamento. No nível mais alto, o sistema operacional pode manter um cache de dados do sistema de arquivos na memória principal. Além disso, discos de estado sólido podem ser usados como memória de alta velocidade acessada pela interface do sistema de arquivos. Grande parte

da memória secundária fica em discos magnéticos. O conteúdo armazenado em disco magnético, por sua vez, é frequentemente duplicado em fitas magnéticas ou discos removíveis para proteção contra a perda de dados no caso de uma falha no disco rígido. Alguns sistemas transferem automaticamente dados de arquivos antigos da memória secundária para a memória terciária, como os jukeboxes de fita, para reduzir o custo de armazenamento (consulte o Capítulo 10).

Nível	1	2	3	4	5
Nome	registradores	cache	memória principal	disco de estado sólido	disco magnético
Tamanho típico	< 1 KB	< 16 MB	< 64 GB	< 1 TB	< 10 TB
Tecnologia de implementação	memória personalizada com várias portas CMOS	CMOS SRAM em chip ou fora do chip	CMOS SRAM	memória flash	disco magnético
Tempo de acesso (ns)	0,25 – 0,5	0,5 – 25	80 – 250	25.000 – 50.000	5.000.000
Largura de banda (MB/s)	20000 – 100000	5000 – 10000	1000 – 5000	500	20 – 150
Gerenciado por	compilador	hardware	sistema operacional	sistema operacional	sistema operacional
Coadjuvado por	cache	memória principal	disco	disco	disco

Figura 1.11 Desempenho de vários níveis de armazenamento.

A movimentação de informações entre os níveis de uma hierarquia de armazenamento pode ser explícita ou implícita, dependendo do projeto de hardware e do software de controle do sistema operacional. Por exemplo, a transferência de dados do cache para a CPU e os registradores é, geralmente, uma função do hardware, sem intervenção do sistema operacional. Por outro lado, a transferência de dados de disco para a memória costuma ser controlada pelo sistema operacional.

Em uma estrutura de armazenamento hierárquica, os mesmos dados podem aparecer em diferentes níveis do sistema de armazenamento. Por exemplo, suponha que um inteiro A que tenha que ser incrementado de 1 esteja localizado no arquivo B e o arquivo B resida em disco magnético. A operação de incremento é efetuada, sendo emitida, em primeiro lugar, uma operação de I/O para copiar, na memória principal, o bloco de disco no qual A reside. Essa operação é seguida pela cópia de A no cache e em um registrador interno. Assim, a cópia de A aparece em vários locais: no disco magnético, na memória principal, no cache e em um registrador interno (consulte a Figura 1.12). Uma vez que o incremento tenha lugar no registrador interno, o valor de A será diferente nos diversos sistemas de armazenamento. O valor de A será o mesmo somente depois que o novo valor de A seja copiado do registrador interno para o disco magnético.

Em um ambiente de computação em que só um processo é executado de cada vez, esse esquema não cria dificuldades, já que um acesso ao inteiro A sempre será efetuado em sua cópia no nível mais alto da hierarquia. No entanto, em um ambiente multitarefa em que a CPU se reveza entre vários processos, deve-se ter extremo cuidado para garantir que, se muitos processos quiserem acessar A, todos eles obtenham o valor mais recente de A.

A situação torna-se mais complicada em um ambiente multiprocessador em que, além de manter registradores internos, cada uma das CPUs também contém um cache local (Figura 1.6). Em um ambiente assim, uma cópia de A pode existir simultaneamente em vários caches. Já que todas as CPUs podem operar em paralelo, temos de nos certificar de que uma atualização do valor de A em um cache seja imediatamente refletida em todos os outros caches em que A reside. Essa situação é chamada de **coerência do cache**, e, em geral, é um problema do hardware (manipulado abaixo do nível do sistema operacional).

Em um ambiente distribuído, a situação torna-se ainda mais complexa. Nesse ambiente, várias cópias (ou réplicas) do mesmo arquivo podem ser mantidas em diferentes computadores. Já que as diversas réplicas podem ser acessadas e atualizadas concorrentemente, alguns sistemas distribuídos asseguram que, quando uma réplica é atualizada em um local, todas as outras réplicas são atualizadas assim que possível. Há várias maneiras de garantir isso, como discutimos no Capítulo 17.

1.8.4 Sistemas de I/O

Um dos objetivos de um sistema operacional é ocultar, dos usuários, as peculiaridades de dispositivos de hardware específicos. Por exemplo, no UNIX, as peculiaridades dos dispositivos de I/O são ocultas de grande parte do próprio sistema operacional pelo **subsistema de I/O**. O subsistema de I/O tem vários componentes:

- Um componente de gerenciamento de memória que inclui o uso de buffer, de cache e spooling
- Uma interface genérica para drivers de dispositivos
- Drivers para dispositivos de hardware específicos

Apenas o driver conhece as peculiaridades do dispositivo específico ao qual é atribuído.

Discutimos, na Seção 1.2.3, como os manipuladores de interrupção e drivers de dispositivos são usados na construção de subsistemas de I/O eficientes. No Capítulo 13, discutimos como o subsistema de I/O interage com outros componentes do sistema, gerencia dispositivos, transfere dados e detecta o término de I/O.

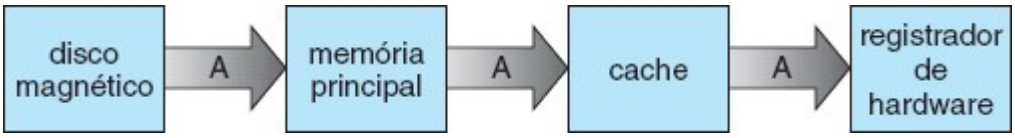

Figura 1.12 Migração de inteiro A de disco para o registrador.

1.9 Proteção e Segurança

Se um sistema de computação tem múltiplos usuários e permite a execução concorrente de múltiplos processos, então o acesso aos dados deve ser regulado. Para atingir esse objetivo, mecanismos asseguram que arquivos, segmentos de memória, CPU e outros recursos possam ser operados apenas pelos processos que receberam autorização apropriada do sistema operacional. Por exemplo, o hardware de endereçamento da memória assegura que um processo só possa ser executado dentro de seu próprio espaço de endereçamento. O timer assegura que nenhum processo possa obter o controle da CPU sem, ao seu tempo, abandonar o controle. Registradores de controle de dispositivo não podem ser acessados por usuários para que a integridade dos diversos dispositivos periféricos seja protegida.

Proteção é, portanto, qualquer mecanismo que controle o acesso de processos ou usuários aos recursos definidos por um sistema de computação. Esse mecanismo deve fornecer os meios para a especificação dos controles a serem impostos e os meios para sua imposição.

A proteção pode aumentar a confiabilidade detectando erros latentes nas interfaces entre subsistemas componentes. A detecção precoce de erros de interface pode impedir, com frequência, a contaminação de um subsistema saudável por outro subsistema defeituoso. Além disso, um recurso desprotegido não pode se defender do uso (ou mau uso) por um usuário não autorizado ou incompetente. Um sistema orientado à proteção fornece meios para a distinção entre o uso autorizado e não autorizado, como discutimos no Capítulo 14.

Um sistema pode ter proteção adequada e, mesmo assim, estar propenso a falhas e permitir acesso inapropriado. Considere um usuário cujas informações de autenticação (seu meio de se identificar para o sistema) foram roubadas. Seus dados poderiam ser copiados ou apagados, mesmo que a proteção de arquivos e memória esteja funcionando. É responsabilidade da **segurança** a defesa de um sistema contra ataques externos e internos. Esses ataques abrangem uma vasta gama e incluem vírus e vermes, ataques de recusa de serviço (que usam todos os recursos de um sistema e, assim, mantêm os usuários legítimos fora do sistema), roubo de identidade e roubo de serviço (uso não autorizado de um sistema). A prevenção contra alguns desses ataques é considerada uma função do sistema operacional em determinados sistemas, enquanto outros sistemas a deixam para a política adotada ou para um software adicional. Em razão do aumento alarmante dos incidentes de segurança, os recursos de segurança do sistema operacional representam uma área de pesquisa e implementação em rápido crescimento. Discutimos segurança no Capítulo 15.

A proteção e a segurança exigem que o sistema seja capaz de identificar todos os seus usuários. A maioria dos sistemas operacionais mantém uma lista de nomes de usuários com **identificadores de usuário (IDs de usuário)** associados. No linguajar do Windows, esse é o **ID de segurança (SID — *security ID*)**. Esses IDs numéricos são exclusivos, um por usuário. Quando um usuário conecta-se ao sistema, o estágio de autenticação determina o ID de usuário apropriado para ele. Esse ID de usuário é associado a todos os processos e threads do usuário. Quando um ID precisa estar legível para um usuário, ele é convertido de volta para o nome do usuário por meio da lista de nomes de usuários.

Em algumas circunstâncias, podemos querer identificar conjuntos de usuários em vez de usuários individuais. Por exemplo, o proprietário de um arquivo em um sistema UNIX pode ter permissão para executar todas as operações sobre esse arquivo, enquanto um conjunto selecionado de usuários pode ter permissão apenas para ler o arquivo. Para usar esse recurso, precisamos definir um nome de grupo e o conjunto de usuários pertencente a esse grupo. A funcionalidade de grupo pode ser implementada como uma lista de nomes de grupo e **identificadores de grupo**, com abrangência em todo o sistema. Um usuário pode estar em um ou mais grupos, dependendo das decisões de projeto do sistema operacional. Os IDs de grupo do usuário também são incluídos em todos os processos e threads associados.

No decorrer do uso normal de um sistema, os IDs de usuário e de grupo de um usuário são suficientes. No entanto, às vezes um usuário precisa **escalar privilégios** para obter permissões adicionais para uma atividade. O usuário pode precisar de acesso a um dispositivo que é restrito, por exemplo. Os sistemas operacionais fornecem vários métodos para permitir a escalação de privilégios. No UNIX, por exemplo, o atributo *setuid* em um programa faz com que esse programa seja executado com o ID de usuário do proprietário do arquivo, em vez do ID do usuário corrente. O processo é executado com esse **UID efetivo** até que privilégios adicionais sejam desativados ou o processo seja concluído.

1.10 Estruturas de Dados do Kernel

Abordamos, a seguir, um tópico essencial para a implementação do sistema operacional: a maneira como os dados são estruturados no sistema. Nesta seção, descrevemos brevemente várias estruturas de dados fundamentais usadas extensivamente em sistemas operacionais. Leitores que necessitem de detalhes adicionais sobre essas estruturas, assim como sobre outras, devem consultar a bibliografia no fim do capítulo.

1.10.1 Listas, Pilhas e Filas

Um array é uma estrutura de dados simples em que cada elemento pode ser acessado diretamente. Por exemplo, a memória principal é construída como um array. Se o item de dados que está sendo armazenado é maior do que um byte, vários bytes podem ser alocados para o item e ele é endereçado como número do item × tamanho do item. Mas, e quanto ao armazenamento de um item cujo tamanho pode variar? E, no caso da remoção de um item, as posições relativas dos itens remanescentes devem ser preservadas? Em tais situações, os arrays dão lugar a outras estruturas de dados.

Depois dos arrays, talvez as listas sejam as estruturas de dados mais fundamentais da ciência da computação. Enquanto cada item de um array pode ser acessado diretamente, os itens de uma lista devem ser acessados em uma ordem específica. Isto é, uma **lista** representa um conjunto de valores de dados como uma sequência. O método mais comum para a implementação dessa estrutura é a **lista encadeada** em que os itens são encadeados uns aos outros. Há vários tipos de listas encadeadas:

- Em uma ***lista simplesmente encadeada***, cada item aponta para seu sucessor, como ilustrado na Figura 1.13.
- Em uma lista ***duplamente encadeada***, determinado item pode referenciar seu predecessor ou seu sucessor, como ilustrado na Figura 1.14.
- Em uma lista ***circularmente encadeada***, o último elemento da lista referencia o primeiro elemento em vez de referenciar nulo, como ilustrado na Figura 1.15.

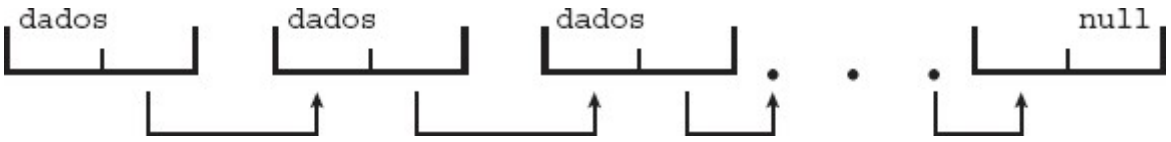

Figura 1.13 Lista simplesmente encadeada.

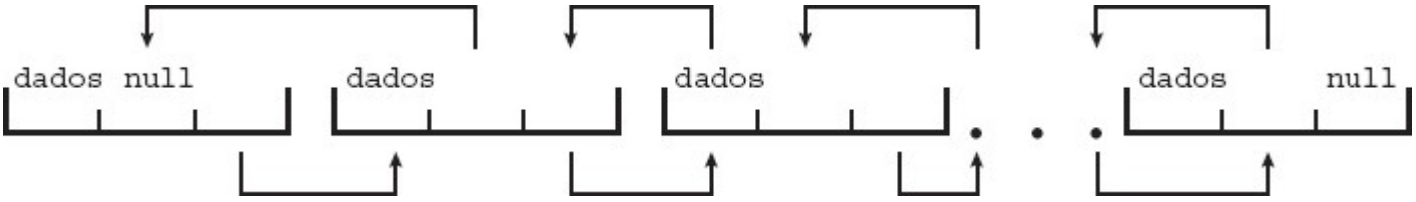

Figura 1.14 Lista duplamente encadeada.

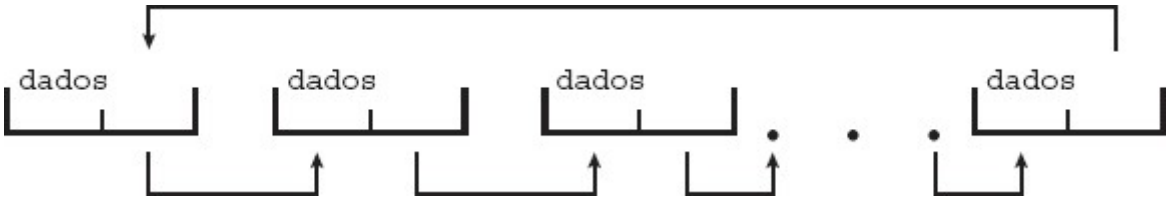

Figura 1.15 Lista circularmente encadeada.

As listas encadeadas acomodam itens de vários tamanhos e permitem a fácil inserção e exclusão de itens. Uma possível desvantagem do uso de uma lista é que o desempenho para recuperação de um item especificado em uma lista

de tamanho n é linear — $O(n)$ — já que, no pior caso, pode ser necessário percorrer todos os n elementos. Às vezes as listas são usadas diretamente por algoritmos do kernel. No entanto, com frequência, elas são utilizadas na construção de estruturas de dados mais poderosas, como as pilhas e filas.

Uma **pilha** é uma estrutura de dados ordenada sequencialmente que usa o princípio último a entrar, primeiro a sair (**LIFO — *last in, first out***) para adicionar e remover itens, ou seja, o último item alocado a uma pilha é o primeiro a ser removido. As operações de inserção de itens em uma pilha e de remoção de itens de uma pilha são conhecidas como ***inclusão* (*push*)** e ***extração* (*pop*)**, respectivamente. Com frequência, os sistemas operacionais usam uma pilha quando invocam chamadas de função. Parâmetros, variáveis locais e o endereço de retorno são incluídos na pilha quando uma função é chamada; o retorno da chamada da função extrai esses itens da pilha.

Uma **fila**, por outro lado, é uma estrutura de dados ordenada sequencialmente que usa o princípio primeiro a entrar, primeiro a sair (**FIFO — *first in, first out***): os itens são removidos de uma fila na ordem em que foram inseridos. Há muitos exemplos cotidianos de filas, incluindo fregueses esperando na fila do caixa em uma loja e carros aguardando enfileirados no semáforo. As filas também são muito comuns nos sistemas operacionais — jobs que são enviados para uma impressora são, normalmente, impressos na ordem em que foram submetidos, por exemplo. Como veremos no Capítulo 6, as tarefas que estão esperando para serem executadas em uma CPU disponível, geralmente são organizadas em filas.

1.10.2 Árvores

Uma **árvore** é uma estrutura de dados que pode ser usada para representar dados hierarquicamente. Os valores de dados de uma estrutura de árvore são vinculados por meio de relacionamentos pai-filho. Em uma **árvore genérica**, um pai pode ter um número ilimitado de filhos. Em uma **árvore binária**, o pai pode ter, no máximo, dois filhos que denominamos o ***filho esquerdo*** e o ***filho direito***. Uma **árvore de pesquisa binária** também requer uma ordem entre os dois filhos em que o *filho_esquerdo* < = *filho_direito*. A Figura 1.16 fornece um exemplo de uma árvore de pesquisa binária. Quando procuramos um item em uma árvore de pesquisa binária, o desempenho, no pior caso, é $O(n)$ (considere como isso pode ocorrer). Para remediar essa situação, podemos usar um algoritmo para criar uma **árvore de pesquisa binária balanceada**. Nesse caso, uma árvore contendo n itens tem, no máximo, $lg\ n$ níveis, assegurando, assim, que o desempenho, na pior hipótese, seja $O(lg\ n)$. Veremos, na Seção 6.7.1, que o Linux utiliza uma árvore de pesquisa binária balanceada como parte de seu algoritmo de scheduling da CPU.

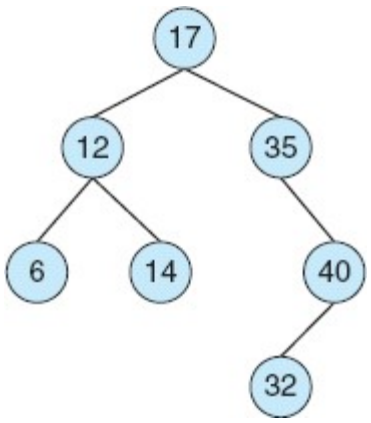

Figura 1.16 Árvore de pesquisa binária.

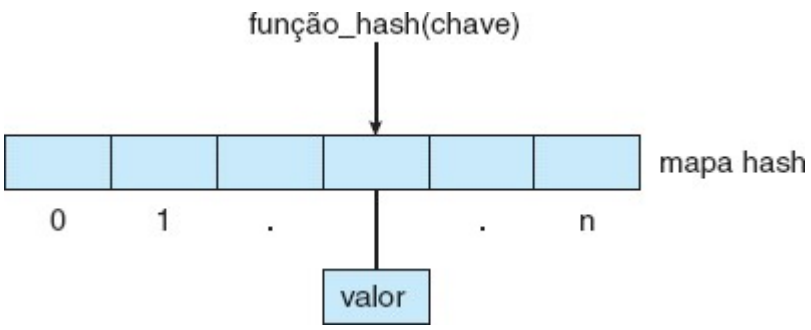

Figura 1.17 Mapa hash.

1.10.3 Funções e Mapas Hash

Uma **função hash** recebe dados como entrada, executa uma operação numérica com os dados e retorna um valor numérico. Esse valor numérico pode, então, ser usado como índice em uma tabela (tipicamente um array) para a recuperação rápida dos dados. Enquanto a procura por um item de dados em uma lista de tamanho *n* pode requerer até $O(n)$ comparações no pior caso, o uso de uma função hash para recuperar dados em uma tabela pode ter um resultado tão bom quanto $O(1)$ no pior caso, dependendo dos detalhes de implementação. Em razão desse desempenho, as funções hash são extensivamente utilizadas nos sistemas operacionais.

Uma dificuldade potencial encontrada nas funções hash é que duas entradas podem resultar no mesmo valor de saída — isto é, podem levar à mesma localização na tabela. Podemos resolver essa ***colisão de hash***, tendo uma lista encadeada nessa localização da tabela contendo todos os itens que tenham o mesmo valor hash. É claro que quanto mais colisões houver, menos eficiente a função hash será.

Um dos usos da função hash é na implementação de um **mapa hash**, que associa (ou ***mapeia***) pares [chave:valor] usando uma função hash. Por exemplo, podemos mapear a chave *operacional* ao valor *sistema*. Estabelecido o mapeamento, aplicamos a função hash à chave para obter o valor no mapa hash (Figura 1.17). Por exemplo, suponha que um nome de usuário seja mapeado para uma senha. A autenticação da senha ocorreria da seguinte forma: um usuário insere seu nome de usuário e senha. A função hash é aplicada ao nome de usuário que é, então, usado para recuperar a senha. Em seguida, a senha recuperada é comparada com a senha inserida pelo usuário para autenticação.

1.10.4 Mapas de Bits

Um **mapa de bits** é uma cadeia de *n* dígitos binários que pode ser usada para representar o *status* de *n* itens. Por exemplo, suponha que tenhamos diversos recursos, e a disponibilidade de cada recurso seja indicada pelo valor de um dígito binário: 0 significa que o recurso está disponível, e 1 indica que ele está indisponível (ou vice-versa). O valor da *i-ésima* posição no mapa de bits está associado ao *i-ésimo* recurso. Como exemplo, considere o mapa de bits mostrado a seguir:

001011101

Os recursos 2, 4, 5, 6 e 8 estão indisponíveis; os recursos 0, 1, 3 e 7 estão disponíveis

O poder dos mapas de bits torna-se evidente quando consideramos sua eficiência na utilização de espaço. Se quisermos usar um valor booleano de oito bits em vez de um único bit, a estrutura de dados resultante será oito vezes maior. Logo, os mapas de bits são, normalmente, usados quando há a necessidade de representar a disponibilidade de um grande número de recursos. Os drives de disco são um bom exemplo. Um drive de disco de tamanho médio pode ser dividido em vários milhares de unidades individuais, chamadas **blocos de disco**. Um mapa de bits pode ser usado para indicar a disponibilidade de cada bloco de disco.

As estruturas de dados estão disseminadas pelas implementações dos sistemas operacionais. Assim, veremos as estruturas discutidas aqui, além de outras, no decorrer deste texto, quando examinarmos algoritmos do kernel e suas implementações.

ESTRUTURAS DE DADOS DO KERNEL DO LINUX

As estruturas de dados usadas no kernel do Linux estão disponíveis no código-fonte do kernel. O arquivo include `<linux/list.h>` fornece detalhes da estrutura de dados lista encadeada usada em todo o kernel. Uma fila no Linux é conhecida como `kfifo` e sua implementação pode ser encontrada no arquivo `kfifo.c` do diretório `kernel` do código-fonte. O Linux também fornece uma implementação de árvore de pesquisa binária balanceada usando ***árvores rubro-negras***. Detalhes podem ser encontrados no arquivo include `<linux/rbtree.h>`.

1.11 Ambientes de Computação

Até aqui, descrevemos brevemente vários aspectos dos sistemas de computação e dos sistemas operacionais que os gerenciam. Passamos agora a uma discussão de como os sistemas operacionais são usados em vários ambientes de computação.

1.11.1 Computação Tradicional

À medida que a computação foi amadurecendo, as linhas que separavam muitos dos ambientes de computação tradicionais embaralharam-se. Considere o "ambiente de escritório típico". Há apenas alguns anos, esse ambiente era composto por PCs conectados a uma rede, com servidores fornecendo serviços de arquivo e impressão. O acesso remoto era complicado e a portabilidade era obtida com o uso de computadores laptop. Terminais conectados a mainframes também eram predominantes em muitas empresas, com ainda menos opções de acesso remoto e portabilidade.

A tendência atual dirige-se ao fornecimento de mais modalidades de acesso a esses ambientes de computação. Tecnologias web e a crescente largura de banda da WAN estão estendendo os limites da computação tradicional. As

empresas estabelecem **portais** que fornecem acessibilidade da web aos seus servidores internos. **Computadores em rede** (ou **clientes magros**) — que, essencialmente, são terminais que entendem a computação baseada na web — são usados no lugar das estações de trabalho tradicionais, em que mais segurança ou manutenção mais fácil é desejada. Computadores móveis podem se sincronizar com PCs para permitir o uso de informações empresariais de maneira bastante portátil. Os computadores móveis também podem se conectar a **redes sem fio** e redes de dados de celular para usar o portal web da empresa (assim como os vários outros recursos da web).

Em casa, a maioria dos usuários tinha um único computador com uma conexão de modem lenta com o escritório, a Internet, ou ambos. Hoje, as velocidades de conexão de rede, antes disponíveis apenas a um alto custo, são relativamente baratas em muitos lugares, dando aos usuários domésticos mais acesso a mais dados. Essas conexões de dados velozes estão permitindo que os computadores domésticos sirvam páginas web e executem redes que incluem impressoras, PCs clientes e servidores. Muitos lares usam **firewalls** para proteger suas redes de brechas na segurança.

Na segunda metade do século XX, os recursos de computação eram relativamente escassos. (Antes disso, nem existiam!) Durante algum tempo, os sistemas eram batch ou interativos. Os sistemas batch processavam jobs em massa, com entradas predeterminadas provenientes de arquivos ou outras fontes de dados. Os sistemas interativos esperavam entradas dos usuários. Para otimizar o uso dos recursos de computação, vários usuários compartilhavam o tempo desses sistemas. Os sistemas de tempo compartilhado usavam um timer e algoritmos de scheduling para revezar rapidamente os processos na CPU, dando a cada usuário uma parcela dos recursos.

Atualmente, os sistemas de tempo compartilhado tradicionais são raros. A mesma técnica de scheduling ainda é usada em computadores desktop, laptops, servidores e até em computadores móveis, mas, com frequência, todos os processos são de propriedade do mesmo usuário (ou de um único usuário e do sistema operacional). Processos de usuário e processos do sistema que fornecem serviços ao usuário são gerenciados de modo que cada um obtenha rapidamente uma parcela de tempo do computador. Considere as janelas criadas enquanto um usuário está trabalhando em um PC, por exemplo, e o fato de que elas podem estar executando diferentes tarefas ao mesmo tempo. Até mesmo um navegador da web pode conter vários processos, um para cada site sendo visitado, com o compartilhamento de tempo sendo aplicado a cada processo do navegador.

1.11.2 Computação Móvel

Computação móvel refere-se à computação em smartphones e tablets portáteis. Esses dispositivos compartilham as características físicas peculiares de portabilidade e leveza. Historicamente, em comparação aos computadores desktop e laptop, os sistemas móveis deixaram de lado o tamanho da tela, a capacidade de memória e a funcionalidade geral em troca do acesso móvel portátil a serviços como e-mail e navegação na web. Nos últimos anos, contudo, os recursos dos dispositivos móveis aumentaram tanto que a diferença de funcionalidades entre, digamos, um laptop e um tablet pode ser difícil de distinguir. Na verdade, podemos argumentar que os recursos de um dispositivo contemporâneo permitem que ele forneça funcionalidades indisponíveis ou impraticáveis em um computador desktop ou laptop.

Hoje, os sistemas móveis são usados não só para o envio de e-mails e a navegação na web, mas também para reproduzir música e vídeo, permitir a leitura de livros digitais, tirar fotos e gravar vídeo de alta definição. Da mesma forma, um enorme crescimento continua ocorrendo no amplo conjunto de aplicativos executados nesses dispositivos. Muitos desenvolvedores já estão projetando aplicativos que se beneficiam dos recursos exclusivos dos dispositivos móveis, como chips do sistema de posicionamento global (GPS — *global positioning system*), acelerômetros e giroscópios. Um chip de GPS embutido permite que um dispositivo móvel use satélites para determinar seu local preciso na Terra. Essa funcionalidade é particularmente útil no projeto de aplicativos que forneçam navegação — por exemplo, para informar aos usuários que caminho tomar a pé ou de carro ou talvez direcioná-los a serviços próximos, como restaurantes. Um acelerômetro permite que o dispositivo móvel detecte sua posição em relação ao chão e identifique algumas outras forças, como inclinação e vibração. Em vários jogos de computador que empregam acelerômetros, os jogadores não interagem com o sistema usando um mouse ou teclado, mas sim pela inclinação, rotação e vibração do dispositivo móvel! Talvez um uso mais prático desses recursos seja encontrado em aplicativos de ***realidade aumentada*** que sobrepõem informações em uma exibição do ambiente corrente. É difícil imaginar como aplicativos equivalentes poderiam ser desenvolvidos em sistemas de computação laptop ou desktop tradicionais.

Para dar acesso a serviços on-line, os dispositivos móveis usam, normalmente, redes sem fio padrão IEEE 802.11 ou redes de dados de celular. A capacidade de memória e a velocidade de processamento dos dispositivos móveis, no entanto, são mais limitadas do que as dos PCs. Enquanto um smartphone ou tablet pode ter 64 GB de memória, não é raro encontrarmos 1 TB de memória em um computador desktop. Da mesma forma, já que o consumo de energia é uma grande preocupação, os dispositivos móveis usam, com frequência, processadores que são menores, mais lentos e oferecem menos núcleos de processamento do que os processadores encontrados em computadores desktop e laptop tradicionais.

Atualmente, dois sistemas operacionais dominam a computação móvel: o **iOS** da **Apple** e o **Android** da **Google**. O iOS foi projetado para ser executado nos dispositivos móveis iPhone e iPad da Apple. O Android capacita os smartphones e tablets disponibilizados por muitos fabricantes. Examinamos esses dois sistemas operacionais móveis com mais detalhes no Capítulo 2.

1.11.3Sistemas Distribuídos

Um sistema distribuído é um conjunto de sistemas de computação fisicamente separados e possivelmente heterogêneos que são conectados em rede para conceder aos usuários acesso aos vários recursos que o sistema mantém. O acesso a um recurso compartilhado aumenta a velocidade de computação, a funcionalidade, a disponibilidade dos dados e a confiabilidade. Alguns sistemas operacionais generalizam o acesso à rede como um tipo de acesso a arquivo, com os detalhes da conexão de rede contidos no driver de dispositivo da interface de rede. Outros fazem os usuários invocarem funções da rede especificamente. Geralmente, os sistemas contêm uma combinação das duas modalidades — por exemplo, o FTP e o NFS. Os protocolos que criam um sistema distribuído podem afetar bastante a utilidade e a popularidade desse sistema.

Uma **rede**, em uma descrição simples, é uma via de comunicação entre dois ou mais sistemas. Os sistemas distribuídos dependem da conexão de rede para oferecer sua funcionalidade. As redes variam de acordo com os protocolos usados, as distâncias entre os nós e a mídia de transporte. O **TCP/IP** é o protocolo de rede mais comum e fornece a arquitetura básica da Internet. A maioria dos sistemas operacionais dá suporte ao TCP/IP, inclusive os de uso geral. Alguns sistemas suportam protocolos proprietários para satisfazer suas necessidades. Para um sistema operacional, um protocolo de rede precisa apenas de um dispositivo de interface — um adaptador de rede, por exemplo — com um driver de dispositivo para gerenciá-lo, assim como um software para manipular os dados. Esses conceitos são discutidos no decorrer deste livro.

As redes são caracterizadas de acordo com as distâncias entre seus nós. Uma **rede local (LAN)** conecta computadores dentro de uma sala, um prédio ou um campus. Uma **rede de longa distância (WAN)** geralmente conecta prédios, cidades ou países. Uma empresa global pode ter uma WAN para conectar seus escritórios mundialmente. Essas redes podem executar um protocolo ou vários. O contínuo surgimento de novas tecnologias traz novos tipos de rede. Por exemplo, uma **rede metropolitana (MAN)** pode conectar prédios dentro de uma cidade. Dispositivos BlueTooth e 802.11 usam tecnologia sem fio para se comunicar por uma distância de vários quilômetros, criando na verdade uma **rede de área pessoal (PAN)** entre um telefone e um fone de ouvido ou um smartphone e um computador desktop.

As mídias que suportam as redes são igualmente variadas. Elas incluem fios de cobre, fibras trançadas e transmissões sem fio entre satélites, parabólicas de ondas curtas e rádios. Quando dispositivos de computação são conectados a telefones celulares, eles criam uma rede. Até mesmo a comunicação em infravermelho de muito pouco alcance pode ser usada na conexão de rede. Em um nível rudimentar, sempre que os computadores se comunicam, eles usam ou criam uma rede. Essas redes também variam em seu desempenho e confiabilidade.

Alguns sistemas operacionais levaram o conceito de redes e sistemas distribuídos para além da noção de fornecimento de conectividade de rede. Um **sistema operacional de rede** é um sistema operacional que fornece recursos como o compartilhamento de arquivos pela rede e um esquema de comunicação que permite que diferentes processos em diferentes computadores troquem mensagens. Um computador executando um sistema operacional de rede atua independentemente de todos os outros computadores da rede, embora tenha conhecimento da rede e seja capaz de se comunicar com outros computadores conectados. Um sistema operacional distribuído fornece um ambiente menos autônomo. Os diferentes computadores se comunicam com proximidade suficiente para dar a impressão de que apenas um único sistema operacional controla a rede. Abordamos redes de computadores e sistemas distribuídos no Capítulo 17.

1.11.4 Computação Cliente-Servidor

Conforme os PCs ficavam mais velozes, poderosos e baratos, os projetistas iam abandonando a arquitetura de sistema centralizado. Atualmente os terminais conectados a sistemas centralizados estão sendo substituídos por PCs e dispositivos móveis. Da mesma forma, a funcionalidade de interface com o usuário, antes manipulada diretamente por sistemas centralizados, está cada vez mais sendo manipulada por PCs, quase sempre por uma interface web. Como resultado, muitos dos sistemas atuais agem como **sistemas servidores** para atender a solicitações geradas por **sistemas clientes**. Esse tipo de sistema distribuído especializado, chamado de sistema **cliente-servidor**, tem a estrutura geral mostrada na Figura 1.18.

Os sistemas servidores podem ser amplamente categorizados como servidores de computação e servidores de arquivo:

- O **sistema servidor de computação** fornece uma interface para a qual um cliente pode enviar uma solicitação para executar uma ação (por exemplo, ler dados). Em resposta, o servidor executa a ação e envia os resultados ao cliente. Um servidor executando um banco de dados que responde a solicitações de dados enviadas por clientes é um exemplo desse tipo de sistema.
- O **sistema servidor de arquivos** fornece uma interface com o sistema de arquivos em que os clientes podem criar, atualizar, ler e excluir arquivos. Um exemplo desse tipo de sistema é um servidor web que distribui arquivos a clientes que estejam executando navegadores da web.

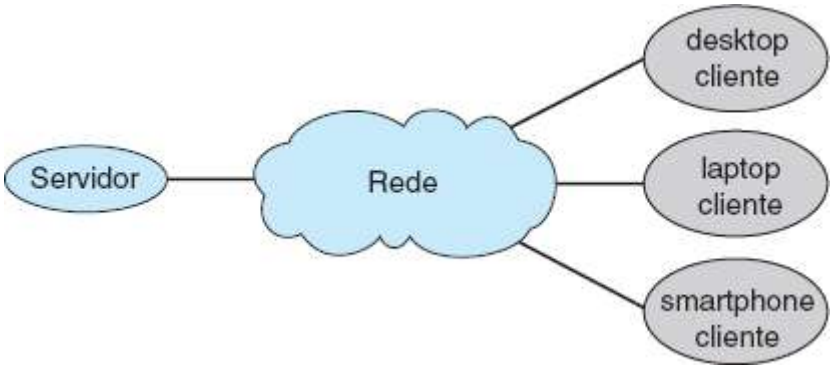

Figura 1.18 Estrutura geral de um sistema cliente-servidor.

1.11.5 Computação entre Pares (*Peer-to-Peer*)

Outra estrutura para um sistema distribuído é o modelo de sistema entre pares (P2P). Nesse modelo, clientes e servidores não são diferentes uns dos outros. Em vez disso, todos os nós do sistema são considerados pares, e cada um pode atuar como cliente ou servidor, dependendo de estar solicitando ou fornecendo um serviço. Os sistemas entre pares oferecem uma vantagem sobre os sistemas cliente-servidor tradicionais. Em um sistema cliente-servidor, o servidor é um gargalo; mas em um sistema entre pares, os serviços podem ser fornecidos por vários nós distribuídos ao longo da rede.

Para participar de um sistema entre pares, um nó deve primeiro conectar-se à rede de pares. Uma vez que o nó tenha se conectado à rede, pode começar a fornecer serviços para — e solicitar serviços de — outros nós da rede. A determinação dos serviços que estão disponíveis é feita de uma entre duas formas gerais:

- Quando um nó se conecta a uma rede, ele registra seu serviço em um serviço de pesquisa centralizado da rede. Qualquer nó que deseje um serviço específico primeiro faz contato com esse serviço de pesquisa centralizado para determinar qual nó fornece o serviço. O resto da comunicação ocorre entre o cliente e o fornecedor do serviço.
- Um esquema alternativo não usa serviço de pesquisa centralizado. Em vez disso, um par atuando como cliente deve descobrir qual nó fornece o serviço desejado transmitindo a solicitação do serviço para todos os outros nós da rede. O nó (ou nós) fornecedor do serviço responde ao par que fez a solicitação. Para suportar essa abordagem, um ***protocolo de descoberta*** deve ser fornecido para permitir que os pares descubram os serviços oferecidos por outros pares da rede. A Figura 1.19 ilustra esse cenário.

As redes entre pares ganharam popularidade no fim dos anos 1990 com vários serviços de compartilhamento de arquivos, como o Napster e o Gnutella que habilitavam pares a trocarem arquivos uns com os outros. O sistema Napster usava uma abordagem semelhante ao primeiro tipo descrito acima: um servidor centralizado mantinha um índice de todos os arquivos armazenados nos nós pares participantes da rede Napster, e a troca real de arquivos ocorria entre os nós pares. O sistema Gnutella usava uma técnica semelhante ao segundo tipo: um cliente transmitia solicitações de arquivo para outros nós do sistema, e os nós que podiam atender a solicitação respondiam diretamente ao cliente. O futuro da troca de arquivos permanece incerto porque as redes entre pares podem ser usadas para a troca anônima de materiais protegidos por direitos autorais (música, por exemplo), e há leis que controlam a distribuição de material com direitos autorais. De qualquer forma, o Napster teve problemas legais por infringir direitos autorais, e seus serviços deixaram de funcionar em 2001.

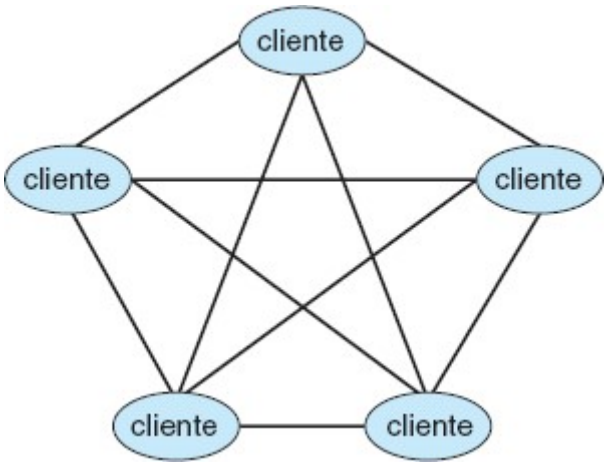

Figura 1.19 Sistema entre pares sem serviço centralizado.

O Skype é outro exemplo de computação entre pares. Ele permite que os clientes façam chamadas de voz e de vídeo e enviem mensagens de texto pela Internet usando uma tecnologia conhecida como **voz sobre IP** (**VoIP — *voice over IP***). O Skype usa uma abordagem entre pares híbrida. Ele inclui um servidor de login centralizado, mas também incorpora pares descentralizados e permite que dois pares se comuniquem.

1.11.6 Virtualização

A virtualização é uma tecnologia que permite que sistemas operacionais sejam executados como aplicações dentro de outros sistemas operacionais. À primeira vista, parece haver poucas razões para a existência dessa funcionalidade. Mas a indústria da virtualização é vasta e vem crescendo, o que prova sua utilidade e importância.

Falando de modo geral, a virtualização é membro de uma classe de softwares que também inclui a emulação. A **emulação** é usada quando o tipo da CPU de origem é diferente do tipo da CPU de destino. Por exemplo, quando a Apple mudou da CPU IBM Power para a Intel x86 em seus computadores desktop e laptop, ela incluiu um recurso de emulação chamado "Rosetta", que permitia aos aplicativos compilados para a CPU IBM serem executados na CPU Intel. Esse mesmo conceito pode ser estendido para permitir que um sistema operacional inteiramente escrito para uma plataforma seja executado em outra. A emulação custa caro, no entanto. Todas as instruções de nível de máquina que são executadas nativamente no sistema de origem devem ser traduzidas para a função equivalente no sistema de destino, o que, com frequência, resulta em várias instruções no sistema de destino. Se as CPUs de origem e de destino tiverem níveis de desempenho semelhantes, o código emulado pode ser executado muito mais lentamente do que o código nativo.

Um exemplo comum de emulação ocorre quando uma linguagem de computação não é compilada para código nativo; em vez disso, é executada em sua forma de alto nível ou traduzida para uma forma intermediária. Isso é conhecido como **interpretação**. Algumas linguagens, como o BASIC, podem ser compiladas ou interpretadas. Java, por outro lado, é sempre interpretada. A interpretação é uma forma de emulação em que o código da linguagem de alto nível é traduzido para instruções nativas da CPU, emulando não outra CPU, mas uma máquina virtual teórica em que essa linguagem poderia ser executada nativamente. Assim, podemos executar programas Java em "máquinas virtuais Java", mas tecnicamente essas máquinas virtuais são emuladores de Java.

Com a **virtualização**, por outro lado, um sistema operacional que é compilado nativamente para uma arquitetura de CPU específica é executado dentro de outro sistema operacional também nativo dessa CPU. A virtualização surgiu, primeiro, em mainframes IBM como um método para que múltiplos usuários executassem tarefas concorrentemente. A execução de múltiplas máquinas virtuais permitia (e ainda permite) que muitos usuários executassem tarefas em um sistema projetado para um único usuário. Posteriormente, em resposta a problemas com a execução de vários aplicativos Microsoft Windows XP na CPU Intel x86, a VMware criou uma nova tecnologia de virtualização na forma de um aplicativo que era executado no XP. Esse aplicativo executava uma ou mais cópias **convidadas** do Windows ou outros sistemas operacionais x86 nativos, cada uma executando seus próprios aplicativos. (Veja a Figura 1.20.) O Windows era o sistema operacional **hospedeiro**, e o aplicativo da VMware era VMM. A VMM executa os sistemas operacionais convidados, gerencia seu uso de recursos e protege os convidados uns dos outros.

Ainda que os sistemas operacionais modernos sejam totalmente capazes de executar múltiplos aplicativos de maneira confiável, o uso da virtualização continua a crescer. Em laptops e desktops, uma VMM permite que o usuário instale múltiplos sistemas operacionais para exploração ou execute aplicativos escritos para sistemas operacionais diferentes do hospedeiro nativo. Por exemplo, um laptop da Apple, executando o Mac OS X na CPU x86, pode executar um convidado Windows para permitir a execução de aplicativos Windows. Empresas que criam software para vários sistemas operacionais podem usar a virtualização para executar todos esses sistemas operacionais no mesmo servidor físico para desenvolvimento, teste e depuração. Dentro dos centros de dados, a virtualização tornou-se um método comum de execução e gerenciamento de ambientes de computação. VMMs como a VMware ESX e a Citrix XenServer não são mais executadas em sistemas operacionais hospedeiros; elas ***são*** os hospedeiros. Detalhes completos dos recursos e da implementação da virtualização são encontrados no Capítulo 16.

1.11.7 Computação em Nuvem

A **computação em nuvem** é um tipo de computação que distribui computação, armazenamento e até mesmo aplicativos como um serviço por uma rede. Em certos aspectos, é uma extensão lógica da virtualização porque a usa como base de sua funcionalidade. Por exemplo, o recurso Elastic Compute Cloud (**EC2**) da Amazon tem milhares de servidores, milhões de máquinas virtuais e petabytes de espaço de armazenamento disponíveis para uso por qualquer pessoa na Internet. Os usuários pagam por mês, dependendo de quanto fizeram uso desses recursos.

Existem atualmente muitos tipos de computação em nuvem, incluindo os descritos a seguir:

- **Nuvem pública** — uma nuvem disponível pela Internet para qualquer pessoa que queira pagar pelos serviços
- **Nuvem privada** — uma nuvem operada por uma empresa para uso próprio dessa empresa
- **Nuvem híbrida** — uma nuvem que inclui componentes de nuvem tanto pública quanto privada
- Software como serviço (**SaaS — *software as a service***) — um ou mais aplicativos (como processadores de texto ou planilhas) disponíveis pela Internet

•Plataforma como serviço (**PaaS — *platform as a service***) — uma pilha de software pronta para o uso de aplicativos pela Internet (por exemplo, um servidor de banco de dados)
•Infraestrutura como serviço (**IaaS — *infrastructure as a service***) — servidores ou espaço de armazenamento disponível pela Internet (por exemplo, espaço de armazenamento disponível para fazer cópias de backup de dados de produção)

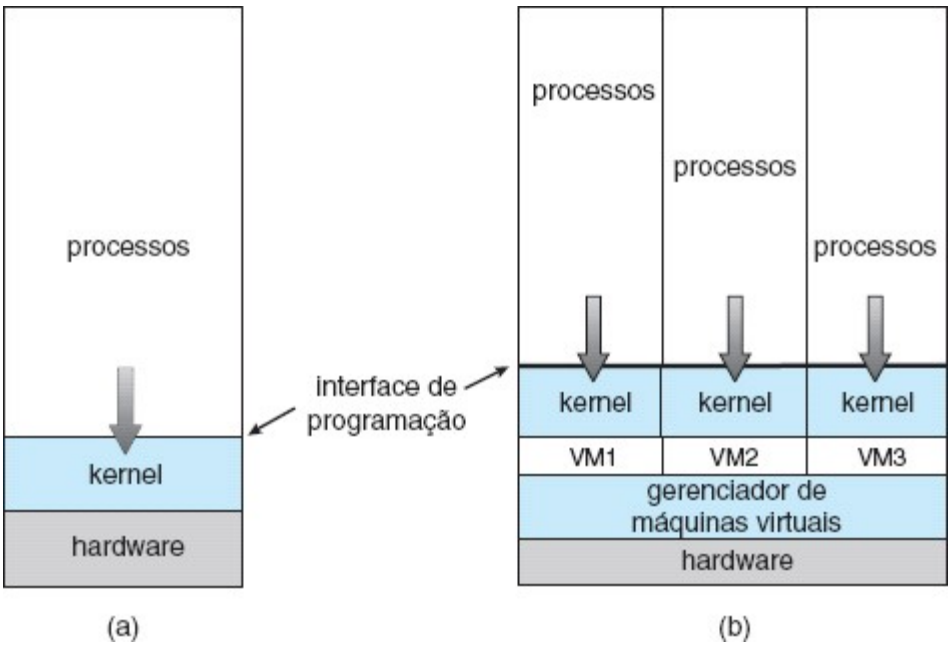

Figura 1.20 VMware.

Esses tipos de computação em nuvem não são exclusivos, já que um ambiente de computação em nuvem pode fornecer uma combinação de vários tipos. Por exemplo, uma empresa pode fornecer tanto o SaaS quanto o IaaS como um serviço disponível publicamente.

Certamente, existem sistemas operacionais tradicionais dentro de muitos dos tipos de infraestrutura de nuvem. Além deles, há VMMs que gerenciam as máquinas virtuais em que os processos de usuário são executados. Em um nível mais alto, as VMMs propriamente ditas são gerenciadas por ferramentas de gerenciamento em nuvem, como o Vware vCloud Director e o conjunto de ferramentas de código-fonte aberto Eucalyptus. Essas ferramentas gerenciam os recursos de uma nuvem específica e fornecem interfaces para os componentes da nuvem, compondo um bom argumento para que sejam consideradas um novo tipo de sistema operacional.

A Figura 1.21 ilustra uma nuvem pública fornecendo o IaaS. Observe que tanto os serviços da nuvem quanto sua interface de usuário são protegidos por um firewall.

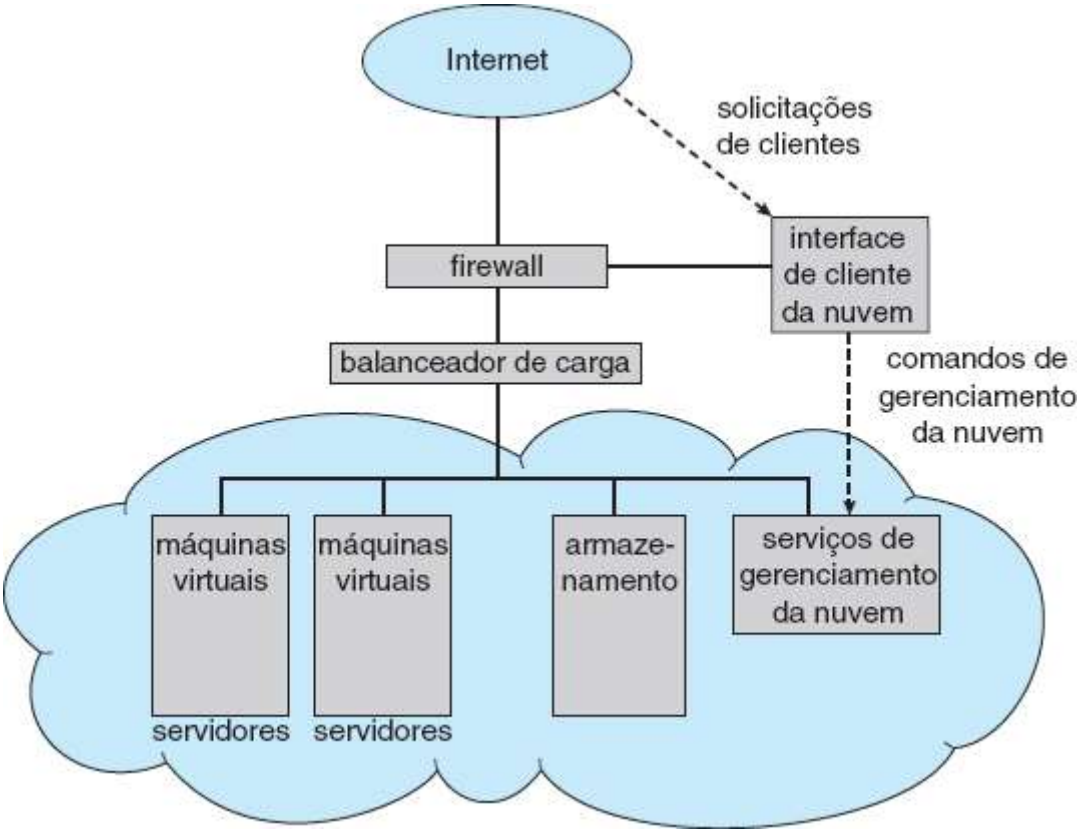

Figura 1.21 Computação em nuvem.

1.11.8 Sistemas Embutidos de Tempo Real

O computador embutido é, atualmente, o tipo de computador mais disseminado. Esses dispositivos são encontrados em todos os lugares, dos motores de carros e robôs industriais aos DVDs e fornos de micro-ondas. Eles tendem a possuir tarefas muito específicas. Os sistemas em que operam costumam ser primitivos e, portanto, os sistemas operacionais fornecem recursos limitados. Geralmente, têm pouca ou nenhuma interface com o usuário, preferindo ocupar seu tempo monitorando e gerenciando dispositivos de hardware, como motores de carro e braços robóticos.

Esses sistemas embutidos variam consideravelmente. Alguns são computadores de uso geral, executando sistemas operacionais padrão — como o Linux — com aplicações de uso específico para implementar a funcionalidade. Outros são dispositivos de hardware com um sistema operacional de uso específico embutido, fornecendo apenas a funcionalidade desejada. Existem ainda outros, que são dispositivos de hardware com circuitos integrados de aplicação específica (**ASICs — *aplication-specific integrated circuits***), que executam suas tarefas sem um sistema operacional.

O uso de sistemas embutidos continua a se expandir. O poder desses dispositivos, tanto como unidades autônomas quanto como elementos de redes e da web, também deve aumentar. Mesmo hoje em dia, residências inteiras podem ser computadorizadas, para que um computador central — um computador de uso geral ou um sistema embutido — possa controlar o aquecimento e a iluminação, sistemas de alarme e até cafeteiras. O acesso à web pode permitir que a proprietária de uma casa peça a ela para se aquecer antes de sua chegada. Algum dia, o refrigerador entrará em contato com o armazém quando notar que o leite acabou.

Os sistemas embutidos quase sempre executam **sistemas operacionais de tempo real**. Um sistema de tempo real é usado quando requisitos de tempo rígidos são exigidos da operação de um processador ou do fluxo de dados; portanto, ele é usado, geralmente, como um dispositivo de controle em uma aplicação dedicada. Sensores trazem dados para o computador. O computador deve analisar os dados e, possivelmente, ajustar controles para modificar as entradas do sensor. Sistemas que controlam experimentos científicos, sistemas médicos de imagens, sistemas de controle industrial e certos sistemas de exibição são sistemas de tempo real. Alguns sistemas de injeção de combustível em motores de automóveis, controladores de utensílios domésticos e sistemas bélicos também são sistemas de tempo real.

Um sistema de tempo real tem restrições de tempo fixas e bem definidas. O processamento ***deve*** ser executado respeitando as restrições definidas, ou o sistema falhará. Por exemplo, não adiantaria um braço robótico ser instruído a parar ***após*** ter golpeado o carro que estava construindo. Um sistema de tempo real só funciona corretamente se retorna o resultado correto dentro de suas restrições de tempo. Compare esse sistema com um sistema de tempo compartilhado em que é desejável (mas não obrigatória) uma resposta rápida, ou um sistema batch que pode não ter quaisquer restrições de tempo.

No Capítulo 6, consideramos o recurso de scheduling necessário à implementação da funcionalidade de tempo real em um sistema operacional. No Capítulo 9, descrevemos o projeto de gerenciamento de memória para a computação de

tempo real. Para concluir, nos Capítulos 18 e 19, descrevemos os componentes de tempo real dos sistemas operacionais Linux e Windows 7.

1.12 Sistemas Operacionais de Código-Fonte Aberto

Mencionamos, no começo deste capítulo, que o estudo dos sistemas operacionais foi facilitado pela disponibilidade de uma vasta quantidade de versões de código-fonte aberto. Os **sistemas operacionais de código-fonte aberto** são aqueles disponíveis em formato de código-fonte e não como código binário compilado. O Linux é o mais famoso sistema operacional de código-fonte aberto, enquanto o Microsoft Windows é um exemplo bem conhecido da abordagem oposta de **código-fonte fechado**. Os sistemas operacionais Mac OS X da Apple e o iOS compõem uma abordagem híbrida. Eles contêm um kernel de código-fonte aberto chamado Darwin, mas também incluem componentes proprietários de código-fonte fechado.

Partir do código-fonte permite que o programador produza código binário que possa ser executado em um sistema. Fazer o oposto — partir dos binários para o código-fonte pela **engenharia reversa** — é bem mais trabalhoso, e itens úteis, como os comentários, nunca são recuperados. O aprendizado dos sistemas operacionais por meio da investigação do código-fonte também tem outros benefícios. Com o código-fonte em mãos, um estudante pode modificar o sistema operacional e, em seguida, compilar e executar o código para testar essas alterações, o que é uma excelente ferramenta de aprendizado. Este texto inclui projetos que envolvem a modificação do código-fonte do sistema operacional, ao mesmo tempo em que descreve algoritmos em alto nível para assegurar que todos os tópicos importantes dos sistemas operacionais tenham sido abordados. No decorrer do texto, indicamos exemplos de código de fonte aberto para um estudo mais aprofundado.

Há muitos benefícios nos sistemas operacionais de código-fonte aberto, inclusive uma comunidade de programadores interessados (e geralmente trabalhando gratuitamente) que contribuem para o desenvolvimento do código, ajudando a depurá-lo, analisá-lo, fornecendo suporte e sugerindo alterações. É discutível se o código-fonte aberto é mais seguro do que o código-fonte fechado porque uma quantidade muito maior de pessoas o visualiza. Certamente o código-fonte aberto tem bugs, mas seus defensores argumentam que os bugs tendem a ser encontrados e corrigidos mais rapidamente, em razão do número de pessoas que usam e visualizam o código. Empresas que faturam com a venda de seus programas, hesitam, com frequência, em abrir seu código-fonte, mas a Red Hat e várias outras empresas estão fazendo exatamente isso e mostrando que empresas comerciais se beneficiam, em vez de serem prejudicadas, quando abrem seu código-fonte. A receita pode ser gerada por meio de contratos de suporte e da venda do hardware em que o software é executado, por exemplo.

1.12.1 História

Nos primórdios da computação moderna (isto é, nos anos 1950), havia muitos softwares disponíveis no formato de código-fonte aberto. Os hackers originais (entusiastas da computação) do Tech Model Railroad Club do MIT deixavam seus programas em gavetas para outras pessoas darem continuidade. Grupos de usuários "da casa" trocavam códigos durante suas reuniões. Posteriormente, grupos de usuários específicos de empresas, como o grupo DEC da Digital Equipment Corporation, passaram a aceitar contribuições de programas em código-fonte, reunindo-as em fitas e distribuindo as fitas a membros interessados.

Fabricantes de computadores e softwares eventualmente tentavam limitar o uso de seu software a computadores autorizados e clientes pagantes. Liberar apenas os arquivos binários compilados a partir do código-fonte, em vez do próprio código-fonte, ajudava-os a atingir esse objetivo assim como a proteger seu código e suas ideias contra seus concorrentes. Outro problema envolvia materiais protegidos por direitos autorais. Sistemas operacionais e outros programas podem limitar, a computadores autorizados, a capacidade de reprodução de filmes e música ou a exibição de livros eletrônicos. Essa **proteção contra cópia** ou **gerenciamento de direitos digitais** (**DRM** — ***digital rights management***) não seria eficaz, se o código-fonte que implementava esses limites fosse publicado. Leis de muitos países, inclusive o U.S. Digital Millenium Copyright Act (DMCA), tornam ilegal usar engenharia reversa em código protegido pelo DRM ou tentar burlar a proteção contra cópia.

Para contra-atacar a tentativa de limitar o uso e redistribuição de software, Richard Stallman, em 1983, iniciou o projeto GNU para criar um sistema operacional compatível com o UNIX, livre e de código-fonte aberto. Em 1985, ele publicou o Manifesto GNU, que argumenta que qualquer software deve ser livre e de código-fonte aberto. Também formou a **Fundação de Software Livre** (**FSF** — ***Free Software Foundation***) com o objetivo de encorajar a livre troca de códigos-fonte de softwares e o livre uso desses softwares. Em vez de patentear seu software, a FSF faz o "copyleft" do software para encorajar o compartilhamento e aperfeiçoamento. A **Licença Pública Geral do GNU** (**GPL** — ***GNU General Public License***) sistematiza o copyleft e é uma licença comum sob a qual softwares livres são lançados. Basicamente, a GPL requer que o código-fonte seja distribuído com todos os binários e que qualquer alteração nele executada seja liberada sob a mesma licença GPL.

1.12.2 Linux

Como exemplo de sistema operacional de código-fonte aberto, considere o **GNU/Linux**. O projeto GNU produziu muitas ferramentas compatíveis com o UNIX, incluindo compiladores, editores e utilitários, mas nunca lançou um kernel. Em 1991, um estudante da Finlândia, Linus Torvalds, lançou um kernel rudimentar, baseado em UNIX, usando os compiladores e ferramentas do GNU, e solicitou contribuições no mundo todo. O advento da Internet significou que qualquer pessoa interessada poderia baixar o código-fonte, modificá-lo e enviar alterações para Torvalds. O lançamento de atualizações uma vez por semana permitiu que o assim chamado sistema operacional Linux crescesse rapidamente, aperfeiçoado por muitos milhares de programadores.

O sistema operacional GNU/Linux resultante gerou centenas de **distribuições** exclusivas, ou construções personalizadas, do sistema. As principais distribuições incluem a RedHat, SUSE, Fedora, Debian, Slackware e Ubuntu. As distribuições variam em função, utilidade, aplicações instaladas, suporte de hardware, interface de usuário e finalidade. Por exemplo, o RedHat Enterprise Linux foi preparado para amplo uso comercial. O PCLinuxOS é um **LiveCD** — um sistema operacional que pode ser inicializado e executado a partir de um CD-ROM sem ser instalado no disco rígido de um sistema. Uma variação do PCLinuxOS — chamada "PCLinuxOS Supergamer DVD" — é um **LiveDVD** que inclui drivers e jogos gráficos. Um jogador pode executá-lo em qualquer sistema compatível, simplesmente inicializando-o a partir do DVD. Quando o jogador termina, uma reinicialização do sistema o reajusta para o seu sistema operacional instalado.

Você pode executar o Linux em um sistema Windows usando a abordagem simples e gratuita a seguir:

1. Baixe a ferramenta gratuita "VMware Player" a partir de

   http://www.vmware.com/download/player/

   e instale-a em seu sistema.
2. Selecione uma versão do Linux entre as centenas de "mecanismos", ou imagens de máquina virtual, disponíveis na VMware em

   http://www.vmware.com/appliances/

   Essas imagens são pré-instaladas com os sistemas operacionais e aplicações e incluem muitas versões do Linux.
3. Inicialize a máquina virtual dentro do VMware Player.

Dentro deste texto, fornecemos uma imagem de máquina virtual do Linux executando a versão Debian. Essa imagem contém o código-fonte do Linux assim como ferramentas para o desenvolvimento de softwares. Abordamos exemplos envolvendo essa imagem do Linux no decorrer do texto assim como em um estudo de caso detalhado no Capítulo 18.

1.12.3 BSD UNIX

O **BSD UNIX** tem uma história mais longa e complicada do que o Linux. Ele surgiu em 1978 como um derivado do UNIX da AT&T. Versões da Universidade da Califórnia em Berkeley (UCB) vinham em forma de código binário e código-fonte, mas não eram de código-fonte aberto porque era requerida uma licença da AT&T. O desenvolvimento do BSD UNIX foi retardado por uma ação judicial da AT&T, mas uma versão de código-fonte aberto totalmente funcional, a 4.4BSD-lite, foi liberada em 1994.

Como ocorre com o Linux, há muitas distribuições do BSD UNIX que incluem o FreeBSD, o NetBSD, o OpenBSD e o DragonflyBSD. Para explorar o código-fonte do FreeBSD, simplesmente baixe a imagem de máquina virtual da versão de interesse e a inicialize dentro do VMware, como descrito acima para o Linux. O código-fonte vem com a distribuição e é armazenado em `/usr/src/`. O código-fonte do kernel fica em `/usr/src/sys`. Por exemplo, para examinar o código de implementação da memória virtual do kernel do FreeBSD, consulte os arquivos em `/usr/src/sys/vm`.

O Darwin, o componente nuclear do kernel do MAC OS X, é baseado no BSD UNIX e também é de código-fonte aberto. O código-fonte está disponível em http://www.opensource.apple.com/. Toda versão do MAC OS X tem seus componentes de código-fonte aberto postados nesse site. O nome do pacote que contém o kernel começa com "xnu". A Apple também fornece várias ferramentas de desenvolvedor, documentação e suporte em http://connect.apple.com. Para mais informações, consulte o Apêndice A.

1.12.4 Solaris

Solaris é o sistema operacional comercial da Sun Microsystems baseado em UNIX. Originalmente, o sistema operacional **SunOS** da Sun era baseado no BSD UNIX. A Sun migrou para o System V UNIX da AT&T, como sua base, em 1991. Em 2005, a Sun abriu a fonte da maior parte do código do Solaris no projeto OpenSolaris. A compra da Sun pela Oracle em 2009, no entanto, deixou incerto o estado desse projeto. O código-fonte, como se encontrava em 2005, ainda está disponível via um navegador de código-fonte e para download em http://src.opensolaris.org/source.

Vários grupos interessados no uso do OpenSolaris partiram dessa base e expandiram seus recursos. O conjunto de seu trabalho é o Projeto Illumos que se expandiu a partir da base fornecida pelo OpenSolaris, para incluir mais recursos e constituir a base de vários produtos. O Illumos está disponível em http://wiki.illumos.org.

1.12.5 Sistemas de Código-Fonte Aberto como Ferramentas de Aprendizado

O movimento do software livre está levando legiões de programadores a criarem milhares de projetos de código-fonte aberto, inclusive sistemas operacionais. Sites como http://freshmeat.net/ e http://distrowatch.com/ fornecem portais para muitos desses projetos. Como mencionado anteriormente, os projetos de código-fonte aberto permitem que os estudantes usem o código-fonte como ferramenta de aprendizado. Eles podem modificar programas e testá-los, ajudar a encontrar e corrigir bugs, ou então explorar sistemas operacionais totalmente desenvolvidos e maduros, compiladores, ferramentas, interfaces de usuário e outros tipos de programa. A disponibilidade do código-fonte de projetos históricos, como o Multics, pode ajudar os estudantes a entenderem esses projetos e a construírem conhecimento, o que auxiliará na implementação de novos projetos.

O GNU/Linux e o BSD UNIX são sistemas operacionais de código-fonte aberto, mas cada um tem seus próprios objetivos, utilidade, licenciamento e finalidade. Às vezes, as licenças não são mutuamente exclusivas e ocorre uma pulverização, permitindo melhorias rápidas nos projetos dos sistemas operacionais. Por exemplo, vários dos componentes principais do OpenSolaris têm sido portados para o BSD UNIX. As vantagens do software livre e do código-fonte aberto devem aumentar a quantidade e a qualidade de projetos de código-fonte aberto, levando a um acréscimo no número de indivíduos e empresas que usam esses projetos.

1.13 Resumo

Um sistema operacional é um software que gerencia o hardware do computador, bem como fornece um ambiente para programas aplicativos serem executados. Talvez o aspecto mais visível de um sistema operacional seja a interface com o sistema de computação que ele fornece ao usuário humano.

Para que um computador realize seu trabalho de execução de programas, os programas devem estar na memória principal. A memória principal é a única grande área de armazenamento que o processador pode acessar diretamente. Ela é um array de bytes, variando em tamanho de milhões a bilhões. Cada byte na memória tem seu próprio endereço. Normalmente, a memória principal é um dispositivo de armazenamento volátil que perde seu conteúdo quando a energia é desligada ou perdida. A maioria dos sistemas de computação fornece memória secundária como uma extensão da memória principal. A memória secundária fornece um tipo de armazenamento não volátil que pode manter grandes quantidades de dados permanentemente. O dispositivo de memória secundária mais comum é o disco magnético, que fornece armazenamento tanto de programas quanto de dados.

A grande variedade de sistemas de armazenamento em um sistema de computação pode ser organizada em uma hierarquia de acordo com a velocidade e o custo. Os níveis mais altos são caros, mas são velozes. Conforme descemos na hierarquia, em geral o custo por bit diminui, enquanto o tempo de acesso aumenta.

O ESTUDO DOS SISTEMAS OPERACIONAIS

Nunca houve uma época tão interessante para estudar os sistemas operacionais, e nunca foi tão fácil. O movimento do código-fonte aberto tomou conta dos sistemas operacionais, fazendo com que muitos deles sejam disponibilizados tanto no formato fonte quanto no formato binário (executável). Essa lista de sistemas operacionais disponíveis nos dois formatos inclui o Linux, o BSD UNIX, o Solaris e parte do Mac OS X. A disponibilidade do código-fonte permite-nos estudar os sistemas operacionais de dentro para fora. Perguntas que antes só podiam ser respondidas pela verificação da documentação ou do comportamento de um sistema operacional já podem ser respondidas pela verificação do próprio código.

Sistemas operacionais que não são mais comercialmente viáveis também passaram a ter o código-fonte aberto, habilitando-nos a estudar como os sistemas operavam em uma época de menos recursos de CPU, memória e armazenamento. Uma extensa, porém incompleta, lista de projetos de sistemas operacionais de código-fonte aberto está disponível em http://dmoz.org/Computers/Software/Operating_Systems/Open_Source/.

Além disso, o surgimento da virtualização como uma função de computação popular (e frequentemente gratuita) torna possível a execução de vários sistemas operacionais no topo de um sistema core. Por exemplo, a VMware (http://www.vmware.com) fornece um "player" gratuito para Windows, em que centenas de "aplicações virtuais" livres podem ser executadas. A Virtualbox (http://www.virtualbox.com) fornece um gerenciador de máquinas virtuais, gratuito e de código-fonte aberto, em muitos sistemas operacionais. Usando essas ferramentas, os estudantes podem testar centenas de sistemas operacionais sem hardware dedicado.

Em alguns casos, simuladores de hardware específico também estão disponíveis, permitindo que o sistema operacional seja executado em hardware "nativo", tudo nos limites de um computador e de um sistema operacional modernos. Por exemplo, um simulador DECSYSTEM-20 sendo executado no Mac OS X pode inicializar o TOPS-20, carregar as fitas de código-fonte e modificar e compilar um novo kernel TOPS-20. Um estudante interessado pode pesquisar na Internet os artigos originais que descrevem o sistema operacional, bem como os manuais originais.

O advento dos sistemas operacionais de código-fonte aberto também torna fácil fazer a transição de estudante para desenvolvedor de sistema operacional. Com algum conhecimento, algum esforço e uma conexão com a Internet, um estudante pode até mesmo criar uma nova distribuição de sistema operacional. Há apenas alguns anos, era difícil ou impossível obter acesso ao código-fonte. Agora, esse acesso é limitado apenas pelo quanto de interesse, tempo e espaço em disco o estudante tem.

Há várias estratégias diferentes para o projeto de um sistema de computação. Sistemas uniprocessadores têm apenas um processador, enquanto sistemas multiprocessadores contêm dois ou mais processadores que compartilham memória física e dispositivos periféricos. O projeto multiprocessador mais comum é o multiprocessamento simétrico (ou SMP), em que todos os processadores são considerados pares e são executados independentemente uns dos outros. Sistemas agrupados (clusters) são um tipo especializado de sistemas multiprocessadores e são compostos por vários sistemas de computação conectados por uma rede local.

Para melhor utilizar a CPU, os sistemas operacionais modernos empregam a multiprogramação que permite a vários jobs ficarem na memória ao mesmo tempo, assegurando, assim, que a CPU sempre tenha um job para executar. Os sistemas de tempo compartilhado são uma extensão da multiprogramação em que algoritmos de scheduling da CPU alternam-se rapidamente entre os jobs, dando a impressão de que cada job está sendo executado concorrentemente.

O sistema operacional deve assegurar a operação correta do sistema de computação. Para impedir que programas de usuário interfiram na operação apropriada do sistema, o hardware possui duas modalidades: modalidade de usuário e modalidade de kernel. Várias instruções (como as instruções de I/O e de interrupção) são privilegiadas e podem ser executadas apenas em modalidade de kernel. A memória em que o sistema operacional reside também deve ser protegida de modificações feitas pelo usuário. Um timer impede loops infinitos. Esses recursos (modalidade dual, instruções privilegiadas, proteção da memória e interrupção por timer) são blocos de construção básicos usados pelos sistemas operacionais para alcançarem a operação correta.

Um processo (ou job) é a unidade básica de trabalho de um sistema operacional. O gerenciamento de processos inclui a criação e exclusão de processos e o fornecimento de mecanismos para a comunicação e sincronização entre os processos. Um sistema operacional gerencia a memória controlando que partes dela estão sendo usadas e por quem. Além disso, o sistema operacional é responsável pela alocação e liberação dinâmica de espaço na memória. O espaço de armazenamento também é gerenciado pelo sistema operacional; isso inclui o fornecimento de sistemas de arquivos para a representação de arquivos e diretórios e o gerenciamento de espaço em dispositivos de armazenamento de massa.

Os sistemas operacionais também devem se preocupar com a proteção e segurança sua e dos usuários. As medidas de proteção controlam o acesso de processos ou usuários aos recursos disponibilizados pelo sistema de computação. As medidas de segurança são responsáveis pela defesa de um sistema de computação contra ataques externos ou internos.

Várias estruturas de dados, fundamentais para a ciência da computação, são amplamente usadas nos sistemas operacionais, incluindo listas, pilhas, filas, árvores, funções hash, mapas e mapas de bits.

A computação ocorre em vários ambientes. A computação tradicional envolve PCs desktop e laptop, geralmente conectados a uma rede de computadores. A computação móvel é a computação dos smartphones portáteis e tablets que oferecem muitos recursos exclusivos. Os sistemas distribuídos permitem que os usuários compartilhem recursos em hospedeiros geograficamente dispersos conectados por uma rede de computadores. Serviços podem ser fornecidos pelo modelo cliente-servidor ou do modelo entre pares. A virtualização envolve a abstração de um hardware de computador em vários ambientes de execução diferentes. A computação em nuvem usa um sistema distribuído para incluir serviços em uma “nuvem”, em que os usuários podem acessá-los de locais remotos. Sistemas operacionais de tempo real são projetados para ambientes embutidos, como dispositivos de consumidores, automóveis e robótica.

O movimento do software livre criou milhares de projetos de código-fonte aberto, inclusive sistemas operacionais. Graças a esses projetos, os estudantes podem usar o código-fonte como ferramenta de aprendizado. Eles podem modificar programas e testá-los, ajudar a encontrar e corrigir bugs, ou então explorar sistemas operacionais maduros e completos, compiladores, ferramentas, interfaces de usuário e outros tipos de programas.

O GNU/Linux e o BSD UNIX são sistemas operacionais de código-fonte aberto. As vantagens do software livre e do código-fonte aberto devem aumentar a quantidade e a qualidade de projetos de código-fonte aberto, levando a um acréscimo no número de indivíduos e empresas que usam esses projetos.