

Sumário

PARTE UM ■ VISÃO GERAL

Capítulo 1 Introdução

1.1 O que Fazem os Sistemas Operacionais	3	1.9 Proteção e Segurança	18
1.2 Organização do Sistema de Computação	5	1.10 Sistemas Distribuídos	18
1.3 Arquitetura do Sistema de Computação	9	1.11 Sistemas de Uso Específico	19
1.4 Estrutura do Sistema Operacional	12	1.12 Ambientes de Computação	20
1.5 Operações do Sistema Operacional	13	1.13 Sistemas Operacionais de Código-fonte Aberto	22
1.6 Gerenciamento de Processos	15	1.14 Resumo	24
1.7 Gerenciamento da Memória	15	Exercícios	25
1.8 Gerenciamento do Armazenamento	16	Notas Bibliográficas	26

Capítulo 2 Estruturas do Sistema Operacional

2.1 Serviços do Sistema Operacional	27	2.7 Estrutura do Sistema Operacional	39
2.2 Interface entre o Usuário e o Sistema Operacional	28	2.8 Máquinas Virtuais	42
2.3 Chamadas de Sistema	30	2.9 Depuração do Sistema Operacional	46
2.4 Tipos de Chamada de Sistema	32	2.10 Geração do Sistema Operacional	48
2.5 Programas de Sistema	37	2.11 Inicialização do Sistema	49
2.6 Projeto e Implementação do Sistema Operacional	37	2.12 Resumo	50
		Exercícios	50
		Notas Bibliográficas	53

PARTE DOIS ■ GERENCIAMENTO DE PROCESSOS

Capítulo 3 Processos

3.1 Conceito de Processo	57	3.6 Comunicação em Sistemas Cliente-servidor	72
3.2 Scheduling de Processos	59	3.7 Resumo	78
3.3 Operações sobre Processos	62	Exercícios	79
3.4 Comunicação entre Processos	66	Notas Bibliográficas	84
3.5 Exemplos de Sistemas IPC	69		

Capítulo 4 Threads

4.1 Visão Geral	85	4.5 Exemplos de Sistemas Operacionais	95
4.2 Modelos de Geração de Multithreads	87	4.6 Resumo	96
4.3 Bibliotecas de Threads	88	Exercícios	97
4.4 Questões Relacionadas à Criação de Threads	92	Notas Bibliográficas	100

Capítulo 5 Scheduling da CPU

- 5.1 Conceitos Básicos 101
- 5.2 Critérios de Scheduling 103
- 5.3 Algoritmos de Scheduling 103
- 5.4 Scheduling de Threads 108
- 5.5 Scheduling com Multiprocessadores 109

- 5.6 Exemplos de Sistemas Operacionais 112
- 5.7 Avaliação de Algoritmos 115
- 5.8 Resumo 118
- Exercícios 119
- Notas Bibliográficas 120

Capítulo 6 Sincronização de Processos

- 6.1 Antecedentes 121
- 6.2 O Problema da Seção Crítica 122
- 6.3 Solução de Peterson 123
- 6.4 Hardware de Sincronização 124
- 6.5 Semáforos 125
- 6.6 Problemas Clássicos de Sincronização 128

- 6.7 Monitores 130
- 6.8 Exemplos de Sincronização 134
- 6.9 Transações Atômicas 136
- 6.10 Resumo 140
- Exercícios 141
- Notas Bibliográficas 146

Capítulo 7 Deadlocks

- 7.1 Modelo de Sistema 148
- 7.2 Caracterização do Deadlock 149
- 7.3 Métodos para Manipulação de Deadlocks 151
- 7.4 Prevenção de Deadlocks 151
- 7.5 Impedimento de Deadlocks 153

- 7.6 Detecção de Deadlocks 156
- 7.7 Recuperação de Deadlocks 158
- 7.8 Resumo 158
- Exercícios 160
- Notas Bibliográficas 161

PARTE TRÊS ■ GERENCIAMENTO DA MEMÓRIA

Capítulo 8 Memória Principal

- 8.1 Antecedentes 165
- 8.2 Permuta entre Processos (Swapping) 168
- 8.3 Alocação de Memória Contígua 170
- 8.4 Paginação 172
- 8.5 Estrutura da Tabela de Páginas 177

- 8.6 Segmentação 180
- 8.7 Exemplo: O Pentium da Intel 181
- 8.8 Resumo 184
- Exercícios 185
- Notas Bibliográficas 186

Capítulo 9 Memória Virtual

- 9.1 Antecedentes 187
- 9.2 Paginação por Demanda 189
- 9.3 Cópia-após-gravação 192
- 9.4 Substituição de Páginas 193
- 9.5 Alocação de Quadros 200
- 9.6 Atividade Improdutiva 202
- 9.7 Arquivos Mapeados em Memória 205

- 9.8 Alocando a Memória do Kernel 208
- 9.9 Outras Considerações 210
- 9.10 Exemplos de Sistemas Operacionais 213
- 9.11 Resumo 214
- Exercícios 216
- Notas Bibliográficas 218

PARTE QUATRO ■ GERENCIAMENTO DE ARMAZENAMENTO

Capítulo 10 Interface do Sistema de Arquivos

- 10.1 Conceito de Arquivo 221
- 10.2 Métodos de Acesso 226
- 10.3 Estrutura de Diretório e Disco 228
- 10.4 Montagem do Sistema de Arquivos 233
- 10.5 Compartilhamento de Arquivos 234

- 10.6 Proteção 237
- 10.7 Resumo 240
- Exercícios 241
- Notas Bibliográficas 241

Capítulo 11 Implementação do Sistema de Arquivos

11.1 Estrutura do Sistema de Arquivos	242	11.7 Recuperação	255
11.2 Implementação do Sistema de Arquivos	243	11.8 NFS	257
11.3 Implementação do Diretório	247	11.9 Exemplo: O Sistema de Arquivos WAFL	261
11.4 Métodos de Alocação	247	11.10 Resumo	262
11.5 Gerenciamento do Espaço Livre	252	Exercícios	263
11.6 Eficiência e Desempenho	253	Notas Bibliográficas	264

Capítulo 12 Estrutura de Armazenamento de Massa

12.1 Visão Geral da Estrutura de Armazenamento de Massa	265	12.7 Estrutura RAID	274
12.2 Estrutura do Disco	266	12.8 Implementação de Espaço de Armazenamento Estável	280
12.3 Conexão do Disco	267	12.9 Estrutura de Armazenamento Terciário	280
12.4 Scheduling de Disco	268	12.10 Resumo	286
12.5 Gerenciamento de Disco	271	Exercícios	288
12.6 Gerenciamento do Espaço de Permuta	273	Notas Bibliográficas	290

Capítulo 13 Sistemas de I/O

13.1 Visão Geral	291	13.6 STREAMS	306
13.2 Hardware de I/O	291	13.7 Desempenho	307
13.3 Interface de I/O da Aplicação	297	13.8 Resumo	309
13.4 Subsistema de I/O do Kernel	300	Exercícios	309
13.5 Transformando Solicitações de I/O em Operações de Hardware	304	Notas Bibliográficas	310

PARTE CINCO ■ PROTEÇÃO E SEGURANÇA

Capítulo 14 Proteção

14.1 Objetivos da Proteção	313	14.7 Revogação dos Direitos de Acesso	321
14.2 Princípios de Proteção	314	14.8 Sistemas Baseados em Competências	322
14.3 Domínio de Proteção	314	14.9 Proteção Baseada em Linguagens	323
14.4 Matriz de Acesso	317	14.10 Resumo	326
14.5 Implementação da Matriz de Acesso	319	Exercícios	327
14.6 Controle de Acesso	320	Notas Bibliográficas	328

Capítulo 15 Segurança

15.1 O Problema da Segurança	329	15.7 Utilizando um Firewall para Proteger Sistemas e Redes	350
15.2 Ameaças a Programas	331	15.8 Classificações de Segurança de Computadores	351
15.3 Ameaças a Sistemas e à Rede	335	15.9 Um Exemplo: Windows XP	352
15.4 Criptografia como Ferramenta de Segurança	338	15.10 Resumo	353
15.5 Autenticação do Usuário	344	Exercícios	353
15.6 Implementando Defesas de Segurança	346	Notas Bibliográficas	354

PARTE SEIS ■ SISTEMAS DISTRIBUÍDOS

Capítulo 16 Estruturas de Sistemas Distribuídos

16.1 Motivação	357	16.7 Robustez	368
16.2 Tipos de Sistemas Operacionais Baseados em Redes	358	16.8 Aspectos de Projeto	369
16.3 Estrutura de Rede	360	16.9 Um Exemplo: Conexão em Rede	370
16.4 Topologia da Rede	362	16.10 Resumo	371
16.5 Estrutura de Comunicação	363	Exercícios	372
16.6 Protocolos de Comunicação	366	Notas Bibliográficas	372

Capítulo 17 Sistemas de Arquivos Distribuídos

17.1 Antecedentes	373	17.5 Replicação de Arquivos	379
17.2 Nomeação e Transparência	374	17.6 Um Exemplo: AFS	380
17.3 Acesso a Arquivos Remotos	376	17.7 Resumo	383
17.4 Serviço com Estado versus Serviço sem Estado	379	Exercícios	383
		Notas Bibliográficas	384

Capítulo 18 Coordenação Distribuída

18.1 Ordenação de Eventos	385	18.6 Algoritmos de Eleição	395
18.2 Exclusão Mútua	386	18.7 Chegando a Acordos	396
18.3 Atomicidade	388	18.8 Resumo	397
18.4 Controle de Concorrência	389	Exercícios	398
18.5 Manipulação de Deadlocks	391	Notas Bibliográficas	398

PARTE SETE ■ SISTEMAS DE USO ESPECÍFICO

Capítulo 19 Sistemas de Tempo Real

19.1 Visão Geral	403	19.5 Scheduling de CPU em Tempo Real	408
19.2 Características dos Sistemas	404	19.6 Um Exemplo: VxWorks 5.x	411
19.3 Recursos dos Kernels de Tempo Real	405	19.7 Resumo	412
19.4 Implementando Sistemas Operacionais de Tempo Real	406	Exercícios	413
		Notas Bibliográficas	413

Capítulo 20 Sistemas Multimídia

20.1 O que É Multimídia?	414	20.6 Gerenciamento de Rede	419
20.2 Compressão	416	20.7 Um Exemplo: CineBlitz	421
20.3 Requisitos de Kernels Multimídia	416	20.8 Resumo	422
20.4 Scheduling da CPU	418	Exercícios	423
20.5 Scheduling de Disco	418	Notas Bibliográficas	424

PARTE OITO ■ ESTUDOS DE CASO

Capítulo 21 O Sistema Linux

21.1 História do Linux	427	21.8 Entrada e Saída	445
21.2 Princípios de Projeto	429	21.9 Comunicação Interprocessos	446
21.3 Módulos do Kernel	431	21.10 Estrutura de Rede	447
21.4 Gerenciamento de Processos	433	21.11 Segurança	448
21.5 Scheduling	435	21.12 Resumo	449
21.6 Gerenciamento de Memória	437	Exercícios	450
21.7 Sistemas de Arquivos	441	Notas Bibliográficas	451

Capítulo 22 Windows XP

22.1 História	452	22.6 Conexão de Rede	473
22.2 Princípios de Projeto	453	22.7 Interface do Programador	477
22.3 Componentes do Sistema	454	22.8 Resumo	481
22.4 Subsistemas Ambientais	467	Exercícios	481
22.5 Sistema de Arquivos	469	Notas Bibliográficas	482

Capítulo 23 Sistemas Operacionais Influentes

23.1 Migração de Recursos	483	23.9 IBM OS/360	490
23.2 Primeiros Sistemas	483	23.10 TOPS-20	490
23.3 Atlas	487	23.11 CP/M e MS/DOS	491
23.4 XDS-940	488	23.12 Sistema Operacional Macintosh e	
23.5 THE	488	Windows	491
23.6 RC 4000	489	23.13 Mach	492
23.7 CTSS	489	23.14 Outros Sistemas	492
23.8 MULTICS	489	Exercícios	493

PARTE NOVE ■ APÊNDICES (Veja material no site www.ltceditora.com.br)

Apêndice A BSD UNIX

A.1 História do UNIX	1	A.7 Sistema de Arquivos	14
A.2 Princípios de Projeto	4	A.8 Sistema de I/O	18
A.3 Interface do Programador	5	A.9 Comunicação Interprocessos	20
A.4 Interface do Usuário	9	A.10 Resumo	23
A.5 Gerenciamento de Processos	11	Exercícios	23
A.6 Gerenciamento de Memória	13	Notas Bibliográficas	24

Apêndice B O Sistema Mach

B.1 História do Sistema Mach	25	B.6 Gerenciamento de Memória	34
B.2 Princípios de Projeto	26	B.7 Interface do Programador	37
B.3 Componentes do Sistema	26	B.8 Resumo	37
B.4 Gerenciamento de Processos	28	Exercícios	38
B.5 Comunicação Interprocessos	31	Notas Bibliográficas	38

Apêndice C Windows 2000

C.1 História	39	C.6 Conexão em Rede	54
C.2 Princípios de Projeto	39	C.7 Interface do Programador	57
C.3 Componentes do Sistema	40	C.8 Resumo	60
C.4 Subsistemas Ambientais	49	Exercícios	61
C.5 Sistema de Arquivos	50	Notas Bibliográficas	61

Bibliografia 494

Créditos 502

Índice 503

Visão Geral

Parte Um

Um sistema operacional atua como um intermediário entre o usuário de um computador e o hardware do computador. A finalidade do sistema operacional é fornecer um ambiente em que o usuário possa executar programas de maneira conveniente e eficiente.

O sistema operacional é um software que gerencia o hardware do computador. O hardware deve fornecer mecanismos apropriados para assegurar a operação correta do sistema de computação e impedir que programas de usuário interfiram na operação apropriada do sistema.

Internamente, os sistemas operacionais variam muito em sua composição, já que estão organizados em muitas linhas diferentes. O projeto de um novo sistema operacional é uma tarefa de peso. É importante que os objetivos do sistema sejam bem definidos antes de o projeto começar. Esses objetivos formam a base das escolhas feitas entre vários algoritmos e estratégias.

Já que um sistema operacional é grande e complexo, deve ser criado por módulos. Cada um desses módulos deve ser uma parcela bem delineada do sistema, com entradas, saídas e funções definidas cuidadosamente.

Introdução



Um *sistema operacional* é um programa que gerencia o hardware do 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 são projetados de modo a oferecer um ambiente no qual o usuário possa interagir facilmente com o computador para executar programas. Assim, alguns sistemas operacionais são projetados para ser *convenientes*, outros para ser *eficientes*, e outros para atender a alguma combinação de ambos os aspectos.

Antes de poder examinar os detalhes da operação do sistema de computação, precisamos saber algo sobre a estrutura do sis-

tema. Começaremos discutindo as funções básicas de inicialização, I/O e armazenamento do sistema. Também descreveremos a arquitetura básica do computador que torna possível criar um sistema operacional funcional.

Já que um sistema operacional é grande e complexo, deve ser criado por módulos. Cada um desses módulos deve ser uma parcela bem delineada do sistema, com entradas, saídas e funções bem definidas. Neste capítulo, fornecemos uma visão geral dos principais componentes de um sistema operacional.

OBJETIVOS DO CAPÍTULO

- Fornecer um trânsito completo pelos principais componentes dos sistemas operacionais.
- Descrever a organização básica dos sistemas de computação.

1.1 O que Fazem os Sistemas Operacionais

Começaremos 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 ele mesmo. Simplesmente proporciona um *ambiente* no qual outros programas possam desempenhar tarefas úteis.

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

1.1.1 O 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 estiver 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* — 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 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 sejam utilizados 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 com outras estações de trabalho e *servidores*. Esses usuários possuem recursos dedicados à sua disposição, mas também compartilham recursos tais como a rede

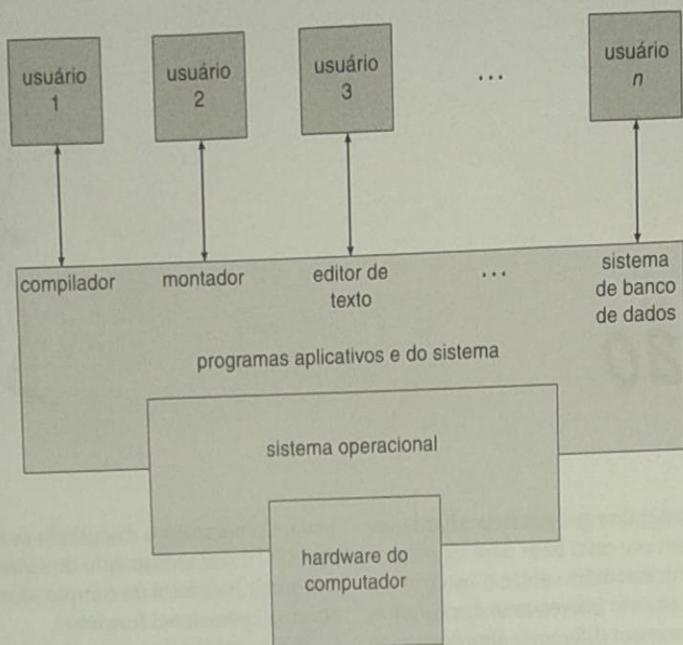


Figura 1.1 Visão abstrata dos componentes de um sistema computacional.

e os servidores — servidores de arquivo, de processamento 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. A maioria desses dispositivos é de unidades autônomas para usuários individuais. Alguns são conectados a redes tanto diretamente por fio como (mais frequentemente) através de modems e conexões de rede sem fio. Devido a limitações de energia, velocidade e interface, eles executam relativamente poucas operações remotas. Seus sistemas operacionais são projetados principalmente objetivando a usabilidade individual, embora o desempenho em relação ao tempo de vida da bateria também seja importante.

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 concorrentes, o sistema operacional precisa decidir como alocá-los a programas e usuários específicos para poder operar o sistema de computação de maneira eficiente e justa. Como vimos, a alocação de recursos é particularmente importante onde muitos usuários acessam o mesmo mainframe ou minicomputador.

Um enfoque ligeiramente diferente relacionado ao sistema operacional enfatiza a necessidade de controle dos diversos dis-

positivos de I/O e programas de usuário. Um sistema operacional é um programa de controle. Um *programa de controle* gerencia a execução dos 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

Examinamos o papel do sistema operacional dos pontos de vista do usuário e do sistema. Como, no entanto, podemos definir o que é um sistema operacional? Em geral, não possuímos uma definição completamente adequada de um sistema operacional. Sistemas operacionais existem porque eles representam uma maneira razoável de resolver o problema de criar um sistema de computação utilitável. 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 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 tipo de software: o sistema operacional.

DEFINIÇÕES E NOTAÇÃO DE ARMAZENAMENTO

Um *bit* é a unidade básica de armazenamento nos computadores. Ele pode conter um entre dois valores, zero e um. 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* compõe-se de 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 a unidade de armazenamento

original de uma determinada arquitetura de computador. Geralmente uma palavra é composta por um ou mais bytes. Por exemplo, um computador pode ter instruções para mover palavras de 64 bits (8 bytes).

Um quilobyte ou KB é igual a 1.024 bytes, um megabyte ou MB equivale a 1.024² bytes, e um gigabyte ou GB é o mesmo que 1.024³ bytes. Os fabricantes de computadores costumam arredondar esses números e dizem que um megabyte corresponde a 1 milhão de bytes e um gigabyte a 1 bilhão de bytes.

Além disso, não temos uma definição universalmente aceita sobre o que compõe o sistema operacional. Um ponto de vista simplista é o de que ele inclui tudo que um vendedor monta quando você encomenda "o sistema operacional". Entretanto, os recursos incluídos variam muito entre os sistemas. Alguns sistemas ocupam menos do que 1 megabyte de espaço e não preenchem nem mesmo uma tela completa do editor, enquanto

outros requerem gigabytes de espaço e são inteiramente baseados em sistemas com janelas gráficas. Uma definição mais comum, que é a que costumamos seguir, é que o sistema operacional é o programa que permanece em execução no computador durante todo o tempo — geralmente chamado de *kernel*. (Além do kernel, há dois outros tipos de programas: os *programas do sistema*, que estão associados ao sistema operacional mas não fazem parte do kernel, e os *programas aplicativos*, que incluem todos os programas não associados à operação do sistema.)

A discussão sobre o que constitui um sistema operacional está se tornando importante. Em 1998, o Departamento de Justiça dos Estados Unidos fez uma representação contra a Microsoft, em essência alegando que ela incluiu funcionalidades demais em seus sistemas operacionais de forma a impedir 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.

1.2 Organização do Sistema de Computação

Antes de podermos examinar os detalhes de como os sistemas de computação funcionam, precisamos de um conhecimento geral de sua estrutura. Nesta seção, examinaremos várias partes dessa estrutura. A seção se preocupará mais com a organização do sistema de computação; portanto, você pode olhá-la superficialmente ou saltá-la se já conhecer os conceitos.

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 fossem disponibilizados tanto no formato de código-fonte quanto no formato binário (executável). Essa lista inclui o Linux, o BSD UNIX, o Solaris e parte do Mac OS X. A disponibilidade do código-fonte nos permite estudar os sistemas operacionais de dentro para fora. Perguntas que antes só podiam ser respondidas através da verificação da documentação ou do comportamento de um sistema operacional já podem ser respondidas através da verificação do próprio código.

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 em que centenas de "aplicações virtuais" podem ser executadas. Usando esse método, os alunos podem testar centenas de sistemas operacionais dentro de seus próprios sistemas sem custo.

Sistemas operacionais que não são mais comercialmente viáveis também passaram a ter o código-fonte aberto, permitindo-nos estudar como os sistemas operavam em uma época de menos recursos de CPU, memória e espaço de 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/. Simuladores de hardware específico também estão disponíveis em alguns casos, 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 os arquivos de código-fonte e modificar e compilar um novo kernel TOPS-20. Um aluno interessado pode procurar na Internet os artigos originais que descrevem o sistema operacional e os manuais originais.

O advento dos sistemas operacionais de código-fonte aberto também torna fácil fazer a transição de aluno para desenvolvedor de sistema operacional. Com algum conhecimento, algum esforço e uma conexão com a Internet, um aluno 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 só é limitado quanto ao tempo e ao espaço em disco que o aluno possui.

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 proporciona acesso à memória compartilhada (Figura 1.2). Cada controlador de dispositivo é responsável por um tipo específico de dispositivo (por exemplo, drives de disco, dispositivos de áudio e exibidores de vídeo). A CPU e os controladores de dispositivos podem operar concorrentemente, competindo por ciclos de memória. Para assegurar acesso ordenado à memória compartilhada, é fornecido um controlador de memória cuja função é sincronizar o acesso à memória.

Para que um computador comece a operar — por exemplo, quando é ligado ou reiniciado — precisa dispor de um programa inicial para executar. Esse programa inicial, ou *programa bootstrap*, tende a ser simples. Normalmente, ele é armazenado em memória de leitura (ROM) ou em memória de leitura programável e eletricamente apagável (EEPROM), conhecida pelo termo geral *firmware*, dentro do hardware do computador. 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 como iniciar

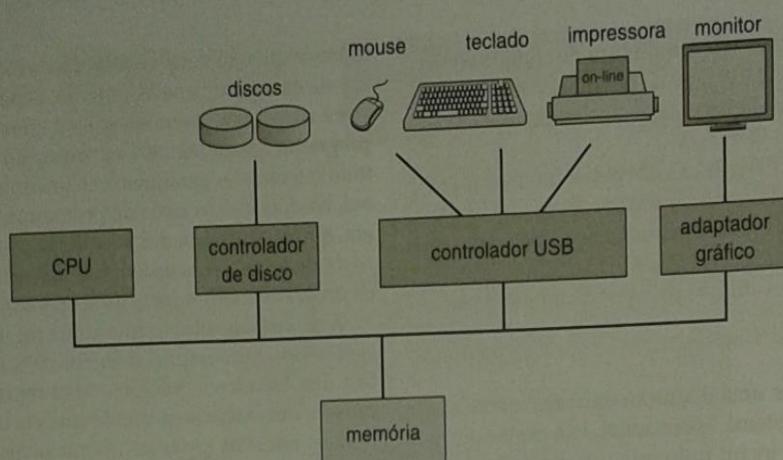


Figura 1.2 Um moderno sistema de computação.

sua execução. Para alcançar esse objetivo, o programa deve alocar e carregar na memória o kernel do sistema operacional. O sistema operacional, então, começa a executar o primeiro processo, como o "init", e aguarda que algum evento ocorra.

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

Quando a CPU é interrompida, ela para o que está fazendo e transfere imediatamente a execução para uma localização fixa. Normalmente essa localização contém o endereço inicial onde se encontra a rotina de serviço da interrupção. A rotina de serviço da interrupção entra em operação; ao completar a execução, a CPU retoma o processamento interrompido. Uma linha de tempo dessa operação é mostrada na Figura 1.3.

As interrupções são uma parte importante da arquitetura de um computador. Cada arquitetura 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 direto 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 através da 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 para os 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 fornecer o endereço da rotina de serviço de interrupção do dispositivo que está sendo interrompido. 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 o processamento interrompido reassume 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

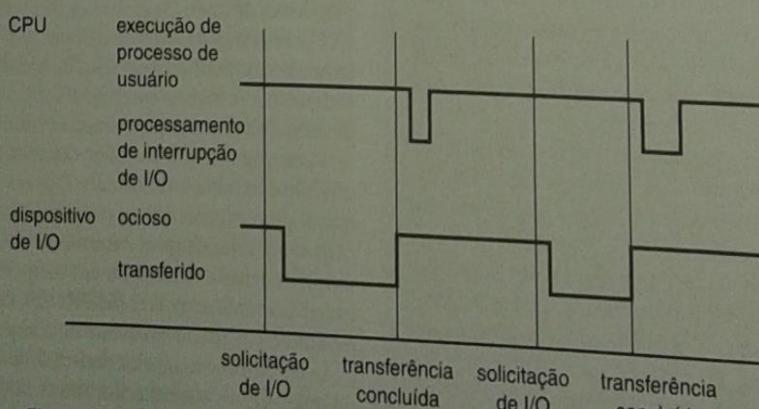


Figura 1.3 Linha de tempo de interrupções de um processo isolado gerando saídas.

nesse local. Computadores de uso geral executam a maioria de seus programas a partir da memória regravável, ou seja, a memória principal (também chamada de *memória de acesso randômico* ou **RAM** — *random-access memory*). Normalmente a memória principal é implementada em uma tecnologia de semicondutor denominada *memória de acesso randômico dinâmica* (**DRAM** — *dynamic random-access memory*). Os computadores também usam outros tipos de memória. Já que a memória de leitura (**ROM**) não pode ser alterada, só programas estáticos são armazenados nela. A imutabilidade da ROM é útil em cartuchos de jogos. A EEPROM não pode ser alterada com frequência e, portanto, contém em sua maioria programas estáticos. Por exemplo, os smartphones usam a EEPROM para armazenar seus programas instalados de fábrica.

Todos os tipos de memória fornecem um array de palavras. Cada palavra 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 uma palavra da memória principal para um registrador interno da CPU, enquanto a instrução *store* move o conteúdo de um registrador para a memória principal. À parte os loads e stores explícitos, a CPU carrega automaticamente instruções a partir da memória principal para execução.

Um típico ciclo instrução-execução, conforme realizado em um sistema com arquitetura *von Neumann*, trará primeiro uma instrução da memória e a armazenará no *registrator de instruções*. A instrução é então decodificada e pode provocar a vinda de operandos da memória e seu armazenamento em algum registrador interno. Após a execução da instrução sobre os operandos, o resultado pode ser retornado à memória. Observe que a unidade de memória enxerga apenas uma cadeia 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 que está em operação.

Em termos ideais, imaginamos que os programas e os dados possam residir na memória principal de modo permanente. Geralmente esse esquema 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. Memória principal é um dispositivo de armazenamento *volátil* que perde seus conteúdos quando a energia é desligada ou por algum outro tipo de perda.

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

O dispositivo de memória secundária mais comum é o *disco magnético*, que propicia armazenamento tanto para programas quanto para dados. A maioria dos programas (do sistema e aplicativos) é armazenada em um disco até que seja carregada na memória. Por isso, muitos programas utilizam o disco tanto como fonte quanto como destino do seu processamento. Logo, o gerenciamento apropriado da memória em disco é de importância capital para um sistema de computação, como discutiremos no Capítulo 12.

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 para armazenar dados e conservá-los 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 em um sistema de computação 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 um 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 a fita de papel e memórias de núcleo, foram relegados aos museus depois que a fita magnética e a *memória de semicondutor* tornaram-se mais rápidas e mais baratas. Os quatro níveis mais altos de memória da Figura 1.4 podem ser construídos com o uso de memória de semicondutor.

Além de possuírem diferentes velocidades e custos, os diversos sistemas de armazenamento podem ser voláteis ou não voláteis. Como mencionado anteriormente, a *memória volátil* perde seus conteúdos quando é retirada a energia disponível para o dispositivo. Na ausência de baterias dispendiosas e sistemas geradores de reserva, 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 eletrônico são voláteis, enquanto os que estão abaixo dele são não voláteis. Um *disco eletrônico* pode ser projetado para ser tanto volátil como não volátil. Durante a operação normal, o disco eletrônico armazena os dados em um extenso array DRAM, que é volátil. Mas muitos dispositivos de disco eletrônico contêm um disco rígido magnético oculto e uma bateria como energia de reserva. Se a força externa é removida, o controlador do disco eletrônico copia os dados da RAM para o disco magnético. Quando a energia externa é restaurada, o controlador devolve os dados à RAM. Outro tipo de disco eletrônico é a memória flash, que é popular

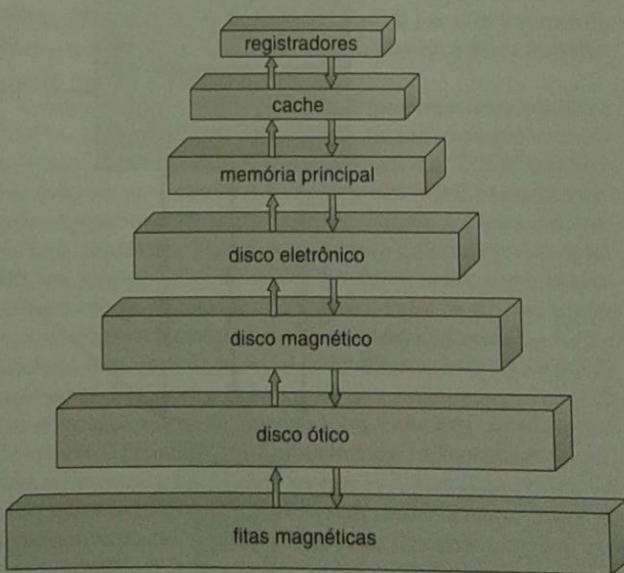


Figura 1.4 Hierarquia de dispositivos de armazenamento.

em câmeras e *assistentes digitais pessoais* (PDAs — *personal digital assistants*), em robôs e cada vez mais como armazenamento removível em computadores de uso geral. A memória flash é mais lenta que a DRAM, mas não precisa de energia para reter seus conteúdos. Outro tipo de armazenamento não volátil é a NVRAM, que é a DRAM com energia de bateria de reserva. 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. Quando existir grande tempo de acesso ou disparidade de taxa de transferência entre dois componentes, caches podem ser instalados para melhorar o desempenho.

1.2.3 Estrutura de I/O

O armazenamento é apenas um dos muitos tipos de dispositivos de I/O de um computador. Uma 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 devido à 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 múltiplos controladores de dispositivos conectados através de um bus comum. Cada controlador de dispositivo é responsável por um tipo específico de dispositivo. Dependendo do controlador, pode haver mais de um dispositivo conectado. Por exemplo, o controlador da *interface de pequenos computadores* (SCSI — *small computer-systems interface*) pode ter sete ou mais dispositivos conectados. O controlador de dispositivo mantém um buffer local e um conjunto de registradores de uso específico. Ele 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 dispositivo. Esse driver de dispositivo

entende o controlador de dispositivo e apresenta uma interface uniforme entre o dispositivo e o resto do sistema operacional.

Para iniciar uma operação de I/O, o driver de dispositivo carrega os registradores apropriados dentro do controlador de dispositivo. Este, por sua vez, examina o conteúdo dos registradores para determinar que ação tomar (como “ler um caractere a partir do teclado”). O controlador inicia a transferência dos dados do dispositivo para o seu buffer local. Uma vez que a transferência estiver concluída, o controlador de dispositivo informa ao driver de dispositivo através de uma interrupção que terminou sua operação. O driver de dispositivo retorna então o controle para o sistema operacional, possivelmente retornando os dados ou um ponteiro para os dados se a operação for uma leitura. Para outras operações, o driver de 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 em massa como no I/O de disco. Para resolver esse problema, é usado o *acesso direto à memória* (DMA — *direct memory access*). Após posicionar buffers, ponteiros e contadores associados ao dispositivo de I/O, o controlador do dispositivo transfere um bloco inteiro de dados diretamente da memória para seu próprio buffer ou a partir dele para a memória, sem intervenção da CPU. Somente uma interrupção é gerada por bloco, para informar ao driver de dispositivo que a operação foi concluída, em vez de uma interrupção por byte gerada pelos dispositivos de baixa velocidade. Enquanto o controlador de dispositivo está executando essas operações, a CPU está disponível para processar outras tarefas.

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 competir 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.

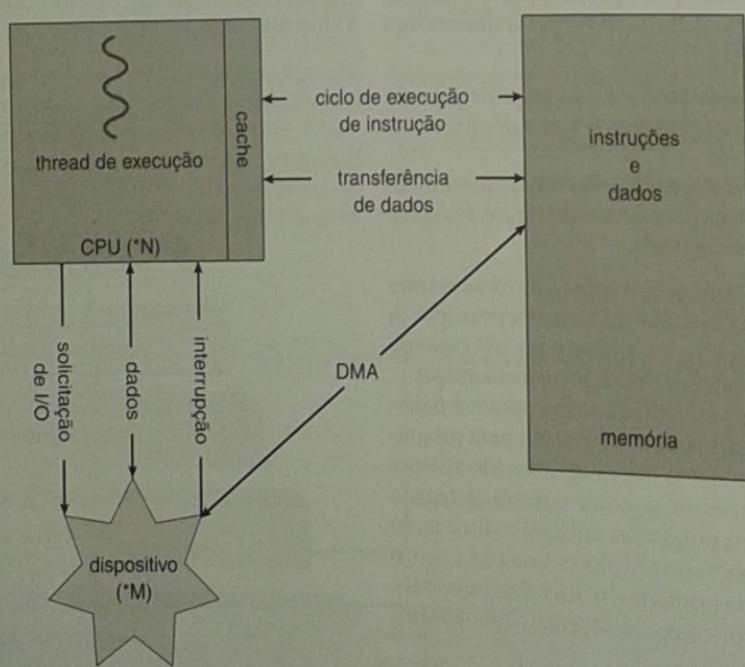


Figura 1.5 Como um moderno sistema de computação funciona.

1.3 Arquitetura do Sistema 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 podemos categorizar de acordo com a quantidade de processadores de uso geral utilizados.

1.3.1 Sistemas com um Único Processador

A maioria dos sistemas usa um único processador. Portanto, a variedade de sistemas de processador único pode ser surpreendente, já que esses sistemas vão dos PDAs aos mainframes. Em um sistema com um único processador, há uma CPU principal capaz de executar um conjunto de instruções de uso geral, inclusive instruções provenientes de processos de usuário. Quase todos os sistemas também possuem 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 movimentam 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 este 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 seu próprio algoritmo de scheduling. Esse esquema libera a CPU principal do overhead de realizar o scheduling do disco. Os PCs contêm um microprocessador no teclado para converter os toques em códigos a serem enviados à 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 com processador único em um multiprocessador. Quando só há uma CPU de uso geral, trata-se de um sistema com processador único.

1.3.2 Sistemas Multiprocessadores

Embora os sistemas com um único processador sejam mais comuns, os *sistemas multiprocessadores* (também conhecidos como *sistemas paralelos* ou *sistemas fortemente acoplados*) estão crescendo em importância. 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 têm três vantagens principais:

1. **Throughput aumentado.** Com o aumento do número de processadores, espera-se obter mais trabalho executado em menor tempo. No entanto, a taxa incremental de velocidade com N processadores não é N , é menor do que N . Quando múltiplos processadores cooperam em uma tarefa, está implícita uma certa quantidade de overhead para que todas as partes continuem trabalhando corretamente. Esse overhead, mais a contingência por recursos compartilhados, diminui o ganho esperado dos processadores adicionais. Do mesmo modo, N programadores trabalhando em estreita cooperação não produzem N vezes o montante de trabalho que um único trabalhador produziria.
2. **Economia de escala.** Sistemas multiprocessadores podem economizar mais dinheiro que múltiplos sistemas com um único processador, porque eles podem compartilhar periféricos, memória de massa e suprimentos de energia. Se vários programas operam sobre o mesmo conjunto de dados, é mais barato armazenar esses dados em um disco, fazendo com que todos os processadores os compartilhem, do que manter muitos computadores com discos locais e muitas cópias dos dados.
3. **Confiabilidade aumentada.** Se as funções podem ser distribuídas apropriadamente entre vários processadores, então a falha de um processador não interrompe o sistema, apenas o torna mais lento. Se temos dez processadores e um falha, cada um dos nove remanescentes pode compartilhar o trabalho do processador que falhou. Assim, o sistema inteiro opera 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 *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 ainda continuar a operar. É bom ressaltar que 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 os resultados diferem é porque uma CPU do par está falhando e as duas são interrompidas. O processo que estava sendo executado é 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 usam *multiprocessamento assimétrico*, no qual a cada processador é designada uma tarefa específica. Um processador mestre controla o sistema; os demais processadores ou se dirigem ao mestre para instruções ou possuem tarefas predefinidas. Esse esquema define um relacionamento mestre-escravo. O processador mestre planeja e aloca trabalho para os processadores escravos.

Os sistemas mais comuns usam *multiprocessamento simétrico* (SMP — *symmetric multiprocessing*), no qual cada processador executa todas as tarefas do sistema operacional. O SMP significa que todos os processadores são pares; não existe um relacionamento mestre-escravo entre os processadores. 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 a memória física. Um exemplo do sistema SMP é o Solaris, uma versão comercial do UNIX projetada pela Sun Microsystems. Um sistema Solaris pode ser configurado de tal modo que empregue dúzias de processadores, todos executando cópias do Solaris. 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 no desempenho. Entretanto, devemos controlar cuidadosamente o I/O para assegurar que os dados alcancem o processador apropriado. Além disso, como as CPUs são separadas,

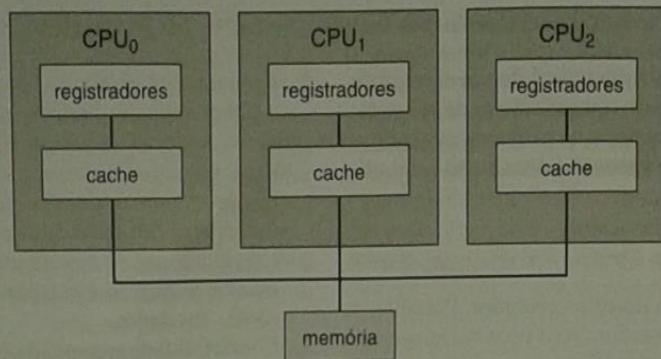


Figura 1.6 Arquitetura de multiprocessamento simétrico.

e 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 6. Praticamente todos os sistemas operacionais modernos — inclusive o Windows, o Windows XP, o Mac OS X e o Linux — já dão suporte ao SMP.

A diferença entre os multiprocessamentos simétrico e assimétrico pode ser resultante 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 fornecia multiprocessamento assimétrico, enquanto a Versão 5 (Solaris) é simétrica no mesmo hardware.

O multiprocessamento adiciona CPUs para aumentar o poder de processamento. Se a CPU tiver um controlador de memória integrado, o acréscimo de CPUs também pode aumentar a quantidade 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 — *uniform memory access*) para o acesso não uniforme (NUMA — *non-uniform memory access*). O UMA é definido como a situação em que o acesso à RAM a partir de qualquer CPU leva o mesmo período de tempo. No NUMA, algumas partes da memória podem demorar mais para ser acessadas do que outras, gerando perda no desempenho. Os sistemas operacionais podem reduzir a perda gerada pelo NUMA através do gerenciamento de recursos, como discutido na Seção 9.5.4.

Uma tendência recente no projeto de CPUs é a inclusão de vários *núcleos* de computação no mesmo chip. Na verdade, esses chips são multiprocessadores. Eles podem ser mais eficientes do que vários chips de núcleo único porque a comunicação dentro do chip é mais veloz do que a comunicação entre chips. Além disso, o chip com vários núcleos usa bem menos energia do que vários chips de núcleo único. Como resultado, os sistemas de vários núcleos (*multicore*) são especialmente adequados para sistemas servidores como os servidores de banco de dados e da Web.

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 à arquitetura, como a disputa por cache, memória e bus, as CPUs multicore aparecem para o sistema ope-

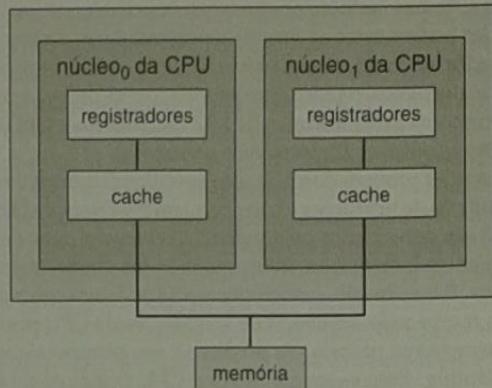


Figura 1.7 Um projeto dual-core com dois núcleos inseridos no mesmo chip.

racionais como N processadores padrões. É uma tendência que pressiona os projetistas de sistemas operacionais — e programadores de aplicações — a fazer uso dessas CPUs.

Para concluir, os *servidores blade* são um desenvolvimento 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 os tipos de computador. Na verdade, esses servidores são compostos por vários sistemas multiprocessadores independentes.

1.3.3 Sistemas Agrupados (clusters)

Outro tipo de sistema com várias CPUs é o *sistema cluster*. Como os sistemas multiprocessadores, os clusters reúnem múltiplas CPUs para desenvolver trabalho computacional. Entretanto, esses sistemas diferem dos sistemas multiprocessadores por serem compostos de dois ou mais sistemas individuais — ou nós — acoplados. A definição do termo *cluster* não é unânime; muitos pacotes comerciais divergem com relação ao que é um 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 através de uma *rede local* (LAN — *local-area network*) (como descrito na Seção 1.10) ou de uma interconexão mais veloz, como a InfiniBand.

O cluster costuma ser usado para proporcionar serviço de *alta disponibilidade*, isto é, o serviço que continua mesmo se

um ou mais sistemas do agrupamento falhar. Geralmente a alta disponibilidade é obtida através da inclusão de um nível de redundância no sistema. Uma camada de software de cluster opera sobre os nós do cluster. Cada nó pode monitorar um ou mais dos outros nós (através da LAN). Se a máquina monitorada falhar, a máquina monitora 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.

CLUSTERS BEOWULF

Os clusters Beowulf são projetados para resolver tarefas de computação de alto desempenho. Esses clusters são construídos com o uso de hardware disponível no comércio — como os computadores pessoais — conectados através de uma simples rede local. O interessante é que o cluster Beowulf não usa um pacote de software específico sendo, em vez disso, composto por um conjunto de bibliotecas de software de código-fonte aberto que permite que os nós de computação do cluster se comuniquem entre si. Portanto, há várias abordagens para a construção de um cluster Beowulf, embora normalmente os nós de computação Beowulf executem 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 conjuntos de computadores pessoais descartados estão usando centenas de nós de computação para resolver problemas de processamento dispendioso da computação científica.

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

Já que um cluster é composto por vários sistemas de computação conectados através de uma rede, os clusters também podem ser usados para fornecer ambientes de *computação de alto desempenho*.

Esses sistemas podem fornecer um poder de processamento significativamente maior do que os sistemas com um único processador ou até mesmos sistemas SMP, porque são capazes de executar uma aplicação concorrentemente em todos os computadores do cluster. No entanto, as aplicações devem ser escritas especificamente para tirar vantagem do cluster com o uso de uma técnica conhecida como *paralelização*, que consiste na divisão de um programa em componentes separados que são executados em paralelo em computadores individuais do cluster. Normalmente, essas aplicações são projetadas para que, quando cada nó de computação do cluster tiver 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 sobre uma rede de longa distância (**WAN** — *wide-area network*) (como descrito na Seção 1.10). Os clusters paralelos permitem que múltiplos hospedeiros tenham acesso aos 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 são obtidos com o uso de versões especiais de software e liberações especiais de aplicações. Por exemplo, o Oracle Real Application Cluster é uma versão do banco de dados Oracle projetada para operar em clusters paralelos. Todas as máquinas executam 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 aos dados, o sistema também deve oferecer controle e bloqueio de acesso para assegurar que não ocorram operações conflitantes. Essa função, normalmente chamada de *gerenciador de segurança distribuído* (**DLM** — *distributed lock manager*), é incluída em algumas tecnologias de cluster.

A tecnologia de cluster está mudando rapidamente. Alguns produtos de cluster dão suporte a vários sistemas em um cluster, assim como a nós clusters separados por quilômetros. Muitas dessas melhorias se tornaram possíveis graças às *redes de armazenamento* (**SANs** — *storage-area networks*), como descrito na Seção 12.3.3, que permitem que muitos sistemas sejam conectados a um pool de armazenamento. Se os aplicativos e seus dados forem armazenados na SAN, o software de cluster poderá designar a aplicação para ser executada em qualquer hospedeiro que estiver 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 cluster.

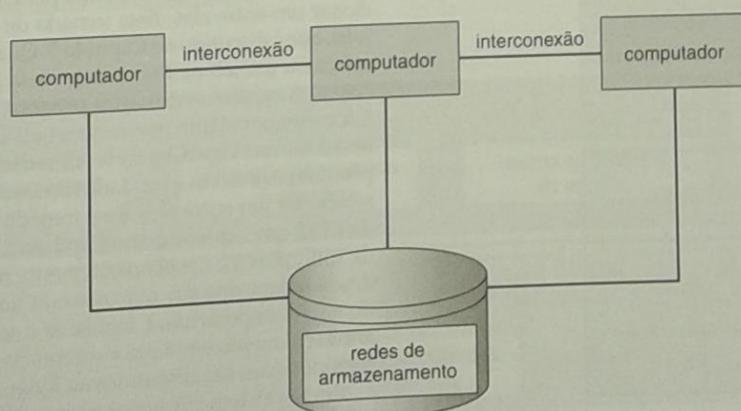


Figura 1.8 Estrutura geral de um sistema em cluster.

1.4 Estrutura do Sistema Operacional

Agora que já discutimos as informações básicas sobre a organização e a arquitetura do sistema de computação, estamos prontos para falar sobre sistemas operacionais. Um sistema operacional fornece o ambiente dentro do qual 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 consideraremos nesta seção.

Um dos aspectos mais importantes dos sistemas operacionais é a capacidade de multiprogramar. Geralmente um único programa não pode manter a CPU ou os dispositivos de I/O ocupados durante todo o tempo. Usuários individuais costumam ter vários programas sendo executados. A *multiprogramação* aumenta a utilização da CPU organizando os jobs (código e dados) de modo que a CPU tenha sempre um deles 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, estes são mantidos inicialmente em disco na *fila de jobs*. Essa fila é composta de todos os processos residentes em disco aguardando alocação na memória principal.

O conjunto de jobs da memória pode ser um subconjunto dos jobs mantidos na fila. O sistema operacional seleciona e começa a executar um dos jobs da memória. Em dado momento, o job pode ter que aguardar que alguma tarefa, como uma operação de I/O, seja concluída. Em um sistema não multiprogramado, a CPU permaneceria ociosa. Em um sistema multiprogramado, o sistema operacional simplesmente passa para um novo job e o executa. Quando esse job tem de aguardar, a CPU é redirecionada para *outro* job e assim por diante. Por fim, o primeiro job sai da espera e volta à CPU. Contanto que pelo menos um job tenha de ser executado, a CPU nunca ficará 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; portanto, existe um 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 computacional. O *tempo compartilhado* (ou a *multitarefa*) é uma extensão lógica da multiprogramação. Em sistemas de tempo compartilhado, a CPU executa múltiplos 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 sendo executado.

O tempo compartilhado requer um *sistema de computação interativo* (ou *hands-on*), que possibilite a 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 um teclado ou um mouse, e espera 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 o outro, cada usuário tem a impressão de que todo o sistema de computação está dedicado ao seu uso, mesmo que ele esteja sendo compartilhado entre muitos usuários.

Um sistema operacional de tempo compartilhado usa a multiprogramação e o scheduling da CPU para disponibilizar uma pequena porção do computador de tempo compartilhado a cada usuário. Cada usuário tem pelo menos um programa separado na memória. Um programa carregado na memória e em execução é chamado de *processo*. Quando um processo entra em execução, normalmente é processado apenas por um breve período antes que termine ou tenha que executar I/O. O I/O pode ser interativo, isto é, a saída é exibida para o usuário e a entrada provém do usuário pelo teclado, o mouse ou e outro dispositivo. 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 direcionará rapidamente a CPU para o programa de algum outro usuário.

O tempo compartilhado e a multiprogramação requerem que vários jobs sejam mantidos simultaneamente na memória. Se vários jobs estiverem prontos para serem trazidos à memória e se não houver espaço suficiente para todos, o sistema deve selecionar um entre eles. Essa tomada de decisão é o *scheduling de jobs*, que é discutido no Capítulo 5. Quando o sistema operacional seleciona um job na fila, 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 é abordado nos Capítulos 8 e 9. Além disso, se vários jobs estiverem prontos para serem executados ao mesmo tempo, o sistema deve selecionar um entre eles. Essa tomada de decisão é o *scheduling da CPU*, que é discutido no Capítulo 5. 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, inclusive o scheduling de processos, o armazenamento em disco e o gerenciamento da memória. Essas considerações são discutidas ao longo do texto.

Em um sistema de tempo compartilhado, o sistema operacional deve assegurar um tempo de resposta razoável, que pode ser obtido através do *swapping*, onde os processos são alternados

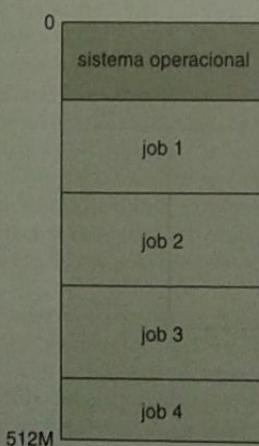


Figura 1.9 Layout da memória de um sistema de multiprogramação.

entre a memória principal e o disco. Um método comum para se atingir esse objetivo 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 vetor de armazenamento, separando a memória física da *memória lógica* visualizada pelo usuário. Esse esquema livra os programadores da preocupação com limitações de armazenamento na memória.

Os sistemas também devem fornecer um sistema de arquivos (Capítulos 10 e 11). O sistema de arquivos reside em um conjunto de discos; portanto, o gerenciamento de discos deve estar disponível (Capítulo 12). Além disso, os sistemas de tempo compartilhado fornecem um mecanismo para a proteção de recursos contra uso inapropriado (Capítulo 14). Para garantir execução ordenada, o sistema deve fornecer mecanismos para sincronização e comunicação entre jobs (Capítulo 6) e deve assegurar que os jobs não fiquem presos em um deadlock, esperando eternamente um pelo outro (Capítulo 7).

1.5 Operações do Sistema Operacional

Como mencionado anteriormente, os sistemas operacionais modernos são *dirigidos por interrupções*. Se não existirem processos para executar, dispositivos de I/O para servir e usuários a quem responder, um sistema operacional permanecerá inativo esperando que algo aconteça. Os eventos são quase sempre indicados pela ocorrência de uma interrupção ou de uma exceção. Uma *exceção* (ou *interceptação*) 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 um sistema operacional dirigido por interrupções define a estrutura geral desse sistema. Para cada tipo de interrupção, segmentos separados de código do sistema operacional determinam que medida deve ser tomada. É fornecida uma rotina de serviço de interrupções responsável por lidar com a interrupção.

Já que o sistema operacional e os usuários compartilham os recursos de hardware e software do sistema de computação, precisamos verificar se um erro em um programa de usuário só causaria problemas 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 ficasse preso em um loop infinito, este loop poderia impedir a operação correta de muitos outros processos. Erros mais sutis podem ocorrer em um sistema com multiprogramação, onde 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 consiga fazer outros programas executarem incorretamente.

1.5.1 Operação em Modo Dual

Para garantir a execução apropriada do sistema operacional, temos de poder distinguir a execução do código do sistema operacional da execução do 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, somos capazes de distinguir entre uma tarefa que é executada em nome do sistema operacional e uma que é executada em nome do usuário. Quando o sistema de computação está operando em nome de uma aplicação de usuário, ele está na modalidade de usuário. No entanto, quando uma aplicação de usuário solicita um serviço do sistema operacional (através de uma chamada de sistema), o sistema deve passar da modalidade de usuário para a de kernel para atender a solicitação. Isso é mostrado na Figura 1.10. Como veremos, essa melhoria na arquitetura também é útil em muitos outros aspectos de operação do sistema.

No momento da inicialização do sistema, o hardware começa a operar em modalidade de kernel. O sistema operacional é então carregado e dá início às 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, modifica para 0 o estado do bit de modalidade). Portanto, sempre que o sistema operacional ganha o controle do computador, ele está na modalidade de kernel. O sistema sem-

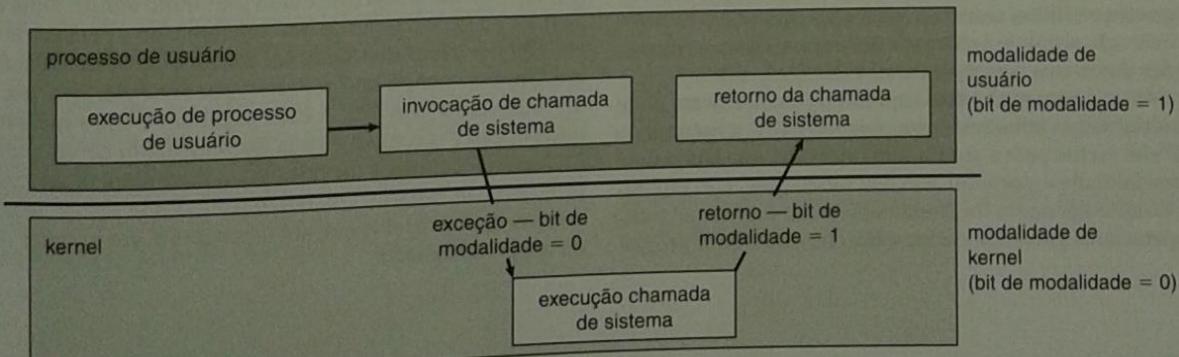


Figura 1.10 Transição da modalidade de usuário para a de kernel.

pre passa para a modalidade de usuário (posicionando o bit de modalidade em 1) antes de passar o controle para um programa de usuário.

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

A instrução de passagem para o modo 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.

Agora podemos ver o ciclo de vida da execução de instruções em um sistema de computação. O controle inicial reside no sistema operacional, onde as instruções são executadas na modalidade de kernel. Quando o controle é passado para uma aplicação de usuário, a modalidade é configurada como modalidade de usuário. O controle acaba sendo devolvido ao sistema operacional através de uma interrupção, uma exceção ou uma chamada de sistema.

As chamadas de sistema fornecem o meio para um programa de usuário solicitar ao sistema operacional que execute tarefas reservadas ao sistema operacional 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 interceptação para um local específico do vetor de interrupções. Essa interceptação pode ser executada por uma instrução `trap` genérica, embora alguns sistemas (como a família MIPS R2000) tenham uma instrução `syscall` específica.

Quando uma chamada de sistema é executada, ela é 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 na 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, através da pilha ou em memória (com os ponteiros para os locais da memória informados em registradores). O kernel verifica se os parâmetros estão corretos e válidos, 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 um modo dual suportado pelo hardware pode causar sérias falhas 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 modo dual. Um programa de usuário operando incorretamente pode tirar do ar o sistema operacional gravando dados sobre ele; e múltiplos progra-

mas são capazes de gravar ao mesmo tempo em um dispositivo, com resultados possivelmente desastrosos. Versões recentes da CPU Intel fornecem operação em modo dual. Como resultado, a maioria dos sistemas operacionais contemporâneos — como o Microsoft Vista e o Windows XP, assim como o Unix, o Linux e o Solaris — tira vantagem desse recurso, o que garante maior proteção ao sistema operacional.

Uma vez que a proteção de hardware esteja implementada, ela detectará erros que violem as modalidades. Normalmente esses erros são manipulados pelo sistema operacional. Se um programa de usuário falhar de alguma forma — tal como tentar executar uma instrução inválida ou acessar memória que não faça parte do espaço de endereços do usuário —, o hardware fará a interceptação para o sistema operacional. A interceptação transfere o controle, através do vetor de interrupções, para o sistema operacional, da mesma forma que a interrupção. Quando ocorre um erro no programa, o sistema operacional deve encerrá-lo anormalmente. Essa situação é manipulada pelo mesmo código usado no caso de um encerramento anormal solicitado pelo usuário. Uma mensagem de erro apropriada é exibida e é possível que a memória do programa seja descarregada. Geralmente o conteúdo 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 preso em um loop infinito ou não chame os serviços do sistema e nunca retorne o controle ao sistema operacional. Para alcançar esse objetivo, podemos usar 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 configura 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 1.024 milissegundos em etapas de 1 milissegundo.

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

Portanto, 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 for positivo, o controle será retornado ao programa do usuário. Quando o contador se tornar negativo, o sistema operacional encerrará o programa por exceder o limite de tempo designado.

1.6 Gerenciamento de Processos

Um programa não faz nada 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 edição 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). 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 permitam que os processos criem subprocessos a serem executados concurrentemente.

Um processo precisa de certos recursos — inclusive tempo de CPU, memória, arquivos e dispositivos de I/O — para cumprir sua tarefa. Esses recursos são fornecidos ao processo quando este 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 de inicialização (entradas) 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 no terminal as informações desejadas. 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 não são jamais 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 um determinado thread.

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

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

- Scheduling de processos e threads nas CPUs
- Criação e exclusão de processos de usuário e de sistema
- Suspensão e retomada de processos
- Fornecimento de mecanismos de sincronização de processos
- Fornecimento de mecanismos de comunicação entre processos

Discutiremos as técnicas de gerenciamento de processos nos Capítulos 3 a 6.

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 palavras ou bytes, variando em tamanho de centenas de milhares a bilhões. Cada palavra ou byte tem seu próprio endereço. A memória principal é um repositório de dados de acesso rápido compartilhado pela CPU e dispositivos de I/O. O processador central lê instruções na memória principal durante o ciclo de busca de instruções (em uma arquitetura von Neumann). Como mencionado anteriormente, a memória principal costuma ser o único dispositivo de armazenamento amplo que a CPU consegue endereçar e acessar diretamente. Por exemplo, para que a CPU processe 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 que a CPU possa executá-las.

Para um programa ser 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 na 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, os computadores de uso geral devem manter vários programas na memória, o que cria a necessidade de gerenciamento da memória. Muitos esquemas diferentes de gerenciamento da memória são usados. Esses esquemas refletem diversas abordagens, e a eficácia de um 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 ao gerenciamento da memória:

- Controlar que partes da memória estão em uso corrente e quem as está usando
- Decidir que processos (ou partes deles) e dados devem ser transferidos para dentro e fora da memória
- Alocar e desalocar espaço na 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. Ele abstrai a partir 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 através 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, 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 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. Finalmente, quando vários usuários têm acesso aos arquivos, pode ser desejável controlar por quem e de que maneiras (por exemplo, para ler, gravar, anexar) eles podem ser acessados.

O sistema operacional é responsável pelas seguintes atividades relacionadas ao 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 para memória secundária
- Criar cópias de arquivos em mídias de armazenamento estáveis (não voláteis)

As técnicas de gerenciamento de arquivos são discutidas nos Capítulos 10 e 11.

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 para criar cópias 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 — inclusive compiladores, montadores, processadores de texto, editores e formatadores — é armazenada em um disco até ser carregada na memória e então usa 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 ao gerenciamento de disco:

- Gerenciamento do espaço livre
- Alocação de espaço de armazenamento
- Scheduling de alocação do disco

Já que a memória secundária é utilizada com frequência, 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 barata (e às vezes de maior capacidade) do que a memória secundária. Backups de dados de disco, dados pouco usados e armazenamento de longo prazo são alguns exemplos. Os drives de fita magnética e suas fitas e os discos e drives de CD e DVD são típicos dispositivos de *memória terciária*. A mídia (fitas e discos ópticos) varia entre os formatos *WORM* (gravar uma vez, ler várias vezes) e *RW* (ler-gravar).

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 de mídia em dispositivos, a alocação e liberação dos dispositivos para uso exclusivo por processos e a migração de dados da memória secundária para a terciária.

As técnicas de gerenciamento das memórias secundária e terciária são discutidas no Capítulo 12.

1.8.3 Armazenamento em Cache (*caching*)

O *caching* é um princípio importante dos sistemas de computação. 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 programáveis internos, 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 que informações devem ser mantidas em registradores e quais devem ficar na memória principal. Também há caches que são implementados totalmente em hardware. Por exemplo, a maioria dos sistemas tem um cache de instruções para armazenar as próximas instruções a serem executadas. Sem esse cache, a CPU teria de esperar vários ciclos enquanto uma instrução é trazida da memória principal. Por razões 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.

Já que os caches têm tamanho limitado, o *gerenciamento do cache* é um importante problema de projeto. A seleção cuidadosa

Nível	1	2	3	4
Nome	regitadores	cache	memória principal	armazenamento em disco
Tamanho típico	< 1 KB	< 16 MB	< 64 GB	> 100 GB
Tecnologia de implementação	memória personalizada com várias portas, CMOS	SRAM de CMOS em chip ou fora do chip	DRAM de CMOS	disco magnético
Tempo de acesso (ns)	0,25 – 0,5	0,5 – 25	80 – 250	5.000.000
Largura de banda (MB/s)	20.000 – 100.000	5000 – 10.000	1000 – 5000	20 – 150
Gerenciado por	compilador	hardware	sistema operacional	sistema operacional
Copiado em	cache	memória principal	disco	CD ou fita

Figura 1.11 Desempenho de vários níveis de armazenamento.

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 para a 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 reside 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 RAM eletrônicos (também conhecidos como *discos de estado sólido*) podem ser usados como memória de alta velocidade acessada através da interface do sistema de arquivos. O corpo 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抗igos da memória secundária para a memória terciária, como jukeboxes de fita, para reduzir o custo de armazenamento (consulte o Capítulo 12).

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

Em uma estrutura de memória hierárquica, os mesmos dados podem aparecer em diferentes níveis do sistema de armazenamento. Por exemplo, suponhamos que um inteiro A que tivesse de ser incrementado em 1 unidade estivesse localizado no arquivo B e o arquivo B residisse em disco magnético. A operação de incremento é efetuada, primeiro, com a emissão de uma operação de I/O destinada a copiar na memória principal o bloco de disco

em que A reside. Essa operação é seguida pela cópia de A no cache e em um registrador interno. Portanto, 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). Assim que o incremento ocorrer no registrador interno, o valor de A será diferente nos diversos sistemas de armazenamento. O valor de A só se tornará igual após o novo valor de A ser 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 de A mais recente.

A situação se torna mais complicada em um ambiente multiprocessado 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 concorrentemente, precisamos nos certificar de que uma atualização no valor de A em um cache se reflita em todos os outros caches em que A reside. Essa situação é chamada de *coerência do cache* e geralmente 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 distribuídos espacialmente. Já que as diversas réplicas podem ser acessadas e atualizadas concorrentemente, alguns sistemas distribuídos garantem que, quando uma réplica for atualizada em um local, todas as outras réplicas sejam atualizadas o mais breve possível. Há várias maneiras de garantir isso, como discutiremos 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

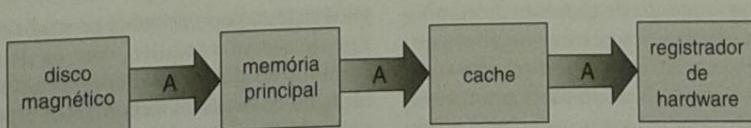


Figura 1.12 Migração do inteiro A do disco para o registrador.

I/O são ocultas do corpo do próprio sistema operacional pelo *subsistema de I/O*. O subsistema de I/O consiste em vários componentes:

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

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

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

1.9 Proteção e Segurança

Se um sistema de computação tem vários usuários e permite a execução concorrente de múltiplos processos, o acesso aos dados deve ser regulado. Com 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 endereços. 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.

Portanto, *proteção* é qualquer mecanismo de controle do 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. Geralmente, a detecção precoce de erros de interface pode impedir a contaminação de um subsistema saudável por outro com defeito. Além disso, um recurso desprotegido não pode se defender do uso (ou mau uso) por um usuário não autorizado ou incompetente. Um sistema orientado para a proteção fornece um meio para a distinção entre o uso autorizado e não autorizado, como discutiremos 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 excluídos, mesmo com a proteção de arquivos e memória funcionando. É responsabilidade da *segurança* defender um sistema de ataques externo e interno. Esses ataques abrangem um vasto grupo e incluem vírus e vermes, ataques de recusa de serviço (que usam todos os recursos de um sistema e assim mantêm 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 deixam a prevenção para a política adotada ou para um software adicional. Devido ao aumento alarmante dos incidentes de segurança, os recursos de segurança do sistema operacional representam uma área de pesquisa e desenvolvimento em rápido crescimento. A segurança é discutida 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 usuário e dos *identificadores de usuário* (*IDs de usuário*) associados. No linguajar do Windows Vista, 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 tem de estar legível para o usuário, ele é convertido novamente para o nome de usuário através da lista de nomes de usuário.

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 usarmos 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, o ID de usuário e o ID de grupo para 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 restrito, por exemplo. Os sistemas operacionais fornecem diversos 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 desativar os privilégios adicionais ou ser concluído.

1.10 Sistemas 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 do processamento, a funcionalidade, a disponibilidade dos dados e a confiabilidade. Al-

guns sistemas operacionais generalizam o acesso à rede como um tipo de acesso a arquivo, com os detalhes da conexão em rede contidos no driver de dispositivo da interface de rede. Outros obrigam os usuários a chamarem especificamente as funções da rede. Geralmente, os sistemas contêm uma combinação dos dois modos — por exemplo, o FTP e o NFS. Os protocolos que criam

um sistema distribuído podem afetar bastante a utilidade e popularidade desse sistema.

Uma *rede*, em uma descrição simples, é uma via de comunicação entre dois ou mais sistemas. Os sistemas distribuídos dependem da 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, embora o ATM e outros protocolos estejam sendo amplamente usados. Da mesma forma, o suporte dos sistemas operacionais aos protocolos varia. A maioria dos sistemas operacionais dá suporte ao TCP/IP, inclusive os sistemas operacionais Windows e UNIX. Alguns sistemas dão suporte a protocolos proprietários para satisfazer suas necessidades. Para um sistema operacional, um protocolo de rede só precisa 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 andar ou um prédio. 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 permitir comunicação por

uma distância de vários quilômetros, criando na verdade uma *rede de pequena distância* como as que poderíamos encontrar em residências.

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 para 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 através da rede e que inclui um esquema de comunicação que permite que diferentes processos em computadores distintos 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 possa se comunicar com outros computadores conectados. Um sistema operacional distribuído fornece um ambiente menos autônomo: os diferentes sistemas operacionais se comunicam com proximidade suficiente para dar a impressão de que apenas um sistema operacional está controlando a rede.

Abordamos redes de computador e sistemas distribuídos nos Capítulos 16 a 18.

IFPB-JP

BIBLIOTECA NILO PECANHA

1.11 Sistemas de Uso Específico

Até agora a discussão enfocou sistemas de computação de uso geral com os quais estamos familiarizados. No entanto, há outros tipos de sistemas de computação cujas funções são mais limitadas e cujo objetivo é lidar com domínios de computação limitados.

1.11.1 Sistemas Embutidos de Tempo Real

O computador embutido é o tipo de computador mais disseminado que existe. 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 executar tarefas muito específicas. Os sistemas em que operam costumam ser primitivos e, portanto, os sistemas operacionais fornecem recursos limitados. Geralmente, eles têm pouca ou nenhuma interface com o usuário, preferindo se dedicar ao monitoramento e gerenciamento dos dispositivos de hardware, como os motores de carro e os braços robóticos.

Esses sistemas embutidos variam consideravelmente. Alguns são computadores de uso geral, executando sistemas operacionais padrões — como o UNIX — 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. Há ainda outros que são dispositivos de hardware com circuitos integrados de aplicação específica (*ASICs — application-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 computado-

rizadas, de modo 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 acabou o leite.

Os sistemas embutidos quase sempre executam *sistemas operacionais de tempo real*. Um sistema de tempo real é usado quando rígidos requisitos de tempo são exigidos da operação de um processador ou do fluxo de dados; portanto, geralmente ele é usado 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 com base em imagens, sistemas de controle industrial e certos sistemas de exibição são sistemas de tempo real. Alguns sistemas de injecção 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 para 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 19, abordamos os sistemas embutidos de tempo real com mais detalhes. No Capítulo 5, 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, no Capítulo 22, descrevemos os componentes de tempo real do sistema operacional Windows XP.

1.11.2 Sistemas Multimídia

A maioria dos sistemas operacionais é projetada para manipular dados convencionais como arquivos de texto, programas, documentos de processamento de texto e planilhas. No entanto, uma tendência recente da tecnologia é a incorporação de *dados multimídia* em sistemas de computação. Os dados multimídia são compostos por arquivos de áudio e vídeo assim como por arquivos convencionais. Esses dados diferem dos dados convencionais porque os dados multimídia — como os quadros de vídeo — devem ser distribuídos (transmitidos em fluxo) de acordo com certas restrições de tempo (por exemplo, 30 quadros por segundo).

O termo multimídia descreve um vasto conjunto de aplicações de uso disseminado correntemente. Elas incluem arquivos de áudio como o MP3, filmes em DVD, videoconferência e clipes de vídeo curtos de propagandas de filmes ou manchetes jornalísticas baixados da Internet. As aplicações multimídia também podem incluir webcasts ao vivo (transmissão através da World Wide Web) de palestras ou eventos esportivos e até mesmo webcams ao vivo que permitem que um usuário em Manhattan observe clientes em um café em Paris. Uma aplicação multimídia não precisa ser em áudio ou vídeo; em vez disso, ela inclui geralmente uma combinação dos dois. Por exemplo, um filme pode ser composto por trilhas de áudio e vídeo separadas. As aplicações multimídia também não precisam ser distribuídas apenas para computadores pessoais de mesa. Cada vez mais, elas estão sendo direcionadas para dispositivos menores, inclusive PDAs e telefones celulares. Por exemplo, um negociante de ações pode ter as cotações distribuídas sem fio e em tempo real para seu PDA.

No Capítulo 20, examinamos as demandas das aplicações multimídia, descrevemos como os dados multimídia diferem dos dados convencionais e explicamos como a natureza desses dados afeta o projeto de sistemas operacionais que dão suporte aos requisitos de sistemas multimídia.

1.11.3 Sistemas Móveis

Os *sistemas móveis* incluem os assistentes digitais pessoais (PDAs), como os PCs de mão ou de bolso, e os telefones celulares, muitos dos quais empregam sistemas operacionais embutidos de uso específico. Desenvolvedores de sistemas e aplicações móveis enfrentam muitos desafios, a maioria deles devido ao tamanho limitado desses dispositivos. Por exemplo, normalmente um PDA tem cerca de 12,5 centímetros de altura e 7,5 centímetros de largura e pesa menos de 250 gramas. Por causa de seu tamanho, grande parte dos dispositivos móveis tem pouca quantidade de memória, processadores lentos e pequenas telas. Examinemos cada uma dessas limitações.

A quantidade de memória física em um dispositivo móvel depende do dispositivo, mas normalmente ela tem entre 1 MB e 1 GB. (Compare isso com um PC ou estação de trabalho comum, que pode ter vários gigabytes de memória.) Como resultado, o sistema operacional e as aplicações devem gerenciar a memória eficientemente. Isso inclui o retorno de toda a memória alocada para o gerenciador de memória quando esta não está sendo usada. No Capítulo 9, examinaremos a memória virtual, que permite aos desenvolvedores escrever programas que se comportam como se o sistema tivesse mais memória do que a disponível fisicamente. Atualmente, poucos dispositivos móveis usam técnicas de memória virtual; portanto, os desenvolvedores de programas devem trabalhar de acordo com restrições de memória física limitada.

Um segundo motivo de preocupação para os desenvolvedores de dispositivos móveis é a velocidade do processador usado nos dispositivos. Os processadores da maioria dos dispositivos móveis são executados em apenas uma fração da velocidade do processador de um PC. Processadores mais rápidos requerem mais energia. A inclusão de um processador mais rápido em um dispositivo móvel demandaria uma bateria maior, que ocuparia mais espaço e teria de ser substituída (ou recarregada) com mais frequência. Grande parte dos dispositivos móveis usa processadores menores e mais lentos que consomem menos energia. Portanto, o sistema operacional e as aplicações devem ser projetados de modo a não carregar o processador.

O último desafio que os desenvolvedores de programas para dispositivos móveis enfrentam é o I/O. A falta de espaço físico limita os métodos de entrada a pequenos teclados, reconhecimento de texto manuscrito ou teclados com base em pequenas telas. As pequenas telas de exibição limitam as opções de saída. Enquanto um monitor de computador doméstico pode medir até 75 centímetros, geralmente a tela de um dispositivo móvel não tem mais do que 7,5 centímetros quadrados. Tarefas familiares como ler e-mails e navegar em páginas da Web devem ser condensadas em telas menores. Uma abordagem para a exibição do conteúdo de páginas da Web é o *recorte Web*, onde só um pequeno subconjunto de uma página da Web é liberado e exibido no dispositivo móvel.

Alguns dispositivos móveis usam tecnologia sem fio, como o BlueTooth ou o padrão 802.11, permitindo o acesso remoto a e-mails e a navegação na Web. Telefones celulares com conectividade com a Internet se enquadram nessa categoria. No entanto, para PDAs que não fornecem acesso sem fio, normalmente o download de dados requer que primeiro o usuário faça o download dos dados em um PC ou estação de trabalho e, em seguida, baixe os dados no PDA. Alguns PDAs permitem que os dados sejam copiados diretamente de um dispositivo para outro com o uso de uma conexão infravermelha.

Geralmente, as limitações na funcionalidade dos PDAs são compensadas por sua conveniência e portabilidade. Seu uso continua a se expandir conforme as conexões de rede se tornam mais disponíveis e outras opções, como as câmeras digitais e reprodutores de MP3, ampliam sua utilidade.

1.12 Ambientes de Computação

Até agora, fornecemos uma visão geral da organização do sistema de computação e dos principais componentes do sistema operacional. Terminaremos com um breve resumo de como esses sistemas são usados em vários ambientes de computação.

1.12.1 Computação Tradicional

Conforme a computação amadurece, as fronteiras que separam muitos dos ambientes de computação tradicionais estão ficando

menos claras. Considere o “típico ambiente de escritório”. 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 difícil e a portabilidade era conseguida com o uso de computadores laptop. Terminais conectados a mainframes também predominavam em muitas empresas, com opções ainda menores de acesso remoto e portabilidade.

A tendência corrente caminha em direção ao fornecimento de mais modos de acesso a esses ambientes de computação. As tecnologias Web estão estendendo as fronteiras da computação tradicional. Empresas estabelecem *portais*, que fornecem acessibilidade Web a seus servidores internos. *Computadores em rede* são essencialmente terminais que reconhecem a computação baseada na Web. Computadores móveis podem ser sincronizados com PCs para permitir o uso de informações da empresa com maior portabilidade. PDAs portáteis também podem se conectar com *redes sem fio* para usar o portal da empresa na Web (assim como os diversos 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. Atualmente, as velocidades de conexão de rede que só eram disponibilizadas a custo alto são relativamente baratas, concedendo aos usuários domésticos mais acesso a mais dados. Essas conexões de dados velozes estão permitindo que computadores domésticos sirvam páginas da Web e operem redes que incluem impressoras, PCs clientes e servidores. Algumas residências têm até mesmo *firewalls* para proteger suas redes de falhas de segurança. Esses firewalls custavam milhares de dólares há alguns anos e nem mesmo existiam há uma década.

Na última metade do século anterior, os recursos de computação eram escassos. (Antes disso, nem existiam!) Durante algum tempo, os sistemas operavam em modo batch ou eram interativos. Os sistemas batch processavam jobs em lotes, com entrada predeterminada (proveniente de arquivos ou outras fontes de dados). Os sistemas interativos esperavam a entrada 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 alternar processos rapidamente na CPU, dando a cada usuário uma parcela dos recursos.

Atualmente, os sistemas tradicionais de tempo compartilhado são raros. A mesma técnica de scheduling ainda é usada em estações de trabalho e servidores, mas geralmente os processos são todos de propriedade do mesmo usuário (ou de um único usuário e do sistema operacional). Os processos de usuário, e os processos de sistema que fornecem serviços para o usuário, são gerenciados para que cada um receba regularmente 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 poderiam estar executando diferentes tarefas ao mesmo tempo.

1.12.2 Computação Cliente-Servidor

Conforme os PCs ficavam mais velozes, poderosos e baratos, os projetistas iam abandonando a arquitetura de sistemas centralizados. Atualmente os terminais conectados a sistemas centralizados estão sendo substituídos por PCs. Da mesma forma, a funcionalidade de interface com o usuário antes manipulada diretamente por sistemas centralizados está cada vez mais sendo manipulada por PCs. Como resultado, muitos dos sistemas atuais agem como *sistemas servidores* para atender solicitações geradas por *sistemas clientes*. Esse tipo de sistema distribuído especializado, chamado de *sistema cliente-servidor*, tem a estrutura geral mostrada na Figura 1.13.

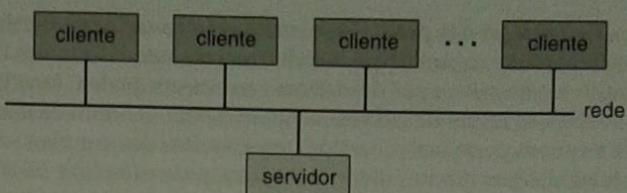


Figura 1.13 Estrutura geral de um sistema cliente-servidor.

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

- O *sistema servidor de processamento* fornece uma interface através da qual um cliente pode enviar uma solicitação para executar uma ação (por exemplo, ler dados); em resposta, o servidor executa a ação e retorna resultados para o 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 seria um servidor Web que distribuisse arquivos para clientes que estivessem executando navegadores da Web.

1.12.3 Computação entre Pares (*peer-to-peer*)

Outra estrutura dos sistemas distribuídos é o modelo de sistema entre pares (P2P). Nesse modelo, clientes e servidores não são diferentes; 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, primeiro o nó deve ingressar na rede de pares. Uma vez que o nó tiver ingressado na rede, poderá começar a fornecer serviços para — e solicitar serviços de — outros nós da rede. A determinação de que serviços estarão disponíveis é feita de uma entre duas maneiras:

- Quando um nó ingressa em uma rede, ele registra seu serviço em um serviço de pesquisa centralizado da rede. Qualquer nó que desejar um serviço específico primeiro fará contato com esse serviço de pesquisa centralizado para determinar que nó fornece o serviço. O resto da comunicação ocorre entre o cliente e o fornecedor do serviço.
- O par que está atuando como cliente deve primeiro descobrir que nó fornece o serviço desejado transmitindo a solicitação do serviço para todos os outros nós da rede. O nó (ou nós) que fornece esse 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 disponibilizados por outros pares da rede.

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 permitem que os pares troquem arquivos uns com os outros. O sistema Napster usa uma abordagem semelhante ao primeiro tipo descrito acima: um servidor centralizado mantém um índice de todos os arquivos armazenados nos nós pares participantes da rede Napster e a troca real de arquivos

ocorre entre os nós pares. O sistema Gnutella usa uma técnica semelhante ao segundo tipo: um cliente transmite solicitações de arquivo para outros nós do sistema e os nós que podem atender à solicitação respondem diretamente ao cliente. O futuro da troca de arquivos permanece incerto porque muitos dos arquivos são protegidos por direitos autorais (música, por exemplo) e há leis que controlam a distribuição de material com direitos autorais. De qualquer forma, no entanto, a tecnologia entre pares fará parte sem dúvida do futuro de muitos serviços, como a pesquisa, a troca de arquivos e o correio eletrônico.

1.12.4 Computação Baseada na Web

A Web se tornou onipresente, levando a mais acessos por uma maior variedade de dispositivos do que se imaginava há alguns anos. Os PCs ainda são os dispositivos de acesso predominantes,

com estações de trabalho, PDAs portáteis e até telefones celulares também fornecendo acesso.

A computação na Web aumentou a ênfase na conexão de rede. Dispositivos que anteriormente não funcionavam em rede agora incluem o acesso com ou sem fio. Dispositivos que funcionavam em rede agora têm uma conectividade mais veloz, fornecida pelo aperfeiçoamento da tecnologia de rede, pela otimização do código de implementação da rede ou ambos.

A implementação da computação baseada na Web fez surgir novas categorias de dispositivos, como os *balanceadores de carga*, que distribuem conexões de rede entre um pool de servidores semelhantes. Sistemas operacionais como o Windows 95, que atuavam como clientes Web, evoluíram para o Linux e o Windows XP, que podem atuar como servidores e clientes Web. De modo geral, a Web aumentou a complexidade dos dispositivos porque seus usuários precisam que eles sejam habilitados para a Web.

1.13 Sistemas Operacionais de Código-Fonte Aberto

O estudo dos sistemas operacionais, como mencionado anteriormente, foi facilitado pela disponibilidade de uma vasta quantidade de versões de código-fonte aberto. Os *sistemas operacionais de código-fonte aberto* são aqueles disponibilizados em formato de código-fonte, em vez de 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*. 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 através de *engenharia reversa* — é bem mais trabalhoso, e itens úteis como os comentários nunca são recuperados. O aprendizado dos sistemas operacionais através da investigação do código-fonte, em vez de através da leitura de explicações sobre esse código, pode ser extremamente útil. Com o código-fonte em mãos, um aluno pode modificar o sistema operacional e, em seguida, compilar e executar o código para testar essas alterações, o que é outra 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-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 os defensores do código-fonte aberto argumentam que os bugs tendem a ser encontrados e corrigidos mais rapidamente devido ao número de pessoas que usam e visualizam o código. Empresas que faturam com a venda de seus programas tendem a hesitar em abrir seu código-fonte, mas a Red Hat, a SUSE, a Sun e várias outras empresas estão fazendo exatamente isso e mostrando que empresas comerciais se beneficiam, em vez de se prejudicarem, quando abrem seu código-fonte. A receita pode ser gerada através de contratos de suporte e da venda do hardware em que o software é executado, por exemplo.

1.13.1 História

Nos primórdios da computação moderna (isto é, os 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 usuário "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, reunir-las em fitas e distribuir as fitas para 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, os ajudava a atingir esse objetivo, assim como a proteger seu código e suas ideias contra seus rivais. Outro problema envolvia materiais protegidos por direitos autorais. Sistemas operacionais e outros programas podem limitar a capacidade de reprodução de filmes e música ou a exibição de livros eletrônicos a computadores autorizados. 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 Millennium Copyright Act (DMCA), tornam ilegal usar de engenharia reversa em código protegido pelo DRM ou tentar burlar a proteção contra cópia.

Para contra-atacar a tentativa de limitar o uso e a redistribuição de software, em 1983, Richard Stallman 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 o 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 feita no código-fonte seja lançada sob a mesma licença GPL.

1.13.2 Linux

Como exemplo de sistema operacional de código-fonte aberto, considere o *GNU/Linux*. O projeto GNU produziu muitas ferramentas compatíveis com o UNIX, inclusive compiladores, editores e utilitários, mas nunca lançou um kernel. Em 1991, um aluno da Finlândia, Linus Torvalds, lançou um kernel rudimentar baseado no UNIX usando os compiladores e ferramentas do GNU e solicitou contribuições no mundo todo. O advento da Internet significava 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 milhares de programadores.

O sistema operacional GNU/Linux resultante gerou centenas de *distribuições* exclusivas, ou versões personalizadas, do sistema. As principais distribuições incluem a RedHat, a SUSE, a Fedora, a Debian, a Slackware e a Ubuntu. As distribuições variam em função, utilidade, aplicações instaladas, suporte de hardware, interface com o usuário e finalidade. Por exemplo, o RedHat Enterprise Linux foi preparado para uso comercial amplo. 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, o “PCLinuxOS Supergamer DVD”, é um *LiveDVD* que inclui drivers gráficos e jogos. Um jogador pode executá-lo em qualquer sistema compatível simplesmente inicializando-o a partir do DVD. Quando o jogador terminar, uma reinitialização do sistema o reconfigurará com o sistema operacional que estava instalado.

O acesso ao código-fonte do Linux varia por versão. Aqui, consideramos o Ubuntu Linux. O Ubuntu é uma distribuição popular do Linux que existe em vários tipos, inclusive os adaptados para desktops, servidores e alunos. Seu criador grava e envia gratuitamente os DVDs contendo o código binário e o código-fonte (o que ajuda a torná-lo popular). As etapas a seguir descrevem uma maneira de examinar o código-fonte do kernel do Ubuntu em sistemas que dão suporte à ferramenta gratuita “VMware Player”:

- Baixe o player a partir de <http://www.vmware.com/download/player/> e instale-o em seu sistema.
- Baixe uma máquina virtual contendo o Ubuntu. Centenas de “mecanismos”, ou imagens de máquina virtual, pré-instalados com sistemas operacionais e aplicações, estão disponíveis na VMware em <http://www.vmware.com/appliances/>.
- Inicialize a máquina virtual dentro do VMware Player.
- Obtenha o código-fonte da versão do kernel de interesse, como a 2.6, executando `wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.18.1.tar.bz2` dentro da máquina virtual do Ubuntu.
- Descompacte o arquivo tar baixado usando `tar xjf linux-2.6.18.1.tar.bz2`.
- Examine o código-fonte do kernel do Ubuntu, que agora está em `./linux-2.6.18.1`.

Para obter mais informações sobre o Linux, consulte o Capítulo 21. Para saber mais sobre máquinas virtuais, consulte a Seção 2.8.

1.13.3 BSD UNIX

O *BSD UNIX* tem uma história mais longa e complicada do que o Linux. Ele surge em 1978 como um derivado do UNIX da AT&T. Versões da Universidade da Califórnia em Berkeley (UCB) vinham na forma de código binário e código-fonte, mas não eram de

código-fonte aberto porque uma licença da AT&T era requerida. 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, acabou sendo lançada 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 examinar o código-fonte do FreeBSD, baixe a imagem da máquina virtual da versão de interesse e inicialize-a dentro do VMware, como descrito acima para o Ubuntu 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 existentes em `/usr/src/sys/vm`.

O Darwin, o principal componente do kernel do MAC, foi baseado no BSD UNIX e também é de código-fonte aberto. O código-fonte está disponível em <http://www.opensource.apple.com/darwinsource/>. Toda versão do MAC tem seus componentes de fonte aberta postados nesse site. O nome do pacote que contém o kernel é “xnu”. O código-fonte do kernel do MAC revisão 1228 (o código-fonte do MAC Leopard) pode ser encontrado em www.opensource.apple.com/darwinsource/tarballs/apsl/xnu-1228.tar.gz. A Apple também fornece várias ferramentas de desenvolvedor, documentação e suporte em <http://connect.apple.com>. Para obter mais informações, consulte o Apêndice A.

1.13.4 Solaris

O *Solaris* é o sistema operacional comercial da Sun Microsystems baseado no 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, ela abriu a fonte de parte do código do Solaris e com o tempo foi aumentando cada vez mais essa base de código-fonte aberto. Infelizmente, o Solaris não é todo de código-fonte aberto, porque parte do código ainda é de propriedade da AT&T e outras empresas. No entanto, pode ser compilado a partir do código-fonte aberto e vinculado a binários dos componentes de código-fonte fechado, logo, ainda pode ser examinado, modificado, compilado e testado.

O código-fonte está disponível em <http://opensolaris.org/os/downloads/>. Também estão disponíveis distribuições pré-compiladas baseadas no código-fonte, documentação e grupos de discussão. Não é necessário baixar todo o código-fonte, porque a Sun permite que os visitantes o examinem on-line através de um navegador de código-fonte.

1.13.5 Utilidade

O movimento do software livre está levando vários programadores a criar 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. Os projetos de código-fonte aberto permitem que os alunos usem o código-fonte como ferramenta de aprendizado. Eles podem modificar programas e testá-los, ajudar a encontrar e corrigir bugs ou então examinar sistemas operacionais totalmente desenvolvidos e completos, compiladores, ferramentas, interfaces de usuário e outros tipos de programa. A disponibilidade do código-fonte de projetos históricos, como o Multics, pode ajudar os alunos a entender esses projetos e fornecer conhecimento que auxiliará na implementação de novos projetos.

O GNU/Linux, o BSD UNIX e o Solaris são todos 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 Solaris foram transferidos para o BSD UNIX. As vantagens

do software livre e de código-fonte aberto devem aumentar a quantidade e a qualidade de projetos de código-fonte aberto, levando a um acréscimo na quantidade de pessoas e empresas que usam esses projetos.

1.14 Resumo

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

Para um computador realizar seu trabalho de execução de programas, os programas devem estar na memória principal. A memória principal é a única área de armazenamento ampla que o processador pode acessar diretamente. Trata-se de um array de palavras ou bytes, variando em tamanho de milhares a bilhões. Cada palavra na memória tem seu próprio endereço. Geralmente a memória principal é um dispositivo de armazenamento volátil que perde seu conteúdo quando a energia é desligada ou acaba. A maioria dos computadores fornece a memória secundária como uma extensão da memória principal. A memória secundária fornece um tipo de armazenamento não volátil que pode conter 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 a hierarquia, geralmente o custo por bit diminui, enquanto o tempo de acesso aumenta.

Há várias estratégias diferentes para o projeto em um sistema de computação. Sistemas uniprocessadores têm um único 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 semelhantes 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 utilizar melhor a CPU, os sistemas operacionais modernos empregam a multiprogramação, que permite que vários jobs fiquem 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 se alternam 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 tem duas modalidades: modalidade de usuário e modalidade de kernel. Várias instruções (como as instruções de I/O e as instruções de interrupção) são privilegiadas e só podem ser executadas na 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 (modo 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 a segurança sua e dos usuários. As medidas de proteção são mecanismos que 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.

Os sistemas distribuídos permitem que os usuários compartilhem recursos em hosts geograficamente dispersos conectados através de uma rede de computadores. Serviços podem ser fornecidos através do modelo cliente-servidor ou do modelo entre pares. Em um sistema cluster, várias máquinas podem executar operações com os dados que residem na memória compartilhada e o processamento pode continuar até mesmo quando algum subconjunto de membros do cluster falha.

As LANs e WANs são os dois tipos básicos de redes. As LANs permitem que processadores distribuídos em uma pequena área geográfica se comuniquem, enquanto as WANs permitem que processadores distribuídos em uma área maior se comuniquem. Normalmente as LANs são mais velozes do que as WANs.

Há vários sistemas de computação que servem a finalidades específicas. Entre eles estão os sistemas operacionais de tempo real projetados para ambientes embutidos como os de aparelhos domésticos, automóveis e robôs. Os sistemas operacionais de tempo real têm restrições de tempo fixo bem definidas. O processamento deve ser executado dentro das restrições definidas ou o sistema falhará. Os sistemas multimídia envolvem a distribuição de dados multimídia e geralmente têm requisitos especiais de exibição ou reprodução de áudio, vídeo ou fluxos sincronizados de áudio e vídeo.

Recentemente, a influência da Internet e da World Wide Web encorajou o desenvolvimento de sistemas operacionais que incluem navegadores da Web e softwares de rede e de comunicação como recursos integrantes.

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

O GNU/Linux, o BSD UNIX e o Solaris são todos sistemas operacionais de código-fonte aberto. As vantagens do software livre e de código-fonte aberto devem aumentar a quantidade e a qualidade de projetos de código-fonte aberto, levando a um acréscimo na quantidade de pessoas e empresas que usam esses projetos.

Exercícios Práticos

- 1.1 Quais são as três finalidades principais de um sistema operacional?
- 1.2 Quais são as principais diferenças entre os sistemas operacionais de computadores mainframe e computadores pessoais?
- 1.3 Liste as quatro etapas que são necessárias para a execução de um programa em uma máquina totalmente dedicada — um computador que estiver executando apenas esse programa.
- 1.4 Enfatizamos a necessidade de o sistema operacional usar eficientemente o hardware do computador. Quando é apropriado que o sistema operacional ignore esse princípio e “desperdice” recursos? Por que um sistema assim não está na verdade sendo ineficiente?
- 1.5 Qual é a principal dificuldade que um programador deve superar ao criar um sistema operacional para um ambiente de tempo real?
- 1.6 Considere as diversas definições de *sistema operacional*. Considere se o sistema operacional deve incluir aplicações como navegadores da Web e programas de e-mail. Defenda tanto que ele deve quanto que ele não deve fazer isso e baseie suas respostas.
- 1.7 Como a diferença entre a modalidade de kernel e a modalidade de usuário funciona como um tipo rudimentar de sistema de proteção (segurança)?
- 1.8 Qual das instruções a seguir deve ser privilegiada?
- Configurar o valor do timer.
 - Ler o relógio.
 - Apagar a memória.
 - Emitir uma instrução de exceção.
 - Desativar interrupções.
 - Modificar entradas na tabela de status de dispositivos.
 - Passar da modalidade de usuário para a de kernel.
 - Acessar dispositivo de I/O.
- 1.9 Alguns computadores antigos protegiam o sistema operacional inserindo-o em uma partição da memória que não podia ser modificada pelo job do usuário ou pelo próprio sistema operacional. Descreva duas dificuldades que você acha que poderiam surgir nesse esquema.
- 1.10 CPUs fornecem mais de duas modalidades de operação. Cite dois usos possíveis para essas múltiplas modalidades.
- 1.11 Os timers poderiam ser usados para exibir a hora corrente. Forneça uma breve descrição de como isso pode ser feito.
- 1.12 A Internet é uma LAN ou WAN?

Exercícios

- 1.13 Em um ambiente de multiprogramação e tempo compartilhado, vários usuários compartilham o sistema simultaneamente. Essa situação pode resultar em diversos problemas de segurança.
- Cite dois desses problemas.
 - Podemos assegurar o mesmo nível de segurança tanto em uma máquina dedicada como em uma máquina de tempo compartilhado? Explique sua resposta.
- 1.14 A questão da utilização de recursos assume formas distintas em diferentes tipos de sistemas operacionais. Liste que recursos devem ser gerenciados cuidadosamente nas configurações a seguir:
- Sistemas mainframe ou de minicomputador
 - Estações de trabalho conectadas a servidores
 - Computadores móveis
- 1.15 Em que circunstâncias seria melhor para o usuário usar um sistema de tempo compartilhado em vez de um PC ou uma estação de trabalho monousuária?
- 1.16 Identifique qual das funcionalidades listadas abaixo tem de ter suporte no sistema operacional para (a) dispositivos móveis e (b) sistemas de tempo real.
- Programação batch
 - Memória virtual
 - Tempo compartilhado
- 1.17 Descreva as diferenças entre os multiprocessamentos simétrico e assimétrico. Cite três vantagens e uma desvantagem de sistemas multiprocessadores.
- 1.18 Em que os sistemas clusters diferem de sistemas multiprocessadores? O que é necessário para duas máquinas pertencentes a um cluster cooperarem para fornecer um serviço de alta disponibilidade?
- 1.19 Qual a diferença entre os modelos de sistema distribuído cliente-servidor e entre pares?
- 1.20 Considere um cluster de computadores composto por dois nós executando um banco de dados. Descreva duas maneiras pelas quais o software do cluster gerencie o acesso aos dados no disco. Discuta as vantagens e desvantagens de cada uma.
- 1.21 Em que os computadores em rede são diferentes dos computadores pessoais tradicionais? Descreva alguns cenários de uso em que é vantajoso usar computadores em rede.
- 1.22 Qual é a finalidade das interrupções? Quais são as diferenças entre uma exceção e uma interrupção? As exceções podem ser geradas intencionalmente por um programa de usuário? Caso possam, com que finalidade?
- 1.23 O acesso direto à memória é usado em dispositivos de I/O de alta velocidade para impedir o aumento da carga de execução da CPU.
- Como a CPU se relaciona com o dispositivo para coordenar a transferência?
 - Como a CPU sabe quando as operações da memória foram concluídas?
 - A CPU pode executar outros programas enquanto o controlador de DMA está transferindo dados. Esse processo interfere na execução dos programas de usuário? Caso interfira, que tipos de interferência são gerados?
- 1.24 Alguns sistemas de computação não fornecem um modo privilegiado de operação de hardware. É possível construir um sistema operacional seguro para esses sistemas de computação? Defenda tanto que é quanto que não é possível.
- 1.25 Cite duas razões da utilidade dos caches. Que problemas eles resolvem? Que problemas eles causam? Se um cache puder ser tão amplo quanto o dispositivo para o qual ele está armazenando (por exemplo, um cache com o mesmo espaço de um disco), por que não dar a ele esse espaço e eliminar o dispositivo?
- 1.26 Considere um sistema SMP semelhante ao que é mostrado na Figura 1.6. Ilustre com um exemplo como os dados que residem na memória poderiam ter dois valores diferentes em cada um dos caches locais?
- 1.27 Discuta, com exemplos, como o problema de manter a co-

- erência dos dados armazenados em cache se manifesta nos ambientes de processamento a seguir:
- Sistemas com um único processador
 - Sistemas multiprocessadores
 - Sistemas distribuídos
- 1.28** Descreva um mecanismo que garantiria a proteção da memória impedindo que um programa modificasse a memória associada a outros programas.
- 1.29** Que configuração de rede atenderia melhor os ambientes a seguir:
- O andar de um albergue
 - Um campus universitário
 - Um estado
 - Uma nação
- 1.30** Defina as propriedades essenciais dos tipos de sistema operacional a seguir:
- Batch
 - Interativo
 - De tempo compartilhado
 - De tempo real
 - De rede
 - Paralelo
 - Distribuído
 - em Cluster
 - Móvel
- 1.31** Quais são as desvantagens próprias dos computadores móveis?
- 1.32** Identifique várias vantagens e desvantagens dos sistemas operacionais de código-fonte aberto. Inclua os tipos de pessoas que considerariam cada aspecto uma vantagem ou desvantagem.

Notas Bibliográficas

Brooks [2003] fornece uma visão geral da ciência da computação.

Uma visão geral do sistema operacional Linux é apresentada em Bovet e Cesati [2006]. Solomon e Russinovich [2000] fornecem uma visão geral do Microsoft Windows e detalhes técnicos consideráveis sobre os mecanismos internos e componentes do sistema. Russinovich e Solomon [2005] atualizam essas informações para o Windows Server 2003 e Windows XP. McDougall e Mauro [2007] abordam os mecanismos internos do sistema operacional Solaris. O Mac OS X é apresentado em <http://www.apple.com/macosx>. Os mecanismos internos do Mac OS X são discutidos em Singh [2007].

A abordagem dos sistemas entre pares é encontrada em Paramewaran et al. [2001], Gong [2002], Ripeanu et al. [2002], Agre [2003], Balakrishnan et al. [2003] e Loo [2003]. Uma discussão dos sistemas de tempo compartilhado entre pares pode ser encontrada em Lee [2003]. Uma boa abordagem da computação em cluster é fornecida por Buyya [1999]. Avanços recentes na computação em cluster são descritos por Ahmed [2000]. Uma pesquisa sobre questões relacionadas ao suporte dado pelos sistemas operacionais a sistemas distribuídos pode ser encontrada em Tanenbaum e Van Renesse [1985].

Muitos livros de conteúdo abrangente abordam sistemas operacionais, inclusive os de Stallings [2000b], Nutt [2004] e Tanenbaum [2001].

Hamacher et al. [2002] descrevem a organização dos computadores e McDougall e Laudon [2006] discutem os processadores multicore. Hennessy e Patterson [2007] fornecem uma abordagem dos buses e sistemas de I/O e da arquitetura dos sistemas em geral. Blaauw e Brooks [1997] descrevem detalhes da arquitetura de muitos sistemas de computação, inclusive vários da IBM. Stokes [2007] fornece uma introdução ilustrada sobre microprocessadores e sobre a arquitetura dos computadores.

Memórias cache, inclusive a memória associativa, são descritas e analisadas por Smith [1982]. Esse texto também inclui uma extensa bibliografia sobre o assunto.

Discussões relacionadas à tecnologia de disco magnético são apresentadas por Freedman [1983] e por Harker et al. [1981]. Os discos ópticos são abordados por Kenville [1982], Fujitani [1984], O'Leary e Kitts [1985], Gait [1988], e Olsen e Kenley [1989]. Discussões sobre disquetes são oferecidas por Pechura e Schoeffler [1983] e por Sarisky [1983]. Discussões gerais relacionadas à tecnologia de armazenamento de massa são oferecidas por Chi [1982] e por Hoagland [1985].

Kurose e Ross [2005] e Tanenbaum [2003] fornecem visões gerais de redes de computadores. Fortier [1989] apresenta uma discussão detalhada sobre hardware e software de rede. Kozierok [2005] discute o TCP detalhadamente. Mullender [1993] fornece uma visão geral de sistemas distribuídos. Wolf [2003] discute avanços recentes no desenvolvimento de sistemas embutidos. Questões relacionadas a dispositivos móveis podem ser encontradas em Myers e Beigl [2003] e Di Pietro e Mancini [2003].

Uma discussão completa da história do código-fonte aberto e seus benefícios e desafios é encontrada em Raymond [1999]. A história da atividade de hacking é discutida em Levy [1994]. A Fundação de Software Livre publicou sua filosofia em seu site da Web: <http://www.gnu.org/philosophy/free-software-for-freedom.html>. Instruções detalhadas sobre como construir o kernel do Ubuntu Linux se encontram em http://www.howtoforge.com/kernel_compilation_ubuntu. Os componentes de código-fonte aberto do MAC estão disponíveis em <http://developer.apple.com/open-source/index.html>.

A Wikipedia (http://en.wikipedia.org/wiki/Richard_Stallman) tem uma entrada informativa sobre Richard Stallman.

O código-fonte do Multics está disponível em http://web.mit.edu/multics-history/source/Multics_Internet_Server/Multics_sources.html.

Estruturas do Sistema Operacional

Um sistema operacional fornece o ambiente dentro do qual os programas são executados. Internamente, os sistemas operacionais variam muito em sua composição, já que estão organizados em muitas linhas diferentes. O projeto de um novo sistema operacional é uma tarefa de peso. É importante que os objetivos do sistema sejam bem definidos antes de o projeto começar. Esses objetivos formarão a base das escolhas feitas entre vários algoritmos e estratégias.

Podemos considerar um sistema operacional de acordo com vários critérios. Um enfoca os serviços que o sistema fornece; outro, a interface que ele torna disponível para usuários e programadores; e um terceiro enfoca seus componentes e suas interconexões. Neste capítulo, examinaremos todos os três aspectos dos sistemas operacionais, mostrando os pontos de vista de usuários, programadores e projetistas dos sistemas. Consideraremos que

serviços um sistema operacional fornece, como eles são fornecidos, como são depurados e que metodologias existem para o projeto desses sistemas. Para concluir, descreveremos como os sistemas operacionais são criados e como um computador inicia seu sistema operacional.

OBJETIVOS DO CAPÍTULO

- Descrever os serviços que um sistema operacional fornece para usuários, processos e outros sistemas.
- Discutir as diversas maneiras de estruturar um sistema operacional.
- Explicar como os sistemas operacionais são instalados e personalizados e como são inicializados.

2.1 Serviços do Sistema Operacional

Um sistema operacional fornece um ambiente para a execução de programas. Ele fornece certos serviços para programas e para os usuários desses programas. É claro que os serviços específicos fornecidos diferem de um sistema operacional para o outro, mas podemos identificar classes comuns. Esses serviços do sistema operacional são fornecidos visando a conveniência do programador, para tornar mais fácil a tarefa de programar. A Figura 2.1 mostra uma representação dos diversos serviços do sistema operacional e como eles estão relacionados.

Um conjunto de serviços do sistema operacional fornece funções que são úteis para o usuário.

- **Interface de usuário.** Quase todos os sistemas operacionais têm uma *interface de usuário* (*UI* — *user interface*). Essa interface pode assumir várias formas. Uma é a *interface de linha de comando* (*CLI* — *command-line interface*) *DTrace*, que usa comandos de texto e um método de inserção (por exemplo, um programa para permitir a inserção e edição de comandos). Outra é a *interface batch*, em que os comandos e suas diretivas de controle são inseridos em arquivos e esses arquivos são executados. O mais comum é o uso de uma *interface gráfica de usuário* (*GUI* — *graphical user interface*). Nesse caso, a interface é um sistema de janelas com um dispositivo apontador para o direcionamento de I/O, a seleção de menus e a escolha de opções e um teclado para inserção de texto. Alguns sistemas fornecem duas dessas variações ou todas as três.
- **Execução de programas.** O sistema deve ser capaz de carregar um programa na memória e executar esse programa,

O programa tem de poder encerrar sua execução, normal ou anormalmente (indicando erro).

- **Operações de I/O.** Um programa em execução pode precisar de operações de I/O e isso pode envolver um arquivo ou um dispositivo de I/O. Para dispositivos específicos, funções especiais podem ser desejáveis (como a gravação em um drive de CD ou DVD ou a limpeza de uma tela). Por eficiência e proteção, geralmente os usuários não podem controlar os dispositivos de I/O diretamente. Portanto, o sistema operacional deve fornecer um meio de execução de operações de I/O.
- **Manipulação do sistema de arquivos.** O sistema de arquivos é de especial interesse. É claro que os programas têm de ler e gravar arquivos e diretórios. Eles também precisam criar e excluí-los pelo nome, procurar um arquivo específico e listar informações de arquivos. Para concluir, alguns programas incluem o gerenciamento de permissões para liberar ou negar acesso a arquivos ou diretórios com base em quem possui os arquivos. Muitos sistemas operacionais fornecem vários sistemas de arquivos, em algumas situações para permitir a escolha pessoal e em outras para fornecer recursos específicos ou características de desempenho.
- **Comunicações.** Há muitas situações em que um processo precisa trocar informações com outro processo. Essa comunicação pode ocorrer entre processos que estão sendo executados no mesmo computador ou entre processos sendo executados em sistemas de computação diferentes conectados.

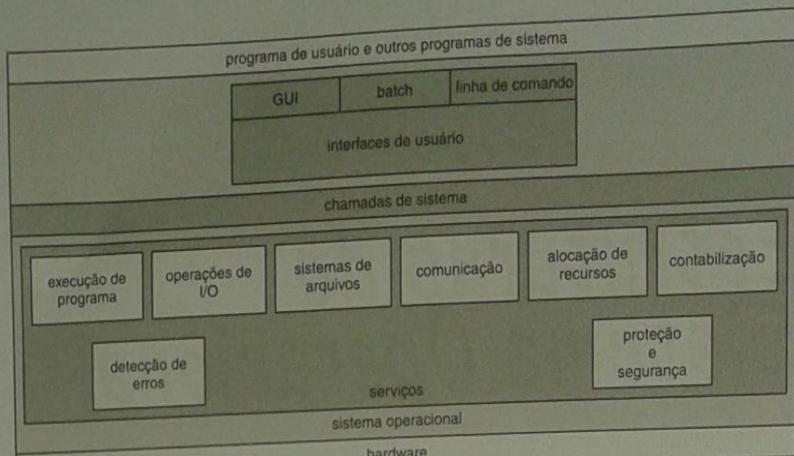


Figura 2.1 Uma visão dos serviços do sistema operacional.

tados por uma rede de computadores. As comunicações podem ser implementadas através de *memória compartilhada* ou da *troca de mensagens*, em que pacotes de informações são transmitidos entre processos pelo sistema operacional.

- **Detecção de erros.** O sistema operacional tem que estar sempre atento a possíveis erros. Os erros podem ocorrer no hardware da CPU e da memória (como um erro de memória ou a falta de energia), em dispositivos de I/O (como um erro de paridade em fita, uma falha de conexão na rede ou a falta de papel na impressora) e no programa do usuário (como um overflow aritmético, uma tentativa de acessar um local inválido na memória ou o uso excessivo de tempo da CPU). Para cada tipo de erro, o sistema operacional deve tomar a medida adequada para garantir o processamento correto e consistente. É claro que há variações em como os sistemas operacionais reagem a erros e os corrigem. Recursos de depuração podem melhorar muito as possibilidades de uso eficiente do sistema pelo usuário e pelo programador.

Existe outro conjunto de funções do sistema operacional cujo objetivo não é ajudar o usuário, e sim assegurar a operação eficiente do próprio sistema. Sistemas com vários usuários podem ganhar eficiência compartilhando os recursos do computador entre os usuários.

- **Alocação de recursos.** Quando há muitos usuários ou jobs ativos ao mesmo tempo, é necessário alocar recursos para cada um deles. Vários tipos diferentes de recursos são gerenciados pelo sistema operacional. Alguns (como os ciclos da CPU, a memória principal e o armazenamento de arquivos) podem ter um código de alocação especial, enquanto outros (como os dispositivos de I/O) podem ter um código muito mais genérico de solicitação e liberação. Por exem-

plo, na determinação da melhor maneira de usar a CPU, os sistemas operacionais têm rotinas de scheduling da CPU que levam em consideração a velocidade da CPU, os jobs que devem ser executados, a quantidade de registradores disponíveis e outros fatores. Também podem existir rotinas de alocação de impressoras, modems, drives de armazenamento USB e outros dispositivos periféricos.

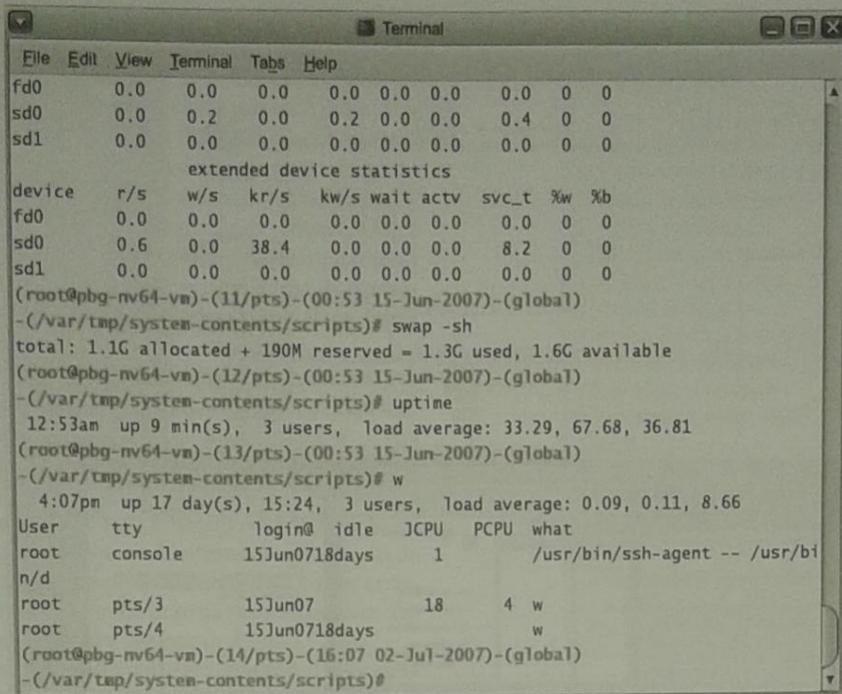
- **Contabilização.** Queremos controlar quais usuários utilizam que quantidade e que tipos de recursos do computador. Essa monitoração pode ser usada a título de responsabilidade (para que os usuários possam ser cobrados) ou simplesmente para acumulação de estatísticas de uso. As estatísticas de uso podem ser uma ferramenta valiosa para pesquisadores que quiserem reconfigurar o sistema para melhorar os serviços de processamento.
- **Proteção e segurança.** Os proprietários de informações armazenadas em um sistema de computação multiusuário ou em rede podem querer controlar o uso dessas informações. Quando vários processos separados são executados concorrentemente, um processo não pode interferir na operação dos outros ou do próprio sistema operacional. Proteção significa garantir que qualquer acesso a recursos do sistema seja controlado. A segurança do sistema contra invasores também é importante. Essa segurança começa com a exigência de que cada usuário se autentique junto ao sistema, geralmente através de uma senha, para ganhar acesso aos seus recursos. Ela se estende à defesa de dispositivos externos de I/O, incluindo modems e adaptadores de rede, contra tentativas de acesso ilegal e à gravação de todas essas conexões para a detecção de invasões. Para um sistema estar protegido e seguro, precauções devem ser tomadas em toda a sua extensão. A força de uma corrente se mede pelo elo mais fraco.

2.2 Interface entre o Usuário e o Sistema Operacional

Mencionamos anteriormente que há várias maneiras de os usuários se comunicarem com o sistema operacional. Aqui, discutiremos duas abordagens básicas. Uma fornece uma interface de linha de comando, ou *interpretador de comandos*, que permite que os usuários insiram diretamente os comandos a serem executados pelo sistema operacional. A outra permite que os usuários se comuniquem com o sistema operacional através de uma interface gráfica de usuário, ou *GUI*.

2.2.1 Interpretador de Comandos

Alguns sistemas operacionais incluem o interpretador de comandos no kernel. Outros, como o Windows XP e o UNIX, tratam o interpretador de comandos como um programa especial que está sendo executado quando um job é iniciado ou quando um usuário estabelece sua primeira conexão (em sistemas interativos). Em sistemas onde é possível escolher entre vários interpretadores de comandos, os interpretadores são conhecidos como *shells*. Por



```

Terminal
File Edit View Terminal Tabs Help
fd0      0.0    0.0    0.0    0.0  0.0  0.0    0.0    0  0
sd0      0.0    0.2    0.0    0.2  0.0  0.0    0.4    0  0
sd1      0.0    0.0    0.0    0.0  0.0  0.0    0.0    0  0
          extended device statistics
device   r/s    w/s    kr/s   kw/s wait activ  svc_t %w %b
fd0      0.0    0.0    0.0    0.0  0.0  0.0    0.0    0  0
sd0      0.6    0.0   38.4    0.0  0.0  0.0    8.2    0  0
sd1      0.0    0.0    0.0    0.0  0.0  0.0    0.0    0  0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-/var/tmp/system-contents/scripts)#
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-/var/tmp/system-contents/scripts)#
uptime
12:53am up 9 min(s), 3 users, load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-/var/tmp/system-contents/scripts)#
w
4:07pm up 17 day(s), 15:24, 3 users, load average: 0.09, 0.11, 8.66
User     tty      login@  idle   JCPU   PCPU what
root    console  15Jun07 18days   1      /usr/bin/ssh-agent -- /usr/bi
n/d
root    pts/3    15Jun07      18      4  w
root    pts/4    15Jun07 18days      w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-/var/tmp/system-contents/scripts)#

```

Figura 2.2 O interpretador de comandos do shell Bourne no Solaris 10.

exemplo, em sistemas UNIX e Linux, um usuário pode escolher entre vários shells diferentes, incluindo o *shell Bourne*, o *shell C*, o *shell Bourne-Again*, o *shell Korn* e outros. Shells de terceiros e shells gratuitos criados por usuários também estão disponíveis. A maioria dos shells fornece funcionalidade semelhante, e geralmente a escolha que o usuário faz do shell a ser usado se baseia na preferência pessoal. A Figura 2.2 mostra o interpretador de comandos do shell Bourne usado no Solaris 10.

A principal função do interpretador de comandos é capturar e executar o próximo comando especificado pelo usuário. Muitos dos comandos fornecidos nesse nível manipulam arquivos: criar, excluir, listar, imprimir, copiar, executar e assim por diante. Os shells do MS-DOS e do UNIX operam dessa forma. Esses comandos podem ser implementados de duas maneiras básicas.

Em uma abordagem, o próprio interpretador de comandos contém o código que executa o comando. Por exemplo, um comando que exclui um arquivo pode fazer o interpretador de comandos saltar para uma seção do seu código que configura os parâmetros e faz a chamada de sistema apropriada. Nesse caso, a quantidade de comandos que pode ser fornecida determina o tamanho do interpretador de comandos, já que cada comando requer seu próprio código de implementação.

Uma abordagem alternativa — usada pelo UNIX, entre outros sistemas operacionais — implementa a maioria dos comandos através de programas do sistema. Nesse caso, o interpretador de comandos definitivamente não entende o comando; ele simplesmente usa o comando para identificar um arquivo a ser carregado na memória e executado. Portanto, o comando UNIX de exclusão de um arquivo

```
rm file.txt
```

procuraria um arquivo chamado *rm*, carregaria o arquivo na memória e o executaria com o parâmetro *file.txt*. A função associada ao comando *rm* seria totalmente definida pelo código existente no arquivo *rm*. Dessa forma, os programadores podem adicionar facilmente novos comandos ao sistema criando novos

arquivos com os nomes apropriados. O programa interpretador de comandos, que pode ser pequeno, não precisa ser alterado para novos comandos serem adicionados.

2.2.2 Interfaces Gráficas de Usuário

Uma segunda estratégia de comunicação com o sistema operacional é através de uma interface de usuário gráfica amigável, ou GUI. Aqui, em vez de inserirem comandos diretamente através de uma interface de linha de comando, os usuários empregam um sistema de janelas e menus baseado no uso do mouse e caracterizado por uma simulação de *área de trabalho*. O usuário move o mouse para posicionar seu ponteiro em imagens, ou *ícones*, na tela (a área de trabalho) que representam programas, arquivos, diretórios e funções do sistema. Dependendo do local em que estiver o ponteiro do mouse, um clique em um de seus botões pode chamar um programa, selecionar um arquivo ou diretório — conhecido como *pasta* — ou abrir um menu contendo comandos.

As interfaces gráficas de usuário surgiram em parte devido a pesquisas que ocorreram no início dos anos 1970 no centro de pesquisas Xerox PARC. A primeira GUI surgiu no computador Xerox Alto em 1973. No entanto, as interfaces gráficas se tornaram mais populares com o advento dos computadores Apple Macintosh nos anos 1980. A interface de usuário do sistema operacional Macintosh (Mac OS) passou por várias alterações com o passar dos anos, sendo a mais significativa a adoção da interface *Aqua*, que surgiu com o Mac OS X. A primeira versão da Microsoft para o Windows — Versão 1.0 — baseava-se na adição de uma GUI ao sistema operacional MS-DOS. Versões posteriores do Windows forneceram alterações superficiais na aparência da GUI, além de várias melhorias em sua funcionalidade, que incluem o Windows Explorer.

Tradicionalmente, os sistemas UNIX têm sido dominados por interfaces de linha de comando. Várias GUIs estão disponíveis, no entanto, inclusive os sistemas Common Desktop Environment

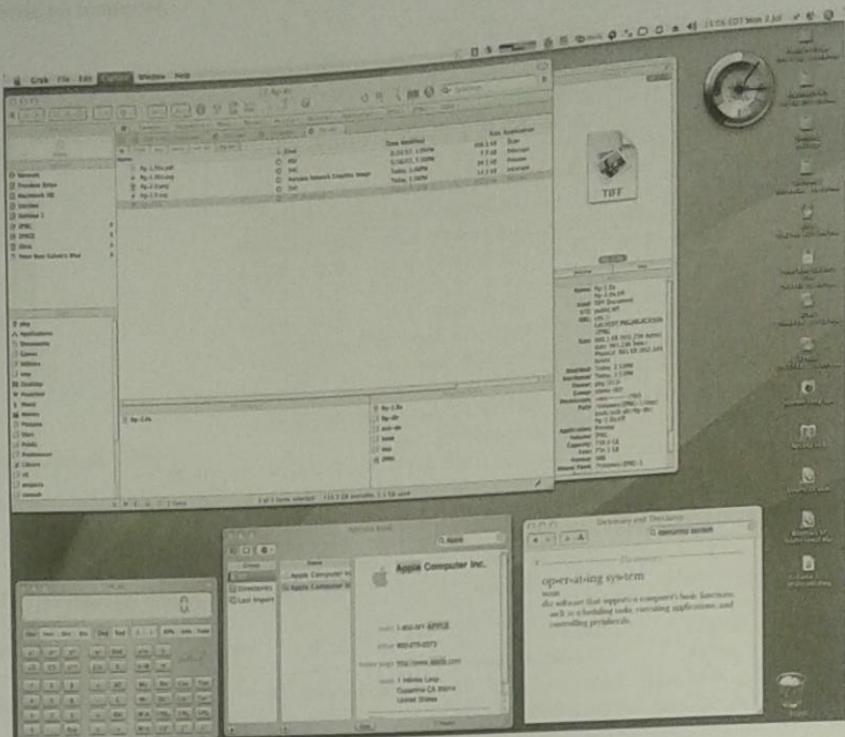


Figura 2.3 A GUI do Mac OS X.

(CDE) e X-Windows, que são comuns em versões comerciais do UNIX, como o Solaris e o sistema AIX da IBM. Além disso, houve desenvolvimentos significativos na aparência das GUIs de vários projetos de *código-fonte aberto*, como o *K Desktop Environment* (ou *KDE*) e a área de trabalho do *GNOME* do projeto *GNU*. As áreas de trabalho do *KDE* e do *GNOME* são executadas no Linux e em vários sistemas UNIX e estão disponíveis sob licenças de código-fonte aberto, o que significa que seu código-fonte pode ser lido e modificado conforme os termos específicos da licença.

A escolha entre usar uma interface de linha de comando ou uma GUI depende em grande parte de preferências pessoais. Como regra geral, muitos usuários do UNIX preferem interfaces de linha de comando, já que com frequência elas fornecem poderosas interfaces de shell. Por outro lado, a maioria dos usuários do Windows gosta de usar o ambiente de GUI do sistema e quase nunca usa a interface de shell do MS-DOS. As diversas alterações pelas quais passaram os sistemas operacionais Macin-

tosh também fornecem um estudo interessante. Historicamente, o Mac OS não fornecia uma interface de linha de comando, sempre demandando que seus usuários se comunicassem com o sistema operacional usando sua GUI. No entanto, com o lançamento do Mac OS X (que é em parte implementado com o uso de um kernel UNIX), agora o sistema operacional fornece tanto uma nova interface Aqua quanto uma interface de linha de comando. A Figura 2.3 é uma tomada de tela da GUI do Mac OS X.

A interface de usuário pode variar de um sistema para outro e até mesmo de usuário para usuário em um sistema. Ela costuma ser removida substancialmente da estrutura do sistema. Portanto, o projeto de uma interface de usuário útil e amigável não depende diretamente do sistema operacional. Neste livro, estamos nos concentrando nos problemas básicos do fornecimento de serviço adequado para programas de usuário. Do ponto de vista do sistema operacional, não fazemos a distinção entre programas de usuário e programas do sistema.

2.3 Chamadas de Sistema

As **chamadas de sistema** fornecem uma interface com os serviços disponibilizados por um sistema operacional. Geralmente essas chamadas estão disponíveis como rotinas escritas em C e C++, embora certas tarefas de baixo nível (por exemplo, tarefas em que o hardware deve ser acessado diretamente) possam ter de ser escritas com o uso de instruções em linguagem de montagem.

Antes de discutirmos como um sistema operacional torna as chamadas de sistema disponíveis, usaremos um exemplo para ilustrar como elas são usadas: a criação de um programa simples para a leitura de dados em um arquivo e sua cópia em outro arquivo. A primeira entrada de que o programa precisará são os nomes dos dois arquivos: o arquivo de entrada e o arquivo de saída. Esses nomes podem ser especificados de muitas formas, dependendo do projeto do sistema operacional. Uma abordagem é aquela em que o programa solicita ao usuário os nomes dos dois arquivos. Em um sistema

interativo, essa abordagem demandará uma sequência de chamadas de sistema, primeiro para exibir uma mensagem de solicitação na tela e, em seguida, para ler a partir do teclado os caracteres que definem os dois arquivos. Em sistemas baseados em ícones e no mouse, geralmente é exibido um menu de nomes de arquivo em uma janela. O usuário pode então usar o mouse para selecionar o nome do arquivo de origem e uma janela pode ser aberta para o nome do arquivo de destino ser especificado. Essa sequência requer muitas chamadas de sistema para operações de I/O.

Uma vez que os dois nomes de arquivo forem obtidos, o programa deve abrir o arquivo de entrada e criar o arquivo de saída. Cada uma dessas operações requer outra chamada de sistema. Também há condições de erro que podem ocorrer para cada operação. Quando o programa tentar abrir o arquivo de entrada, pode descobrir que não há arquivo com esse nome ou que o arquivo está protegido

contra acesso. Nesses casos, o programa deve exibir uma mensagem no console (outra sequência de chamadas de sistema) e então terminar anormalmente (outra chamada de sistema). Se o arquivo de entrada existir, devemos criar um novo arquivo de saída. Podemos descobrir que já existe um arquivo de saída com o mesmo nome. Essa situação pode fazer o programa abortar (uma chamada de sistema) ou podemos excluir o arquivo existente (outra chamada de sistema) e criar um novo (mais uma chamada de sistema). Outra opção, em um sistema interativo, é perguntar ao usuário (através de uma sequência de chamadas de sistema de exibição da mensagem de solicitação e de leitura da resposta no terminal) se deseja substituir o arquivo existente ou abortar o programa.

Agora que os dois arquivos foram definidos, entramos em um loop que lê o arquivo de entrada (uma chamada de sistema) e grava no arquivo de saída (outra chamada de sistema). Cada operação de leitura e gravação deve retornar informações de status referente a várias condições de erro possíveis. Na entrada, o programa pode entender que o fim do arquivo foi alcançado ou que houve uma falha de hardware na leitura (como um erro de paridade). A operação de gravação pode encontrar vários erros, dependendo do dispositivo de saída (não há mais espaço em disco, a impressora está sem papel e assim por diante).

Para concluir, após o arquivo inteiro ser copiado, o programa pode fechar os dois arquivos (outra chamada de sistema), exibir uma mensagem no console ou janela (mais chamadas de sistema) e por fim terminar normalmente (a última chamada de sistema). Como podemos ver, até mesmo programas simples podem usar bastante o sistema operacional. Geralmente, os sistemas executam milhares de chamadas de sistema por segundo. Essa sequência de chamadas de sistema é mostrada na Figura 2.4.

No entanto, a maioria dos programadores nunca vê esse nível de detalhe. Normalmente, os desenvolvedores de aplicações projetam programas de acordo com uma *interface de programação de aplicações* (API — *application programming interface*). A API especifica um conjunto de funções que estão disponíveis para o programador de aplicações, inclusive os parâmetros que são passados para cada função e os valores de retorno que o programador pode esperar. As três APIs mais comuns para programadores de aplicações são a API Win32 para sistemas Windows, a API POSIX para sistemas baseados em POSIX (que incluem praticamente todas as versões do UNIX, Linux e Mac OS X) e a API Java para o projeto de programas que são executados na máquina virtual Java. Lembre que — exceto se especificado — os nomes das chamadas de sistema usados em todo este texto são exemplos genéricos. Cada sistema operacional tem seu próprio nome para cada chamada de sistema.

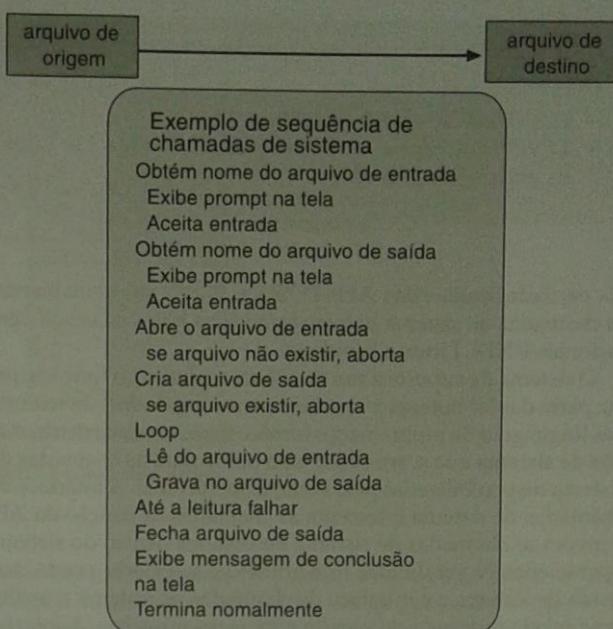


Figura 2.4 Exemplo de como as chamadas de sistema são usadas.

Em segundo plano, normalmente as funções que compõem uma API invocam as chamadas de sistema reais em nome do programador da aplicação. Por exemplo, a função `CreateProcess()` do Win32 (que por acaso é usada na criação de um novo processo) na verdade invoca a chamada de sistema `NTCreateProcess()` no kernel do Windows. Por que um programador de aplicações iria preferir programar de acordo com uma API em vez de invocar as chamadas de sistema reais? Há várias razões para se fazer isso. Um benefício de programar de acordo com uma API está relacionado à portabilidade do programa: quando um programador de aplicações projeta um programa usando uma API ele espera que seu programa seja compilado e executado em qualquer sistema que dê suporte à mesma API (embora, na verdade, diferenças na arquitetura geralmente tornem isso mais difícil do que parece). Além do mais, as chamadas de sistema reais costumam ser mais detalhadas e difíceis de manipular do que a API disponível para um programador de aplicações. De qualquer forma, sempre há uma forte relação entre uma função da API e a chamada de sistema associada a ela dentro do kernel.

EXEMPLO DE API PADRÃO

Como exemplo de uma API padrão, considere a função `ReadFile()` da API Win32 — uma função para a leitura de um arquivo. A API dessa função aparece na Figura 2.5.

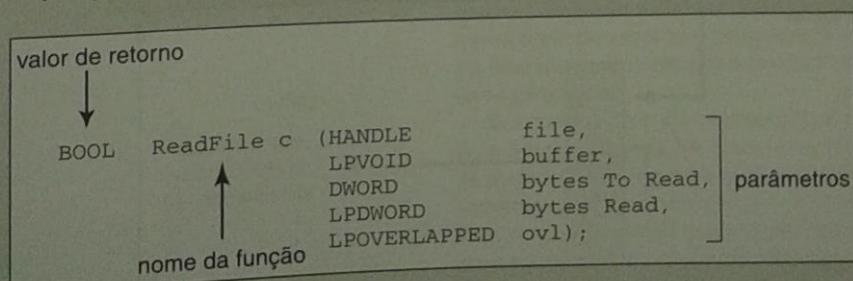


Figura 2.5 A API da função `ReadFile()`.

A seguir temos uma descrição dos parâmetros passados para `ReadFile()`:

- `HANDLE file` — o arquivo a ser lido
- `LPVOID buffer` — um buffer onde os dados serão lidos e de onde serão gravados

- `DWORD bytesToRead` — a quantidade de bytes a ser lida no buffer
- `LPDWORD bytesRead` — a quantidade de bytes lida durante a última leitura
- `LPOVERLAPPED ovlp` — indica se o I/O sobreposto está sendo usado

Na verdade, muitas das APIs POSIX e Win32 são semelhantes às chamadas de sistema nativas fornecidas pelos sistemas operacionais UNIX, Linux e Windows.

O sistema de suporte a run time (um conjunto de funções que faz parte das bibliotecas incluídas com o compilador) da maioria das linguagens de programação fornece uma interface de chamadas de sistema que serve como uma ponte para as chamadas de sistema disponibilizadas pelo sistema operacional. A interface de chamadas de sistema intercepta as chamadas de função da API e invoca as chamadas de sistema necessárias dentro do sistema operacional. Normalmente, um número é associado a cada chamada de sistema e a interface de chamadas de sistema mantém uma tabela indexada de acordo com esses números. A interface de chamadas de sistema invoca então a chamada de sistema desejada no kernel do sistema operacional e retorna o status da chamada e qualquer valor de retorno associado.

O invocador não precisa saber coisa alguma sobre como a chamada de sistema é implementada ou o que ela faz durante a execução. Em vez disso, ele só precisa seguir a API e saber o que o sistema operacional fará como resultado da execução dessa chamada de sistema. Portanto, a maioria dos detalhes da interface do sistema operacional é oculta do programador pela API e gerenciada pela biblioteca de suporte a run time. O relacionamento entre uma API, a interface de chamadas de sistema e o sistema operacional é mostrado na Figura 2.6, que ilustra como o siste-

ma operacional manipula uma aplicação de usuário invocando a chamada de sistema `open()`.

As chamadas de sistema ocorrem de diferentes maneiras, dependendo do computador que estiver sendo usado. Geralmente, são necessárias mais informações além da identidade da chamada de sistema desejada. A quantidade e o tipo exatos das informações variam de acordo com a chamada e o sistema operacional específicos. Por exemplo, para obter entradas, podemos ter de especificar o arquivo ou dispositivo a ser usado como origem, assim como o endereço e o tamanho do buffer de memória em que a entrada deve ser lida. É claro que o dispositivo ou arquivo e o tamanho podem estar implícitos na chamada.

Três métodos gerais são usados na passagem de parâmetros para o sistema operacional. A abordagem mais simples é a passagem dos parâmetros em *registradores*. Em alguns casos, no entanto, pode haver mais parâmetros do que registradores. Nesses casos, geralmente os parâmetros são armazenados em um *bloco*, ou tabela, na memória, e o endereço do bloco é passado como parâmetro em um registrador (Figura 2.7). Essa é a abordagem adotada pelo Linux e Solaris. Os parâmetros também podem ser colocados, ou *incluídos*, na *pilha* pelo programa e *extraídos* da pilha pelo sistema operacional. Alguns sistemas operacionais preferem o método de bloco ou pilha porque essas abordagens não limitam a quantidade ou a extensão dos parâmetros que estão sendo passados.

2.4 Tipos de Chamadas de Sistema

Basicamente as chamadas de sistema podem ser agrupadas em seis categorias principais: controle de processo, manipulação de arquivo, manipulação de dispositivo, manutenção de in-

formação, comunicações e proteção. Nas Seções 2.4.1 a 2.4.6, discutiremos brevemente os tipos de chamadas de sistema que podem ser fornecidos por um sistema operacional. A maioria

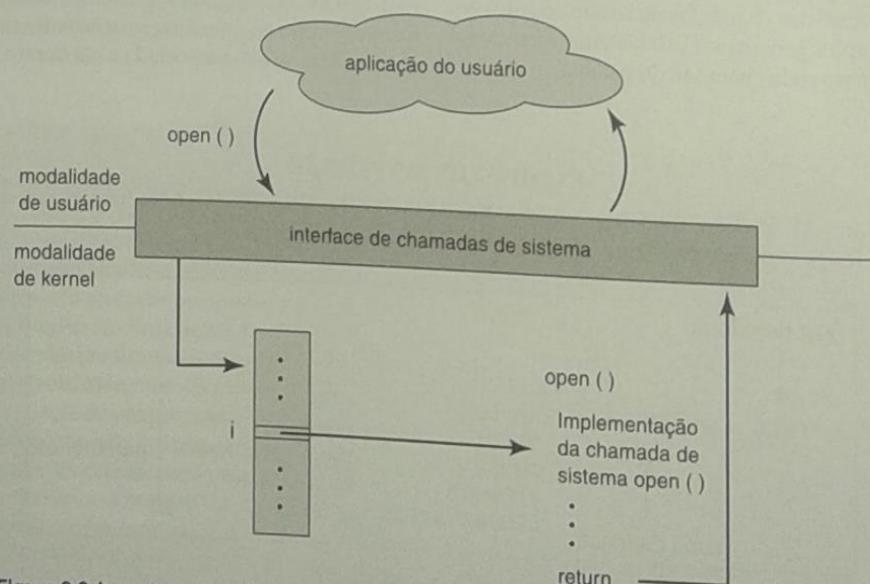


Figura 2.6 A manipulação de uma aplicação de usuário que invoca a chamada de sistema `open()`.

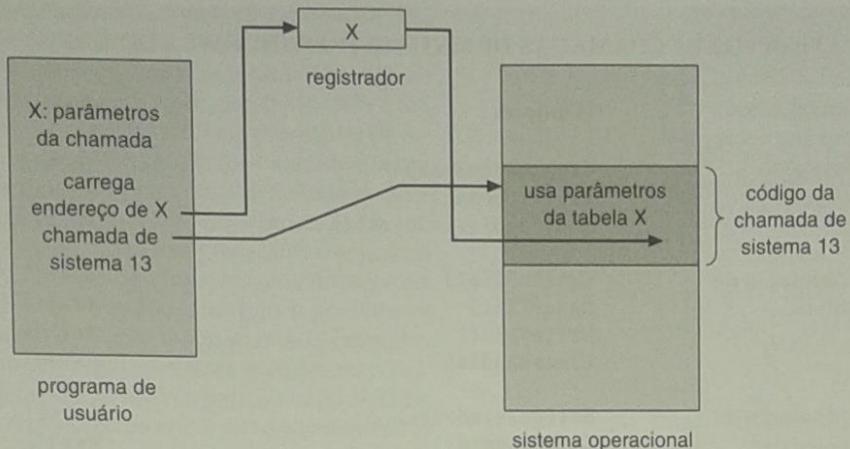


Figura 2.7 Passagem de parâmetros como uma tabela.

dessas chamadas de sistema dá suporte a (ou é suportada por) conceitos e funções que são discutidos em capítulos posteriores. A Figura 2.8 resume os tipos de chamadas de sistema normalmente fornecidos por um sistema operacional.

2.4.1 Controle de Processos

Um programa em execução tem de poder interromper seu processamento normal (`end`) ou anormalmente (`abort`). Se uma chamada de sistema for feita para encerrar o programa que está sendo executado corretamente de modo anormal ou se o pro-

- Controle de processo
 - `end`, `abort`
 - `load`, `execute`
 - `creat process`, `terminate process`
 - `get process attributes`, `set process attributes`
 - `wait for time`
 - `wait event`, `signal event`
 - `allocate and free memory`
- Gerenciamento de arquivo
 - `create file`, `delete file`
 - `open`, `close`
 - `read`, `write`, `reposition`
 - `get file attributes`, `set file attributes`
- Gerenciamento de dispositivo
 - `request device`, `release device`
 - `read`, `write`, `reposition`
 - `get device attributes`, `set device attributes`
 - `logically attach or detach devices`
- Manutenção de informações
 - `get time or date`, `set time or date`
 - `get system data`, `set system data`
 - `get process`, `file or device attributes`
 - `set process`, `file or device attributes`
- Comunicações
 - `create`, `delete communication connection`
 - `send`, `receive messages`
 - `transfer status information`
 - `attach or detach remote devices`

Figura 2.8 Tipos de chamadas de sistema.

ma encontrar um problema e causar uma exceção por erro, uma imagem da memória pode ser descarregada e uma mensagem de erro gerada. A imagem é gravada em disco, podendo ser examinada por um *depurador* — um programa do sistema projetado para ajudar o programador a encontrar e corrigir bugs — para determinar a causa do problema. Sob circunstâncias normais ou anormais, o sistema operacional deve transferir o controle para o interpretador de comandos. Em seguida, o interpretador lerá o próximo comando. Em um sistema interativo, o interpretador de comandos simplesmente passa para o próximo comando; assume-se que o usuário emitirá um comando apropriado para responder a qualquer erro. Em um sistema de GUI, uma janela pop-up pode alertar o usuário sobre o erro e solicitar orientações. Em um sistema batch, geralmente o interpretador de comandos encerra o job inteiro e continua no próximo job. Alguns sistemas permitem que cartões de controle indiquem ações especiais de recuperação no caso de um erro. Um *cartão de controle* é um conceito do sistema batch. É um comando para o gerenciamento da execução de um processo. Se o programa descobrir um erro em sua entrada e quiser encerrar anormalmente, também pode querer definir um nível de erro. Erros mais graves podem ser indicados por um parâmetro de erro de nível mais alto. Assim é possível combinar o encerramento normal e o anormal definindo um encerramento normal como um erro de nível 0. O interpretador de comandos ou o programa seguinte pode usar esse nível de erro para determinar a próxima ação automaticamente.

Um processo ou job que estiver executando um programa pode querer carregar (`load`) e executar (`execute`) outro programa. Esse recurso permite que o interpretador de comandos execute um programa a partir de um comando de usuário, um clique no mouse ou um comando batch, por exemplo. Uma questão interessante é para onde retornar o controle quando o programa carregado terminar. Essa questão está relacionada com o problema de saber se o programa existente foi perdido, salvo ou liberado para continuar a execução concorrentemente com o novo programa.

Se o controle tiver de retornar para o programa existente quando o novo programa for encerrado, devemos salvar a imagem de memória do programa existente; assim, teremos criado efetivamente um mecanismo para um programa chamar outro. Se os dois programas continuarem concorrentemente, teremos criado um novo job ou processo para ser multiprogramado. Geralmente, há uma chamada de sistema especificamente para essa finalidade (`create process` ou `submit job`).

EXEMPLOS DE CHAMADAS DE SISTEMA DO WINDOWS E DO UNIX

Windows	Unix	
Controle de processos	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
Manipulação de arquivos	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Manipulação de dispositivos	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Manutenção de informações	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Comunicação	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Proteção	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Se criarmos um novo job ou processo, ou talvez até mesmo um conjunto de jobs ou processos, devemos poder controlar sua execução. Esse controle requer a habilidade de determinar e redefinir os atributos de um job ou processo, inclusive a prioridade do job, o tempo de execução máximo permitido e assim por diante (`get process attributes` e `set process attributes`). Também podemos querer encerrar um job ou processo que criamos (`terminate process`) se acharmos que ele está incorreto ou não é mais necessário.

Tendo criado novos jobs ou processos, podemos ter de esperar o término de sua execução. Podemos querer esperar por um certo período de tempo (`wait time`); porém, o mais provável é que esperemos que um evento específico ocorra (`wait event`). Os jobs ou processos devem então sinalizar quando esse evento ocorrer (`signal event`). Com bastante frequência, dois ou mais processos compartilham dados. Para assegurar a integridade dos dados que estão sendo compartilhados, os sistemas operacionais costumam fornecer chamadas de sistema que permitem que um processo bloquee dados compartilhados, impedindo dessa forma que outro processo acesse os dados enquanto estão bloqueados. Normalmente essas chamadas de sistema são `acquire lock` e `release lock`. Esses tipos de chamadas de sistema, que lidam com a coordenação de processos concorrentes, são discutidos com maiores detalhes no Capítulo 6.

EXEMPLO DA BIBLIOTECA C PADRÃO

A biblioteca C padrão fornece parte da interface de chamadas de sistema de muitas versões do UNIX e Linux. Como exemplo, suponhamos que um programa C chamassem a instrução `printf()`. A biblioteca C intercepta essa cha-

mada e invoca a(s) chamada(s) de sistema necessária(s) no sistema operacional — nesse exemplo, a chamada de sistema `write()`. A biblioteca pega o valor retornado por `write()` e o passa para o programa do usuário. Isso é mostrado na Figura 2.9.

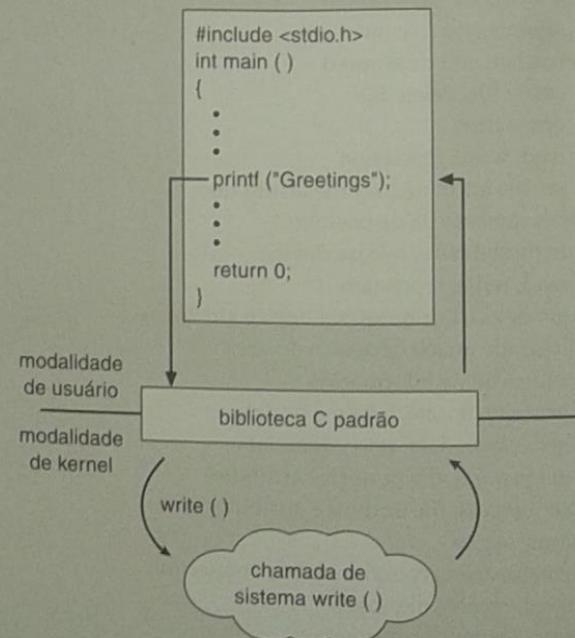


Figura 2.9 Manipulação de `write()` pela biblioteca C padrão.

Há tantas nuances e variações no controle de processos e jobs que a seguir usaremos dois exemplos — um envolvendo um sistema monotarefa e o outro um sistema multitarefa — para esclarecer esses conceitos. O sistema operacional MS-DOS é um exemplo de sistema monotarefa. Ele tem um interpretador de comandos que é chamado quando o computador é iniciado (Figura 2.10(a)). Já que o MS-DOS é monotarefa, usa um método simples para executar um programa e não cria um novo processo. Ele carrega o programa na memória, fazendo a gravação sobre grande parte dele próprio para dar ao programa o máximo de memória possível (Figura 2.10(b)). Em seguida, direciona o ponteiro de instruções para a primeira instrução do programa. O programa é então executado e ou um erro causa uma exceção, ou o programa executa uma chamada de sistema para ser encerrado. De uma forma ou de outra, o código de erro é salvo na memória do sistema para uso posterior. Após essa ação, a pequena parcela do interpretador de comandos que não foi sobreposta retoma a execução. Sua primeira tarefa é recarregar o resto do interpretador de comandos a partir do disco. Em seguida, o interpretador de comandos torna o código de erro anterior disponível para o usuário ou para o próximo programa.

O FreeBSD (derivado do Berkeley UNIX) é um exemplo de sistema multitarefa. Quando um usuário se conecta ao sistema, o shell que ele escolheu é executado. Esse shell é semelhante ao shell do MS-DOS já que aceita os comandos e executa os programas que o usuário solicita. No entanto, como o FreeBSD é um sistema multitarefa, o interpretador de comandos pode continuar em execução enquanto outro programa é processado (Figura 2.11). Para iniciar um novo processo, o shell executa uma chamada de sistema `fork()`. Em seguida, o programa selecionado é carregado na memória através de uma chamada de sistema `exec()` e é executado. Dependendo da maneira como o comando foi emitido, o shell esperará o processo terminar ou o executará “em segundo plano”. No último caso, o shell solicita imediatamente outro comando. Quando um processo está sendo executado em segundo plano, ele não pode receber entradas diretamente do teclado, porque o shell está usando esse recurso. Portanto, a operação de I/O é executada através de arquivos ou de uma GUI. Enquanto isso, o usuário pode solicitar ao shell que execute outros programas, monitore o progresso do processo que está sendo executado, altere a prioridade desse programa e assim por diante. Quando o processo é concluído, ele executa uma chamada de sistema `exit()` para ser encerrado,

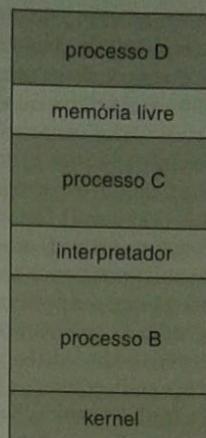


Figura 2.11 FreeBSD executando vários programas.

retornando para o processo que o chamou o código de status 0 ou um código de erro diferente de zero. Esse código de status ou erro fica então disponível para o shell ou outros programas. Os processos são discutidos no Capítulo 3 com um exemplo de programa que usa as chamadas de sistema `fork()` e `exec()`.

2.4.2 Gerenciamento de Arquivos

O sistema de arquivos é discutido com mais detalhes nos Capítulos 10 e 11. Podemos, no entanto, identificar várias chamadas de sistema comuns que lidam com arquivos.

Primeiro temos de poder criar (`create`) e excluir (`delete`) arquivos. As duas chamadas de sistema precisam do nome do arquivo e talvez alguns de seus atributos. Uma vez que o arquivo seja criado, ele deverá ser aberto (`open`) e usado. Também podemos ler (`read`), gravar (`write`) ou reposicionar (`reposition`) o arquivo (retornar ao início ou saltar para o fim do arquivo, por exemplo). Para concluir, temos de fechar (`close`) o arquivo, indicando que ele não está mais sendo usado.

Podemos precisar desses mesmos conjuntos de operações para diretórios se tivermos uma estrutura de diretórios para a organização de arquivos no sistema de arquivos. Além disso, no caso de arquivos ou diretórios, temos de poder determinar os valores de vários atributos e talvez redefinir-los se necessário. Os atributos do arquivo incluem seu nome, tipo, códigos de proteção, informações de contabilização e assim por diante. Pelo menos duas chamadas de sistema, `get_file_attribute` e `set_file_attribute`, são necessárias para essa função. Alguns sistemas operacionais fornecem muitas outras chamadas, como as chamadas de movimentação (`move`) e cópia (`copy`) do arquivo. Outros podem fornecer uma API que execute essas operações usando código e chamadas de sistema diferentes, e outros ainda podem fornecer apenas programas do sistema para executar essas tarefas. Se os programas do sistema puderem ser chamados por outros programas, cada um deles poderá ser considerado uma API por outros programas do sistema.

2.4.3 Gerenciamento de Dispositivos

Um processo pode precisar de vários recursos para ser executado — memória principal, drives de disco, acesso a arquivos e assim por diante. Quando os recursos estão disponíveis, eles podem ser cedidos e o controle é passado para o processo do usuário. Caso contrário, o processo terá de esperar até recursos suficientes estarem disponíveis.

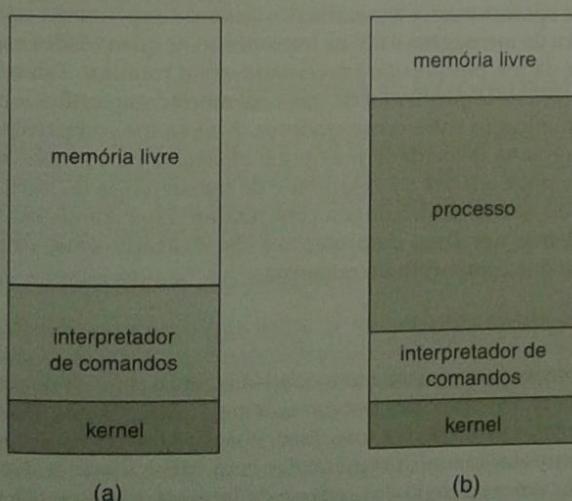


Figura 2.10 Execução do MS-DOS. (a) Na inicialização do sistema. (b) Processando um programa.

Os diversos recursos controlados pelo sistema operacional podem ser considerados dispositivos. Alguns desses dispositivos são físicos (por exemplo, drives de disco), enquanto outros podem ser considerados dispositivos abstratos ou virtuais (por exemplo, arquivos). Um sistema com vários usuários pode demandar que primeiro seja feita a solicitação (`request`) do dispositivo, para garantir seu uso exclusivo. Quando o dispositivo não é mais necessário, ele pode ser liberado (`release`). Essas funções são o mesmo que as chamadas de sistema `open` e `close` são para os arquivos. Outros sistemas operacionais permitem o acesso não gerenciado a dispositivos. Nesse caso o perigo é o potencial para a disputa por dispositivos e talvez o deadlock, que é descrito no Capítulo 7.

Uma vez que o dispositivo tenha sido solicitado (e alocado para nosso uso), poderemos ler (`read`), gravar (`write`) e (possivelmente) reposicioná-lo (`reposition`), da mesma forma que fazemos com os arquivos. Na verdade, a semelhança entre arquivos e dispositivos de I/O é tão grande que muitos sistemas operacionais, inclusive o UNIX, fundem os dois em uma estrutura arquivo-dispositivo combinada. Nesse caso, um conjunto de chamadas de sistema é usado tanto para arquivos quanto para dispositivos. Em algumas situações, os dispositivos de I/O são identificados por nomes de arquivo especiais, localização de diretórios ou atributos de arquivo.

A interface de usuário também pode fazer arquivos e dispositivos parecerem semelhantes, ainda que as chamadas de sistema subjacentes sejam diferentes. Esse é outro exemplo das muitas decisões de projeto que fazem parte da construção de um sistema operacional e da interface de usuário.

2.4.4 Manutenção de Informações

Muitas chamadas de sistema só existem para fins de transferência de informações entre o programa do usuário e o sistema operacional. Por exemplo, a maioria dos sistemas tem uma chamada de sistema que retorna a hora (`time`) e a data (`date`) correntes. Outras chamadas de sistema podem retornar informações sobre o sistema, como a quantidade de usuários correntes, o número da versão do sistema operacional, a quantidade de espaço livre na memória ou em disco e assim por diante.

Outro conjunto de chamadas de sistema é útil na depuração de um programa. Muitos sistemas fornecem chamadas para o despejo (`dump`) da memória. Esse recurso é útil na depuração. Um rastreamento (`trace`) de programa lista cada chamada de sistema quando ela é executada. Até mesmo microprocessadores fornecem um modo de CPU conhecido como *passo único*, em que uma exceção é executada pela CPU após cada instrução. Geralmente, a exceção é capturada por um depurador.

Muitos sistemas operacionais fornecem um perfil de tempo do programa que indica por quanto tempo ele é executado em uma locação ou conjunto de locações específico. Um perfil de tempo requer um recurso de rastreamento ou interrupções periódicas de timer. A cada ocorrência de interrupção do timer, o valor do contador do programa é registrado. Com interrupções suficientemente frequentes do timer, um cenário estatístico do tempo gasto em várias partes do programa pode ser obtido.

Além disso, o sistema operacional mantém informações sobre todos os seus processos, e chamadas de sistema são usadas no acesso a essas informações. Geralmente, chamadas também são usadas na redefinição das informações do processo (`get process attributes` e `set process attributes`). Na Seção 3.1.3, discutiremos que informações costumam ser mantidas.

2.4.5 Comunicação

Há dois modelos comuns de comunicação entre processos: o modelo de transmissão de mensagens e o de memória compartilhada.

No modelo de **transmissão de mensagens**, os processos de comunicação trocam mensagens uns com os outros para transferir informações. As mensagens podem ser trocadas entre os processos direta ou indiretamente através de uma caixa de correio comum. Antes que a comunicação possa ocorrer, uma conexão deve ser aberta. O nome do outro interlocutor deve ser conhecido, seja ele outro processo do mesmo sistema ou um processo em outro computador conectado por uma rede de comunicação. Cada computador de uma rede tem um *nome de hospedeiro* pelo qual é conhecido. O hospedeiro também tem um identificador de rede que pode ser um endereço IP. Da mesma forma, cada processo tem um *nome de processo*, e esse nome é convertido em um identificador através do qual o sistema operacional pode referenciar o processo. As chamadas de sistema `get hostid` e `get processid` fazem essa conversão. Os identificadores são então passados para as chamadas de uso geral `open` e `close` fornecidas pelo sistema de arquivos ou para chamadas `open connection` e `close connection` específicas, dependendo do modelo de comunicação do sistema. Geralmente o processo destinatário deve dar sua permissão com uma chamada `accept connection` para que a comunicação possa ocorrer. A maioria dos processos que recebem conexões é de *daemons* de uso específico, que são programas de sistema fornecidos para esse fim. Eles executam uma chamada `wait for connection` e são ativados quando uma conexão é estabelecida. Em seguida, a fonte da comunicação, conhecida como *cliente*, e o daemon destinatário, conhecido como *servidor*, trocam mensagens usando chamadas de sistema `read message` e `write message`. A chamada `close connection` encerra a comunicação.

No modelo de **memória compartilhada**, os processos usam chamadas `shared memory create` e `shared memory attach` para criar e acessar regiões da memória ocupadas por outros processos. Lembre-se de que, normalmente, o sistema operacional tenta impedir que um processo acesse a memória usada por outro processo. Na memória compartilhada dois ou mais processos têm de concordar com a remoção dessa restrição. Assim eles podem trocar informações lendo e gravando dados nas áreas compartilhadas. A forma dos dados é determinada pelos processos e não ficam sob o controle do sistema operacional. Os processos também devem garantir que não farão gravações no mesmo local simultaneamente. Esses mecanismos são discutidos no Capítulo 6. No Capítulo 4, examinaremos uma variação do modelo de processo — os threads — em que a memória é compartilhada por default.

Os dois modelos que acabamos de discutir são comuns em sistemas operacionais, e a maioria dos sistemas implementa ambos. A troca de mensagens é útil na transmissão de quantidades menores de dados, porque não é necessário evitar conflitos. Também é mais fácil de implementar do que a memória compartilhada para a comunicação entre computadores. A memória compartilhada proporciona velocidade máxima e eficiência na comunicação, já que pode ocorrer na velocidade de transferência da memória quando ocorre dentro de um computador. No entanto, existem problemas nas áreas de proteção e sincronização entre os processos que compartilham memória.

2.4.6 Proteção

A proteção proporciona um mecanismo para o controle de acesso aos recursos fornecidos por um sistema de computação. Historicamente, a proteção era uma preocupação somente em sistemas de computação multiprogramados com vários usuários. No entanto, com o advento das redes e da Internet, todos os sistemas de computação, de servidores a PDAs, devem se preocupar com proteção.

Normalmente, as chamadas de sistema que fornecem proteção são `set permission` e `get permission`, que manipulam as configurações de permissões de recursos como arquivos e discos. As chamadas de sistema `allow user` e `deny user`

determinam se usuários específicos podem — ou não — acessar certos recursos.

Abordaremos a proteção no Capítulo 14 e o tópico muito mais abrangente da segurança no Capítulo 15.

2.5 Programas de Sistema

Outra característica de um sistema moderno é a presença de um conjunto de programas de sistema. Volte à Figura 1.1, que demonstra a hierarquia lógica do computador. No nível mais baixo está o hardware. Em seguida está o sistema operacional, depois os programas de sistema e finalmente os programas aplicativos. Os **programas de sistema**, também conhecidos como **utilitários de sistema**, fornecem um ambiente conveniente para o desenvolvimento e a execução de programas. Alguns são simplesmente interfaces de usuário para chamadas de sistema; outros são consideravelmente mais complexos. Eles podem ser divididos nas categorias a seguir:

- **Gerenciamento de arquivos.** Esses programas criam, excluem, copiam, renomeiam, imprimem, descarregam, listam e geralmente manipulam arquivos e diretórios.
- **Informações de status.** Alguns programas simplesmente solicitam ao sistema a data, a hora, o espaço disponível na memória ou em disco, a quantidade de usuários ou informações de status semelhantes. Outros são mais complexos, fornecendo informações detalhadas sobre desempenho, registro em log e depuração. Normalmente, esses programas formatam e exibem a saída no terminal ou em outros dispositivos ou arquivos de saída, ou a exibem em uma janela GUI. Alguns sistemas também dão suporte a um *registro de configuração*, que é usado no armazenamento e recuperação de informações de configuração.
- **Modificação de arquivos.** Vários editores de texto podem estar disponíveis para a criação e modificação do conteúdo de arquivos armazenados em disco ou outros dispositivos de armazenamento. Pode também haver comandos especiais para a procura de conteúdos de arquivos ou a execução de alterações no texto.
- **Suporte a linguagens de programação.** Geralmente, compiladores, montadores, depuradores e interpretadores de linguagens de programação comuns (como C, C++, Java, Visual Basic e PERL) são fornecidos para o usuário com o sistema operacional.

- **Carga e execução de programas.** Uma vez que um programa é montado ou compilado, ele deve ser carregado na memória para ser executado. O sistema pode fornecer carregadores absolutos, carregadores relocáveis, linkage editors e carregadores de overlay. Sistemas de depuração tanto para linguagens de alto nível quanto para linguagem de máquina também são necessários.
- **Comunicações.** Esses programas fornecem o mecanismo para a criação de conexões virtuais entre processos, usuários e sistemas de computação. Eles permitem que os usuários enviem mensagens de uma tela para outra, naveguem em páginas da Web, enviem mensagens de correio eletrônico, conectem-se remotamente ou transfiram arquivos de uma máquina para outra.

Além dos programas de sistema, a maioria dos sistemas operacionais vem com programas que são úteis na resolução de problemas comuns ou na execução de operações corriqueiras. Esses **programas aplicativos** incluem os navegadores da Web, processadores e formatadores de texto, planilhas, sistemas de bancos de dados, compiladores, pacotes de plotagem e análise estatística e jogos.

A visão que a maioria dos usuários tem do sistema operacional é definida pelos programas aplicativos e de sistema, e não pelas chamadas de sistema. Considere o PC de um usuário. Quando o computador de um usuário está executando o sistema operacional Mac OS X, o usuário pode ver a GUI fornecendo uma interface baseada no mouse e em janelas. Alternativamente, ou até mesmo em uma das janelas, o usuário pode ter um shell UNIX de linha de comando. Os dois usam o mesmo conjunto de chamadas de sistema, mas elas têm aparências diferentes e agem de maneira distinta. Para confundir mais a visão do usuário, imaginemos que ele executasse a inicialização dual do Windows Vista a partir do Mac OS X. Agora o mesmo usuário no mesmo hardware tem duas interfaces totalmente diferentes e dois conjuntos de aplicações usando os mesmos recursos físicos. Portanto, no mesmo hardware um usuário pode ser exposto a várias interfaces de usuário sequencial ou concorrentemente.

2.6 Projeto e Implementação do Sistema Operacional

Nesta seção, discutiremos os problemas que surgem no projeto e na implementação de um sistema operacional. É claro que não há soluções definitivas para esses problemas, mas há abordagens que se mostraram bem-sucedidas.

2.6.1 Objetivos do Projeto

O primeiro problema que surge no projeto de um sistema é a definição de objetivos e especificações. Em um nível mais alto, o projeto do sistema será afetado pela escolha do hardware e do tipo de sistema: batch, tempo compartilhado, monusuário, multiusuário, distribuído, tempo real ou uso geral.

Além desse nível mais geral do projeto, os requisitos podem ser muito mais difíceis de especificar. No entanto, eles podem ser divididos em dois grupos básicos: objetivos do usuário e objetivos do sistema.

Os usuários desejam certas propriedades óbvias em um sistema. O sistema deve ser de uso eficiente, fácil de aprender e usar, confiável, seguro e veloz. É claro que essas especificações não são particularmente úteis no projeto do sistema, já que não há um consenso geral sobre como atendê-las.

Um conjunto semelhante de requisitos pode ser definido pelas pessoas responsáveis por projetar, criar, manter e operar o sistema. O sistema deve ser fácil de projetar, implementar e manter; e deve ser flexível, confiável, sem erros e eficiente. Novamente, esses requisitos são vagos e podem ser interpretados de várias maneiras.

Resumindo, não há uma solução única para o problema de definição dos requisitos de um sistema operacional. A grande quantidade de sistemas existentes mostra que diferentes requisitos podem resultar em várias soluções para ambientes distintos.

Por exemplo, os requisitos do VxWorks, um sistema operacional de tempo real para sistemas embutidos, devem ter sido substancialmente diferentes dos definidos para o MVS, um sistema operacional multiusuário e multiacesso de grande porte para mainframes IBM.

Definir a especificação e o projeto de um sistema operacional é uma tarefa altamente criativa. Embora nenhum livro possa lhe dizer como fazê-lo, foram desenvolvidos princípios gerais no campo da engenharia de software e passaremos à discussão de alguns desses princípios.

2.6.2 Mecanismos e Políticas

Um princípio importante é a separação entre **política** e **mecanismo**. Os mecanismos determinam *como* fazer algo; as políticas determinam *o que* será feito. Por exemplo, o timer (consulte a Seção 1.5.2) é um mecanismo que assegura a proteção da CPU, mas a definição de por quanto tempo ele deve ser configurado para um usuário específico é uma decisão política.

A separação entre política e mecanismo é importante para a flexibilidade. As políticas podem ser diferentes em locais distintos e com o passar do tempo. Na pior das hipóteses, cada mudança na política demandaria uma mudança no mecanismo subjacente. Um mecanismo genérico não afetado pelas mudanças na política seria mais desejável. Mudanças na política só demandariam a redefinição de certos parâmetros do sistema. Por exemplo, considere um mecanismo que definisse a prioridade de determinados tipos de programas sobre outros. Se o mecanismo for apropriadamente separado da política, ele poderá ser usado para suportar a decisão política de que programas com muitas operações de I/O devem ter prioridade sobre os que fazem uso pesado da CPU ou dar suporte à política oposta.

Sistemas operacionais baseados em microkernel (Seção 2.7.3) levam a separação entre mecanismo e política a um extremo implementando um conjunto básico de blocos de construção primitivos. Esses blocos são quase independentes de política, permitindo que mecanismos e políticas mais avançados sejam adicionados através de módulos de kernel criados pelo usuário ou através dos próprios programas dos usuários. Como exemplo, considere a história do UNIX. Inicialmente, ele tinha um scheduler de tempo compartilhado. Na última versão do Solaris, o scheduling é controlado por tabelas carregáveis. Dependendo da tabela carregada correntemente, o sistema pode ser de tempo compartilhado, processamento batch, tempo real, compartilhamento justo ou qualquer combinação. Fazer com que o mecanismo de scheduling passe a ser de uso geral permite que amplas alterações sejam feitas na política com um único comando `load-new-table`. No outro extremo está um sistema como o Windows, em que tanto o mecanismo quanto a política são codificados no sistema para impor uma aparência global. Todas as aplicações têm interfaces semelhantes, porque a própria interface é construída nas bibliotecas do kernel e do sistema. O sistema operacional Mac OS X tem funcionalidade semelhante.

As decisões políticas são importantes em qualquer alocação de recursos. Sempre que for necessário decidir se um recurso deve ou não ser alocado, uma decisão política deve ser tomada. Sempre que o problema for *como* em vez de *o que*, um mecanismo é que deve ser determinado.

2.6.3 Implementação

Uma vez que o sistema operacional tenha sido projetado, ele deve ser implementado. Normalmente, os sistemas operacionais eram escritos em linguagem de montagem. Atualmente, no entanto, eles costumam ser escritos em linguagens de alto nível como C ou C++.

Provavelmente o primeiro sistema a não ser escrito em linguagem de montagem foi o Master Control Program (MCP) para computadores Burroughs. O MCP foi escrito em uma variante de ALGOL. O MULTICS, desenvolvido no MIT, foi escrito principalmente em PL/1. Os sistemas operacionais Linux e Windows XP foram, em grande parte, escritos em C, embora haja algumas pequenas seções de código de montagem para drivers de dispositivo e para a gravação e recuperação do estado dos registradores.

As vantagens do uso de uma linguagem de alto nível, ou pelo menos uma linguagem de implementação de sistemas, para implementar sistemas operacionais são as mesmas obtidas quando a linguagem é usada para programas aplicativos: o código pode ser escrito mais rapidamente, é mais compacto e mais fácil de entender e depurar. Além disso, avanços na tecnologia dos compiladores melhoraram o código gerado para o sistema operacional inteiro através de uma simples recompilação. Para concluir, um sistema operacional é muito mais fácil de *portar* — mover para outro hardware — quando é escrito em uma linguagem de alto nível. Por exemplo, o MS-DOS foi escrito na linguagem de montagem Intel 8088. Como resultado, ele só é executado nativamente na família de CPUs Intel X86. (Embora o MS-DOS só seja executado nativamente no Intel X86, emuladores do conjunto de instruções do X86 permitem que o sistema operacional seja executado de modo não nativo — mais lentamente, com maior uso de recursos — em outras CPUs. *Emuladores* são programas que replicam a funcionalidade de um sistema com outro sistema.) O sistema operacional Linux, por outro lado, é escrito quase todo em C e está disponível nativamente em várias CPUs diferentes, como Intel X86, Sun SPARC e IBM PowerPC.

As únicas possíveis desvantagens da implementação de um sistema operacional em uma linguagem de alto nível são a diminuição da velocidade e o aumento dos requisitos de armazenamento. No entanto, isso não é mais um grande problema nos sistemas atuais. Embora um programador especialista em linguagem de montagem possa produzir rotinas pequenas e eficientes, para programas grandes um compilador moderno pode executar análises complexas e aplicar otimizações sofisticadas que produzem excelente código. Os processadores modernos têm fortes interligações e várias unidades funcionais que podem manipular os detalhes de dependências complexas muito mais facilmente do que a mente humana.

Como ocorre em outros sistemas, provavelmente os principais avanços no desempenho dos sistemas operacionais sejam resultado de melhores estruturas de dados e algoritmos, e não de um ótimo código em linguagem de montagem. Além disso, embora os sistemas operacionais sejam grandes, apenas uma pequena parte do código é crítica para o alto desempenho; é provável que o gerenciador de memória e o scheduler da CPU sejam as rotinas mais críticas. Depois que o sistema é escrito e está funcionando corretamente, rotinas que representem gargalos podem ser identificadas e substituídas por equivalentes em linguagem de montagem.

2.7 Estrutura do Sistema Operacional

Um sistema tão grande e complexo como um sistema operacional moderno deve ser construído cuidadosamente para funcionar de maneira apropriada e ser de fácil modificação. Uma abordagem comum é a divisão da tarefa em componentes pequenos em vez da criação de um sistema monolítico. Cada um desses módulos deve ser uma parte bem definida do sistema, com entradas, saídas e funções cuidadosamente definidas. Já discutimos brevemente no Capítulo 1 os componentes comuns dos sistemas operacionais. Nesta seção, discutiremos como esses componentes são interconectados e moldados em um kernel.

2.7.1 Estrutura Simples

Muitos sistemas operacionais comerciais não têm estruturas bem definidas. Geralmente, esses sistemas começam como sistemas pequenos, simples e limitados e então crescem para além de seu escopo original. O MS-DOS é um exemplo desse tipo de sistema. Foi originalmente projetado e implementado por algumas pessoas que não tinham ideia de que ele se tornaria tão popular. Ele foi criado para fornecer o máximo de funcionalidade no menor espaço; portanto, não foi dividido cuidadosamente em módulos. A Figura 2.12 mostra sua estrutura.

No MS-DOS, as interfaces e níveis de funcionalidade não estão bem separados. Por exemplo, programas aplicativos podem acessar as rotinas básicas de I/O para gravar diretamente em tela e em drives de disco. Essa liberdade deixa o MS-DOS vulnerável a programas oportunistas (ou maliciosos), fazendo com que o sistema inteiro seja interrompido quando programas de usuário falham. É claro que o MS-DOS também tinha limitações devido ao hardware de sua época. Já que o Intel 8088 para o qual ele foi escrito não fornece modo dual e proteção de hardware, os projetistas do MS-DOS não tinham outra opção a não ser deixar o hardware básico acessível.

Outro exemplo de estrutura limitada é o sistema operacional UNIX original. Como o MS-DOS, inicialmente o UNIX era limitado pela funcionalidade do hardware. Ele é composto por duas partes separadas: o kernel e os programas de sistema. Por sua vez, o kernel é separado em uma série de interfaces e drivers de dispositivo, que foram adicionados e expandidos com o passar dos anos conforme o UNIX evoluía. Podemos considerar o sistema operacional UNIX tradicional como uma estrutura em camadas, como mostrado na Figura 2.13. Tudo que está abaixo da interface de chamadas de sistema e acima do hardware físico

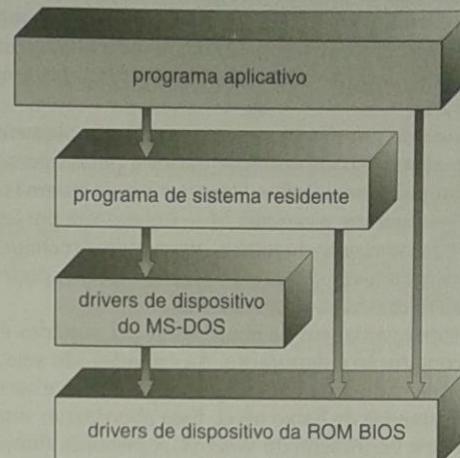


Figura 2.12 Estrutura de camadas do MS-DOS.

é o kernel. O kernel fornece o sistema de arquivos, o scheduling da CPU, o gerenciamento de memória e outras funções do sistema operacional através de chamadas de sistema. Tudo somado, temos uma enorme quantidade de funcionalidades combinadas no mesmo nível. Essa estrutura monolítica era difícil de implementar e manter.

2.7.2 A Abordagem em Camadas

Com suporte de hardware apropriado, os sistemas operacionais podem ser divididos em partes que sejam menores e mais adequadas do que as permitidas pelos sistemas MS-DOS e UNIX originais. Assim, o sistema operacional pode ter um controle muito maior sobre o computador e sobre as aplicações que fazem uso desse computador. Os implementadores têm mais liberdade para alterar os mecanismos internos do sistema e criar sistemas operacionais modulares. Em uma abordagem top-down, a funcionalidade e os recursos gerais são determinados e separados em componentes. A ocultação de informações também é importante, porque deixa os programadores livres para implementar as rotinas de baixo nível como acharem melhor, contanto que a interface externa da rotina permaneça inalterada e a rotina propriamente dita execute a tarefa anunciada.

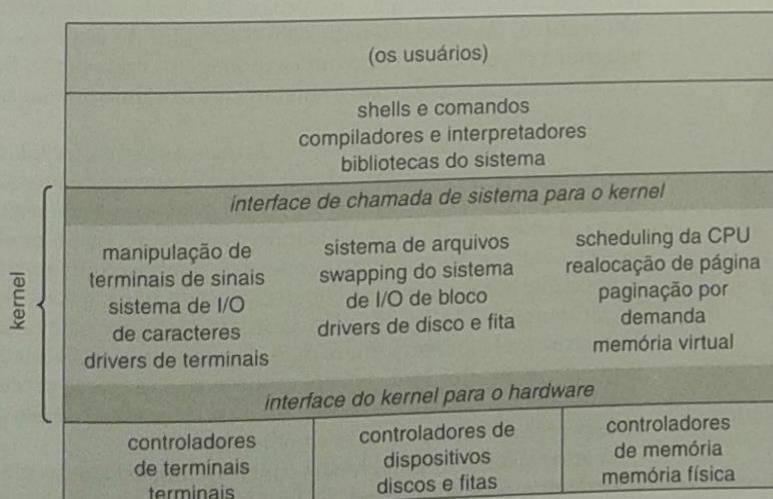


Figura 2.13 Estrutura tradicional do sistema UNIX.

Um sistema pode ser modularizado de várias maneiras. Um dos métodos é a **abordagem em camadas**, no qual o sistema operacional é dividido em várias camadas (níveis). A camada inferior (camada 0) é o hardware; a camada mais alta (camada N) é a interface de usuário. Essa estrutura em camadas é mostrada na Figura 2.14.

A camada de um sistema operacional é a implementação de um objeto abstrato composto por dados e pelas operações que podem manipular esses dados. Uma camada de sistema operacional típica — digamos, a camada M — é composta por estruturas de dados e um conjunto de rotinas que podem ser chamadas por camadas de alto nível. A camada M, por sua vez, pode chamar operações em camadas de baixo nível.

A principal vantagem da abordagem em camadas é a facilidade de construção e depuração. As camadas são selecionadas de modo que cada uma use funções (operações) e serviços somente de camadas de baixo nível. Essa abordagem simplifica a depuração e a verificação do sistema. A primeira camada pode ser depurada sem qualquer preocupação com o resto do sistema, porque, por definição, ela só usa o hardware básico (que se supõe esteja correto) para implementar suas funções. Uma vez que a primeira camada esteja depurada, seu funcionamento correto pode ser presumido enquanto a segunda camada é depurada e assim por diante. Se um erro for encontrado durante a depuração de uma camada específica, ele deve estar nessa camada, porque as de baixo já foram depuradas. Portanto, o projeto e a implementação do sistema são simplificados.

Cada camada é implementada somente com as operações fornecidas por camadas de baixo nível. A camada não precisa saber como essas operações são implementadas; só precisa saber o que essas operações fazem. Portanto, cada camada oculta das camadas de nível superior a existência de certas estruturas de dados, operações e hardware.

A principal dificuldade da abordagem em camadas envolve a definição apropriada das diversas camadas. Já que uma camada só pode usar camadas de baixo nível, um planejamento cuidadoso é necessário. Por exemplo, o driver de dispositivo do backing store (espaço em disco usado por algoritmos de memória virtual) deve estar em um nível mais baixo do que as rotinas de gerenciamento da memória, porque o gerenciamento da memória demanda o uso do backing store.

Outros requisitos podem não ser tão óbvios. Normalmente o driver do backing store fica acima do módulo de scheduler da CPU, porque ele pode ter de esperar que as operações de I/O e a CPU sofram um rescheduling durante esse intervalo. No entanto, em um sistema grande, o scheduler da CPU pode ter mais informações sobre todos os processos ativos do que a memória pode armazenar. Logo, essas informações podem ter de ser inseridas e extraídas da memória, o que demandaria que a rotina do driver de backing store ficasse abaixo do scheduler da CPU.

Um problema final das implementações em camadas é que elas tendem a ser menos eficientes do que outras abordagens. Por exemplo, quando um programa de usuário executa uma operação de I/O, ele executa uma chamada de sistema que é interceptada para a camada de I/O, que chama a camada de gerenciamento da memória, a qual por sua vez chama a camada de scheduling da CPU, que é então passada para o hardware. Em cada camada, os parâmetros podem ser modificados, dados podem ter de ser passados e assim por diante. Cada camada adiciona overhead à chamada de sistema; o resultado final é uma chamada de sistema que demora mais do que em um sistema não estruturado em camadas.

Ultimamente essas limitações têm causado alguma reação contra a estruturação em camadas. Menos camadas com mais funcionalidades estão sendo projetadas, fornecendo a maioria das vantagens do código modularizado e evitando ao mesmo tempo os difíceis problemas de definição e interação de camadas.

2.7.3 Microkernels

Já vimos que, conforme o UNIX se expandiu, o kernel se tornou maior e mais difícil de gerenciar. Na metade dos anos 80, pesquisadores da Universidade Carnegie Mellon desenvolveram um sistema operacional chamado Mach que modularizou o kernel usando a abordagem de **microkernel**. Esse método estrutura o sistema operacional removendo todos os componentes não essenciais do kernel e implementando-os como programas de nível de sistema e usuário. O resultado é um kernel menor. Há pouco consenso sobre que serviços devem permanecer no kernel e quais devem ser implementados no espaço do usuário. Normalmente, no entanto, os microkernels fornecem um gerenciamento mínimo dos processos e da memória, além de um recurso de comunicação.

A principal função do microkernel é fornecer um recurso de comunicação entre o programa cliente e os diversos serviços que também estão sendo executados no espaço do usuário. A comunicação é fornecida pela *transmissão de mensagens*, que foi descrita na Seção 2.4.5. Por exemplo, se o programa cliente quiser acessar um arquivo, ele deve interagir com o servidor de arquivos. O programa cliente e o serviço nunca interagem diretamente. Em vez disso, eles se comunicam indiretamente trocando mensagens com o microkernel.

Um dos benefícios da abordagem de microkernel é a facilidade de extensão do sistema operacional. Todos os serviços novos são adicionados ao espaço do usuário e consequentemente não requerem a modificação do kernel. Quando o kernel precisa ser modificado, as alterações tendem a ser poucas, porque o microkernel é um kernel menor. O sistema operacional resultante é mais fácil de ser transportado de uma plataforma de hardware para outra. O microkernel também fornece mais segurança e confiabilidade, já que a maioria dos serviços está sendo executada como processos de usuário — e não do kernel. Se um serviço falha, o resto do sistema operacional permanece inalterado.

Vários sistemas operacionais contemporâneos têm usado a abordagem de microkernel. O Tru64 UNIX (antes conhecido

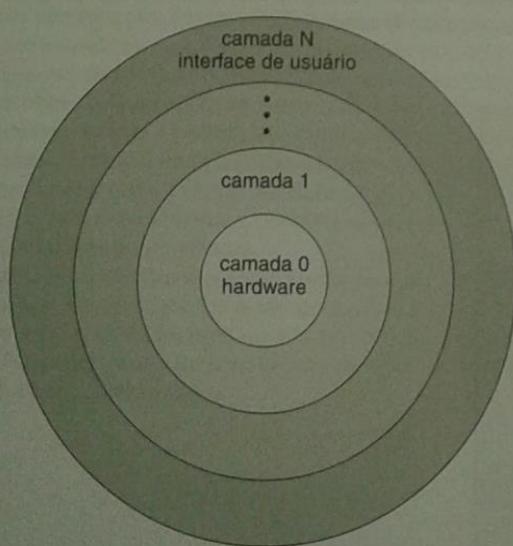


Figura 2.14 Um sistema operacional em camadas.

como Digital UNIX) fornece uma interface UNIX para o usuário, mas é implementado com um kernel Mach. O kernel Mach converte chamadas de sistema UNIX em mensagens enviadas aos serviços de nível de usuário apropriados. O kernel do Mac OS X (também conhecido como *Darwin*) também se baseia no microkernel Mach.

Outro exemplo é o QNX, um sistema operacional de tempo real. O microkernel do QNX fornece serviços de transmissão de mensagens e scheduling de processos. Ele também manipula comunicação de rede de baixo nível e interrupções de hardware. Todos os outros serviços do QNX são fornecidos por processos padrão executados fora do kernel em modo de usuário.

Infelizmente, os microkernels podem comprometer o desempenho devido ao overhead de funções do sistema. Considere a história do Windows NT. A primeira versão tinha uma organização de microkernel em camadas. No entanto, essa versão tinha baixo desempenho se comparada com a do Windows 95. O Windows NT 4.0 resolveu parcialmente o problema de desempenho movendo camadas do espaço do usuário para o espaço do kernel e integrando-as mais proximamente. Quando o Windows XP foi projetado, sua arquitetura era mais monolítica do que de microkernel.

2.7.4 Módulos

Talvez a melhor metodologia corrente para o projeto de sistemas operacionais envolva o uso de técnicas de programação orientada a objetos na criação de um kernel modular. Aqui, o kernel tem um conjunto de componentes e links básicos em serviços adicionais durante o tempo de inicialização ou de execução. Essa estratégia usa módulos dinamicamente carregáveis e é comum em implementações modernas do UNIX, como o Solaris, o Linux e o Mac OS X. Por exemplo, a estrutura do sistema operacional Solaris, mostrada na Figura 2.15, é organizada ao redor de um kernel básico com sete tipos de módulos de kernel carregáveis:

1. Classes de scheduling
2. Sistemas de arquivos
3. Chamadas de sistema carregáveis
4. Formatos executáveis
5. Módulos STREAMS
6. Miscelâneas
7. Drivers de dispositivo e de bus

Esse tipo de projeto permite que o kernel forneça serviços básicos e também que certos recursos sejam implementados dinamicamente. Por exemplo, drivers de dispositivo e de bus de

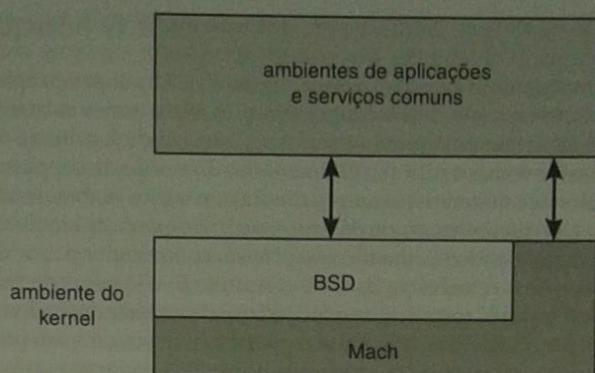


Figura 2.16 A estrutura do Mac OS X.

um hardware específico podem ser adicionados ao kernel, e o suporte a diferentes sistemas de arquivos pode ser adicionado na forma de módulos carregáveis. O resultado final lembra um sistema em camadas em que cada seção do kernel tem interfaces definidas e protegidas; porém é mais flexível do que um sistema em camadas porque um módulo pode chamar qualquer outro módulo. Além disso, a abordagem é como a de microkernel já que o módulo principal só tem funções básicas e o conhecimento de como carregar e se comunicar com outros módulos. No entanto, é mais eficiente, porque os módulos não precisa chamar a transmissão de mensagens para se comunicar.

O sistema operacional Mac OS X da Apple usa uma estrutura híbrida. Ele é um sistema em camadas em que uma camada é composta pelo microkernel Mach. A estrutura do Mac OS X aparece na Figura 2.16. As camadas superiores incluem ambientes de aplicação e um conjunto de serviços fornecendo uma interface gráfica para as aplicações. Abaixo dessas camadas está o ambiente do kernel, que é composto principalmente pelo microkernel Mach e o kernel BSD. O Mach fornece gerenciamento da memória; suporte a chamadas de procedimento remotas (RPCs) e recursos de comunicação entre processos (IPC), inclusive a transmissão de mensagens; e o scheduling de threads. O componente BSD fornece uma interface de linha de comando BSD, suporte à conexão de rede e sistemas de arquivos e uma implementação de APIs POSIX, incluindo Pthreads. Além do MAC e do BSD, o ambiente do kernel fornece um kit de I/O para o desenvolvimento de drivers de dispositivo e módulos dinamicamente carregáveis (que o Mac OS X chama de extensões do kernel). Como mostrado na figura, aplicações e serviços comuns podem fazer uso dos recursos do Mach ou do BSD diretamente.

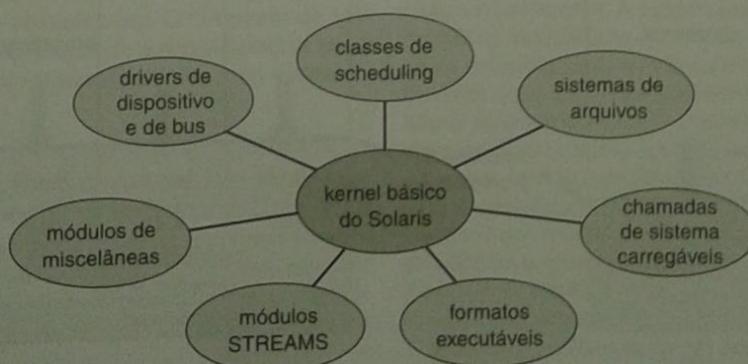


Figura 2.15 Módulos carregáveis do Solaris.

2.8 Máquinas Virtuais

A abordagem em camadas descrita na Seção 2.7.2 atinge seu ápice lógico no conceito de *máquina virtual*. A ideia básica existente por trás de uma máquina virtual é a abstração do hardware de um único computador (CPU, memória, drives de disco, placas de interface de rede e assim por diante) em vários ambientes de execução diferentes, o que dá a impressão de que cada ambiente de execução está executando seu próprio computador privado.

Usando o scheduling da CPU (Capítulo 5) e técnicas de memória virtual (Capítulo 9), um *hospedeiro* do sistema operacional pode dar a impressão de que um processo tem seu próprio processador com sua própria memória (virtual). A máquina virtual fornece uma interface *idêntica* ao hardware puro subjacente. Cada processo *convidado* recebe uma cópia (virtual) do computador subjacente (Figura 2.17). Geralmente, o processo convidado é na verdade um sistema operacional, e é assim que uma única máquina física pode executar vários sistemas operacionais concorrentemente, cada um em sua própria máquina virtual.

2.8.1 História

As máquinas virtuais surgiram pela primeira vez comercialmente em mainframes IBM através do sistema operacional VM em 1972. O VM evoluiu e ainda está disponível, e muitos dos seus conceitos originais são encontrados em outros sistemas, o que torna válido o estudo desse recurso.

O VM370 da IBM dividia um mainframe em várias máquinas virtuais, cada uma executando seu próprio sistema operacional. Um grande problema da abordagem de máquina virtual do VM envolvia os sistemas de disco. Suponhamos que a máquina física tivesse três drives de disco, mas quisesse dar suporte a sete máquinas virtuais. É claro que ela não poderia alocar um drive de disco a cada máquina virtual, porque o próprio software da máquina virtual precisava de espaço substancial em disco para fornecer memória virtual e spooling. A solução era fornecer discos virtuais — chamados de *minidiscos* no sistema operacional VM da IBM — idênticos em todos os aspectos exceto no tamanho. O sistema implementava cada minidisco alocando nos discos físicos quantas trilhas os minidiscos precisassem.

Uma vez que essas máquinas virtuais eram criadas, os usuários podiam executar qualquer um dos sistemas operacionais ou pacotes de software que estavam disponíveis na máquina subjacente. Para o sistema VM da IBM, normalmente o usuário executava o CMS — um sistema operacional interativo monusuário.

2.8.2 Benefícios

Há várias razões para a criação de uma máquina virtual. Basicamente, a maioria delas está relacionada com a capacidade de compartilhar o mesmo hardware e ainda assim processar vários ambientes de execução diferentes (isto é, diferentes sistemas operacionais) concorrentemente.

Uma vantagem importante é que o sistema hospedeiro fica protegido das máquinas virtuais, da mesma forma que as máquinas virtuais são protegidasumas das outras. Um vírus dentro de um sistema operacional convidado poderia danificar esse sistema operacional, mas provavelmente não afetará o hospedeiro ou os outros convidados. Já que cada máquina virtual fica totalmente isolada de todas as outras máquinas virtuais, não há problemas de proteção. Ao mesmo tempo, no entanto, não há compartilhamento direto de recursos. Duas abordagens para fornecer compartilhamento têm sido implementadas. Na primeira, é possível compartilhar um volume do sistema de arquivos e, portanto, compartilhar arquivos. Na segunda, é possível definir uma rede de máquinas virtuais, cada uma podendo enviar informações através da rede de comunicações virtual. A rede é modelada de acordo com as redes de comunicação físicas, mas é implementada em software.

Um sistema de máquina virtual é um veículo perfeito para a pesquisa e desenvolvimento de sistemas operacionais. Normalmente, mudar um sistema operacional é uma tarefa difícil. Os sistemas operacionais são programas grandes e complexos e é difícil garantir que uma alteração em uma parte não fará com que bugs obscuros apareçam em alguma outra parte. O poder do sistema operacional torna sua alteração particularmente perigosa. Já que o sistema operacional é executado na modalidade de kernel, uma al-

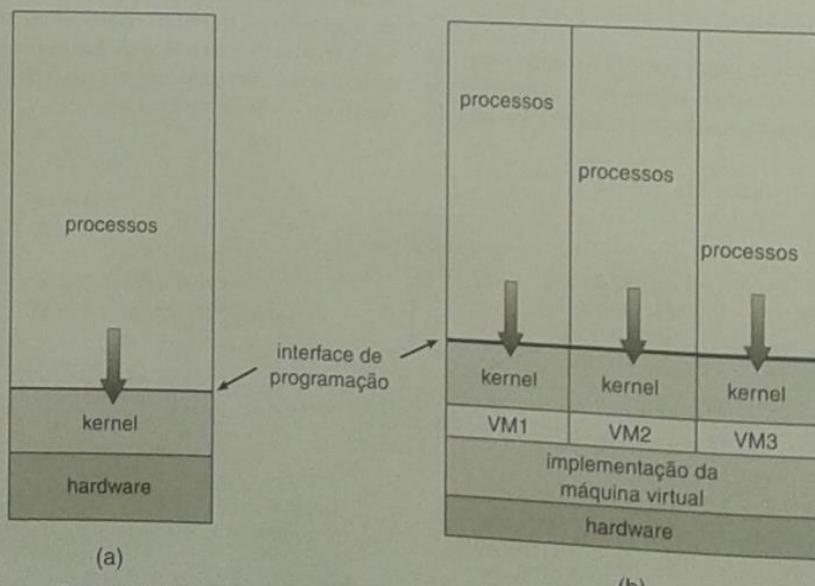


Figura 2.17 Modelos de sistema. (a) Máquina não virtual. (b) Máquina virtual.

teração errada em um ponteiro poderia causar um erro que destruiria o sistema de arquivos inteiro. Portanto, é necessário testar todas as alterações feitas no sistema operacional cuidadosamente.

No entanto, os sistemas operacionais são executados na máquina inteira e a controlam. Logo, o sistema corrente deve ser interrompido e deixar de ser usado enquanto alterações estiverem sendo feitas e testadas. Normalmente esse período é chamado *tempo de desenvolvimento do sistema*. Já que ele torna o sistema indisponível para os usuários, o tempo para desenvolvimento do sistema costuma ser agendado para tarde da noite ou nos fins de semana, quando a carga do sistema é baixa.

Um sistema de máquina virtual pode eliminar grande parte desse problema. Os programadores de sistema recebem sua própria máquina virtual, e o desenvolvimento é feito na máquina virtual em vez de em uma máquina física. Raramente a operação normal do sistema precisa ser interrompida pela atividade de desenvolvimento.

Outra vantagem das máquinas virtuais para os desenvolvedores é que vários sistemas operacionais podem ser executados na estação de trabalho do desenvolvedor concorrentemente. Essa estação de trabalho virtualizada permite a transferência rápida dos programas e o seu teste em vários ambientes. Da mesma forma, os engenheiros de gestão de qualidade podem testar suas aplicações em vários ambientes sem comprar, ligar e manter um computador para cada ambiente.

Uma grande vantagem das máquinas virtuais quando em centros de dados de produção é a *consolidação* do sistema o que envolve executar dois ou mais sistemas separados em máquinas virtuais sobre um único sistema. Essas conversões de físico para virtual resultam na otimização de recursos, já que sistemas pouco usados podem ser combinados para criar um único sistema de maior uso.

Se o uso de máquinas virtuais continuar aumentando, a implantação de aplicações também evoluirá. Se um sistema puder adicionar, remover e mover facilmente uma máquina virtual, por que instalar aplicações nesse sistema diretamente? Em vez disso, os desenvolvedores de aplicações instalariam previamente a aplicação em um sistema operacional ajustado e personalizado em uma máquina virtual. Tal ambiente virtual seria o mecanismo de liberação da aplicação. Esse método seria uma melhoria para os desenvolvedores de aplicações; o gerenciamento de aplicações ficaria mais fácil, menos ajustes seriam necessários e o suporte técnico à aplicação seria mais direto. Os administradores de sistemas também achariam o ambiente mais fácil de gerenciar. A instalação seria simples, e uma nova implantação da aplicação em outro sistema seria muito mais fácil do que as etapas normais de desinstalação e reinstalação. No entanto, para a ampla adoção dessa metodologia ocorrer, o formato das máquinas virtuais deve ser padronizado de modo que a máquina virtual seja executada em qualquer plataforma de virtualização. O "Formato de Máquina Virtual Aberta" é uma tentativa de fazer exatamente isso e poderia conseguir unificar os formatos das máquinas virtuais.

2.8.3 Simulação

A virtualização de sistemas como discutimos até agora é apenas uma das muitas metodologias de emulação de sistemas. A virtualização é a mais comum porque faz aplicações e sistemas operacionais convidados "acreditarem" que estão sendo executados em hardware nativo. Já que só os recursos do sistema precisam ser virtualizados, esses convidados são executados quase à velocidade máxima.

Outra metodologia é a *simulação*, na qual o sistema hospedeiro tem uma arquitetura e o sistema convidado foi compilado

em uma arquitetura diferente. Por exemplo, suponhamos que uma empresa tivesse substituído seu sistema de computação desatualizado por um novo sistema, mas quisesse continuar a executar certos programas importantes que foram compilados no sistema antigo. Os programas poderiam ser executados em um emulador que convertesse as instruções do sistema desatualizado para o conjunto de instruções nativo do novo sistema. A emulação pode aumentar a vida dos programas e permitir o uso de arquiteturas antigas sem o emprego de uma máquina antiga real, mas seu principal desafio é o desempenho. A emulação do conjunto de instruções pode ser executada em uma ordem de grandeza mais lenta do que as instruções nativas. Portanto, a menos que a nova máquina seja dez vezes mais rápida do que a antiga, o programa que estiver sendo executado na nova máquina será processado mais lentamente do que seria em seu hardware nativo. Outro desafio é o fato de ser difícil criar um emulador correto porque, na verdade, isso envolve a criação de uma CPU inteira em software.

2.8.4 Paravirtualização

A *paravirtualização* é outra variação desse tema. Em vez de tentar fazer com que um sistema operacional convidado acredite que ele tem um sistema para si próprio, a paravirtualização apresenta o convidado a um sistema que é semelhante, mas não idêntico, ao sistema que ele prefere. O convidado deve ser modificado para ser executado no hardware paravirtualizado. A vantagem desse trabalho adicional é o uso mais eficiente dos recursos e uma camada de virtualização menor.

O Solaris 10 inclui *contêineres*, ou *zonas*, que criam uma camada virtual entre o sistema operacional e as aplicações. Nesse sistema, só um kernel é instalado e o hardware não é virtualizado. Em vez disso, o sistema operacional e seus dispositivos são virtualizados, fornecendo processos dentro de um contêiner e dando a impressão de que eles são os únicos processos do sistema. Um ou mais contêineres podem ser criados e cada um pode ter suas próprias aplicações, pilhas de rede, endereço de rede e portas, contas de usuário e assim por diante. Os recursos da CPU podem ser divididos entre os contêineres e os processos de todo o sistema. A Figura 2.18 mostra um sistema Solaris 10 com dois contêineres e o espaço do usuário "global" padrão.

2.8.5 Implementação

Embora o conceito de máquina virtual seja útil, é difícil de implementar. Muito trabalho é necessário para o fornecimento de uma *duplicata* exata da máquina subjacente. Lembre-se de que normalmente a máquina subjacente tem duas modalidades: modalidade de usuário e modalidade de kernel. O software da máquina virtual pode ser executado na modalidade de kernel, já que é o sistema operacional. A própria máquina virtual só pode ser executada na modalidade de usuário. No entanto, da mesma forma que a máquina física tem duas modalidades, a máquina virtual também deve ter. Consequentemente, devemos ter uma modalidade de usuário virtual e uma modalidade de kernel virtual, as duas sendo executadas em uma modalidade de usuário física. As ações que causam uma transferência da modalidade de usuário para a modalidade de kernel em uma máquina real (como uma chamada de sistema ou uma tentativa de executar uma instrução privilegiada) também devem causar uma transferência da modalidade de usuário virtual para a modalidade de kernel virtual em uma máquina virtual.

Essa transferência pode ser feita da seguinte forma. Quando uma chamada de sistema, por exemplo, é feita por um programa sendo executado em uma máquina virtual na modalidade

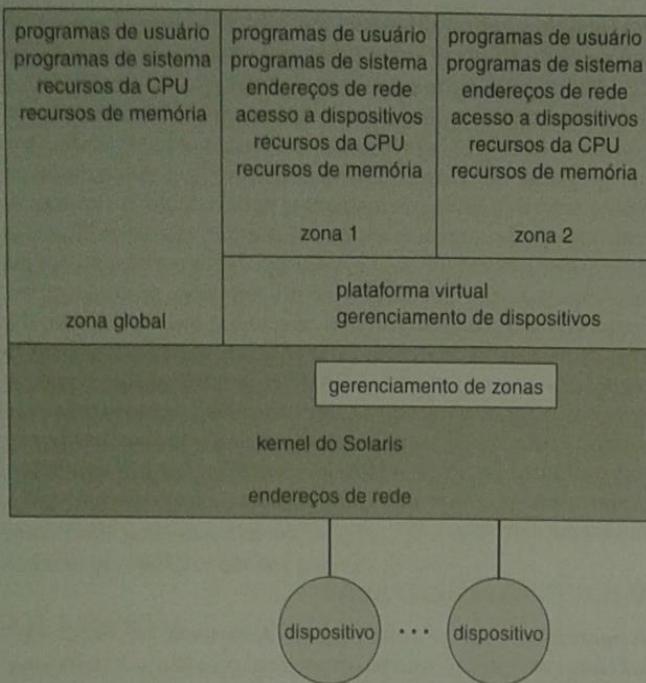


Figura 2.18 O Solaris 10 com dois contêineres.

de usuário virtual, ela causa uma transferência para o kernel da máquina virtual na máquina real. Quando o kernel da máquina virtual obtém o controle, ele pode alterar os conteúdos dos registradores e o contador do programa da máquina virtual para simular o efeito da chamada de sistema. Então, pode reiniciar a máquina virtual, observando que agora ela está na modalidade de kernel virtual.

É claro que a principal diferença é o tempo. Enquanto o I/O real pode levar 100 milissegundos, o I/O virtual pode levar menos tempo (porque entra em spool) ou mais tempo (porque é interpretado). Além disso, a CPU está sendo multiprogramada entre muitas máquinas virtuais, retardando ainda mais as máquinas virtuais de maneiras imprevisíveis. Em um caso extremo, pode ser necessária a simulação de todas as instruções para fornecer uma máquina virtual real. O VM, discutido anteriormente, funciona para máquinas IBM porque instruções normais das máquinas virtuais podem ser executadas diretamente no hardware. Só as instruções privilegiadas (necessárias principalmente para I/O) devem ser simuladas e, portanto, são executadas mais lentamente.

Sem algum nível de suporte de hardware, a virtualização seria impossível. Quanto maior o suporte de hardware disponível em um sistema, mais recursos, estabilidade e desempenho satisfatório as máquinas virtuais apresentam. Todas as principais CPUs de uso geral fornecem algum nível de suporte de hardware para a virtualização. Por exemplo, a tecnologia de virtualização AMD é encontrada em vários processadores AMD. Ela define dois novos modos de operação — hospedeiro e convidado. O software da máquina virtual pode ativar o modo de hospedeiro, definir as características de cada máquina virtual convidada e, então, passar o sistema para o modo de convidado, passando o controle para o sistema operacional convidado que está sendo executado na máquina virtual. No modo de convidado, o sistema operacional virtualizado pensa que está sendo executado em hardware nativo e vê certos dispositivos (aqueles incluídos na definição do hospedeiro para o convidado). Se o convidado tentar acessar um recurso virtualizado, o controle será passado ao hospedeiro para o gerenciamento dessa interação.

2.8.6 Exemplos

Apesar das vantagens das máquinas virtuais, elas receberam pouca atenção por vários anos após terem sido desenvolvidas. Atualmente, no entanto, as máquinas virtuais estão se tornando populares como um meio de resolver problemas de compatibilidade de sistemas. Nesta seção, examinaremos duas máquinas virtuais contemporâneas populares: o VMware Workstation e a máquina virtual Java. Como você verá, normalmente essas máquinas virtuais podem ser executadas sobre os sistemas operacionais de qualquer um dos tipos de projeto discutidos anteriormente. Portanto, os métodos de projeto do sistema operacional — camadas simples, microkernels, módulos e máquinas virtuais — não são mutuamente exclusivos.

2.8.6.1 VMware

A maior parte das técnicas de virtualização discutidas nesta seção requer que a virtualização tenha suporte no kernel. Outro método envolve a criação da ferramenta de virtualização para ser executada em modalidade de usuário como uma aplicação sobre o sistema operacional. As máquinas virtuais executadas dentro dessa ferramenta acreditam que estão sendo executadas em hardware puro, mas na verdade são executadas dentro de uma aplicação de nível de usuário.

O *VMware Workstation* é uma aplicação comercial popular que simula o Intel X86 e hardware compatível em máquinas virtuais isoladas. Ele é executado como uma aplicação em um sistema operacional hospedeiro, como o Windows ou o Linux, e permite que esse sistema hospedeiro execute concorrentemente vários sistemas operacionais convidados diferentes como máquinas virtuais independentes.

A arquitetura de um sistema assim é mostrada na Figura 2.19. Nesse cenário, o Linux está sendo executado como o sistema operacional hospedeiro, e o FreeBSD, o Windows NT e o Windows XP estão sendo executados como sistemas operacionais convidados. A camada de virtualização é a parte principal do VMware, já que ela simula o hardware físico em máquinas virtuais isoladas sendo executadas como sistemas operacionais convidados. Cada máquina virtual tem sua própria CPU virtual, memória, drives de disco, interfaces de rede e assim por diante.

O disco físico que o convidado possui e gerencia é na verdade apenas um arquivo dentro do sistema de arquivos do sistema operacional hospedeiro. Para criar uma instância convidada idêntica, só precisamos copiar o arquivo. A cópia do arquivo em outra localização protege a instância convidada contra uma falha na localização original. A transferência do arquivo para outra localização transfere o sistema convidado. Esses cenários mostram como a virtualização pode melhorar a eficiência da administração e o uso de recursos do sistema.

2.8.6.2 A Máquina Virtual Java

Java é uma linguagem de programação orientada a objetos popular introduzida pela Sun Microsystems em 1995. Além de uma especificação da linguagem e uma extensa biblioteca de APIs, o ambiente Java também fornece uma especificação da máquina virtual Java — ou JVM.

Os objetos Java são especificados com o construtor `class`; um programa Java é composto por uma ou mais classes. Para cada classe Java, o compilador produz um arquivo (`.class`) de saída com `bytecode` independente da arquitetura que será executado em qualquer implementação da JVM.

A JVM é uma especificação de um computador abstrato. Ela é composta por um **carregador de classes** e um **interpretador**

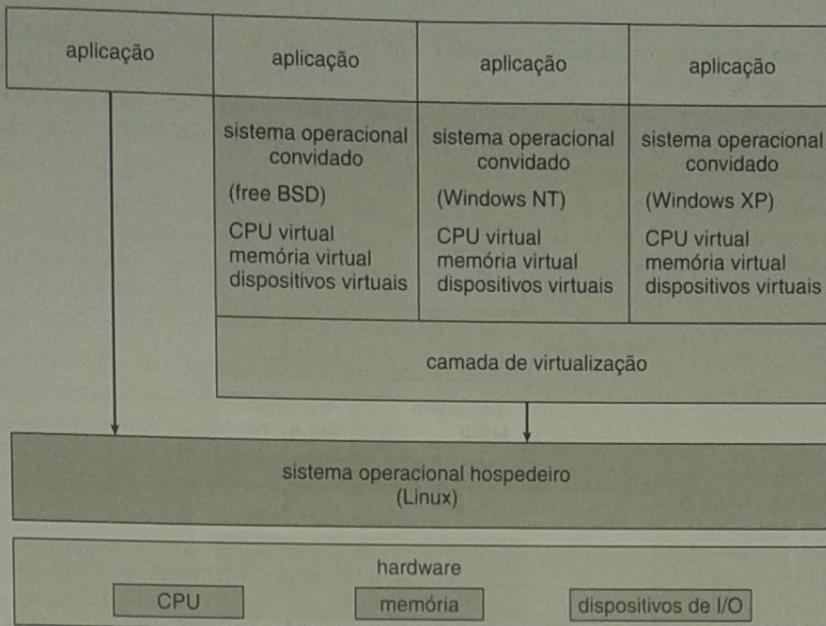


Figura 2.19 Arquitetura do VMware.

Java que executa os bytecodes independentes de arquitetura, como diagramado na Figura 2.20. O carregador de classes carrega os arquivos .class compilados a partir tanto do programa Java quanto da API Java para execução pelo interpretador Java. Após uma classe ser carregada, o verificador analisa se o arquivo .class tem bytecodes Java válidos e não estoura a pilha ou a subutiliza. Ele também garante que o bytecode não execute aritmética de ponteiros, o que poderia liberar acesso ilegal à memória. Se a classe passar na verificação, ela será executada pelo interpretador Java. A JVM também gerencia a memória automaticamente executando a **coleta de lixo** — a prática de reclamar a memória dos objetos que não estão mais sendo usados e devolvê-la ao sistema. Muitas pesquisas estudam o efeito de algoritmos de coleta de lixo para melhorar o desempenho de programas Java na máquina virtual.

A JVM pode ser implementada em software sobre um sistema operacional hospedeiro, como o Windows, o Linux ou o Mac OS X, ou como parte de um navegador Web. Alternativamente, a JVM pode ser implementada em hardware em um chip especificamente projetado para executar programas Java. Se a JVM for implementada em software, o interpretador Java interpreta as operações de bytecodes uma de cada vez. Uma técnica de software mais rápida é usar um compilador **just-in-time (JIT)**. Neste caso, na primeira vez em que um método Java é invocado,

os bytecodes para o método são convertidos em linguagem de máquina nativa do sistema hospedeiro. Essas operações são então armazenadas em cache para que invocações subsequentes de um método sejam executadas com o uso das instruções da máquina nativa e as operações de bytecodes não precisem ser interpretadas novamente. Uma técnica que é potencialmente ainda mais rápida é a execução da JVM em hardware em um chip Java especial que execute as operações de bytecodes Java como código nativo, eliminando assim a necessidade de um interpretador de software ou de um compilador just-in-time.

O AMBIENTE .NET

O Ambiente .Net é um conjunto de tecnologias, que inclui um grupo de bibliotecas de classes e um ambiente de execução que são disponibilizados conjuntamente para fornecer uma plataforma de desenvolvimento de software. Essa plataforma permite que sejam escritos programas direcionados ao Ambiente .NET em vez de a uma arquitetura específica. Um programa escrito para o Ambiente .NET não precisa se preocupar com as especificações do hardware ou do sistema operacional no qual será executado. Portanto, qualquer arquitetura que implementar o Ambiente .NET poderá executar com sucesso o programa. Isso ocorre porque o ambiente de execução ignora esses detalhes e fornece uma máquina virtual como intermediária entre o programa a ser executado e a arquitetura subjacente.

No centro do Ambiente .NET está o Common Language Runtime (CLR). O CLR é a implementação da máquina virtual .NET. Ele fornece um ambiente para a execução de programas escritos em qualquer uma das linguagens consideradas no Ambiente .NET. Programas escritos em linguagens como C# (pronuncia-se C-sharp) e VB.NET são compilados em uma linguagem intermediária independente de arquitetura chamada Microsoft Intermediate Language (MS-IL). Esses arquivos compilados, chamados de montagens, incluem instruções e metadados MS-IL. Eles têm extensões de arquivo .EXE ou .DLL. Na execução de

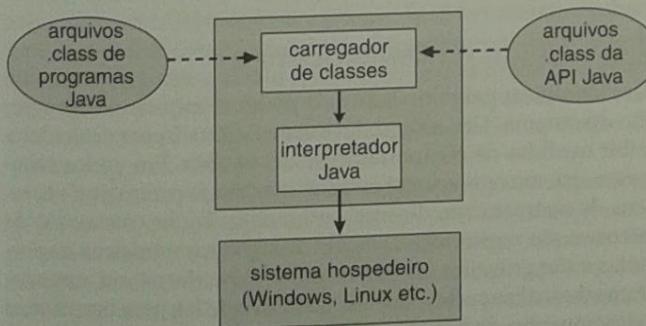


Figura 2.20 A máquina virtual Java.

um programa, o CLR carrega as montagens no que é conhecido como o **Domínio da Aplicação**. Quando instruções são solicitadas pelo programa que está em execução, o CLR converte as instruções MS-IL das montagens em código nativo que é específico da arquitetura subjacente, usando a compilação just-

in-time. Uma vez que as instruções tenham sido convertidas para código nativo, elas serão armazenadas e continuarão a ser executadas como código nativo para a CPU. A arquitetura do CLR para o Ambiente .NET é mostrada na Figura 2.21.

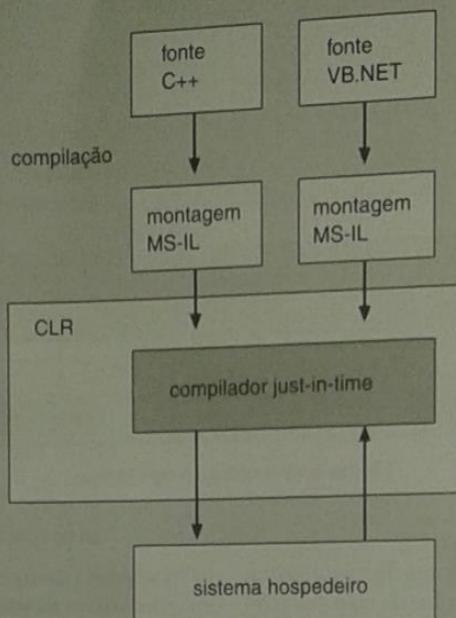


Figura 2.21 Arquitetura do CLR para o Ambiente .NET.

2.9 Depuração do Sistema Operacional

Em sentido amplo, **depuração** é a atividade de encontrar e corrigir erros, ou *bugs*, em um sistema. A depuração tenta encontrar e corrigir erros tanto de hardware quanto de software. Problemas de desempenho são considerados bugs; portanto, a depuração também pode incluir o *ajuste no desempenho*, que tenta melhorá-lo removendo *gargalos* do processamento que estiverem ocorrendo em um sistema. Uma discussão sobre a depuração de hardware não faz parte do escopo deste texto. Nesta seção, examinaremos a depuração do kernel, erros em processos e problemas de desempenho.

2.9.1 Análise de Falhas

Quando um processo falha, a maioria dos sistemas operacionais grava as informações de erro em um *arquivo de log* para alertar aos operadores ou usuários do sistema que o problema ocorreu. O sistema operacional também pode receber uma *descarga do núcleo* — uma captura da memória (conhecida como “núcleo” nos primórdios da computação) do processo. Essa imagem de núcleo é armazenada em um arquivo para análise posterior. Programas em execução e imagens do núcleo podem ser examinados por um *depurador*, uma ferramenta projetada para permitir que o programador examine o código e a memória de um processo.

A depuração do código de processos de nível de usuário é um desafio. A depuração do kernel do sistema operacional é ainda mais complexa por causa do tamanho e complexidade do kernel, seu controle sobre o hardware e a falta de ferramentas de depuração de nível de usuário. Uma falha no kernel é chamada de *desastre*. Como ocorre em uma falha de processo, as informações

de erro são salvas em um arquivo de log e o estado da memória é descarregado em um *arquivo de imagem do desastre*.

Geralmente a depuração de sistemas operacionais usa ferramentas e técnicas diferentes da depuração de processos devido à natureza tão distinta dessas duas tarefas. Considere que uma falha de kernel no código do sistema de arquivos tornaria arriscado para o kernel tentar salvar seu estado em um arquivo do sistema de arquivos antes da sua reinicialização. Uma técnica comum é salvar o estado da memória do kernel em uma seção de disco que não contenha sistema de arquivos e seja configurada para essa finalidade. Se o kernel detectar um erro irrecuperável, gravará todo o conteúdo da memória, ou pelo menos as partes da memória do sistema de propriedade do kernel, nessa área do disco. Quando o sistema for reinicializado, um processo será executado para coletar os dados dessa área e gravá-los em um arquivo de imagem do desastre dentro de um sistema de arquivos para análise.

2.9.2 Ajuste no Desempenho

Para identificar gargalos, temos de poder monitorar o desempenho do sistema. Um código deve ser adicionado para calcular e exibir medidas de comportamento do sistema. Em vários sistemas, o sistema operacional executa essa tarefa produzindo listagens de rastreamento do comportamento. Todos os eventos de interesse são registrados com sua duração e parâmetros importantes e são gravados em um arquivo. Posteriormente, um programa de análise pode processar o arquivo de log para determinar o desempenho do sistema e identificar gargalos e ineficiências. Esses mesmos rastreamentos podem ser executados como entra-

das para a simulação de uma proposta de otimização do sistema. Os rastreamentos também podem ajudar as pessoas a encontrar erros no comportamento do sistema operacional.

Lei de Kernighan

"Depurar é duas vezes mais difícil do que criar o código. Portanto, se você escrever o código da maneira mais inteligente possível, por definição, você não será suficientemente inteligente para depurá-lo."

Outra abordagem para o ajuste do desempenho é a inclusão de ferramentas interativas no sistema que permitam que os usuários e administradores examinem o estado de vários componentes do sistema para procurar gargalos. O comando UNIX `top` exibe os recursos usados no sistema, assim como uma lista ordenada dos processos que mais usam recursos. Outras ferramentas exibem o estado do I/O de disco, a alocação de memória e o tráfego de rede. Os autores dessas ferramentas de uso exclusivo tentam adivinhar o que um usuário gostaria de ver ao analisar um sistema e procuram fornecer essas informações.

Tornar a execução de sistemas operacionais mais fácil de entender, depurar e ajustar é uma área ativa da pesquisa e implementação de sistemas operacionais. O ciclo de ativação do rastreamento quando problemas ocorrem no sistema e de análise posterior dos rastreamentos está sendo rompido por uma nova geração de ferramentas de análise de desempenho habilitadas para o kernel. Além disso, essas ferramentas não são de uso exclusivo ou simplesmente destinadas às seções de código que foram criadas para emitir dados de depuração. O recurso de rastreamento dinâmico DTrace do Solaris 10 é um exemplo pioneiro desse tipo de ferramenta.

2.9.3 DTrace

O *DTrace* é um recurso que adiciona dinamicamente sondagens a um sistema em execução, tanto em processos de usuário quanto em processos do kernel. Essas sondagens podem ser consultadas através da linguagem de programação D para a visualização de uma quantidade enorme de informações sobre o kernel, o estado do sistema e as atividades dos processos. Por exemplo, a Figura 2.22 acompanha uma aplicação enquanto ela executa uma chamada de sistema (`ioctl`) e também mostra as chamadas funcionais dentro do kernel quando elas são processadas para executar a chamada. As linhas que terminam com "U" são executadas em modalidade de usuário e as que terminam em "K" em modalidade de kernel.

A depuração das interações entre código de kernel e de nível de usuário é quase impossível sem um grupo de ferramentas que entenda os dois conjuntos de código e possa instrumentar as operações. Para esse conjunto de ferramentas ser realmente útil, ele precisa poder depurar qualquer área de um sistema, inclusive áreas que não foram criadas visando a depuração, e fazer isso sem afetar a confiabilidade do sistema. Essa ferramenta também deve ter um impacto mínimo no desempenho — o ideal é que ela não cause impacto quando não estiver sendo usada e só cause um impacto proporcional durante o uso. A ferramenta DTrace atende esses requisitos e fornece um ambiente de depuração dinâmico, seguro e de baixo impacto.

Até a estrutura e as ferramentas do DTrace ficarem disponíveis no Solaris 10, a depuração do kernel era uma tarefa enigmática executada através de códigos e ferramentas casuais e arcaicos. Por exemplo, as CPUs têm um recurso de ponto de interrupção que interrompe a execução e permite que o depurador examine o estado do sistema. Em seguida, a execução pode continuar até o próximo ponto de interrupção ou o encerramento. Esse método não pode ser

```
# ./all.d 'pgrep xclock' XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                                U
  0  -> _XEventsQueued                            U
  0  -> _X11TransBytesReadable                      U
  0  <- _X11TransBytesReadable                      U
  0  -> _X11TransSocketBytesReadable                U
  0  <- _X11TransSocketBytesReadable                U
  0  -> ioctl                                      U
  0   -> ioctl                                    K
  0   -> getf                                     K
  0    -> set_active_fd                           K
  0    <- set_active_fd                           K
  0   <- getf                                    K
  0   -> get_udatamodel                         K
  0   <- get_udatamodel                         K
...
  0   -> releaseef                               K
  0    -> clear_active_fd                        K
  0    <- clear_active_fd                        K
  0   -> cv_broadcast                           K
  0    -> cv_broadcast                           K
  0    <- releaseef                             K
  0    <- ioctl                                 K
  0   <- ioctl                                 U
  0  <- _XEventsQueued                          U
  0 <- XEventsQueued                           U
```

Figura 2.22 O comando dtrace do Solaris 10 acompanha uma chamada de sistema dentro do kernel.

usado em um kernel de sistema operacional multiusuário sem afetar negativamente todos os usuários do sistema. A *geração de perfis*, que periodicamente coleta amostras do ponteiro de instruções para determinar que código está sendo executado, pode mostrar tendências estatísticas, mas não atividades individuais. Um código poderia ser incluído no kernel para a emissão de dados específicos sob determinadas circunstâncias, mas esse código tornaria o kernel lento e poderia não ser incluído na parte em que o problema específico que está sendo depurado está ocorrendo.

Por outro lado, o DTrace é executado em sistemas em produção — sistemas que estão executando aplicações importantes ou críticas — e não causam danos ao sistema. Ele retarda as atividades enquanto está ativo, mas após a execução reposiciona o sistema no seu estado pré-depuração. Além disso, é uma ferramenta abrangente e profunda. Pode depurar de forma abrangente tudo que está acontecendo no sistema (tanto nos níveis de usuário e de kernel quanto entre as camadas do usuário e do kernel). O DTrace também pode examinar profundamente o código, exibindo instruções individuais de CPU ou atividades de sub-rotina do kernel.

O *DTrace* é composto por um compilador, uma estrutura, *provedores de sondagens* criados dentro dessa estrutura e *consumidores* dessas sondagens. Os provedores do DTrace criam sondagens. Existem estruturas no kernel que controlam todas as sondagens que os provedores criarem. As sondagens são armazenadas em uma estrutura de dados de tabela hash que é codificada por nome e indexada de acordo com identificadores de sondagem exclusivos. Quando uma sondagem é ativada, um bloco de código da área a ser sondada é reescrito para chamar `dtrace_probe(probe identifier)` e continuar então com a operação original do código. Diferentes provedores criam diferentes tipos de sondagens. Por exemplo, uma sondagem de chamadas de sistema do kernel

funciona diferentemente de uma sondagem de processos do usuário, que é diferente de uma sondagem de I/O.

O DTrace vem com um compilador que gera um bytecode que é executado no kernel. O compilador garante a "confiabilidade" desse código. Por exemplo, loops não são permitidos e só certas modificações no estado do kernel são autorizadas quando solicitadas especificamente. Apenas usuários com os "privilegios" (ou usuários "root") do DTrace podem usá-lo, já que ele pode recuperar dados privados do kernel (e modificar dados se solicitado). O código gerado é executado no kernel e habilita sondagens. Ele também habilita consumidores na modalidade de usuário e permite comunicações entre os dois.

Um consumidor do DTrace é um código interessado em uma sondagem e em seus resultados. O consumidor solicita que o provedor crie uma ou mais sondagens. Quando uma sondagem é acionada, ela emite dados que são gerenciados pelo kernel. Dentro do kernel, ações chamadas *blocos de controle de ativação*, ou *ECBs* (*enabling control blocks*), são executadas quando as sondagens são acionadas. Uma sondagem pode fazer com que vários ECBs sejam executados se mais de um consumidor estiver interessado nela. Cada ECB contém um predicado ("comando if") que pode filtrá-la. Caso contrário, a lista de ações da ECB será executada. A ação mais comum é a captura de algum fragmento de dados, como o valor de uma variável no ponto da execução da sondagem. Através da coleta desses dados, um cenário completo de uma ação do usuário ou do kernel pode ser construído. Além disso, sondagens acionadas tanto a partir do espaço do usuário quanto do kernel podem mostrar como uma ação de nível de usuário causou reações no nível do kernel. Esses dados são inestimáveis para a monitoração do desempenho e a otimização de código.

Assim que o consumidor da sondagem terminar, seus ECBs serão removidos. Se não houver ECBs consumindo uma sondagem, ela será removida. Isso envolve a recriação do código para remover a chamada `dtrace_probe` e reinserir o código original. Portanto, antes de uma sondagem ser criada e após ela ser destruída, o sistema fica exatamente igual, como se nenhuma sondagem tivesse ocorrido.

O DTrace se encarrega de assegurar que as sondagens não usem capacidade de CPU ou memória demais, o que poderia danificar o sistema que está em execução. Os buffers usados no armazenamento dos resultados da sondagem são monitorados quanto à não ultrapassagem dos limites default e máximo. O tempo de CPU para a execução de sondagens também é monitorado. Se os limites forem excedidos, o consumidor será encerrado, junto com as sondagens ofensivas. Buffers são alocados por CPU para evitar a disputa e a perda de dados.

Um exemplo de código D e sua saída mostra parte de sua utilidade. O programa a seguir mostra o código DTrace que ativa sondagens no scheduler e regista o tempo de CPU de cada

# dtrace -s sched.d	
dtrace: script 'sched.d' matched 6 probes	
^C	
gnome-settings-d	142354
gnome-vfs-daemon	158243
dsdm	189804
wnck-applet	200030
gnome-panel	277864
clock-applet	374916
mapping-daemon	385475
xscreensaver	514177
metacity	539281
Xorg	2579646
gnome-terminal	5007269
mixer_applet2	7388447
java	10769137

Figura 2.23 Saída do código D.

processo que está sendo executado com ID de usuário 101 enquanto essas sondagens estão ativas (isto é, enquanto o programa é executado):

```

sched:::on-cpu
uid == 101
{
    self->ts = timestamp;
}

sched:::off-cpu
self->ts
{
    @time[execname] = sum(timestamp - self->ts);
    self->ts = 0;
}

```

A saída do programa, exibindo os processos e quanto tempo (em nanosegundos) eles permanecem sendo executados nas CPUs, é mostrada na Figura 2.23.

Como o DTrace faz parte do sistema operacional de código-fonte aberto Solaris 10, ele está sendo adicionado a outros sistemas operacionais quando esses sistemas não têm contratos de licença conflitantes. Por exemplo, o DTrace foi adicionado ao Mac OS X 10.5 e ao FreeBSD e provavelmente terá uma disseminação maior devido a seus recursos únicos. Outros sistemas operacionais, principalmente os derivados do Linux, também estão adicionando a funcionalidade de rastreamento do kernel. E há sistemas operacionais que estão começando a incluir ferramentas de desempenho e rastreamento fomentadas por pesquisas em várias instituições, o que inclui o projeto Paradyn.

2.10 Geração do Sistema Operacional

É possível projetar, codificar e implementar um sistema operacional especificamente para uma máquina de um determinado local. Geralmente, no entanto, os sistemas operacionais são projetados para execução em qualquer máquina de uma determinada classe de máquinas, em diversos locais, com inúmeras configurações periféricas. O sistema deve então ser configurado ou gerado para cada local específico do computador, um processo também conhecido como *geração do sistema* (SYSGEN).

Normalmente, o sistema operacional é distribuído em disco, em CD-ROM ou DVD-ROM, ou como uma imagem "ISO", que é um arquivo no formato de um CD-ROM ou DVD-ROM. Para

gerar um sistema, usamos um programa especial. Esse programa SYSGEN faz leituras em um determinado arquivo ou solicita ao operador do sistema informações relacionadas à configuração específica do sistema de hardware ou sonda o hardware diretamente para determinar que componentes estão disponíveis. Os tipos de informações a seguir devem ser determinados.

- Que CPU será usada? Que opções (conjuntos de instruções estendidos, aritmética de ponto flutuante e assim por diante) estão instaladas? Para sistemas com várias CPUs, cada CPU deve ser descrita.

- Como o disco de inicialização será formatado? Em quantas seções, ou “partições”, ele será dividido e o que haverá em cada partição?
- Qual a quantidade de memória disponível? Alguns sistemas determinarão esse valor eles próprios referenciando locação a locação da memória até que uma falha de “endereço inválido” seja gerada. Esse procedimento define o endereço final válido e, portanto, a quantidade de memória disponível.
- Que dispositivos estão disponíveis? O sistema terá de saber como endereçar cada dispositivo (o número do dispositivo), seu número de interrupção, seu tipo e modelo e qualquer característica especial do dispositivo.
- Que opções do sistema operacional são desejadas ou que valores de parâmetros devem ser usados? Essas opções ou valores podem incluir a quantidade e o tamanho dos buffers a serem usados, que tipo de algoritmo de scheduling da CPU é desejado, a quantidade máxima de processos suportada e assim por diante.

Assim que essas informações forem determinadas, elas podem ser usadas de várias maneiras. Em um extremo, um administrador de sistemas pode usá-las para modificar uma cópia do código-fonte do sistema operacional. O sistema operacional será

então totalmente compilado. Declarações de dados, inicializações e constantes, junto com uma compilação adicional, produzirão uma versão-objeto do sistema operacional personalizada para o sistema descrito.

Em um nível de personalização um pouco menor, a descrição do sistema pode levar à criação de tabelas e à seleção de módulos em uma biblioteca pré-compilada. Esses módulos são ligados para gerar o sistema operacional. A seleção supõe que a biblioteca contenha os drivers de todos os dispositivos de I/O suportados, mas só os necessários são ligados ao sistema operacional. Já que o sistema não é recompilado, sua geração é mais rápida, mas o sistema resultante pode ser excessivamente genérico.

No outro extremo, é possível construir um sistema totalmente dirigido por tabelas. O código inteiro faz parte do sistema o tempo todo, e a seleção ocorre em tempo de execução, em vez de em tempo de compilação ou ligação. A geração do sistema envolve simplesmente a criação das tabelas apropriadas para descrevê-lo.

As principais diferenças entre essas abordagens são o tamanho e generalidade do sistema gerado e a facilidade de modificá-lo quando a configuração do hardware muda. Considere o custo de modificação do sistema para dar suporte a um terminal gráfico recém-adquirido ou outro drive de disco. É claro que, como contraponto a esse custo, temos a frequência (ou infrequência) dessas alterações.

2.11 Inicialização do Sistema

Após um sistema operacional ser gerado, ele deve ser disponibilizado para uso pelo hardware. Mas como o hardware sabe onde está o kernel ou como carregar esse kernel? O procedimento de iniciar um computador a partir da carga do kernel é conhecido como *inicialização* do sistema. Na maioria dos sistemas de computação, um pequeno bloco de código conhecido como **programa bootstrap** ou **carregador bootstrap** localiza o kernel, carrega-o na memória principal e inicia sua execução. Alguns sistemas de computação, como os PCs, usam um processo de duas etapas em que um carregador bootstrap simples acessa um programa de inicialização mais complexo no disco, que, por sua vez, carrega o kernel.

Quando uma CPU recebe um evento de reinicialização — por exemplo, quando é ligada ou reinicializada —, o registrador de instruções é carregado com uma localização de memória predefinida e a execução começa aí. Nessa localização está o programa bootstrap inicial. Esse programa se encontra na forma de **memória de leitura (ROM)**, porque a RAM está em um estado desconhecido na inicialização do sistema. A ROM é conveniente porque não precisa de inicialização e não pode ser infectada facilmente por um vírus de computador.

O programa bootstrap pode executar várias tarefas. Geralmente, uma das tarefas é a execução de diagnósticos para a determinação do estado da máquina. Se os diagnósticos forem bem-sucedidos, o programa poderá continuar com as etapas de inicialização. Ele também pode inicializar todos os aspectos do sistema, dos registradores da CPU aos controladores de dispositivo e o conteúdo da memória principal. Assim que possível, ele inicia o sistema operacional.

Alguns sistemas — como os telefones celulares, os PDAs e consoles de jogos — armazenam o sistema operacional inteiro em ROM. O armazenamento do sistema operacional em ROM é adequado para sistemas operacionais pequenos, hardware de suporte simples e operações irregulares. Um problema dessa abordagem é que a alteração do código bootstrap requer a mudança dos chips de hardware da ROM. Alguns sistemas resolvem esse problema

usando **memória de leitura apagável e programável (EPROM)**, que é somente de leitura exceto quando recebe explicitamente um comando para se tornar gravável. Todos os tipos de ROM também são conhecidos como **firmware**, já que suas características se encaixam em algum ponto entre as de hardware e as de software. Um problema comum no firmware é que a execução do código nesse local é mais lenta do que em RAM. Alguns sistemas armazenam o sistema operacional em firmware e o copiam em RAM para a execução ser rápida. Um último problema do firmware é o fato de ele ser relativamente caro; portanto, geralmente só pequenas quantidades estão disponíveis.

Para sistemas operacionais grandes (inclusive a maioria dos sistemas operacionais de uso geral como o Windows, o Mac OS X e o UNIX) ou para sistemas que mudam com frequência, o carregador bootstrap é armazenado em firmware e o sistema operacional fica em disco. Nesse caso, o programa bootstrap executa diagnósticos e tem um trecho de código que pode ler um único bloco em uma localização fixa (por exemplo, o bloco zero) do disco, enviar para a memória e executar o código a partir desse **bloco de inicialização**. O programa armazenado no bloco de inicialização pode ser suficientemente sofisticado para carregar o sistema operacional inteiro na memória e começar sua execução. Normalmente, é um código simples (já que cabe em um único bloco do disco) e só conhece o endereço em disco e o tamanho do resto do programa bootstrap. O **GRUB** é um exemplo de programa bootstrap de código-fonte aberto para sistemas Linux. Toda a inicialização baseada em disco e o próprio sistema operacional podem ser facilmente alterados a partir da criação de novas versões em disco. Um disco que contém uma partição de inicialização (veja mais sobre isso na Seção 12.5.1) é chamado de **disco de inicialização** ou **disco do sistema**.

Agora que o programa bootstrap inteiro foi carregado, ele pode percorrer o sistema de arquivos para encontrar o kernel do sistema operacional, carregá-lo na memória e iniciar sua execução. Só nesse momento é que consideramos que o sistema está em execução.

12.12 Resumo

Os sistemas operacionais fornecem vários serviços. No baixo nível, chamadas de sistemas permitem que um programa em execução faça solicitações diretamente ao sistema operacional. Em um alto nível, o shell ou interpretador de comandos fornece um mecanismo para o usuário emitir uma solicitação sem escrever um programa. Os comandos podem ser provenientes de arquivos durante a execução em modo batch ou diretamente de um terminal quando é usado um modo interativo ou de tempo compartilhado. Programas de sistema são fornecidos para satisfazer solicitações muito comuns dos usuários.

Os tipos de solicitação variam de acordo com o nível. O nível de chamadas de sistema deve fornecer as funções básicas, como o controle de processos e a manipulação de arquivos e dispositivos. Solicitações de alto nível, atendidas pelo interpretador de comandos ou por programas do sistema, são convertidas em uma sequência de chamadas de sistema. Os serviços do sistema podem ser classificados em várias categorias: controle de programas, solicitações de status e solicitações de I/O. Os erros de programa podem ser considerados pedidos implícitos de serviço.

Uma vez que os serviços do sistema estejam definidos, a estrutura do sistema operacional pode ser desenvolvida. Várias tabelas serão necessárias para o registro das informações que definem o estado do sistema de computação e o status dos jobs do sistema.

O projeto de um novo sistema operacional é uma tarefa de peso. É importante que os objetivos do sistema sejam bem definidos antes de o projeto começar. O tipo de sistema desejado será a base das escolhas feitas entre os vários algoritmos e estratégias que serão necessários.

Como um sistema operacional é extenso, a modularidade é importante. Projetar um sistema como uma sequência de camadas ou usar um microkernel são consideradas boas técnicas. O conceito de máquina virtual adota a abordagem em camadas e trata tanto o kernel do sistema operacional quanto o hardware como se fossem hardware. Outros sistemas operacionais também podem ser carregados sobre essa máquina virtual.

Durante todo o ciclo de projeto do sistema operacional, devemos ter o cuidado de separar decisões políticas de detalhes (mecanismos) de implementação. Essa separação permite flexibilidade máxima se as decisões políticas tiverem de ser alteradas posteriormente.

Atualmente, os sistemas operacionais são quase sempre escritos em uma linguagem de implementação de sistemas ou em uma linguagem de alto nível. Essa característica melhora sua implementação, manutenção e portabilidade. Para criar um sistema operacional para uma configuração de máquina específica, devemos executar a geração do sistema.

A depuração de falhas em processos e no kernel pode ser feita com o uso de depuradores e outras ferramentas que analisem imagens do núcleo. Ferramentas como o DTrace analisam sistemas em produção para encontrar gargalos e entender outros comportamentos do sistema.

Para um sistema de computação começar a ser executado, a CPU deve ser inicializada e começar a executar o programa bootstrap em firmware. O programa bootstrap pode executar o sistema operacional diretamente se este também estiver no firmware ou pode concluir uma sequência em que carregue progressivamente programas mais inteligentes a partir do firmware e do disco até o próprio sistema operacional ser carregado na memória e executado.

Exercícios Práticos

- 2.1 Qual é a finalidade das chamadas de sistema?
- 2.2 Quais são as cinco principais atividades de um sistema operacional relacionadas ao gerenciamento de processos?
- 2.3 Quais são as três principais atividades de um sistema operacional relacionadas ao gerenciamento de memória?
- 2.4 Quais são as três principais atividades de um sistema operacional relacionadas ao gerenciamento de memória secundária?
- 2.5 Qual é a finalidade do interpretador de comandos? Por que geralmente ele é separado do kernel?
- 2.6 Que chamadas de sistema têm de ser executadas por um shell ou interpretador de comandos para iniciar um novo processo?
- 2.7 Qual é a finalidade dos programas de sistema?
- 2.8 Qual é a principal vantagem da abordagem em camadas para o projeto de sistemas? Quais são as desvantagens do uso da abordagem em camadas?
- 2.9 Liste cinco serviços fornecidos por um sistema operacional e explique por que cada um deles é conveniente para os usuários. Em que casos seria impossível programas de nível de usuário fornecerem esses serviços? Explique sua resposta.
- 2.10 Por que alguns sistemas armazenam o sistema operacional em firmware enquanto outros o armazenam em disco?
- 2.11 Como seria o projeto de um sistema que permitisse a escolha do sistema operacional a partir do qual se dará a inicialização? O que o programa bootstrap teria que fazer?

Exercícios

- 2.12 Os serviços e funções fornecidos por um sistema operacional podem ser divididos em duas categorias principais. Descreva resumidamente as duas categorias e discuta em que elas diferem.
- 2.13 Descreva três métodos gerais de passagem de parâmetros para o sistema operacional.
- 2.14 Descreva como você poderia obter um perfil estatístico do período de tempo gasto por um programa executando diferentes seções de seu código. Discuta a importância da obtenção desse perfil estatístico.
- 2.15 Quais são as cinco atividades principais de um sistema operacional relacionadas ao gerenciamento de arquivos?
- 2.16 Quais são as vantagens e desvantagens do uso da mesma interface de chamadas de sistema na manipulação tanto de arquivos quanto de dispositivos?
- 2.17 Seria possível para o usuário desenvolver um novo interpretador de comandos usando a interface de chamadas de sistema fornecida pelo sistema operacional?
- 2.18 Quais são os dois modelos de comunicação entre processos?

- Quais são as vantagens e desvantagens das duas abordagens?
- 2.19 Por que a separação entre mecanismo e política é desejável?
- 2.20 As vezes é difícil adotar uma abordagem em camadas quando dois componentes do sistema operacional dependem um do outro. Identifique um cenário em que não fica claro como devemos dispor em camadas dois componentes do sistema que requerem estreita integração de suas funcionalidades.
- 2.21 Qual é a principal vantagem da abordagem de microkernel para o projeto de sistemas? Como os programas de usuário e serviços do sistema interagem em uma arquitetura de microkernel? Quais são as desvantagens do uso da abordagem de microkernel?
- 2.22 De que maneiras a abordagem de kernel modular é semelhante à abordagem em camadas? De que maneiras ela difere da abordagem em camadas?
- 2.23 Qual é a principal vantagem de usar uma arquitetura de máquina virtual para um projetista de sistema operacional? Qual é a principal vantagem para o usuário?
- 2.24 Por que um compilador just-in-time é útil na execução de programas Java?
- 2.25 Qual é o relacionamento entre um sistema operacional convidado e um sistema operacional hospedeiro em um sistema como o VMware? Que fatores têm de ser considerados na seleção do sistema operacional hospedeiro?
- 2.26 O sistema operacional experimental Synthesis tem um montador incorporado ao kernel. Para otimizar o desempenho das chamadas de sistema, o kernel monta rotinas dentro de seu próprio espaço para reduzir o caminho que a chamada deve percorrer dentro dele. Essa abordagem é a antítese da abordagem em camadas, em que o caminho percorrido no kernel é estendido para tornar a construção do sistema operacional mais fácil. Discuta as vantagens e desvantagens da abordagem de projeto do kernel e de otimização do desempenho do sistema no Synthesis.

Problema de Programação

- 2.27 Na Seção 2.3, descrevemos um programa que copia o conteúdo de um arquivo em um arquivo de destino. Ao ser executado, primeiro esse programa solicita ao usuário o nome dos arquivos de origem e destino. Escreva o programa usando a API Win32 e a API POSIX. Certifique-se de inserir toda a verificação de erros necessária, verificando inclusive se o arquivo de origem existe.

Uma vez que você tenha projetado e testado corretamente o programa, se você usou um sistema que dê suporte a isso, execute o programa usando um utilitário que rastreie chamadas de sistema. Os sistemas Linux fornecem o utilitário `ptrace`, e os sistemas Solaris usam o comando `truss` ou `dtrace`. No Mac OS X, o recurso `ktrace` fornece funcionalidade semelhante. Já que os sistemas Windows não fornecem esses recursos, você terá de rastrear a versão Win32 desse programa usando um depurador.

Projeto de Programação

- 2.28 Adicionando uma chamada de sistema ao kernel do Linux.

Nesse projeto, você estudará a interface de chamadas de sistema fornecida pelo sistema operacional Linux e aprenderá como os programas de usuário se comunicam com o kernel do sistema operacional através dessa interface. Sua tarefa é incorporar uma nova chamada de sistema ao kernel, expandindo assim a funcionalidade do sistema operacional.

Parte 1: Iniciando

Uma chamada de procedimento na modalidade de usuário é executada passando argumentos para o procedimento na pilha ou através de registradores, salvando o estado corrente e do valor do contador do programa e saltando para o início do código correspondente ao procedimento chamado. O processo continua tendo os mesmos privilégios de antes.

As chamadas de sistema aparecem como chamadas de procedimento para os programas de usuário, mas resultam em uma alteração no contexto e nos privilégios de execução. No Linux sendo executado na arquitetura Intel 386, uma chamada de sistema é feita através do armazenamento do número da chamada no registrador EAX, do armazenamento de argumentos para a chamada em outros registradores de hardware e da execução de uma instrução de exceção (que é a instrução de montagem INT 0x80). Após a exceção ser executada, o número da chamada de sistema é usado na indexação em uma tabela de ponteiros de código para a obtenção do endereço inicial do código manipulador que implementa a chamada. O processo então

salta para esse endereço e seus privilégios são alterados da modalidade de usuário para a de kernel. Com os privilégios expandidos, o processo pode executar código de kernel, o que pode incluir instruções privilegiadas que não podem ser executadas na modalidade de usuário. O código de kernel pode então executar os serviços solicitados, como a interação com dispositivos de I/O, e pode executar o gerenciamento de processos e outras atividades que não podem ser executadas na modalidade de usuário.

Os números de chamada de sistema em versões recentes do kernel do Linux são listados em `/usr/src/linux-2.x/include/asm-i386/unistd.h`. (Por exemplo, `_NR_close` corresponde à chamada de sistema `close()`, que é invocada para o fechamento de um descritor de arquivo e é definida com o valor 6). Normalmente, a lista de ponteiros que conduzem a manipuladores de chamadas de sistema é armazenada no arquivo `/usr/src/linux-2.x/arch/i386/kernel/entry.S` sob o cabeçalho `ENTRY(sys_call_table)`. Observe que `sys_close` está armazenado na entrada número 6 da tabela para manter a consistência com o número de chamada de sistema definido no arquivo `unistd.h`. (A palavra-chave `.long` significa que a entrada ocupará a mesma quantidade de bytes de um valor de dados de tipo `long`.).

Parte 2: Construindo um Novo Kernel

Antes de adicionar uma chamada de sistema ao kernel, você deve se familiarizar com a tarefa de construir o binário de um kernel a partir de seu código-fonte e inicializar a máquina

com o kernel recém-construído. Essa atividade é composta pelas tarefas a seguir, algumas das quais dependem da instalação específica do sistema operacional Linux em uso.

- Obtenha o código-fonte do kernel da distribuição do Linux. Se o pacote do código-fonte já tiver sido instalado em sua máquina, os arquivos correspondentes podem estar disponíveis em `/usr/src/linux` ou `/usr/src/linux-2.x` (onde o sufixo corresponde ao número de versão do kernel). Se o pacote ainda não tiver sido instalado, pode ser baixado do fornecedor de sua distribuição do Linux ou de <http://www.kernel.org>.
- Aprenda como configurar, compilar e instalar o binário do kernel. Isso variará entre as diferentes distribuições do kernel, mas alguns comandos típicos para sua construção (após o acesso ao diretório em que o código-fonte do kernel está armazenado) incluem:
 - `make xconfig`
 - `make dep`
 - `make bzImage`
- Adicione uma nova entrada ao conjunto de kernels inicializáveis suportados pelo sistema. Normalmente, o sistema operacional Linux usa utilitários como `lilo` e `grub` para manter uma lista de kernels inicializáveis entre os quais os usuários podem escolher durante a inicialização da máquina. Se seu sistema der suporte ao `lilo`, adicione uma entrada a `lilo.conf`, como em:

```
image=/boot/bzImage.mykernel
label=mykernel
root=/dev/hda5
read-only
```

onde `/boot/bzImage.mykernel` é a imagem do kernel e `mykernel` é o rótulo associado ao novo kernel. Essa etapa permitirá que você selecione o novo kernel durante o processo de inicialização. Você terá então a opção de inicializar o novo kernel ou inicializar o kernel inalterado se o kernel recém-construído não funcionar apropriadamente.

Parte 3: Estendendo o Código-Fonte do Kernel

Agora você pode tentar adicionar um novo arquivo ao conjunto de arquivos-fonte usados na compilação do kernel. Normalmente, o código-fonte é armazenado no diretório `/usr/src/linux-2.x/kernel`, embora essa localização possa ser diferente em sua distribuição do Linux. Há duas opções para a inclusão da chamada de sistema. A primeira é adicionar a chamada de sistema a um arquivo-fonte existente nesse diretório. A segunda é criar um novo arquivo no diretório-fonte e modificar `/usr/src/linux-2.x/kernel/Makefile` para incluir o arquivo recém-criado no processo de compilação. A vantagem da primeira abordagem é que, quando modificamos um arquivo existente que já faz parte do processo de compilação, o arquivo `Makefile` não precisa ser modificado.

Parte 4: Adicionando uma Chamada de Sistema ao Kernel

Agora que você está familiarizado com as diversas tarefas secundárias correspondentes à construção e inicialização de

kernels Linux, pode começar o processo de inclusão de uma nova chamada de sistema no kernel. Nesse projeto, a chamada de sistema terá funcionalidade limitada; ela simplesmente passará da modalidade de usuário para a modalidade de kernel, exibirá uma mensagem que será registrada com as mensagens do kernel e passará novamente para a modalidade de usuário. Chamaremos essa chamada de chamada de sistema `helloworld`. Mesmo tendo funcionalidade limitada, ela ilustra o mecanismo das chamadas de sistema e demonstra a interação entre os programas de usuário e o kernel.

- Crie um novo arquivo chamado `helloworld.c` para definir sua chamada de sistema. Inclua os arquivos de cabeçalho `linux/linkage.h` e `linux/kernel.h`. Adicione o código a seguir a esse arquivo:

```
#include <linux/linkage.h>
#include <linux/kernel.h>
asm linkage int sys_helloworld() {
    printk(KERN_EMERG "hello world!");
}
```

Isso criará uma chamada de sistema com o nome `sys_helloworld()`. Se você decidir adicionar essa chamada de sistema a um arquivo existente no diretório-fonte, só terá de adicionar a função `sys_helloworld()` ao arquivo que escolher. No código, `asm linkage` é um remanescente dos dias em que o Linux usava tanto código C++ quanto C e é usado para indicar que o código foi escrito em C. A função `printk()` é usada para gravar mensagens em um arquivo de log do kernel e, portanto, só pode ser chamada a partir do kernel. As mensagens do kernel especificadas no parâmetro para `printk()` são registradas no arquivo `/var/log/kernel/warnings`. O protótipo de função para a chamada de `printk()` está definido em `/usr/include/linux/kernel.h`.

- Defina um novo número de chamada de sistema para `_NR_helloworld` em `/usr/src/linux-2.x/include/asm-i386/unistd.h`. Um programa de usuário pode usar esse número para identificar a chamada de sistema recém-adicionada. Certifique-se também de incrementar o valor de `_NR_syscalls`, que é armazenado no mesmo arquivo. Essa constante rastreia a quantidade de chamadas de sistema definidas correntemente no kernel.
- Adicione uma entrada `.long sys_helloworld` à tabela `sys_call_table` definida no arquivo `/usr/src/linux-2.x/arch/i386/kernel/entry.S`. Como discutido anteriormente, o número da chamada de sistema é usado como índice nessa tabela para encontrar a localização da posição do código manipulador da chamada de sistema invocada.
- Adicione seu arquivo `helloworld.c` ao arquivo `Makefile` (se você criou um novo arquivo para sua chamada de sistema). Salve uma cópia de sua antiga imagem binária do kernel (para o caso de haver problemas com o kernel recém-criado). Agora você pode construir o novo kernel, renomeá-lo para distingui-lo do kernel inalterado e adicionar uma entrada aos arquivos de configuração do carregador (como `lilo.conf`). Após concluir essas etapas, você poderá inicializar o kernel antigo ou o novo kernel que contém sua chamada de sistema.

Parte 5: Usando a Chamada de Sistema a Partir de um Programa de Usuário

Quando você executar a inicialização com o novo kernel, ele dará suporte à chamada de sistema recém-definida; agora você só precisa invocar essa chamada de sistema a partir de um programa de usuário. Normalmente, a biblioteca C padrão dá suporte a uma interface de chamadas de sistema definida para o sistema operacional Linux. Já que sua nova chamada de sistema não está ligada à biblioteca C padrão, sua invocação demandará intervenção manual.

Como mencionado anteriormente, uma chamada de sistema é invocada pelo armazenamento do valor apropriado em um registrador de hardware e pela execução de uma instrução de exceção. Infelizmente, essas operações de baixo nível não podem ser executadas com o uso de comandos da linguagem C e em vez disso requerem instruções de montagem. Por sorte, o Linux fornece macros para a instânciação de funções encapsuladoras contendo as instruções de montagem apropriadas. Por exemplo, o programa C a seguir usa a macro `_syscall10()` para invocar a chamada de sistema recém-definida:

```
#include <linux/errno.h>
#include <sys/syscall.h>
#include <linux/unistd.h>

_syscall10(int, helloworld);

main()
{
    helloworld();
}
```

- A macro `_syscall10` tem dois argumentos. O primeiro especifica o tipo do valor retornado pela chamada de sistema; o segundo é o nome da chamada de sistema. O nome é usado para identificar o número da chamada de sistema que é armazenado no registrador de hardware antes de a instrução de exceção ser executada. Se sua chamada de sistema demandar argumentos, uma macro diferente (como `_syscall10`, onde o sufixo indica a quantidade de argumentos) poderia ser usada para instanciar o código de montagem necessário à execução da chamada.
- Compile e execute o programa com o kernel recém-construído. Deve haver uma mensagem “hello world!” no arquivo de log `/var/log/kernel/warnings` do kernel para indicar que a chamada de sistema foi executada.

Como próxima etapa, considere a expansão da funcionalidade de sua chamada de sistema. Como você passaria um valor inteiro ou uma cadeia de caracteres para a chamada de sistema e o faria ser gravado no arquivo de log do kernel? Quais são as implicações de passar ponteiros para dados armazenados no espaço de endereços do programa de usuário em vez de simplesmente passar um valor inteiro do programa do usuário para o kernel usando registradores de hardware?

Notas Bibliográficas

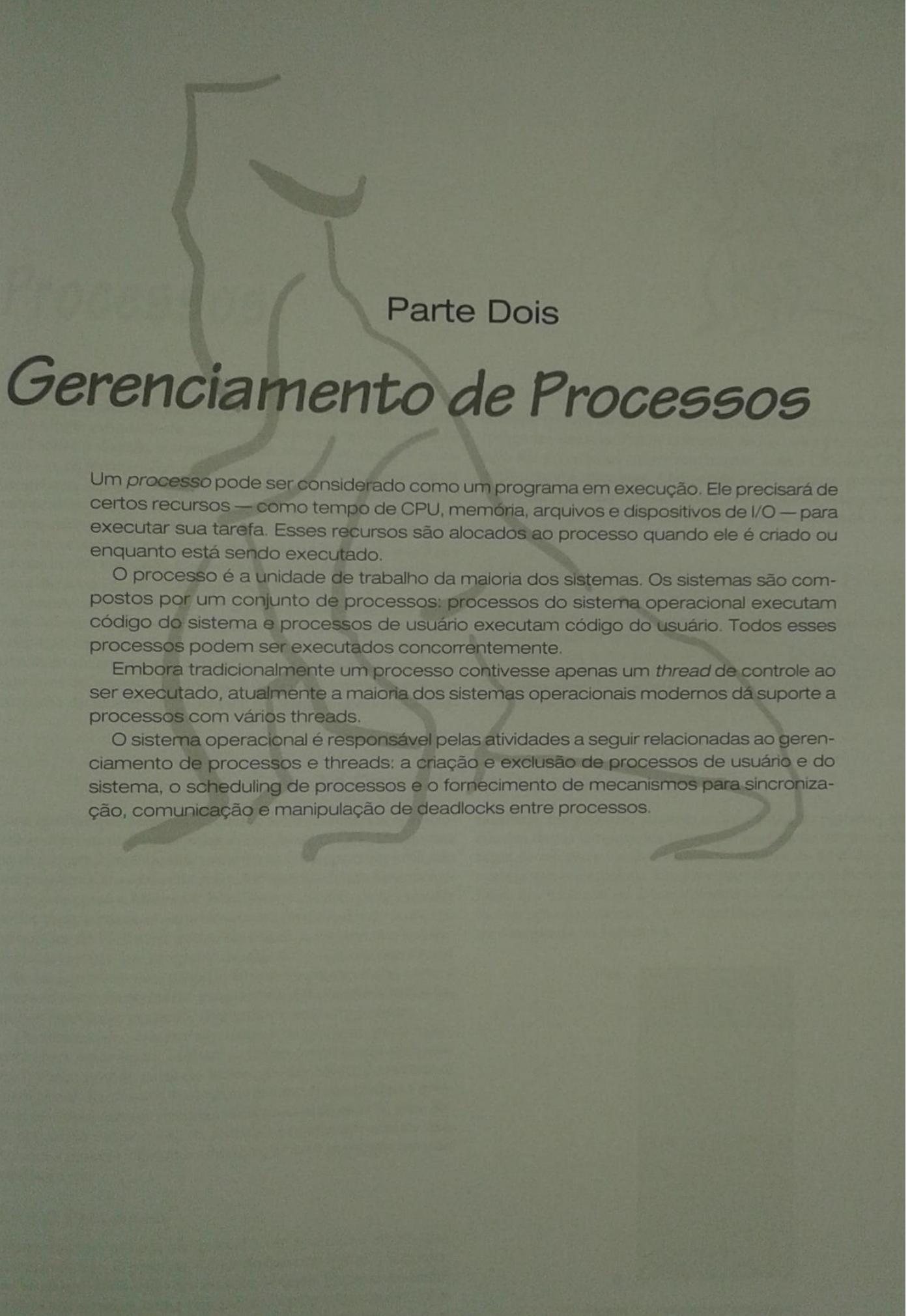
Dijkstra [1968] defendeu a abordagem em camadas para o projeto de sistemas operacionais. Brinch-Hansen [1970] foi um propONENTE pIONEIRO DA CONSTRUÇÃO DE UM SISTEMA OPERACIONAL COMO UM KERNEL (OU NÚCLEO) SOBRE O QUAL SISTEMAS MAIS COMPLETOS PUDESSEM SER CONSTRUÍDOS.

A instrumentação do sistema e o rastreamento dinâmico são descritos em Tamches e Miller [1999]. O DTrace é discutido em Cantrill et al. [2004]. O código-fonte do DTrace está disponível em <http://src.opensolaris.org/source/>. Cheung e Long [1995] examinam questões relacionadas à estrutura do sistema operacional abordando desde sistemas de microkernel a sistemas extensíveis.

O MS-DOS, versão 3.1, é descrito pela Microsoft [1986]. O Windows NT e o Windows 2000 são descritos por Solomon [1998] e Solomon e Russinovich [2000]. Os mecanismos internos do Windows 2003 e do Windows XP são descritos em Russinovich e Solomon [2005]. Hart [2005] aborda a programação de sistemas do Windows em detalhes. O BSD UNIX é descrito em McKusick et al. [1996]. Bovet e Cesati [2006] discutem detalhadamente o kernel do Linux. Vários sistemas UNIX — inclusive o Mach — são tratados com detalhes em Vahalia [1996]. O Mac OS X é apresentado em <http://www.apple.com/macosx> e em Singh [2007]. O Solaris é descrito detalhadamente em McDougall e Mauro [2007].

O primeiro sistema operacional a fornecer uma máquina virtual foi o CP/67 em um IBM 360/67. O sistema operacional comercialmente disponível VM/370 da IBM foi derivado do CP/67. Detalhes relacionados ao Mach, um sistema operacional baseado em microkernel, podem ser encontrados em Young et al. [1987]. Kaashoek et al. [1997] apresentam detalhes relacionados a sistemas operacionais de exokernel, em que a arquitetura separa as questões de gerenciamento das de proteção, dando assim a oportunidade a softwares não confiáveis de exercerem controle sobre recursos de hardware e software.

As especificações da linguagem Java e da máquina virtual Java são apresentadas por Gosling et al. [1996] e por Lindholm e Yellin [1999], respectivamente. Os mecanismos internos da máquina virtual Java são descritos detalhadamente por Venners [1998]. Golm et al. [2002] destacam o sistema operacional JX; Back et al. [2000] abordam várias questões relacionadas ao projeto de sistemas operacionais Java. Mais informações sobre Java estão disponíveis na Web em <http://www.javasoft.com>. Detalhes sobre a implementação do VMware podem ser encontrados em Sugerman et al. [2001]. Informações sobre o Formato de Máquina Virtual Aberta podem ser encontrados em <http://www.vmware.com/appliances/learn.ovf.html>.



Parte Dois

Gerenciamento de Processos

Um processo pode ser considerado como um programa em execução. Ele precisará de certos recursos — como tempo de CPU, memória, arquivos e dispositivos de I/O — para executar sua tarefa. Esses recursos são alocados ao processo quando ele é criado ou enquanto está sendo executado.

O processo é a unidade de trabalho da maioria dos sistemas. Os sistemas são compostos por um conjunto de processos: processos do sistema operacional executam código do sistema e processos de usuário executam código do usuário. Todos esses processos podem ser executados concorrentemente.

Embora tradicionalmente um processo contivesse apenas um *thread* de controle ao ser executado, atualmente a maioria dos sistemas operacionais modernos dão suporte a processos com vários threads.

O sistema operacional é responsável pelas atividades a seguir relacionadas ao gerenciamento de processos e threads: a criação e exclusão de processos de usuário e do sistema, o scheduling de processos e o fornecimento de mecanismos para sincronização, comunicação e manipulação de deadlocks entre processos.