O Livro do MINIX

Sistemas Operacionais

Projeto e Implementação

TERCEIRA EDIÇÃO

INCLUI CD-ROM

Andrew S. Tanenbaum
Albert S. Woodhull

Bookman®

T164s Tanenbaum, Andrewa S.
Sistemas operacionais [recurso eletrônico] : projeto e implementação / Andrew S. Tanenbaum, Albert S. Woodhull ; tradução João Tortello. – 3. ed. – Dados eletrônicos. – Porto Alegre : Bookman, 2008.

Editado também como livro impresso em 2008.
ISBN 978-85-7780-285-2

1. Sistemas operacionais. I. Woodhull, Albert S. II. Título.

CDU 004.4

Catalogação na publicação: Mônica Ballejo Canto – CRB 10/1023

nada à impressora. Quando um programa tiver terminado, o sistema operacional envia para a impressora a sua saída armazenada no arquivo em disco, enquanto, ao mesmo tempo, um outro programa poderá continuar gerando mais saída, ignorando o fato de que ela não está realmente indo para a impressora (ainda).

Quando um computador (ou uma rede) tem vários usuários, a necessidade de gerenciar e proteger a memória, dispositivos de E/S e outros recursos é ainda maior, pois os usuários poderiam interferir uns com os outros. Além disso, os usuários freqüentemente precisam compartilhar não apenas o hardware, mas também informações (arquivos, bancos de dados etc.). Em resumo, essa visão do sistema operacional sustenta que sua principal tarefa é controlar quem está usando qual recurso, garantir os pedidos de recursos, medir a utilização e mediar pedidos conflitantes de diferentes programas e usuários.

O gerenciamento de recursos inclui a multiplexação (compartilhamento) de recursos de duas maneiras: no tempo e no espaço. Quando um recurso é multiplexado no tempo, diferentes programas ou usuários o utilizam por turnos: primeiro um deles utiliza o recurso, depois outro e assim por diante. Por exemplo, com apenas uma CPU e vários programas que queiram ser executados nela, o sistema operacional primeiramente aloca a CPU para um programa e, depois, após o programa ter executado o suficiente, outro programa utiliza a CPU, depois outro e, finalmente, o primeiro programa novamente. Determinar como o recurso é multiplexado no tempo – quem vem em seguida e por quanto tempo – é tarefa do sistema operacional. Outro exemplo de multiplexação no tempo é o compartilhamento da impressora. Quando várias tarefas de impressão são enfileiradas em uma única impressora, uma decisão precisa ser tomada com relação a qual das tarefas deve ser impressa a seguir.

O outro tipo de multiplexação é a no espaço. Em vez dos clientes atuarem por turnos, cada um deles recebe parte do recurso. Por exemplo, normalmente, a memória principal é dividida entre vários programas que estejam em execução para que cada um possa estar residente ao mesmo tempo (por exemplo, para utilizar a CPU por turnos). Supondo que haja memória suficiente para conter múltiplos programas, é mais eficiente manter vários programas na memória de uma vez do que alocar toda ela para um único programa, especialmente se ele precisar apenas de uma pequena fração do total. É claro que isso levanta problemas de imparcialidade, proteção etc., e fica por conta do sistema operacional resolvê-los. Outro recurso multiplexado no espaço é o disco (rígido). Em muitos sistemas, um único disco pode conter arquivos de muitos usuários ao mesmo tempo. Alocar espaço em disco e controlar quem está usando quais blocos de disco é uma típica tarefa de gerenciamento de recursos do sistema operacional.

1.2 HISTÓRIA DOS SISTEMAS OPERACIONAIS

Os sistemas operacionais vêm evoluindo ao longo dos anos. Nas seções a seguir, veremos resumidamente alguns dos destaques. Como, historicamente, os sistemas operacionais têm sido intimamente ligados à arquitetura dos computadores em que são executados, examinaremos as sucessivas gerações de computadores para vermos como eram seus sistemas operacionais. Esse mapeamento das gerações de sistema operacional para gerações de computador é grosseiro, mas oferece uma base que de outra forma não teríamos.

O primeiro computador digital foi projetado pelo matemático inglês Charles Babbage (1792–1871). Embora Babbage tenha gasto a maior parte de sua vida e de sua fortuna tentando construir sua "máquina analítica", nunca conseguiu fazê-la funcionar corretamente, pois ela era puramente mecânica e a tecnologia de seu tempo não podia produzir as rodas e engrenagens exigidas com a alta precisão que necessitava. É desnecessário dizer que a máquina analítica não tinha sistema operacional.

Como um dado histórico interessante, Babbage percebeu que precisaria de software para sua máquina analítica; assim, contratou como a primeira programadora do mundo, uma jovem chamada Ada Lovelace, que era filha do famoso poeta britânico Lord Byron. A linguagem de programação Ada® recebeu esse nome em sua homenagem.

1.2.1 A primeira geração (1945–1955): válvulas e painéis de conectores

Depois dos esforços mal-sucedidos de Babbage, pouco progresso foi feito na construção de computadores digitais até a II Guerra Mundial. Em meados da década de 40, Howard Aiken, da Universidade de Harvard, John von Neumann, do Instituto de Estudos Avançados de Princeton, J. Presper Eckert e John Mauchley, da Universidade da Pensilvânia, e Konrad Zuse, na Alemanha, entre outros, tiveram êxito na construção de máquinas de calcular. As primeiras delas usavam relés mecânicos, mas eram muito lentas, com tempos de ciclo medidos em segundos. Posteriormente, os relés foram substituídos por válvulas a vácuo. Essas máquinas eram enormes, ocupavam salas inteiras com dezenas de milhares de válvulas, mas ainda eram milhões de vezes mais lentas do que os computadores pessoais mais baratos de hoje.

Naqueles tempos, um único grupo de pessoas projetava, construía, programava, operava e mantinha cada máquina. Toda a programação era feita em linguagem de máquina pura, freqüentemente interligando fios através de painéis de conectores para controlar as funções básicas da máquina. As linguagens de programação não existiam (nem mesmo a linguagem *assembly*). Ninguém tinha ouvido falar de sistemas operacionais. O modo de operação normal era o programador reservar um período de tempo em uma folha de reserva afixada na parede, depois descer à sala da máquina, inserir seu painel de conectores no computador e passar as próximas horas esperando que nenhuma das quase 20.000 válvulas queimasse durante a execução. Praticamente todos os problemas resolvidos eram cálculos numéricos simples, como a geração de tabelas de senos, co-senos e logaritmos.

No início da década de 50, a rotina havia melhorado um pouco, com a introdução dos cartões perfurados. Agora era possível, em vez de usar painéis de conectores, escrever programas em cartões de papel e lê-los; fora isso, o procedimento era o mesmo.

1.2.2 A segunda geração (1955–1965): transistores e sistemas de lote

A introdução do transistor, em meados da década de 50, mudou o quadro radicalmente. Os computadores se tornaram confiáveis o bastante para serem fabricados e vendidos para clientes com a expectativa de que continuariam a funcionar por tempo suficiente para realizarem algum trabalho útil. Pela primeira vez, havia uma separação clara entre projetistas, construtores, operadores, programadores e pessoal de manutenção.

Essas máquinas, agora chamadas de **computadores de grande porte** (ou *mainframes*), eram postas em salas especiais com ar-condicionado e com equipes de operadores profissionais especialmente treinadas para mantê-las funcionando. Somente grandes empresas, importantes órgãos do governo ou universidades podiam arcar com seu preço, na casa dos milhões de dólares. Para executar um ***job*** (tarefa), isto é, um programa ou um conjunto de programas, um programador primeiro escrevia o programa no papel (em FORTRAN ou possivelmente até em linguagem *assembly*) e depois o transformava em cartões perfurados. Então, ele levava a pilha de cartões para a sala de submissão de *jobs*, o entregava para um dos operadores e ia tomar café até que a saída estivesse pronta. Quando o computador terminava o *job* que estava executando, um operador ia até a impressora, destacava a saída impressa e a levava para uma sala de saída, para que o programador pudesse pegá-la posteriormente. Então, o operador pegava uma das pilhas de cartões que tinham sido trazidas para a sala de submissão

e os inseria na máquina de leitura. Se o compilador FORTRAN fosse necessário, o operador teria que pegá-lo em um gabinete de arquivos e inseri-lo para leitura. Enquanto os operadores andavam pelas salas da máquina, de submissão e de saída, o computador ficava ocioso. Dado o alto custo do equipamento, não é de surpreender que as pessoas procurassem rapidamente maneiras de reduzir o tempo desperdiçado. A solução geralmente adotada era o **sistema de processamento em lotes** (*batch system*). A idéia era reunir em uma bandeja (*tray*) um conjunto de *jobs* da sala de submissão e então lê-los em uma fita magnética usando um computador relativamente pequeno e barato, como o IBM 1401, que era muito bom para ler cartões, copiar fitas e imprimir a saída, mas péssimo para cálculos numéricos. Outras máquinas muito mais caras, como o IBM 7094, eram usadas para a computação de fato. Essa situação aparece na Figura 1-2.

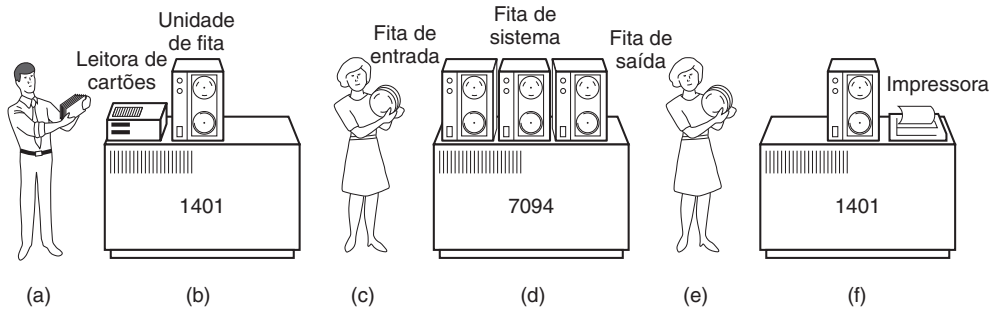

Figura 1-2 Um sistema de processamento em lotes primitivo. (a) Os programadores trazem os cartões para o 1401. (b) O 1401 lê o lote de *jobs* na fita. (c) O operador leva a fita de entrada para o 7094. (d) O 7094 realiza a computação. (e) O operador leva a fita de saída para o 1401. (f) O 1401 imprime a saída.

Após cerca de uma hora de leitura de lotes de *jobs*, a fita era rebobinada e levada para a sala da máquina, onde era montada em uma unidade de fita. O operador carregava então um programa especial (o ancestral do sistema operacional de hoje), que lia o primeiro *job* da fita e a executava. A saída era gravada em uma segunda fita, em vez de ser impressa. Depois que cada *job* terminava, o sistema operacional lia automaticamente o próximo *job* da fita e começava a executá-lo. Quando o lote inteiro estava pronto, o operador removia as fitas de entrada e saída, substituía a fita de entrada pelo próximo lote e levava a fita de saída para um 1401 imprimir ***off line*** (isto é, não conectado ao computador principal).

A estrutura típica de um *job* aparece na Figura 1-3. Ele começava com um cartão $JOB, especificando o tempo de execução máximo, em minutos, o número da conta a ser cobrada e o nome do programador. Em seguida, vinha um cartão $FORTRAN, instruindo o sistema operacional a carregar o compilador FORTRAN da fita de sistema. Depois, vinha o programa a ser compilado e, então, um cartão $LOAD, orientando o sistema operacional a carregar o programa-objeto recém compilado. (Freqüentemente, os programas compilados eram gravados em fitas virgens e tinham de ser carregados explicitamente.) Em seguida, vinha o cartão $RUN, instruindo o sistema operacional a executar o programa com os dados que o seguiam. Finalmente, o cartão $END marcava o fim do *job*. Esses cartões de controle primitivos foram os precursores das linguagens de controle de *jobs* modernos e dos interpretadores de comandos.

Grandes computadores de segunda geração eram usados principalmente para cálculos científicos e de engenharia, como a solução de equações diferenciais parciais que freqüentemente ocorrem na física e na engenharia. Eles eram programados principalmente em

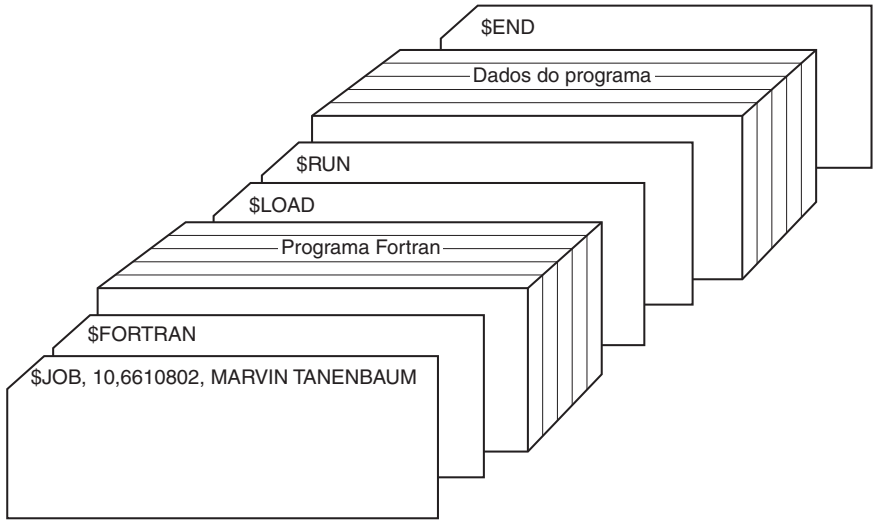

Figura 1-3 Estrutura típica de um *job* FMS.

FORTRAN e em linguagem *assembly*. Sistemas operacionais típicos eram o FMS (*o Fortran Monitor System*) e o IBSYS, sistema operacional da IBM para o 7094.

1.2.3 A terceira geração (1965–1980): CIs e multiprogramação

No início da década de 60, a maioria dos fabricantes de computadores tinha duas linhas de produtos distintas e totalmente incompatíveis. Por um lado, havia os computadores científicos de grande escala, baseados em palavras binárias, como o 7094, que eram utilizados para cálculos numéricos em ciência e engenharia. Por outro, havia os computadores comerciais, baseados em caracteres, como o 1401, que eram amplamente usados por bancos e companhias de seguro para ordenar e imprimir fitas.

Desenvolver, manter e comercializar duas linhas de produtos completamente diferentes era uma proposta cara para os fabricantes de computadores. Além disso, muitos clientes novos necessitavam, inicialmente, de uma máquina pequena, mas posteriormente cresciam e queriam uma máquina maior, que tivesse a mesma arquitetura da atual para poderem executar todos os seus programas antigos, só que mais rapidamente.

A IBM tentou resolver esses dois problemas de uma só vez introduzindo o System/360. O 360 era uma série de máquinas de software compatível que variavam desde a capacidade de um 1401 até um muito mais poderoso do que o 7094. As máquinas diferiam apenas no preço e no desempenho (capacidade máxima de memória, velocidade do processador, número de dispositivos de E/S permitidos etc.). Como todas as máquinas tinham a mesma arquitetura e o mesmo conjunto de instruções, os programas escritos para uma podiam ser executados em todas as outras, pelo menos teoricamente. Além disso, o 360 foi projetado para realizar computação científica (isto é, numérica) e comercial. Assim, uma única família de máquinas podia satisfazer as necessidades de todos os clientes. Nos anos seguintes, a IBM lançou novos produtos sucessores usando tecnologia mais moderna, compatíveis com a linha 360, conhecidos como as séries 370, 4300, 3080, 3090 e Z.

O 360 foi a primeira linha de computadores importante a usar circuitos integrados (CIs), oferecendo assim uma importante vantagem de preço/desempenho em relação às máquinas de segunda geração, que eram construídas a partir de transistores individuais. Ele teve sucesso imediato e a idéia de uma família de computadores compatíveis logo foi ado-

tada por todos os outros principais fabricantes. As descendentes dessas máquinas ainda são empregadas nos centros de computação atuais. Hoje em dia, elas são freqüentemente usadas para gerenciamento de bancos de dados grandes (por exemplo, para sistemas de reservas de passagens aéreas) ou como servidores de sites web que precisam processar milhares de pedidos por segundo.

A maior força da idéia de "uma família" foi ao mesmo tempo sua maior fraqueza. A intenção era que todo software, incluindo o sistema operacional, o **OS/360**, funcionasse em todos os modelos. Ele tinha que funcionar em sistemas pequenos, que freqüentemente apenas substituíam os 1401 para copiar cartões em fita, e em sistemas muito grandes, que muitas vezes substituíam os 7094 para fazer previsão do tempo e outros cálculos pesados. Ele tinha que ser bom em sistemas com poucos periféricos e em sistemas com muitos periféricos. Tinha que funcionar em ambientes comerciais e em ambientes científicos. Acima de tudo, ele tinha que ser eficiente em todos esses diferentes usos.

Não havia como a IBM (ou quem quer que fosse) escrever um software para atender a todos esses requisitos conflitantes. O resultado foi um sistema operacional enorme e extraordinariamente complexo, provavelmente duas ou três vezes maior do que o FMS. Ele consistia em milhões de linhas de linguagem *assembly*, escritas por milhares de programadores, e continha milhares e milhares de erros, que necessitavam um fluxo contínuo de novas versões na tentativa de corrigi-los. Cada nova versão corrigia alguns erros e introduzia outros, de modo que o número de erros provavelmente permanecia constante com o tempo.

Posteriormente, um dos projetistas do OS/360, Fred Brooks, escreveu um livro espirituoso e incisivo descrevendo suas experiências com o OS/360 (Brooks, 1995). Embora seja impossível resumir o livro aqui, basta dizer que a capa mostra uma manada de animais pré-históricos atolados em uma vala de alcatrão. A capa do livro de Silberschatz *et al*. (2004) faz uma comparação semelhante, sobre o fato de os sistemas operacionais serem dinossauros.

Apesar de seu tamanho enorme e de seus problemas, o OS/360 e os sistemas operacionais de terceira geração semelhantes, produzidos por outros fabricantes de computadores, satisfizeram a maioria de seus clientes razoavelmente bem. Eles também popularizaram várias técnicas importantes, ausentes nos sistemas operacionais de segunda geração. Provavelmente a mais importante delas foi a **multiprogramação**. No 7094, quando o *job* corrente fazia uma pausa para esperar a conclusão de uma operação de fita, ou de outro dispositivo de E/S, a CPU simplesmente ficava ociosa até que a operação terminasse. No caso de cálculos científicos, que exigem muito da CPU, as operações de E/S não são freqüentes, de modo que esse tempo desperdiçado não é significativo. No caso do processamento de dados comerciais, o tempo de espera pelas operações de E/S freqüentemente chegava a 80 ou 90 porcento do tempo total; portanto, algo tinha que ser feito para evitar que a CPU (cara) ficasse tão ociosa.

A solução desenvolvida foi dividir a memória em várias partições, com um *job* diferente em cada partição, como mostra a Figura 1-4. Enquanto um *job* estava esperando a conclusão da operação de E/S, outro podia usar a CPU. Se *jobs* suficientes pudessem ser mantidos si-

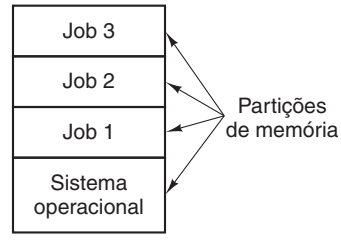

Figura 1-4 Um sistema de multiprogramação com três *jobs* na memória.

multaneamente na memória principal, a CPU poderia ficar ocupada praticamente 100% do tempo. Manter vários *jobs* simultâneos, com segurança, em memória, exige hardware especial para proteger cada um deles, evitando que um interfira e cause danos no outro, mas o 360 e outros sistemas de terceira geração estavam equipados com esse hardware.

Outro recurso importante apresentado nos sistemas operacionais de terceira geração foi a capacidade de ler *jobs* de cartões para o disco assim que eram trazidos para a sala do computador. Então, quando um *job* em execução acabava, o sistema operacional podia carregar um novo *job* do disco na partição, agora vazia, e executá-lo. Essa técnica é chamada de ***spooling*** (de *Simultaneous Peripheral Operation On Line* – Operação Periférica Simultânea *On-line*) e também era usada para saída. Com o *spooling*, os 1401 não eram mais necessários e acabava grande parte do trabalho de carga das fitas.

Embora os sistemas operacionais de terceira geração fossem convenientes para cálculos científicos pesados e execuções de processamento de dados comerciais de grande volume, basicamente eles ainda eram sistemas de lote. Muitos programadores sentiam falta dos tempos da primeira geração, quando eles tinham a máquina toda para si por algumas horas e, assim, podiam depurar seus programas rapidamente. Com os sistemas de terceira geração, o tempo entre submeter um *job* e receber a saída era freqüentemente de várias horas, de modo que uma vírgula colocada em lugar errado podia fazer uma compilação falhar e o programador perder metade do dia.

Essa necessidade de um tempo de resposta curto abriu caminho para o **compartilhamento do tempo** (*time sharing*), uma variante da multiprogramação na qual cada usuário tem um terminal *on-line*. Em um sistema de tempo compartilhado, se 20 usuários estivessem conectados e 17 deles estivessem pensando, conversando ou tomando café, a CPU podia ser alocada por turnos para os três *jobs* que quisessem o serviço. Como as pessoas que depuram programas normalmente utilizam comandos curtos (por exemplo, compilar uma *procedure*[†] de cinco páginas) em vez de longos (por exemplo, ordenar um arquivo de um milhão de registros), o computador pode oferecer um serviço rápido e interativo para vários usuários e também trabalhar em segundo plano (*background*) com *jobs* grandes em lotes, quando a CPU estiver ociosa. O primeiro sistema sério de compartilhamento de tempo, o **CTSS** (*Compatible Time Sharing System*), foi desenvolvido no M.I.T. em um 7094 especialmente modificado (Corbató *et al*., 1962). Entretanto, o compartilhamento de tempo não se tornou popular até que o hardware de proteção necessário se tornou difundido, durante a terceira geração.

Após o sucesso do sistema CTSS, o MIT, o Bell Labs e a General Electric (na época, um importante fabricante de computadores) decidiram dedicar-se ao desenvolvimento de um "*computer utility*"*, uma máquina que suportaria centenas de usuários de tempo compartilhado simultaneamente. Seu modelo era o sistema de distribuição de eletricidade – quando precisa de energia elétrica, você simplesmente insere um plugue na tomada da parede e, dentro do possível, toda a energia que precisar estará lá. Os projetistas desse sistema, conhecido como **MULTICS** (*MULTiplexed Information and Computing Service* – Serviço de Computação e Informação Multiplexado), imaginaram uma única máquina enorme fornecendo poder de computação para todos na região de Boston. A idéia de que máquinas muito mais poderosas do que seu computador de grande porte GE-645 seriam vendidas aos milhões por menos de mil dólares, apenas 30 anos depois, era pura ficção científica, como seria nos dias de hoje a idéia de trens supersônicos e transatlânticos submarinos.

[†] Usaremos os termos *procedure*, procedimento, sub-rotina e função indistintamente neste livro.

* N. de R. T.: *Utility* neste caso tem o sentido de um serviço público, indicando um recurso computacional amplamente disponível.

O MULTICS foi um sucesso misto. Ele foi projetado para suportar centenas de usuários em uma máquina apenas ligeiramente mais poderosa do que um PC baseado no Intel 80386, embora tivesse muito mais capacidade de E/S. Isso não é tão louco quanto parece, pois naquela época as pessoas sabiam escrever programas pequenos e eficientes, uma habilidade que subseqüentemente foi perdida. Houve muitas razões para o MULTICS não tomar conta do mundo, não sendo a menor delas, o fato de ter sido escrito em PL/I e que o compilador de PL/I estava anos atrasado e mal funcionava quando finalmente chegou ao mercado. Além disso, o MULTICS era enormemente ambicioso para sua época, semelhantemente à máquina analítica de Charles Babbage no século XIX.

O MULTICS introduziu muitas idéias embrionárias na literatura sobre computadores, mas transformá-lo em um produto sério e em um sucesso comercial foi muito mais difícil do que o esperado. O Bell Labs retirou-se do projeto e a General Electric desistiu completamente do negócio de computadores. Entretanto, o M.I.T. persistiu e finalmente fez o MULTICS funcionar. Por fim, ele foi vendido como um produto comercial pela empresa que adquiriu o ramo de computadores da GE (a Honeywell) e instalado por cerca de 80 empresas e universidades importantes do mundo todo. Embora em número pequeno, os usuários do MULTICS eram extremamente leais. A General Motors, a Ford e a Agência de Segurança Nacional dos EUA, por exemplo, só desligaram seus sistemas MULTICS no final dos anos 90. O último MULTICS que estava em funcionamento, no Departamento de Defesa Nacional canadense, foi desligado em outubro de 2000. Apesar de sua falta de sucesso comercial, o MULTICS teve uma enorme influência sobre os sistemas operacionais subseqüentes. Existem muitas informações sobre ele (Corbató *et al*., 1972; Corbató e Vyssotsky, 1965; Daley e Dennis, 1968; Organick, 1972; e Saltzer, 1974). Ele também tem um site web ainda ativo, *www.multicians.org*, com muitas informações sobre o sistema, seus projetistas e seus usuários.

Não se ouve falar mais da expressão *computer utility*, mas a idéia ganhou vida nova recentemente. Em sua forma mais simples, os PCs ou **estações de trabalho** (PCs de topo de linha) em uma empresa, ou em uma sala de aula, podem estar conectados, por meio de uma **rede local** (**LAN**), a um **servidor de arquivos** no qual todos os programas e dados estão armazenados. Então, um administrador precisa instalar e proteger apenas um conjunto de programas e dados, e pode reinstalar facilmente software local em um PC ou estação de trabalho que não esteja funcionando bem, sem se preocupar com a recuperação ou com a preservação de dados locais. Em ambientes mais heterogêneos, foi desenvolvida uma classe de software chamada ***middleware*** para fazer a ligação entre usuários locais e arquivos e entre programas e bancos de dados armazenados em servidores remotos. O *middleware* faz os computadores interligados em rede parecerem locais para os PCs ou para as estações de trabalho dos usuários individuais e apresenta uma interface uniforme com o usuário, mesmo que haja uma ampla variedade de servidores, PCs e estações de trabalho diferentes em uso. A *World Wide Web* é um exemplo. Um navegador web apresenta documentos para um usuário de maneira uniforme e um documento visualizado no navegador de um usuário pode ser composto por texto de um servidor e elementos gráficos de um outro, apresentados em um formato determinado por uma folha de estilos (*style sheets*) de um terceiro servidor. Normalmente, as empresas e universidades utilizam uma interface web para acessar bancos de dados e executar programas em um computador em outro prédio ou mesmo em outra cidade. O *middleware* parece ser o sistema operacional de um **sistema distribuído**, mas na realidade não é um sistema operacional e esse assunto está fora dos objetivos deste livro. Para ver mais informações sobre sistemas distribuídos, consulte Tanenbaum e Van Steen (2002).

Outro desenvolvimento importante durante a terceira geração foi o fenomenal crescimento dos minicomputadores, começando com o PDP-1 da DEC (*Digital Equipment Company*), em 1961. O PDP-1 tinha apenas 4K de palavras de 18 bits, mas a US$120.000 por

máquina (menos de 5% do preço de um 7094), foi um grande sucesso de vendas. Para certos tipos de trabalho não numéricos, ele era quase tão rápido quanto o 7094 e deu origem a toda uma nova indústria. Rapidamente, foi seguido por uma série de outros PDPs (ao contrário da família da IBM, todos incompatíveis), culminando no PDP-11.

Um dos cientistas da computação do Bell Labs, que tinha trabalhado no projeto do MULTICS, Ken Thompson, encontrou um pequeno minicomputador PDP-7 que ninguém usava e começou a escrever uma versão monousuário simplificada do MULTICS. Posteriormente, esse trabalho transformou-se no sistema operacional **UNIX**, que se tornou popular no mundo acadêmico, entre órgãos do governo e em muitas empresas.

A história do UNIX foi contada em outros textos (por exemplo, Salus, 1994). Como o código-fonte estava amplamente disponível, várias organizações desenvolveram suas próprias versões (incompatíveis), que levaram ao caos. Duas versões importantes foram desenvolvidas, o **System V**, da AT&T, e a **BSD** (*Berkeley Software Distribution*), da Universidade da Califórnia, em Berkeley. Elas também tiveram pequenas variantes, agora incluindo FreeBSD, OpenBSD e NetBSD. Para tornar possível escrever programas que pudessem ser executados em qualquer sistema UNIX, o IEEE desenvolveu um padrão para o UNIX, chamado **POSIX**, que a maioria das versões de UNIX agora o suportam. O padrão POSIX define uma interface mínima de chamadas de sistema que os sistemas UNIX compatíveis devem suportar. Na verdade, agora, outros sistemas operacionais também oferecem suporte a interface POSIX. As informações necessárias para se escrever software compatível com o padrão POSIX estão disponíveis em livros (IEEE, 1990; Lewine, 1991) e *on-line*, como a "*Single UNIX Specification*" do Open Group, que se encontra no endereço *www.unix.org*. Posteriormente, neste capítulo, quando nos referirmos ao UNIX, incluímos também todos esses sistemas, a não ser que mencionemos de outra forma. Embora sejam diferentes internamente, todos eles suportam o padrão POSIX; portanto, para o programador, eles são bastante semelhantes.

1.2.4 A quarta geração (1980–hoje): computadores pessoais

Com o desenvolvimento dos circuitos LSI (*Large Scale Integration* – integração em larga escala), *chips* contendo milhares de transistores em um centímetro quadrado de silício, surgiu a era do computador pessoal baseado em **microprocessador**. Em termos de arquitetura, os computadores pessoais (inicialmente chamados de **microcomputadores**) não eram muito diferentes dos minicomputadores da classe PDP-11, mas em termos de preço, eles certamente eram diferentes. O minicomputador também tornou possível que um departamento de uma empresa ou universidade tivesse seu próprio computador. O microcomputador tornou possível que uma pessoa tivesse seu próprio computador.

Havia várias famílias de microcomputadores. Em 1974, a Intel apareceu com o 8080, o primeiro microprocessador de 8 bits de propósito geral. Diversas empresas produziram sistemas completos usando o 8080 (ou o microprocessador compatível da Zilog, o Z80) e o sistema operacional **CP/M** (*Control Program for Microcomputers*), de uma empresa chamada Digital Research, foi amplamente usado neles. Muitos programas aplicativos foram escritos para executar no CP/M e ele dominou o mundo da computação pessoal por cerca de 5 anos.

A Motorola também produziu um microprocessador de 8 bits, o 6800. Um grupo de engenheiros deixou a Motorola para formar a MOS Technology e fabricar a CPU 6502, após a Motorola ter rejeitado as melhorias sugeridas por eles para o 6800. O 6502 foi a CPU de vários sistemas antigos. Um deles, o Apple II, se tornou um importante concorrente dos sistemas CP/M nos mercados doméstico e educacional. Mas o CP/M era tão popular que muitos proprietários de computadores Apple II adquiriram placas com o coprocessador Z-80 para executar CP/M, pois a CPU 6502 não era compatível com este sistema operacional. As pla-

cas CP/M eram comercializadas por uma pequena empresa chamada Microsoft, que também tinha um nicho de mercado, fornecendo interpretadores BASIC, usado por vários microcomputadores que executavam o CP/M.

A geração seguinte de microprocessadores foram os sistemas de 16 bits. A Intel apareceu com o 8086 e, no início dos anos 80, a IBM projetou o IBM PC utilizando o 8088 da Intel (internamente, um 8086, com um caminho de dados externo de 8 bits). A Microsoft ofereceu à IBM um pacote que incluía o BASIC e um sistema operacional, o **DOS** (*Disk Operating System*), originalmente desenvolvido por outra empresa – a Microsoft comprou o produto e contratou o autor original para aprimorá-lo. O sistema revisado foi chamado de **MS-DOS** (*MicroSoft Disk Operating System*) e rapidamente dominou o mercado do IBM PC.

O CP/M, o MS-DOS e o Apple DOS eram todos sistemas de linha de comando: os usuários digitavam comandos no teclado. Anos antes, Doug Engelbart, do Stanford Research Institute, tinha inventado a **GUI** (***Graphical User Interface*** – interface gráfica com o usuário), contendo janelas, ícones, menus e mouse. Steve Jobs, da Apple, viu a possibilidade de um computador pessoal realmente **amigável** (para usuários que não sabiam nada sobre computadores e não queriam aprender) e o Macintosh da Apple foi anunciado no início de 1984. Ele usava a CPU 68000 de 16 bits da Motorola e tinha 64 KB de memória **ROM** (*Read Only Memory* – memória somente de leitura), para suportar a GUI. Com o passar dos anos, o Macintosh evoluiu. As CPUs subseqüentes da Motorola eram verdadeiros sistemas de 32 bits e posteriormente a Apple mudou para as CPUs PowerPC da IBM, com arquitetura RISC de 32 bits (e, posteriormente, 64 bits). Em 2001, a Apple fez uma mudança importante no sistema operacional, lançando o **Mac OS X**, com uma nova versão da GUI Macintosh sobre o UNIX de Berkeley. E, em 2005, a Apple anunciou que estaria mudando para processadores Intel.

Para concorrer com o Macintosh, a Microsoft inventou o Windows. Originalmente, o Windows era apenas um ambiente gráfico sobre o MS-DOS de 16 bits (isto é, era mais um *shell* do que um verdadeiro sistema operacional). Entretanto, as versões atuais do Windows são descendentes do Windows NT, um sistema de 32 bits completo, reescrito desde o início.

O outro concorrente importante no mundo dos computadores pessoais é o UNIX (e seus vários derivados). O UNIX é mais poderoso em estações de trabalho e em outros computadores topo de linha, como os servidores de rede. Ele é especialmente difundido em máquinas equipadas com *chips* RISC de alto desempenho. Em computadores baseados em Pentium, o Linux está se tornando uma alternativa popular ao Windows para os estudantes e, cada vez mais, para muitos usuários corporativos. (Neste livro, usaremos o termo "Pentium" para nos referirmos à família Pentium inteira, incluindo os microprocessadores Celeron, de baixo poder computacional e os Xeon, de mais alto poder computacional e seus compatíveis AMD).

Embora muitos usuários de UNIX, especialmente programadores experientes, prefiram uma interface baseada em comandos a uma GUI, praticamente todos os sistemas UNIX suportam um sistema de janelas chamado **X Window**, desenvolvido no M.I.T. Esse sistema trata do gerenciamento básico de janelas, permitindo aos usuários criar, excluir, mover e redimensionar janelas usando um mouse. Freqüentemente, uma GUI completa, como a **Motif**, é disponibilizada para funcionar sobre o sistema X Window, proporcionando ao UNIX a aparência e o comportamento do Macintosh ou do Microsoft Windows para os usuários UNIX que assim o desejarem.

Um desenvolvimento interessante, que começou durante meados dos anos 80, é o crescimento das redes de computadores pessoais executando **sistemas operacionais de rede** e **sistemas operacionais distribuídos** (Tanenbaum e Van Steen, 2002). Em um sistema operacional de rede, os usuários sabem da existência de vários computadores e podem se conectar a máquinas remotas e copiar arquivos de uma para outra. Cada máquina executa seu próprio

sistema operacional local e tem seu próprio usuário (ou usuários) local. Basicamente, as máquinas são independentes entre si.

Os sistemas operacionais de rede não são fundamentalmente diferentes dos sistemas operacionais locais a uma máquina. Obviamente, eles precisam de uma controladora de interface de rede e de algum software de baixo nível para fazê-los funcionar, assim como de programas para obter *login* remoto e acesso remoto aos arquivos, mas essas adições não mudam a estrutura básica do sistema operacional.

Em contraste, um sistema operacional distribuído é aquele que aparece para seus usuários como um sistema de um processador tradicional, mesmo sendo composto, na verdade, por vários processadores. Os usuários não devem saber onde seus programas estão sendo executados nem onde seus arquivos estão localizados; tudo isso deve ser manipulado automática e eficientemente pelo sistema operacional.

Os verdadeiros sistemas operacionais distribuídos exigem mais do que apenas adicionar um código em um sistema operacional centralizados, pois os sistemas distribuídos e os centralizados diferem de maneiras importantes. Os sistemas distribuídos, por exemplo, freqüentemente permitem que os aplicativos sejam executados em vários processadores ao mesmo tempo, exigindo assim algoritmos de escalonamento mais complexos para otimizar o volume de paralelismo.

Muitas vezes, os atrasos de comunicação em rede significam que esses algoritmos (e outros) devam ser executados com informações incompletas, desatualizadas ou mesmo incorretas. Essa situação é radicalmente diferente de um sistema operacional centralizado, no qual se tem informações completas sobre o estado do sistema.

1.2.5 A história do MINIX 3

No início, o código-fonte do UNIX (versão 6) estava amplamente disponível, sob licença da AT&T, e era muito estudado. John Lions, da Universidade de New South Wales, na Austrália, escreveu um livro descrevendo seu funcionamento, linha por linha (Lions, 1996), e ele foi usado (com permissão da AT&T) como livro texto em muitos cursos universitários sobre sistemas operacionais.

Quando a AT&T lançou a versão 7, começou a perceber que o UNIX era um produto comercial valioso e, assim, distribuiu essa versão com uma licença proibindo o estudo do código-fonte em cursos para evitar o risco de expor seu status de segredo comercial. Muitas universidades obedeceram simplesmente eliminando o estudo do UNIX e ensinando apenas a teoria.

Infelizmente, ensinar apenas a teoria deixa o aluno com uma visão prejudicada do que um sistema operacional realmente é. Os assuntos teóricos normalmente abordados em detalhes em cursos e livros sobre sistemas operacionais, como os algoritmos de escalonamento, não têm tanta importância na prática. Os assuntos realmente importantes, como E/S e sistemas de arquivos, geralmente são abandonados, pois há pouca teoria a respeito.

Para corrigir essa situação, um dos autores deste livro (Tanenbaum) decidiu escrever um novo sistema operacional a partir de zero, que seria compatível com o UNIX do ponto de vista do usuário, mas completamente diferente por dentro. Por não usar sequer uma linha do código da AT&T, esse sistema evita as restrições de licenciamento; assim, ele pode ser usado para estudo individual ou em classe. Desse modo, os leitores podem dissecar um sistema operacional real para ver o que há por dentro, exatamente como os alunos de biologia dissecam rãs. Ele foi chamado de **MINIX** e foi lançado em 1987, com seu código-fonte completo para qualquer um estudar ou modificar. O nome MINIX significa mini-UNIX, pois ele é pequeno o bastante até para quem não é especialista poder entender seu funcionamento.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.