Capítulo 1

Conceitos básicos

Um sistema de computação é constituído basicamente por hardware e software. O hardware é composto por circuitos eletrônicos (processador, memória, portas de entrada/saída, etc.) e periféricos eletro-óptico-mecânicos (teclados, mouses, discos rígidos, unidades de disquete, CD ou DVD, dispositivos USB, etc.). Por sua vez, o software de aplicação é representado por programas destinados ao usuário do sistema, que constituem a razão final de seu uso, como editores de texto, navegadores Internet ou jogos. Entre os aplicativos e o hardware reside uma camada de software multifacetada e complexa, denominada genericamente de *Sistema Operacional* (SO). Neste capítulo veremos quais os objetivos básicos do sistema operacional, quais desafios ele deve resolver e como ele é estruturado para alcançar seus objetivos.

1.1 Objetivos de um SO

Existe uma grande distância entre os circuitos eletrônicos e dispositivos de hardware e os programas aplicativos em software. Os circuitos são complexos, acessados através de interfaces de baixo nível (geralmente usando as portas de entrada/saída do processador) e muitas vezes suas características e seu comportamento dependem da tecnologia usada em sua construção. Por exemplo, a forma de acesso de baixo nível a discos rígidos IDE difere da forma de acesso a discos SCSI ou leitores de CD. Essa grande diversidade pode ser uma fonte de dor de cabeça para o desenvolvedor de aplicativos. Portanto, é desejável oferecer aos programas aplicativos uma forma de acesso homogênea aos dispositivos físicos, que permita abstrair sua complexidade e as diferenças tecnológicas entre eles.

O sistema operacional é uma camada de software que opera entre o hardware e os programas aplicativos voltados ao usuário final. Trata-se de uma estrutura de software ampla, muitas vezes complexa, que incorpora aspectos de baixo nível (como *drivers* de dispositivos e gerência de memória física) e de alto nível (como programas utilitários e a própria interface gráfica).

A Figura 1.1 ilustra a estrutura geral de um sistema de computação típico. Nela, podemos observar elementos de hardware, o sistema operacional e alguns programas aplicativos.

Os objetivos básicos de um sistema operacional podem ser sintetizados em duas palavras-chave: “abstração” e “gerência”, cujos principais aspectos são detalhados a seguir.

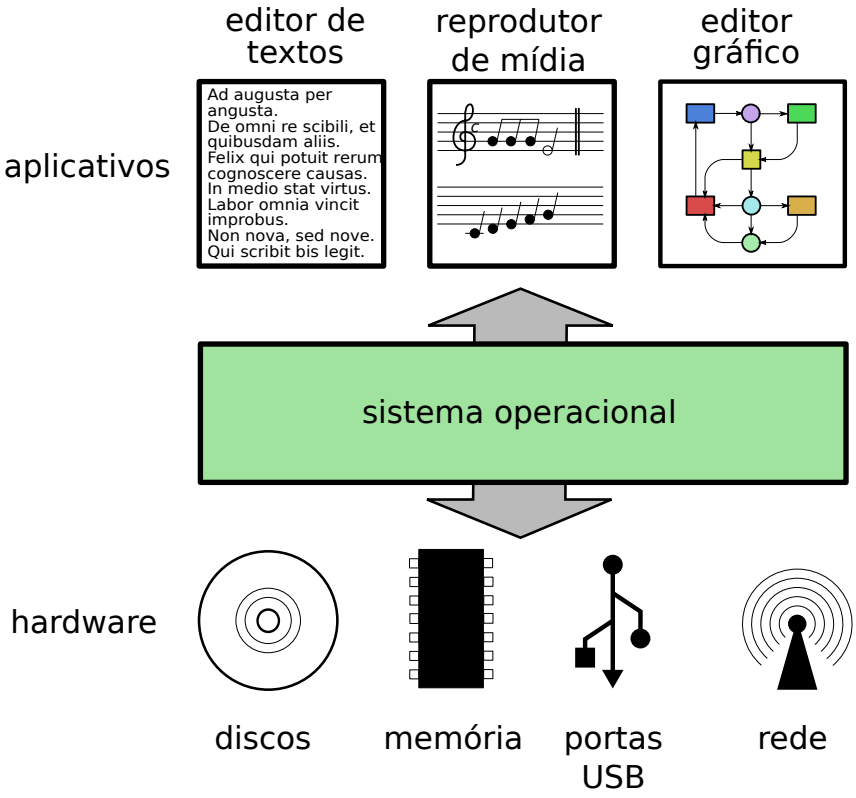

Figura 1.1: Estrutura de um sistema de computação típico

1.1.1 Abstração de recursos

Acessar os recursos de hardware de um sistema de computação pode ser uma tarefa complexa, devido às características específicas de cada dispositivo físico e a complexidade de suas interfaces. Por exemplo, a sequência a seguir apresenta os principais passos envolvidos na abertura de um arquivo (operação `open`) em um disco:

1. verificar se os parâmetros informados estão corretos (nome do arquivo, identificador do disco, buffer de leitura, etc.);
2. verificar se o disco está disponível;
3. ligar o motor do disco e aguardar atingir a velocidade de rotação correta;
4. posicionar a cabeça de leitura sobre a trilha onde está a tabela de diretório;
5. ler a tabela de diretório e localizar o arquivo ou subdiretório desejado;
6. mover a cabeça de leitura para a posição do bloco inicial do arquivo;
7. ler o bloco inicial do arquivo e depositá-lo em um buffer de memória.

Assim, o sistema operacional deve definir interfaces abstratas para os recursos do hardware, visando atender os seguintes objetivos:

- *Prover interfaces de acesso aos dispositivos, mais simples de usar que as interfaces de baixo nível*, para simplificar a construção de programas aplicativos. Por exemplo: para ler dados de um disco rígido, um programador de aplicação usa o conceito

de *arquivo*, que implementa uma visão abstrata do disco rígido, acessível através de operações como `open`, `read` e `close`. Caso tivesse de acessar o disco diretamente, seria necessário manipular portas de entrada/saída e registradores com comandos para o controlador de disco (sem falar na dificuldade de localizar os dados desejados dentro do disco).

- *Tornar os aplicativos independentes do hardware*. Ao definir uma interface abstrata de acesso a um dispositivo de hardware, o sistema operacional desacopla o hardware dos aplicativos e permite que ambos evoluam de forma mais autônoma. Por exemplo, o código de um editor de textos não deve ser dependente da tecnologia de discos utilizada no sistema.

- *Definir interfaces de acesso homogêneas para dispositivos com tecnologias distintas*. Através de suas abstrações, o sistema operacional permite aos aplicativos usar a mesma interface para dispositivos diversos. Por exemplo, um aplicativo acessa dados em disco através de arquivos e diretórios, sem precisar se preocupar com a estrutura real de armazenamento dos dados, que podem estar em um disquete, um disco SATA, uma máquina fotográfica digital conectada à porta USB, um CD ou mesmo um disco remoto, compartilhado através da rede.

1.1.2 Gerência de recursos

Os programas aplicativos usam o hardware para atingir seus objetivos: ler e armazenar dados, editar e imprimir documentos, navegar na Internet, tocar música, etc. Em um sistema com várias atividades simultâneas, podem surgir conflitos no uso do hardware, quando dois ou mais aplicativos precisam dos mesmos recursos para poder executar. Cabe ao sistema operacional definir *políticas* para gerenciar o uso dos recursos de hardware pelos aplicativos, e resolver eventuais disputas e conflitos. Vejamos algumas situações onde a gerência de recursos do hardware se faz necessária:

- Cada computador normalmente possui menos processadores que o número de tarefas em execução. Por isso, o uso desses processadores deve ser distribuído entre os aplicativos presentes no sistema, de forma que cada um deles possa executar na velocidade adequada para cumprir suas funções sem prejudicar os demais. O mesmo ocorre com a memória RAM, que deve ser distribuída de forma justa entre as aplicações.

- A impressora é um recurso cujo acesso deve ser efetuado de forma mutuamente exclusiva (apenas um aplicativo por vez), para não ocorrer mistura de conteúdo nos documentos impressos. O sistema operacional resolve essa questão definindo uma fila de trabalhos a imprimir (*print jobs*) normalmente atendidos de forma sequencial (FIFO).

- Ataques de negação de serviço (*DoS – Denial of Service*) são comuns na Internet. Eles consistem em usar diversas técnicas para forçar um servidor de rede a dedicar seus recursos para atender um determinado usuário, em detrimento dos demais. Por exemplo, ao abrir milhares de conexões simultâneas em um servidor de e-mail, um atacante pode reservar para si todos os recursos do servidor (processos, conexões de rede, memória e processador), fazendo com que os demais usuários não sejam mais atendidos. É responsabilidade do

sistema operacional do servidor detectar tais situações e impedir que todos os recursos do sistema sejam monopolizados por um só usuário (ou um pequeno grupo).

Assim, um sistema operacional visa abstrair o acesso e gerenciar os recursos de hardware, provendo aos aplicativos um ambiente de execução abstrato, no qual o acesso aos recursos se faz através de interfaces simples, independentes das características e detalhes de baixo nível, e no qual os conflitos no uso do hardware são minimizados.

1.2 Funcionalidades

Para cumprir seus objetivos de abstração e gerência, o sistema operacional deve atuar em várias frentes. Cada um dos recursos do sistema possui suas particularidades, o que impõe exigências específicas para gerenciar e abstrair os mesmos. Sob esta perspectiva, as principais funcionalidades implementadas por um sistema operacional típico são:

Gerência do processador: esta funcionalidade, também conhecida como gerência de processos, de tarefas ou de atividades, visa distribuir a capacidade de processamento de forma justa[1] entre as aplicações, evitando que uma aplicação monopolize esse recurso e respeitando as prioridades definidas pelos usuários.

O sistema operacional provê a ilusão de que existe um processador independente para cada tarefa, o que facilita o trabalho dos programadores de aplicações e permite a construção de sistemas mais interativos. Também faz parte da gerência de atividades fornecer abstrações para sincronizar atividades interdependentes e prover formas de comunicação entre elas.

Gerência de memória: tem como objetivo fornecer a cada aplicação uma área de memória própria, independente e isolada das demais aplicações e inclusive do sistema operacional. O isolamento das áreas de memória das aplicações melhora a estabilidade e segurança do sistema como um todo, pois impede aplicações com erros (ou aplicações maliciosas) de interferir no funcionamento das demais aplicações. Além disso, caso a memória RAM existente seja insuficiente para as aplicações, o sistema operacional pode aumentá-la de forma transparente às aplicações, usando o espaço disponível em um meio de armazenamento secundário (como um disco rígido).

Uma importante abstração construída pela gerência de memória, com o auxílio do hardware, é a noção de *memória virtual*, que desvincula os endereços de memória vistos por cada aplicação dos endereços acessados pelo processador na memória RAM. Com isso, uma aplicação pode ser carregada em qualquer posição livre da memória, sem que seu programador tenha de se preocupar com os endereços de memória onde ela irá executar.

Gerência de dispositivos: cada periférico do computador possui suas particularidades; assim, o procedimento de interação com uma placa de rede é completamente

[1] Distribuir de forma justa, mas não necessariamente igual, pois as aplicações têm demandas distintas de processamento. Por exemplo, um navegador de Internet demanda menos o processador que um aplicativo de edição de vídeo, e por isso o navegador pode receber menos tempo de processador.

diferente da interação com um disco rígido SATA. Todavia, existem muitos problemas e abordagens em comum para o acesso aos periféricos. Por exemplo, é possível criar uma abstração única para a maioria dos dispositivos de armazenamento como *pendrives*, discos SATA ou IDE, CDROMs, etc., na forma de um vetor de blocos de dados. A função da gerência de dispositivos (também conhecida como *gerência de entrada/saída*) é implementar a interação com cada dispositivo por meio de *drivers* e criar modelos abstratos que permitam agrupar vários dispositivos similares sob a mesma interface de acesso.

Gerência de arquivos: esta funcionalidade é construída sobre a gerência de dispositivos e visa criar arquivos e diretórios, definindo sua interface de acesso e as regras para seu uso. É importante observar que os conceitos abstratos de arquivo e diretório são tão importantes e difundidos que muitos sistemas operacionais os usam para permitir o acesso a recursos que nada tem a ver com armazenamento. Exemplos disso são as conexões de rede (nos sistemas UNIX e Windows, cada socket TCP é visto como um descritor de arquivo no qual pode-se ler ou escrever dados) e as informações internas do sistema operacional (como o diretório `/proc` do UNIX). No sistema experimental *Plan 9* [Pike et al., 1993], por exemplo, todos os recursos do sistema operacional são vistos como arquivos.

Gerência de proteção: com computadores conectados em rede e compartilhados por vários usuários, é importante definir claramente os recursos que cada usuário pode acessar, as formas de acesso permitidas (leitura, escrita, etc.) e garantir que essas definições sejam cumpridas. Para proteger os recursos do sistema contra acessos indevidos, é necessário: a) definir usuários e grupos de usuários; b) identificar os usuários que se conectam ao sistema, através de procedimentos de autenticação; c) definir e aplicar regras de controle de acesso aos recursos, relacionando todos os usuários, recursos e formas de acesso e aplicando essas regras através de procedimentos de autorização; e finalmente d) registrar o uso dos recursos pelos usuários, para fins de auditoria e contabilização.

Além dessas funcionalidades básicas oferecidas pela maioria dos sistemas operacionais, várias outras vêm se agregar aos sistemas modernos, para cobrir aspectos complementares, como a interface gráfica, suporte de rede, fluxos multimídia, fontes de energia, etc. As funcionalidades do sistema operacional geralmente são interdependentes: por exemplo, a gerência do processador depende de aspectos da gerência de memória, assim como a gerência de memória depende da gerência de dispositivos e da gerência de proteção.

Uma regra importante a ser observada na construção de um sistema operacional é a separação entre os conceitos de **política** e **mecanismo**[2] [Levin et al., 1975]. Como *política* consideram-se os aspectos de decisão mais abstratos, que podem ser resolvidos por algoritmos de nível mais alto, como por exemplo decidir a quantidade de memória que cada aplicação ativa deve receber, ou qual o próximo pacote de rede a enviar para satisfazer determinadas especificações de qualidade de serviço.

Por outro lado, como *mecanismo* consideram-se os procedimentos de baixo nível usados para implementar as políticas, ou seja, para atribuir ou retirar memória de uma aplicação, enviar ou receber um pacote de rede, etc. Os mecanismos devem

[2]Na verdade essa regra é tão importante que deveria ser levada em conta na construção de qualquer sistema computacional complexo.

ser suficientemente genéricos para suportar mudanças de política sem necessidade de modificações. Essa separação entre os conceitos de política e mecanismo traz uma grande flexibilidade aos sistemas operacionais, permitindo alterar sua personalidade (sistemas mais interativos ou mais eficientes) sem ter de alterar o código que interage diretamente com o hardware. Alguns sistemas, como o InfoKernel [Arpaci-Dusseau et al., 2003], permitem que as aplicações escolham as políticas do sistema mais adequadas às suas necessidades específicas.

1.3 Categorias

Os sistemas operacionais podem ser classificados segundo diversos parâmetros e aspectos, como tamanho de código, velocidade, suporte a recursos específicos, acesso à rede, etc. A seguir são apresentados alguns tipos de sistemas operacionais usuais (muitos sistemas operacionais se encaixam bem em mais de uma das categorias apresentadas):

Batch (de lote): os sistemas operacionais mais antigos trabalhavam "por lote", ou seja, todos os programas a executar eram colocados em uma fila, com seus dados e demais informações para a execução. O processador recebia os programas e os processava sem interagir com os usuários, o que permitia um alto grau de utilização do sistema. Atualmente, este conceito se aplica a sistemas que processam tarefas sem interação direta com os usuários, como os sistemas de processamento de transações bancárias. Além disso, o termo "em lote" também é usado para designar um conjunto de comandos que deve ser executado em sequência, sem interferência do usuário. Exemplos clássicos desses sistemas incluem o IBM OS/360 e o VAX/VMS, entre outros.

De rede: um sistema operacional de rede deve possuir suporte à operação em rede, ou seja, a capacidade de oferecer às aplicações locais recursos que estejam localizados em outros computadores da rede, como arquivos e impressoras. Ele também deve disponibilizar seus recursos locais aos demais computadores, de forma controlada. A maioria dos sistemas operacionais atuais oferece esse tipo de funcionalidade.

Distribuído: em um sistema operacional distribuído, os recursos de cada computador estão disponíveis a todos na rede, de forma transparente aos usuários. Ao lançar uma aplicação, o usuário interage com sua interface, mas não sabe onde ela está executando ou armazenando seus arquivos: o sistema é quem decide, de forma transparente ao usuário. Sistemas operacionais distribuídos já existem há muito tempo (por exemplo, o Amoeba [Tanenbaum et al., 1991]); recentemente, os ambientes de computação em nuvem têm implementado esse conceito. Em uma aplicação na nuvem, o usuário interage com a interface da aplicação em um computador ou telefone, mas não tem uma visão clara das máquinas onde seus dados estão sendo processados e armazenados.

Multiusuário: um sistema operacional multiusuário deve suportar a identificação do "dono" de cada recurso dentro do sistema (arquivos, processos, áreas de memória, conexões de rede) e impor regras de controle de acesso para impedir o uso desses recursos por usuários não autorizados. Essa funcionalidade é

fundamental para a segurança dos sistemas operacionais de rede e distribuídos. Grande parte dos sistemas atuais são multiusuários.

Servidor: um sistema operacional servidor deve permitir a gestão eficiente de grandes quantidades de recursos (disco, memória, processadores), impondo prioridades e limites sobre o uso dos recursos pelos usuários e seus aplicativos. Normalmente um sistema operacional servidor também tem suporte a rede e multiusuários.

Desktop: um sistema operacional "de mesa" é voltado ao atendimento do usuário doméstico e corporativo para a realização de atividades corriqueiras, como edição de textos e gráficos, navegação na Internet e reprodução de mídia. Suas principais características são a interface gráfica, o suporte à interatividade e a operação em rede. Exemplos de sistemas *desktop* são os vários sistemas Windows (XP, Vista, 7, 10, etc.), MacOS e Linux.

Móvel: um sistema operacional móvel é usado em equipamentos de uso pessoal compactos, como *smartphones* e *tablets*. Nesse contexto, as principais prioridades são a gestão eficiente da energia (bateria), a conectividade nos diversos tipos de rede (*wifi*, GSM, Bluetooth, NFC, etc) e a interação com uma grande variedade de sensores (GPS, giroscópio, luminosidade, tela de toque, leitor de digitais, etc). Android e iOS são bons exemplos desta categoria.

Embarcado: um sistema operacional é dito embarcado (embutido ou *embedded*) quando é construído para operar sobre um hardware com poucos recursos de processamento, armazenamento e energia. Aplicações típicas desse tipo de sistema aparecem em sistemas de automação e controladores automotivos, equipamentos eletrônicos de uso doméstico (leitores de DVD, TVs, fornos de microondas, centrais de alarme, etc.). Muitas vezes um sistema operacional embarcado se apresenta na forma de uma biblioteca a ser ligada ao programa da aplicação durante sua compilação. LynxOS, TinyOS, Contiki e VxWorks são exemplos de sistemas operacionais embarcados.

Tempo real: são sistemas nos quais o tempo é essencial. Ao contrário da ideia usual, um sistema operacional de tempo real não precisa ser necessariamente ultrarrápido; sua característica essencial é ter um comportamento temporal previsível, ou seja, seu tempo de resposta deve ser previsível no melhor e no pior caso de operação. A estrutura interna de um sistema operacional de tempo real deve ser construída de forma a minimizar esperas e latências imprevisíveis, como tempos de acesso a disco e sincronizações excessivas. Exemplos de sistemas operacionais de tempo real incluem o QNX, RT-Linux e VxWorks. Muitos sistemas embarcados têm características de tempo real, e vice-versa.

Existem sistemas de tempo real *críticos* (*hard real-time systems*), nos quais a perda de um prazo pelo sistema pode perturbar seriamente o sistema físico sob seu controle, com graves consequências humanas, econômicas ou ambientais. Exemplos desse tipo de sistema seriam o controle de funcionamento de uma turbina de avião ou de um freio ABS. Por outro lado, nos sistemas de tempo-real *não-críticos* (*soft real-time systems*), a perda de um prazo é perceptível e degrada o serviço prestado, sem maiores consequências. Exemplos desse tipo de sistema são os softwares de reprodução de mídia: em caso de atrasos, podem ocorrer falhas na música que está sendo tocada.

1.4 Um breve histórico dos SOs

Os primeiros sistemas de computação, no final dos anos 1940, não possuíam sistema operacional: as aplicações eram executadas diretamente sobre o hardware. Por outro lado, os sistemas de computação atuais possuem sistemas operacionais grandes, complexos e em constante evolução. A seguir são apresentados alguns dos marcos mais relevantes na história dos sistemas operacionais [Wikipedia, 2018]:

Anos 40: cada programa executava sozinho e tinha total controle do computador. A carga do programa em memória, a varredura dos periféricos de entrada para busca de dados, a computação propriamente dita e o envio dos resultados para os periférico de saída, byte a byte, tudo devia ser programado detalhadamente pelo desenvolvedor da aplicação.

Anos 50: os sistemas de computação fornecem "bibliotecas de sistema" (*system libraries*) que encapsulam o acesso aos periféricos, para facilitar a programação de aplicações. Algumas vezes um programa "monitor" (*system monitor*) auxilia a carga e descarga de aplicações e/ou dados entre a memória e periféricos (geralmente leitoras de cartão perfurado, fitas magnéticas e impressoras de caracteres).

1961: o grupo do pesquisador Fernando Corbató, do MIT, anuncia o desenvolvimento do CTSS – *Compatible Time-Sharing System [Corbató et al., 1962]*, o primeiro sistema operacional com compartilhamento de tempo.

1965: a IBM lança o OS/360, um sistema operacional avançado, com compartilhamento de tempo e excelente suporte a discos.

1965: um projeto conjunto entre MIT, GE e Bell Labs define o sistema operacional *Multics*, cujas ideias inovadoras irão influenciar novos sistemas durante décadas.

1969: Ken Thompson e Dennis Ritchie, pesquisadores dos Bell Labs, criam a primeira versão do UNIX.

1981: a Microsoft lança o MS-DOS, um sistema operacional comprado da empresa *Seattle Computer Products* em 1980.

1984: a Apple lança o sistema operacional Mac OS 1.0 para os computadores da linha Macintosh, o primeiro a ter uma interface gráfica totalmente incorporada ao sistema.

1985: primeira tentativa da Microsoft no campo dos sistemas operacionais com interface gráfica, através do MS-Windows 1.0.

1987: Andrew Tanenbaum, um professor de computação holandês, desenvolve um sistema operacional didático simplificado, mas respeitando a API do UNIX, que foi batizado como *Minix*.

1987: IBM e Microsoft apresentam a primeira versão do OS/2, um sistema multitarefa destinado a substituir o MS-DOS e o Windows. Mais tarde, as duas empresas rompem a parceria; a IBM continua no OS/2 e a Microsoft investe no ambiente Windows.

1991: Linus Torvalds, um estudante de graduação finlandês, inicia o desenvolvimento do Linux, lançando na rede Usenet o núcleo 0.01, logo abraçado por centenas de programadores ao redor do mundo.

1993: a Microsoft lança o Windows NT, o primeiro sistema 32 bits da empresa, que contava com uma arquitetura interna inovadora.

1993: lançamento dos UNIX de código aberto FreeBSD e NetBSD.

1993: a Apple lança o Newton OS, considerado o primeiro sistema operacional móvel, com gestão de energia e suporte para tela de toque.

1995: a AT&T lança o Plan 9, um sistema operacional distribuído.

1999: a empresa VMWare lança um ambiente de virtualização para sistemas operacionais de mercado.

2001: a Apple lança o MacOS X, um sistema operacional com arquitetura distinta de suas versões anteriores, derivada da família UNIX BSD.

2005: lançado o Minix 3, um sistema operacional micro-núcleo para aplicações embarcadas. O Minix 3 faz parte do *firmware* dos processadores Intel mais recentes.

2006: lançamento do Windows Vista.

2007: lançamento do *iPhone* e seu sistema operacional iOS, derivado do sistema operacional *Darwin*.

2007: lançamento do Android, um sistema operacional baseado no núcleo Linux para dispositivos móveis.

2010: Windows Phone, SO para celulares pela Microsoft.

2015: Microsoft lança o Windows 10.

Esse histórico reflete apenas o surgimento de alguns sistemas operacionais relativamente populares; diversos sistemas acadêmicos ou industriais de grande importância pelas contribuições inovadoras, como *Mach* [Rashid et al., 1989], *Chorus* [Rozier and Martins, 1987], *QNX* e outros, não estão representados.

Exercícios

1. Quais os dois principais objetivos de um sistema operacional?

2. Por que a abstração de recursos é importante para os desenvolvedores de aplicações? Ela tem alguma utilidade para os desenvolvedores do próprio sistema operacional?

3. A gerência de atividades permite compartilhar o processador, executando mais de uma aplicação ao mesmo tempo. Identifique as principais vantagens trazidas por essa funcionalidade e os desafios a resolver para implementá-la.

4. O que caracteriza um sistema operacional de tempo real? Quais as duas classificações de sistemas operacionais de tempo real e suas diferenças?

5. Relacione as afirmações aos respectivos tipos de sistemas operacionais: distribuído (D), multi-usuário (M), desktop (K), servidor (S), embarcado (E) ou de tempo-real (T):

   [ ] Deve ter um comportamento temporal previsível, com prazos de resposta claramente definidos.

   [ ] Sistema operacional usado por uma empresa para executar seu banco de dados corporativo.

   [ ] São tipicamente usados em telefones celulares e sistemas eletrônicos dedicados.

   [ ] Neste tipo de sistema, a localização física dos recursos do sistema computacional é transparente para os usuários.

   [ ] Todos os recursos do sistema têm proprietários e existem regras controlando o acesso aos mesmos pelos usuários.

   [ ] A gerência de energia é muito importante neste tipo de sistema.

   [ ] Sistema que prioriza a gerência da interface gráfica e a interação com o usuário.

   [ ] Construído para gerenciar de forma eficiente grandes volumes de recursos.

   [ ] O MacOS X é um exemplo típico deste tipo de sistema.

   [ ] São sistemas operacionais compactos, construídos para executar aplicações específicas sobre plataformas com poucos recursos.

6. Sobre as afirmações a seguir, relativas aos diversos tipos de sistemas operacionais, indique quais são **incorretas**, justificando sua resposta:

   (a) Em um sistema operacional de **tempo real**, a rapidez de resposta é menos importante que a previsibilidade do tempo de resposta.

   (b) Um sistema operacional **multi-usuários** associa um proprietário a cada recurso do sistema e gerencia as permissões de acesso a esses recursos.

   (c) Nos sistemas operacionais **de rede** a localização dos recursos é transparente para os usuários.

   (d) Um sistema operacional **de tempo real** deve priorizar as tarefas que interagem com o usuário.

   (e) Um sistema operacional **embarcado** é projetado para operar em hardware com poucos recursos.

Referências

A. Arpaci-Dusseau, R. Arpaci-Dusseau, N. Burnett, T. Denehy, T. Engle, H. Gunawi, J. Nugent, and F. Popovici. Transforming policies into mechanisms with InfoKernel. In *19th ACM Symposium on Operating Systems Principles*, October 2003.

F. Corbató, M. Daggett, and R. Daley. An experimental time-sharing system. In *Proceedings of the Spring Joint Computer Conference*, 1962.

R. Levin, E. Cohen, W. Corwin, F. Pollack, and W. Wulf. Policy/mechanism separation in Hydra. *SIGOPS Operating Systems Review*, 9(5):132–140, Nov. 1975.

R. Pike, D. Presotto, K. Thompson, H. Trickey, and P. Winterbottom. The use of name spaces in Plan 9. *Operating Systems Review*, 27(2):72–76, April 1993.

R. Rashid, D. Julin, D. Orr, R. Sanzi, R. Baron, A. Forin, D. Golub, and M. B. Jones. Mach: a system software kernel. In *Proceedings of the 1989 IEEE International Conference, COMPCON*, pages 176–178, San Francisco, CA, USA, 1989. IEEE Comput. Soc. Press.

M. Rozier and J. L. Martins. The Chorus distributed operating system: Some design issues. In Y. Paker, J.-P. Banatre, and M. Bozyiğit, editors, *Distributed Operating Systems*, pages 261–287, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

A. Tanenbaum, M. Kaashoek, R. van Renesse, and H. Bal. The Amoeba distributed operating system – a status report. *Computer Communications*, 14:324–335, July 1991.

Wikipedia. Wikipedia online enciclopedia. http://www.wikipedia.org, 2018.