UNIP

UNIVERSIDADE PAULISTA

Sistemas Operacionais

Autor: Prof. Michel Bernardo Fernandes da Silva
Colaboradoras: Profa. Vanessa Lessa
Profa. Larissa Rodrigues Damiani

Professor conteudista: Michel Bernardo Fernandes da Silva

Michel Bernardo Fernandes da Silva é formado em Engenharia Elétrica, opção Telecomunicações pela Escola Politécnica da Universidade de São Paulo (Poli-USP). Concluiu os cursos de pós-graduação em Finanças e MBA Executivo no Instituto de Ensino e Pesquisa (Insper) e finalizou o mestrado em Engenharia Elétrica na Escola Politécnica. Possui experiência professional de mais de 12 anos na área financeira e já atuou como professor universitário e de pós-graduação em diversas instituições como FMU, Uninove e Anhanguera. Atualmente é doutorando em Engenharia Elétrica na Poli-USP e desde 2013 atua como professor dos cursos superiores de tecnologia em diversos campi da Universidade Paulista (UNIP). É autor do livro *Cibersegurança*: uma visão panorâmica sobre a segurança de informação na internet (2023).

Dados Internacionais de Catalogação na Publicação (CIP)

S586s Silva, Michel Bernardo Fernandes da.

Sistemas Operacionais / Michel Bernardo Fernandes da Silva. – São Paulo: Editora Sol, 2023.

164 p., il.

Nota: este volume está publicado nos Cadernos de Estudos e Pesquisas da UNIP, Série Didática, ISSN 1517-9230.

1. Sistemas operacionais. 2. Gerenciamento. 3. Linux. I. Título.

CDU 681.3.066

U517.61 – 23

Sumário

Sistemas Operacionais

Unidade II

Unidade III

APRESENTAÇÃO

O objetivo geral da disciplina Sistemas Operacionais é compreender seu funcionamento, analisando tarefas como gerenciamento de processos, gerenciamentos de memória, gerenciamento de entrada e saída, sistema de arquivos e aspectos de segurança.

Uma finalidade específica dela é a apresentação das principais características que o sistema operacional deve ter, os conceitos sobre sistemas operacionais, a arquitetura desses sistemas, os algoritmos utilizados nas diferentes atividades de gerência e as formas de implementação para os sistemas.

Todos já tivemos alguma experiência com sistemas operacionais como Windows, Linux e Mac OS X em um computador pessoal ou com sistemas operacionais Android e iOS utilizados em smartphones, tablets e outros dispositivos móveis. Entretanto, na verdade, interagimos com uma interface gráfica do sistema operacional ou *graphical user interface* (GUI) e um interpretador de comandos denominado shell. Tanto a interface gráfica como o interpretador de comandos são programas que não fazem parte do núcleo do sistema operacional.

Um sistema operacional é um software complexo e com numerosos módulos a serem criados, e cada um deles precisa ter uma delimitação clara do sistema e possuir entradas, saídas e funções definidas criteriosamente. Somente o núcleo do sistema operacional Linux, cujo código é livre e aberto, tem dezenas de milhões de linhas de código-fonte!

Podemos considerar o sistema operacional como um software básico para o computador, pois sobre ele serão construídos outros softwares. Ele facilita a operação dos outros softwares aplicativos, pois contribui com todas as interações necessárias com os recursos de hardware.

INTRODUÇÃO

O computador é um dispositivo físico que necessita ser gerenciado para poder funcionar. Na ausência de software, o hardware do computador fica inutilizado, sendo apenas caixa plástica contendo uma coleção de circuitos eletrônicos.

De forma geral, é possível classificar os softwares em dois tipos: programas aplicativos e programas de sistemas. Os programas aplicativos são implementados para atender as diferentes necessidades dos usuários, como assistir vídeos, realizar pesquisas na internet e enviar e-mail – atividades que precisam de softwares específicos.

Por sua vez, os programas de sistemas são responsáveis por gerenciar a operação do computador. O principal programa de sistema é o sistema operacional, que controla os recursos computacionais disponíveis, como memória, processador, acesso a dispositivos de entrada/saída e estabelece acessos ao hardware que os programas aplicativos utilizam para suas funções. Os sistemas operacionais são o foco deste livro-texto.

O conhecimento profundo de sistemas operacionais é extremamente útil para hackers e profissionais de computação, porque os mecanismos implementados pelo sistema operacional influenciam diretamente o desempenho e o comportamento das aplicações do usuário. Adicionalmente, por meio dele é realizada uma série de configurações de serviços de rede e segurança do sistema.

Segundo Deitel, Deitel e Choffnes (2005), o sistema operacional reveza sua execução com a de outros programas, de forma que aparenta estar vigiando, controlando e orquestrando todo o processo computacional. Podemos considerá-lo como um software básico sobre o qual serão desenvolvidos outros softwares.

O conhecimento profundo de sistemas operacionais é importante para os profissionais de computação, já que os mecanismos desenvolvidos pelo sistema operacional influenciam de forma direta no comportamento e no desempenho das aplicações. Mesmo dispositivos computacionais de menor porte, tais como smartphones, smartwatches ou assistentes virtuais, possuem sistema operacional para realizar suas operações. Entretanto, em alguns aparelhos, ele é projetado para funcionar sem intervenção do usuário.

O presente livro-texto está dividido em três unidades. Na primeira delas, será abordada a visão básica de sistemas operacionais, a sua evolução histórica, as suas funções essenciais de gerência de recursos e abstração de recursos, as principais características e a organização desse tipo de software. Adicionalmente, serão apresentados os conceitos de processos, um programa em execução, situações aplicáveis aos processos, como os deadlocks, e os principais algoritmos de escalonamento aplicáveis à gerência de processos e arquiteturas com múltiplos núcleos de processamento.

Na segunda unidade, serão descritos os conceitos e as técnicas aplicáveis à gerência de memória, tais como memória virtual e alocação de memória. Também serão apresentados os requisitos de sistema de arquivos e a organização de diretórios.

Na terceira unidade, é detalhada a gerência de dispositivos de entrada e saída para o sistema operacional. Adicionalmente, serão apresentados o funcionamento do sistema operacional Linux, o histórico do Linux, as diversas distribuições de Linux e a configuração básica desse sistema. Por fim, serão exibidos os principais comandos e qual a organização das atividades elementares de gerência no sistema operacional.

Aproveite a leitura e bons estudos!

Unidade I

1 CONCEITO DE SISTEMAS OPERACIONAIS

Um sistema operacional ou *operating system*, em inglês, é um programa, ou seja, um software responsável por gerenciar o hardware de um computador. Tal programa também fornece uma base para os programas aplicativos e atua como intermediário entre o usuário e o hardware do computador. Um aspecto interessante dele é assumir diferentes abordagens ao cumprir essas tarefas.

Caso as aplicações fossem desenvolvidas sem a utilização de um sistema operacional, o tempo de programação para o desenvolvimento do código-fonte seria muito maior, pois o programador precisaria implementar as rotinas de entrada/saída necessárias, que são previstas pelo sistema operacional. Com isso, a complexidade do software desenvolvido será maior. Adicionalmente, o usuário deverá estar preocupado com os detalhes de hardware, tais como as configurações técnicas de cada dispositivo.

O sistema operacional é considerado um software básico para o sistema computacional e é essencial, uma vez que permite maior dedicação aos problemas de alto nível, isto é, não ligados ao hardware e voltados para as aplicações. Além disso, provê maior portabilidade entre dispositivos diferentes, possibilitando que uma mesma aplicação funcione para diversas configurações de hardware.

Machado e Maia (2013, p. 3) definem um sistema operacional como: "[...] um conjunto de rotinas executado pelo processador, de forma semelhante aos programas dos usuários". Sua principal função é o controle da operação do computador, pelo gerenciamento, utilização e compartilhamento de vários hardwares existentes, tais como memória, dispositivos de entrada/saída e processadores.

Uma caraterística que distingue um sistema operacional de outros programas aplicativos é a forma pela qual as rotinas são processadas ao longo do tempo. No sistema operacional, a forma de execução das rotinas é não linear, pois elas são executadas de forma concorrente e acionadas por eventos assíncronos, ou seja, podem ocorrer a qualquer instante.

Com sistemas operacionais, são possíveis a criação de uma máquina mais flexível e o uso eficiente e controlado dos componentes de hardware. Além disso, permitem a utilização compartilhada e protegida dos diversos componentes de hardware e software por múltiplos usuários.

Apresentaremos a seguir a visão geral de um sistema computacional e detalharemos suas funções básicas, com seu histórico e seus diferentes tipos e classificações.

1.1 Visão geral de um sistema computacional

De forma simplificada, um sistema computacional pode ser dividido em quatro componentes: hardware, sistema operacional, softwares aplicativos e os usuários, conforme apresentado na figura a seguir.

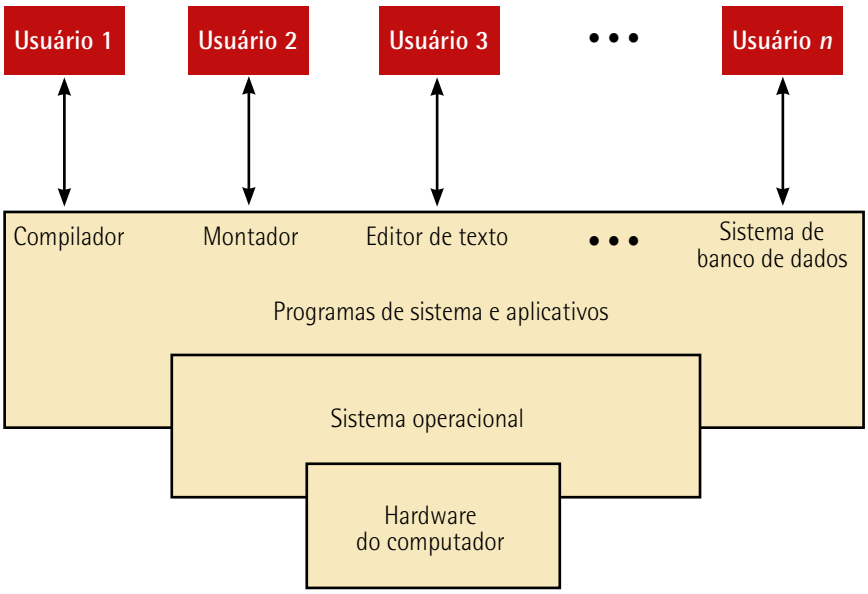


Figura 1 – Visão abstrata de um sistema computacional

Fonte: Silberschatz, Galvin e Gagne (2015, p. 3).

O hardware fornece os recursos básicos de computação do sistema, tais como a unidade central de processamento (CPU), os dispositivos de armazenamento e os dispositivos de entrada/saída (E/S) ou input/output (I/O). Os computadores atuais possuem mais de um núcleo de processamento ou core, o que permite realizar processamentos de forma paralela. Entretanto, essa situação requer a coordenação entre os diferentes núcleos.

Os programas aplicativos realizam as diversas tarefas desejadas pelos usuários e definem os meios pelos quais os recursos computacionais serão consumidos visando atender às demandas do usuário. Eles incluem todos os programas não associados à operação do sistema: navegadores, editores de imagem e correio eletrônico são exemplos de seus exemplos.

Os programas do sistema estão associados ao sistema operacional, mas não necessariamente fazem parte do kernel, por exemplo a calculadora e o gerenciador de arquivos. Por sua vez, o sistema operacional é responsável pelo controle do hardware e pela coordenação de utilização dos recursos para atender aos vários programas aplicativos de diversos usuários. Trata-se do software envolvido diretamente com o hardware. Nesse contexto, ele pode ser considerado como um alocador de recursos ou gerenciador de recursos. Um sistema de computação pode demandar muitos recursos para a resolução de um problema, tais como tempo de processamento da unidade central de processamento

(CPU), espaço de armazenamento em disco para os arquivos, espaço disponível na memória, dispositivos de entrada/saída (E/S), entre outros.

Adicionalmente, o sistema operacional é o primeiro programa a ser executado pela máquina quando ela for acionada através do processo denominado bootstrapping. A partir desse momento, o sistema operacional permanece em funcionamento até que o computador seja desligado.

1.2 Funções essenciais do sistema operacional

Nesta seção, serão descritas as duas funções essenciais e não relacionadas do sistema operacional: gerenciar os recursos do hardware e fornecer um conjunto de recursos de hardware abstratos para os programas aplicativos ou, de forma simplificada, gerência de recursos e abstração de recursos.

1.2.1 Abstração de recursos

Os recursos de hardware de um computador utilizam uma linguagem de máquina mais antiga, normalmente denominada de linguagem de baixo nível, que é extremamente complexa para programação, em especial dos dispositivos de entrada/saída.

Para o usuário comum, a operação de leitura de um arquivo em um disco é aparentemente trivial. Entretanto, existem numerosas rotinas específicas que são responsáveis por acionar o mecanismo de leitura e gravação do disco rígido, posicionar o leitor na trilha e setor corretos, transferir os dados para memória e, por fim, avisar o término da operação à aplicação. Todas essas etapas necessárias são realizadas e controladas pelo sistema operacional.

Seria muito difícil para o programador lidar com as especificidades de cada hardware e ser obrigado a programar cada função de leitura e gravação no disco rígido, por exemplo, além de se preocupar com os detalhes relacionados a operações de leitura de gravação.

Para essas tarefas, o sistema operacional utiliza o driver do dispositivo, que irá se comunicar com o hardware e possibilita a retirada dos detalhes complexos do funcionamento de cada hardware para a aplicação. Consequentemente, o programador de software de aplicativos não precisará lidar com as operações do hardware, pois o sistema operacional será responsável por proporcionar uma interface entre os usuários e os recursos de hardware disponíveis para aquele sistema computacional. Assim, a comunicação se torna transparente, permite um trabalho mais eficiente aos usuários e as chances de erros nas aplicações ficam reduzidas.

No caso dos discos de armazenamento, uma abstração típica é o disco possuir um conjunto de arquivos com nomes, de forma que cada arquivo possa ser aberto para leitura ou escrita através de uma interface do sistema operacional. O programador não precisa se preocupar com detalhes como interrupções, temporizadores, frequências utilizadas e outros recursos de baixo nível. De certa forma, o sistema operacional esconde o hardware e oferece uma abstração mais simples e de maior facilidade de acesso aos recursos do sistema.

1.2.2 Gerência de recursos

O sistema computacional é composto de recursos como processadores, discos rígidos, interfaces de rede, temporizadores, teclado, mouse, impressora, entre outros. Ele atua como um gerenciador de recursos pois estabelece uma alocação ordenada e controlada desses recursos entre as várias aplicações que concorrem pela utilização desses dispositivos.

Os sistemas operacionais modernos permitem a alocação na memória e a execução de múltiplos programas. Pense na seguinte situação: três programas executados em um mesmo computador pedem simultaneamente acesso a uma única impressora. Caso não existisse o sistema operacional, não haveria uma orquestração da ordem de utilização dessa impressora e o resultado seria caótico, pois a impressão de um programa poderia interferir na saída dos demais.

Para resolver esses conflitos de utilização concorrente entre os diferentes programas, o sistema operacional é o responsável por permitir o acesso concorrente a esse e aos outros recursos de maneira protegida e organizada.

Tais conflitos são mais frequentes ainda quando existem diversos usuários operando em uma rede de computadores e a necessidade de gerenciamento e proteção da memória e dos dispositivos de entrada/saída são mais evidentes, uma vez que a utilização de um indivíduo pode interferir em outros. Além do compartilhamento dos recursos de hardware, existem as informações da rede, tais como arquivos e bancos de dados, que podem ser compartilhados.

De acordo com Tanenbaum e Bos (2016), o gerenciamento de recursos inclui o compartilhamento ou a multiplexação desses mecanismo, e pode ocorrer de duas formas diferentes: no tempo e no espaço. Na multiplexação no tempo, diferentes programas e usuários revezam a utilização do recurso, e a tarefa de determinar por quanto tempo e quando cada aplicação pode utilizar o recurso é tarefa do sistema operacional. No caso da multiplexação no espaço, cada programa terá direito a uma parte dos recursos, mas não há revezamento entre os programas. Esse tipo de multiplexação é comum na memória principal, na qual vários programas ficam residentes, para serem executados, em partes diferentes da memória de trabalho.

Ao lidar com numerosas solicitações de recursos, de múltiplas aplicações diferentes e possivelmente conflitantes, o sistema operacional precisa decidir como alocar os recursos disponíveis a programas e usuários específicos, de forma a possibilitar a operação de sistema de computação de maneira eficiente e justa.

Observação

A distribuição justa de recursos é diferente da distribuição igualitária. Cada aplicação possui requisitos, necessidades de hardware e prioridades diferentes. A alocação de recursos é particularmente importante quando muitos usuários acessam o mesmo mainframe ou múltiplas tarefas devem ser executadas em um computador.

Os principais recursos a serem gerenciados pelo sistema operacional são o processador ou conjunto de processadores, a memória, os dispositivos de entrada e saída e o sistema de arquivos.

Segundo Deitel, Deitel e Choffnes (2010), o sistema operacional realiza o revezamento da sua execução com a de demais softwares, de modo a orquestrar todo o processo computacional.

1.3 Interação do usuário com sistema operacional

O usuário pode interagir com o sistema operacional utilizando uma linguagem de comunicação especial denominada linguagem de comando. A figura a seguir apresenta a interação entre os usuários e o sistema operacional.

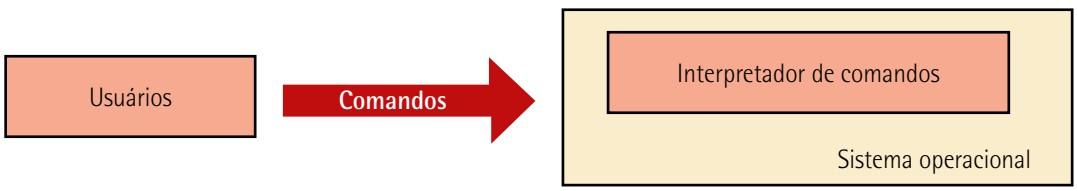


Figura 2 – Interação entre usuários e sistema operacional

Existem diversos mecanismos para essa interação. Para isso pode ser utilizada a linguagem JCL, do inglês *job control language*. Em muitos sistemas operacionais, a interação do usuário ocorre de forma textual. A figura a seguir apresenta um exemplo de interface textual no sistema Unix.

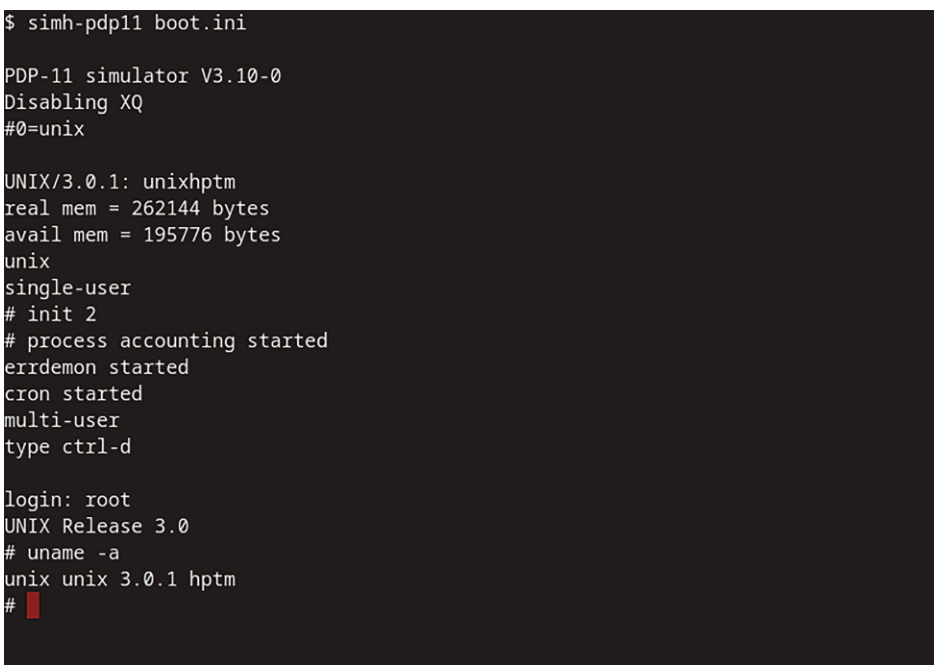


Figura 3 – Interface textual do sistema Unix

Disponível em: https://bit.ly/429WTd7. Acesso em: 14 mar. 2023.

Outros sistemas possuem uma interface gráfica ou *graphical user interface* (GUI), em inglês, que utiliza janelas e ícones e proporciona maior facilidade de interação do usuário, já que nem todos os usuários conseguiriam digitar comandos no prompt do sistema operacional. A figura a seguir apresenta a interface gráfica de um sistema operacional Linux Ubuntu.

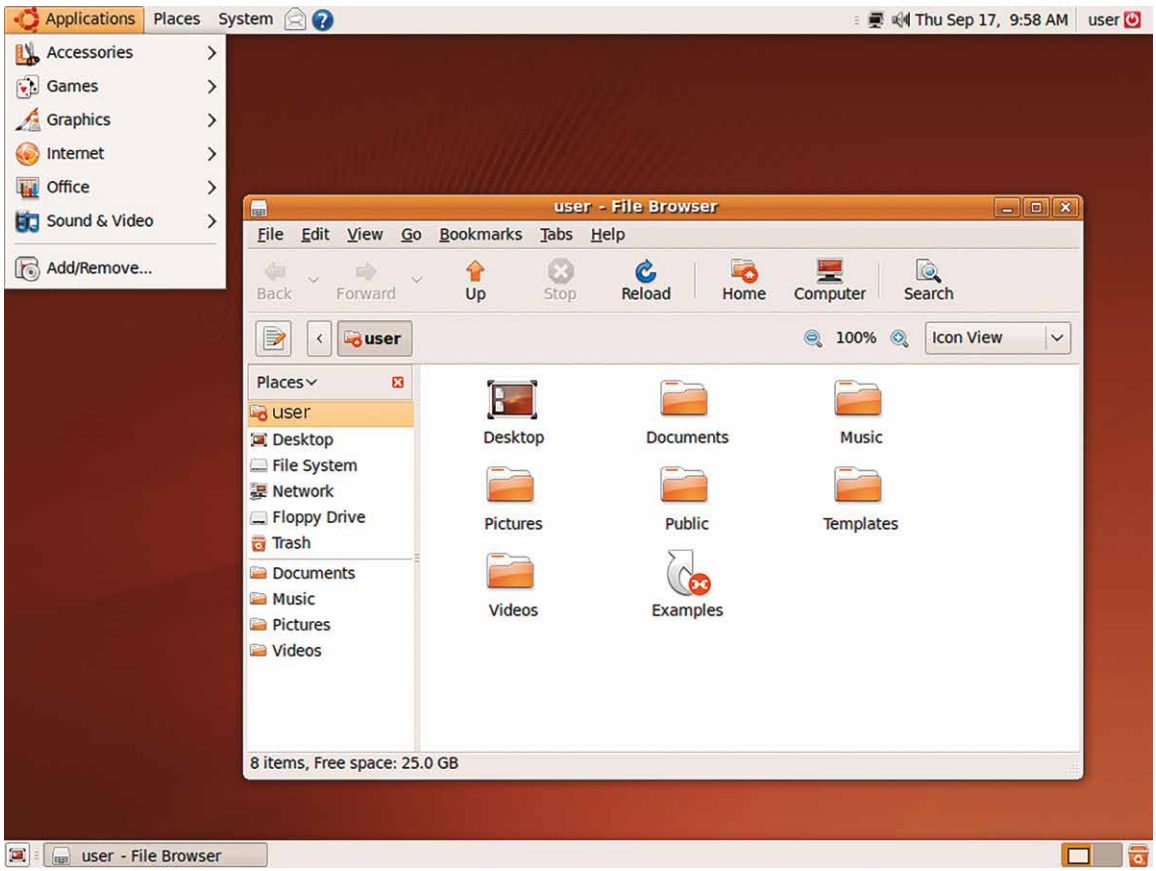


Figura 4 – Interface gráfica do sistema Linux Ubuntu

Disponível em: https://bit.ly/3yEINUK. Acesso em: 14 mar. 2023.

Além dos usuários, os outros softwares interagem com o sistema operacional. Os programas de usuário invocam serviços do sistema operacional por meio das chamadas de sistema ou system calls. As chamadas de sistema permitem o controle mais eficiente sobre as operações do sistema e realizam operações para cada sistema operacional, tais como abertura e gravação e fechamento de arquivo, solicitação e liberação de dispositivo de entrada/saída, obtenção de informação de data ou hora do sistema.

Quadro 1 – Exemplos de chamadas de sistema

Tipo de operação	Operação	Nome da chamada de sistema	
		Windows	Linux
Controle de processos	Criação de processos	CreateProcess( )	fork( )
	Término de processos	ExitProcess( )	exit( )
	Espera para processo	WaitForSingleObject( )	wait( )
Manipulação de arquivos	Criação de arquivo	CreateFile( )	open()
	Leitura de arquivo	ReadFile( )	read( )
	Gravação de arquivo	WriteFile( )	write( )
	Fechamento de arquivo	CloseHandle( )	close( )

1.4 Histórico do sistema operacional

Da mesma forma que o hardware, os sistemas operacionais evoluíram ao longo dos anos e, nesta seção, apresentaremos um breve histórico deles. Há uma vinculação do sistema operacional com a arquitetura dos computadores onde são executados, mudando conforme as gerações de computadores. Segundo Tanenbaum e Bos (2016), o mapeamento da evolução do sistema frente às gerações de computadores é impreciso, mas trata-se da única forma encontrada para mostrá-la.

Saiba mais

O site do Computer Museum History apresenta a linha do tempo histórica da evolução do computador. Acesse-a no link a seguir:

COMPUTER MUSEUM HISTORY. *Timeline of computer history*. Califórnia: CHM, [s.d.]. Disponível em: https://bit.ly/2L452LO. Acesso em: 14 mar. 2023.

Até o período da Segunda Guerra Mundial, não houve sucesso na construção de computadores digitais. O projeto de maior destaque foi a "máquina analítica" do matemático inglês Charles Babbage (1792-1871).

Por falta de tecnologia na época, Babbage não pôde construir a "máquina analítica", pois era necessário produzir engrenagens de alta precisão mecânica. Esse dispositivo não possuía um sistema operacional e a foto a seguir apresenta a construção da máquina de Babbage depois de seu falecimento.

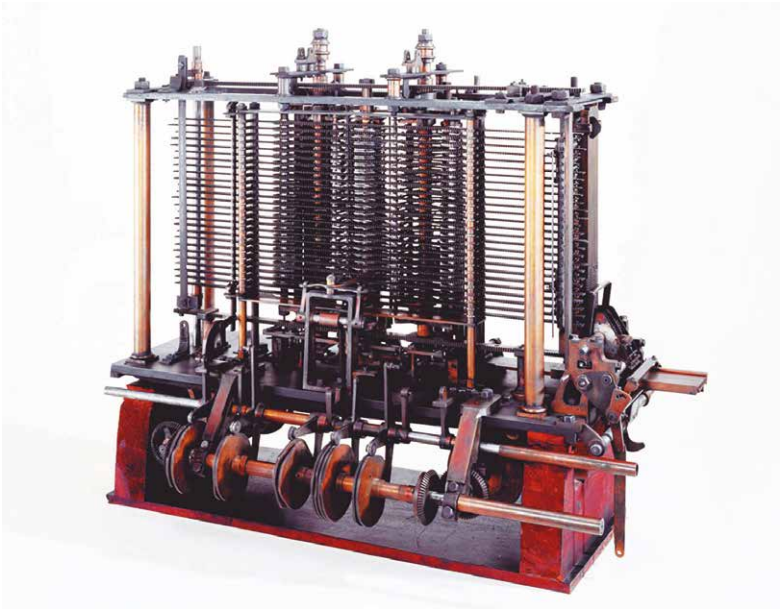


Figura 5 – Máquina de Babbage

Disponível em: https://bit.ly/2pKqMzF. Acesso em: 14 mar. 2023.

A Segunda Guerra Mundial desencadeou o desenvolvimento de máquinas que pudessem acelerar os procedimentos manuais realizados para fins militares. Nesse período surgiram os primeiros computadores eletromecânicos, que eram calculadoras de grande capacidade, formados por milhares de válvulas, e ocupavam áreas enormes, sendo de funcionamento lento e sem confiabilidade.

1.4.1 Primeira geração (1945–1955)

A primeira geração de computadores é caracterizada pela utilização de válvulas como principal componente eletrônico. Eram computadores primitivos, ocupavam áreas enormes e demoravam segundos para realizar os cálculos mais simples.

Para esses computadores, o mesmo grupo de pessoas era responsável por projetar, construir, programar, operar e manter cada máquina. Em muitos casos, a programação ocorria por meio de circuitos elétricos conectando cabos elétricos com painéis para o controle de funções básicas da máquina. As linguagens de programação eram desconhecidas e o sistema operacional era inexistente.

Na operação desse computador, o programador reservava um bloco de tempo e inseria seu painel de programação na sala de máquinas. Entretanto, a operação poderia ser interrompida caso algumas das dezenas de milhares de válvulas queimassem. Praticamente todos os problemas resolvidos por esses computadores eram cálculos matemáticos ou cálculos de trajetórias balísticas.

Considera-se como o primeiro computador digital e eletrônico o Electronic Numerical Integrator and Calculator (Eniac), desenvolvido pelos engenheiros J. Presper Eckert e John W. Mauchly na Universidade da Pensilvânia, e sua principal função era a realização de cálculos balísticos e, posteriormente, a utilização

no projeto da bomba de hidrogênio. A estrutura do dispositivo era composta de 17 mil válvulas, 10 mil capacitores e ele pesava 30 toneladas. O consumo de energia era de 140 quilowatts e era capaz de realizar 5 mil adições por segundo. A figura a seguir apresenta parte da estrutura do computador.

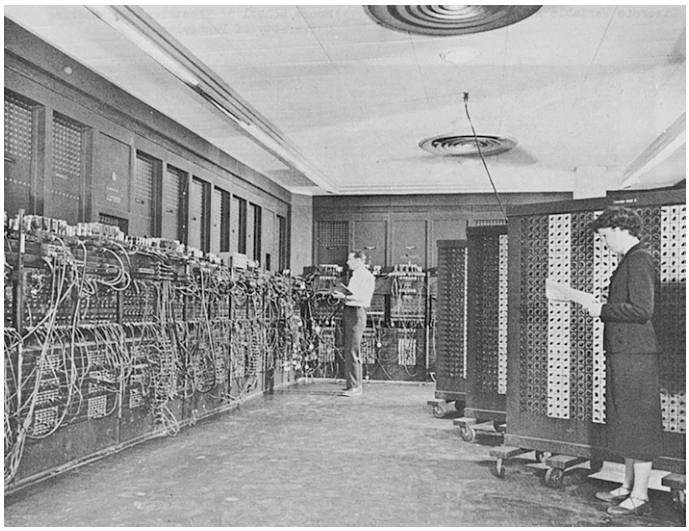


Figura 6 – Computador Eniac

Disponível em: https://bit.ly/3JGMYVn. Acesso em: 14 mar. 2023.

No início da década de 1950, os painéis de programação foram substituídos por máquinas perfuradoras de cartões, que propiciaram o avanço no processo.

1.4.2 Segunda geração (1955-1965)

Em 1947, foi descoberto o transistor, um dispositivo semicondutor capaz de amplificar um sinal eletrônico ou funcionar como uma chave eletrônica, que impulsionou a eletrônica e o desenvolvimento dos computadores. Ele substituiu a válvula na construção de computadores, pois tinha as vantagens de ocupar menos espaço, consumir menos potência e ter maior confiabilidade. O transistor é uma invenção creditada a três pesquisadores do Bell Labs: William Bradford Shockley, John Bardeen e Walter Houser Brattain, que receberam o Prêmio Nobel de Física em 1956.

Saiba mais

O artigo a seguir detalha a evolução do transistor ao microprocessador de forma didática e com muitas figuras dos dispositivos.

MEHL, E. L. M. *Do transistor ao microprocessador*. Curitiba: Universidade Federal do Paraná (UFPR), [s.d.]. Disponível em: https://bit.ly/3LpdZh2. Acesso em: 14 mar. 2023.

Nesta geração apareceu a distinção entre as funções de projetistas, construtores, programadores e responsáveis técnicos de manutenção que apresentavam tarefas específicas.

As máquinas desenvolvidas na época são denominadas de computadores de grande porte ou mainframes e eram instaladas em salas amplas, climatizadas e projetadas para este fim. Para o pleno funcionamento desses computadores, era necessária uma equipe de alta qualificação técnica. Devido ao alto custo dos mainframes, somente grandes empresas, universidades importantes ou agências do governo poderiam adquiri-los.

Esses computadores executavam tarefas como um programa ou conjunto de programas. Esses programas eram escritos com cartão perfurado utilizando linguagem Fortran ou linguagem de máquina Assembly e o programador utilizava uma perfuradora de cartões. Com os cartões perfurados, o programador se dirigia à sala de entradas. Como normalmente existia uma fila de programas para execução, o profissional aguardava até que seu programa fosse executado. Após isso, a saída era impressa e o operador iria até a impressora retirar o resultado da execução do programa.

A)

B)

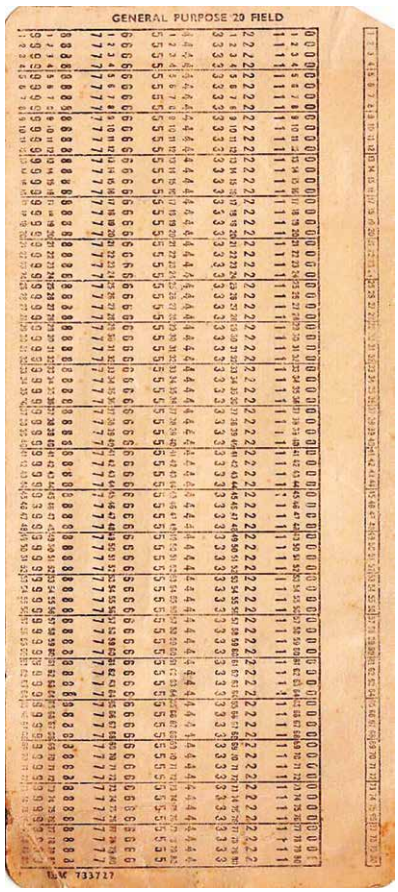
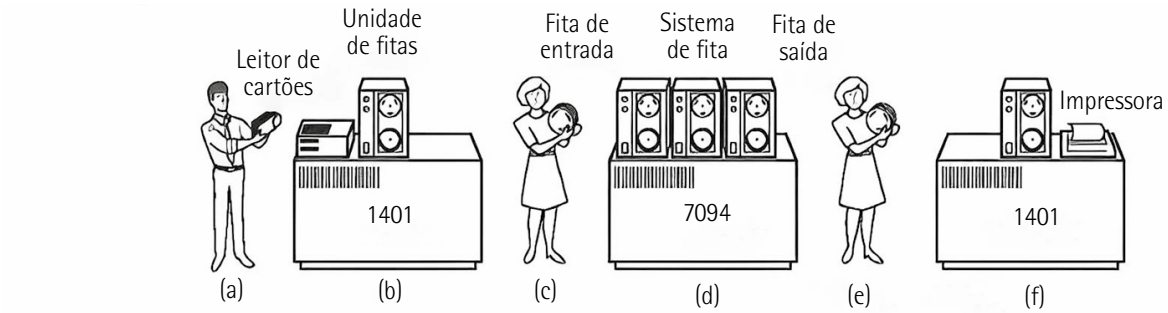


Figura 7 – A) Foto de perfuradora de cartões e B) cartão perfurado (Museu da IBM em São Paulo)

Nesta operação havia muito desperdício de tempo, e o custo dos computadores era muito alto, portanto a solução foi adotar um sistema em lote ou batch. Nela, um lote de tarefas era reunido na sala de entradas e passado para uma fita magnética, utilizando um computador dedicado à leitura de cartões, cópias de fitas e impressão das saídas dos programas, fatos que geravam um custo muito menor que os computadores destinados a cálculos matemáticos.

Quando um lote inteiro estava finalizado, o operador retirava as fitas de entrada e saída, substituía a fita de entrada para o próximo lote e levava a fita de saída para impressão, sem estar conectado ao computador principal. A figura a seguir ilustra a operação desse sistema.



(a) Programadores levavam cartões para o 1401.
(b) 1401 lia o lote de tarefas em uma fita.
(c) Operador levava a fita de entrada para o 7094.
(d) 7094 executava o processamento.
(e) Operador levava a fita de saída para o 1401.
(f) 1401 imprimia as saídas.

Figura 8 – Operação de sistema em lote ou batch

Fonte: Tanenbaum e Bos (2016, p. 6).

Nesta geração surgem as primeiras linguagens de programação de alto nível, tais como Cobol, Fortran e Algol. Com a introdução dessas linguagens, o papel do programador foi facilitado, pois ele não precisava mais ter uma relação direta com o hardware do computador, que é extremamente complexo. Portanto, o desenvolvimento de programas se tornou mais rápido, bem como a sua manutenção.

De forma análoga às linguagens de programação, o sistema operacional também evoluiu de forma a possibilitar as funções de codificar, executar e depurar os programas. Para tornar isso possível, os sistemas operacionais incluíram o próprio conjunto de rotinas para operações de entrada/saída ou input/output control system (IOCS). Com a introdução do IOCS, foi eliminada a necessidade de os programadores desenvolverem rotinas de leitura ou gravação destinadas a cada dispositivo, criando o conceito de independência de dispositivo.

Figura 9 – Foto do Mainframe IBM 704

Disponível em: https://bit.ly/4097WBl. Acesso em: 14 mar. 2023.

1.4.3 Terceira geração (1965-1980)

A terceira geração de computadores incorporou inúmeras inovações nos sistemas operacionais, empregando técnicas utilizadas até os dias atuais, tais como multiprogramações, multiprocessamento, memória virtual e compartilhamento de tempo ou time-sharing.

Antes da multiprogramação, apenas um programa ficava residente na memória e, quando ele realizava uma operação com os dispositivos de entrada/saída, o processador não operava e aguardava o término da leitura ou gravação. Para cálculos científicos, nos quais o uso do processador é muito intenso e há baixa frequência de leituras ou escritas, o tempo ocioso não era significativo. Entretanto, para processamento de informações empresariais, o tempo de espera por operações de entrada/saída poderia representar de 80% a 90% do tempo de operação, segundo Tanenbaum e Woodhull (2008). Com a introdução da multiprogramação, vários programas poderiam compartilhar a memória simultaneamente e, durante uma operação de entrada/saída, o processador poderia executar outro software, reduzindo a ociosidade do processador.

De acordo com Machado e Maia (2013), visando diminuir o tempo necessário para o desenvolvedor de software depurar a sua aplicação, foram introduzidas interfaces para o usuário. Adicionalmente, cada programa somente poderia utilizar o processador durante pequenos intervalos. Esse processo de divisão do tempo de processamento foi denominado compartilhamento de tempo ou time-sharing, servindo para fins diversos, em inglês, *compatible time-sharing systems* (CTSS).

Com objetivo de criar uma interface entre usuário e computadores, bem como interação do usuário durante a aplicação, novos dispositivos de entrada/saída foram disponibilizados, tais como o teclado e o terminal de vídeo.

Uma importante evolução no hardware ocorreu com a introdução de circuitos integrados (CIs), que são circuitos eletrônicos miniaturizados, construídos com material semicondutor. Computadores construídos com CIs possuem uma importante vantagem em relação a preço e desempenho quando comparados a máquinas feitas com transistores individuais.

A IBM introduziu o mainframe System/360, que era composto de uma série de máquinas com softwares compatíveis, com a mesma arquitetura interna e o mesmo conjunto de instruções. Com isso, os programas escritos em uma máquina poderiam operar em todas as outras, formando uma família de computadores que atenderia a necessidade dos clientes.

Figura 10 – Foto do Mainframe IBM System 360

Disponível em: https://bit.ly/3YSvmdb. Acesso em: 14 mar. 2023.

Em 1969, Ken Thompson, que tinha participado no projeto do sistema operacional Multics, implementou seu próprio sistema operacional, que viria a ser conhecido como Unix e que foi construído a partir da linguagem de programação C, criada por Dennis Ritchie. Suas principais características técnicas são a portabilidade, que permitia que fosse utilizado em diferentes configurações de hardware, assim como a capacidade de multiusuário e de multitarefa e estabilidade.

1.4.4 Quarta geração (1980-1990)

A evolução da eletrônica possibilitou o aumento da capacidade de integração, permitindo a construção de milhares de transistores em placas de silício de circuitos integrados através da união em larga escala ou *large scale integration* (LSI) e a integração em muito larga escala ou *very large scale integration* (VLSI).

Com o aumento da quantidade de transistores em uma mesma área de placa de circuito integrado, foi possível iniciar o projeto de miniaturização e redução de preços dos computadores, pois os processadores e as memórias estavam cada vez mais rápidos e baratos e os dispositivos de E/S ampliaram sua capacidade de armazenamento e velocidade, além de terem as suas dimensões reduzidas.

Como resultado dessa evolução, são lançados os primeiros computadores pessoais ou *personal computers* (PCs). Surgem os primeiros sistemas operacionais para esses computadores pessoais, como o MS-DOS, o sistema operacional de disco da Microsoft, em inglês *MicroSoft disk operating system*. O MS-DOS foi adotado nos PCs da IBM e a interface com usuário era textual, com a digitação de comandos no teclado pelos usuários.

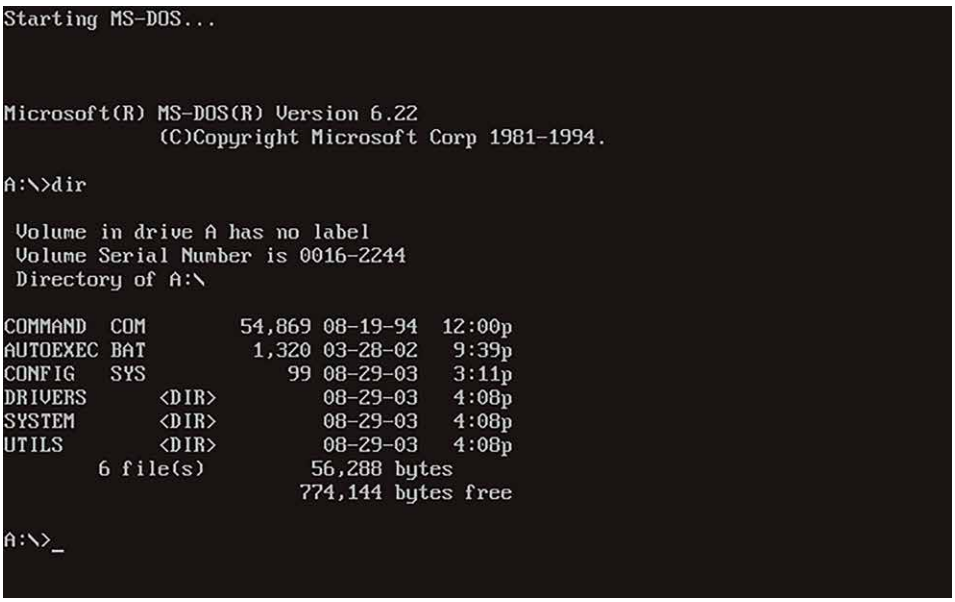


Figura 11 – Interface do MS-DOS

Disponível em: https://bit.ly/3I5MQFu. Acesso em: 14 mar. 2023.

Apesar da ideia de utilizar uma interface gráfica para o usuário (GUI) ter surgido no Centro de Pesquisas da Xerox na década de 1970, a empresa Xerox foi lenta para o lançamento de um produto comercial com essa solução. A Apple Macintosh introduziu a interface gráfica GUI na linha de computadores da Apple e foi lançada em janeiro de 1984, com a apresentação de Steve Jobs. A figura a seguir mostra o aparelho e seu criador.

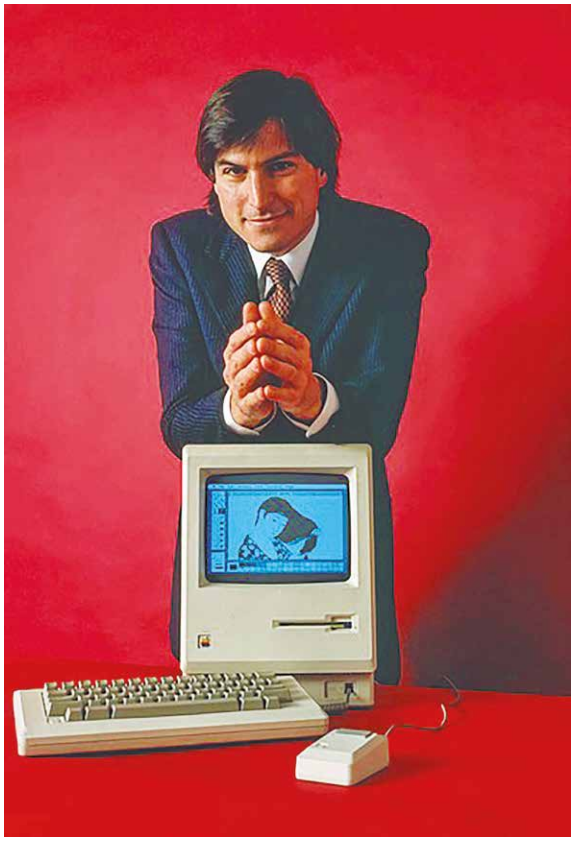


Figura 12 – Apple Macintosh e Steve Jobs em 1984

Disponível em: https://bit.ly/3Tddn07. Acesso em: 14 mar. 2023.

Devido à evolução dos microprocessadores, principalmente da família Intel, surgiram os primeiros sistemas operacionais comerciais que ofereciam interface gráfica para interação com o usuário, como o Microsoft Windows e o OS/2.

1.4.5 Quinta geração (1990-hoje)

A década de 2000 também é um marco no desenvolvimento de softwares, hardwares e telecomunicações e o seu crescimento, junto à ascensão dos notebooks, netbooks e celulares. Com esse progresso, a evolução da internet e suas inúmeras possibilidades de realização de transações comerciais dispararam com os sistemas de busca como Google e ainda houve a consolidação das redes sociais.

De forma geral, houve o aprimoramento dos sistemas operacionais da década de 1990, e a evolução na interface teve como objetivo torná-la mais intuitiva e melhorar a experiência do usuário. Foram introduzidos procedimentos de atualização e correção de erros nos sistemas operacionais por meio de conexão com a internet, sendo que ocorriam verificações automáticas de novas versões. Essas atualizações são fundamentais para correção de falhas críticas de segurança que podem representar vulnerabilidades para o usuário.

Em 1991, o engenheiro de software finlandês Linus Torvalds iniciou o desenvolvimento do sistema operacional Linux, um acrônimo de seu primeiro nome com Unix, que evoluiu a partir da colaboração de vários programadores que ajudaram no progresso de kernel, utilitários e vários aplicativos. Atualmente o Linux é utilizado tanto para fins acadêmicos como comerciais e possui centenas de distribuições.

Em 2010, a inovação foi o desenvolvimento da computação em nuvem, ou cloud computing, que pode ser entendida como a entrega de recursos de TI sob demanda através da internet e com preço definido pela utilização. Desse modo, as empresas não precisam construir e manter datacenters e servidores físicos, pois basta o acesso a serviços conforme a necessidade, utilizando um provedor da nuvem, como a Amazon Web Services (AWS) e o Microsoft Azure (AWS, s.d.).

Com a difusão da internet, foi necessário ao sistema operacional oferecer suporte à pilha de protocolos TCP/IP na web. Adicionalmente, houve a adoção de paradigmas cliente/servidor que acarretaram no lançamento das versões de sistemas operacionais específicas para servidores, com maior capacidade de gerenciamento de recursos e de sistemas de operacionais para os clientes, nas versões desktop do sistema operacional.

Desde o início da década de 2010, houve a popularização de dispositivos móveis, como smartphones e tablets, ao redor do mundo. Com isso, as empresas desenvolvedoras de tecnologia investiram fortemente na construção de sistemas operacionais para este tipo de dispositivo com diversas funcionalidades de comunicação, interface intuitiva e compatibilidade com seus sistemas para servidores e desktops. Portanto, houve o surgimento de sistemas operacionais como o Android, da empresa de tecnologia Google, cujo maior concorrente é o IOS, de outra gigante, a Apple.

1.5 Tipos de sistemas operacionais

Do mesmo modo que fazemos com automóveis, hardware e software, os sistemas operacionais podem ser classificados em função de diversos aspectos, tais como porte do computador onde será instalado, suporte a hardware específico, acesso à rede, utilização do processador por uma ou múltiplas tarefas, entre outros.

De acordo com Machado e Maia (2013), os tipos de sistemas operacionais e sua evolução estão conectados diretamente com o desenvolvimento do hardware e das aplicações suportadas por ele. Quando o hardware altera a forma de interação com o usuário ou existe uma mudança na forma de trabalho do processador, é necessária uma adequação das técnicas e nos elementos do sistema operacional para que haja a tentativa de refletir uma nova forma de interação ou processamento. Consequentemente, alterações relevantes no hardware levam a modificações no sistema operacional para suporte ao hardware modificado.

As formas de classificação mais usuais dos sistemas operacionais são estas:

- Em relação ao compartilhamento de hardware:
  - Sistema operacional monoprogramável ou monotarefa
  - Sistema operacional multiprogramável ou multitarefa
    - Sistema operacional batch ou em lote
- Em relação ao tipo de equipamento em que irá operar:
  - Sistema operacional desktop
  - Sistema operacional servidor
  - Sistema operacional embarcado
- Em relação ao acesso a hardware:
  - Sistema operacional de rede
  - Sistema operacional distribuído
- Em relação à quantidade de usuários:
  - Sistema operacional monousuário
  - Sistema operacional multiusuário
- Em relação à criticidade de respostas em função do tempo:
  - Sistemas de tempo real

Deve-se ressaltar que um mesmo sistema operacional pode se enquadrar em mais de uma classificação.

1.5.1 Sistema operacional monoprogramável ou monotarefa

Este tipo de sistema permite apenas que um programa seja executado em determinado período, e esse aplicativo deve permanecer na memória principal até o fim da execução. Todos os recursos de hardware, como memória, processador e periféricos, são alocados exclusivamente para um único programa.

Com isso, os recursos de hardware podem ficar muito tempo ociosos. Caso o programa executado necessite de processamento, mas não tenha acesso ao disco rígido, este ficará ocioso e não será utilizado

por outro programa. Uma vantagem desse tipo de sistema operacional é a simplicidade da sua parte gerencial, que pode ser facilmente implementada. Podemos citar como exemplos MS-DOS, sensores e sistemas embarcados com uma função específica.

1.5.2 Sistema operacional multiprogramável ou multitarefa

Neste tipo de sistema, mais de um programa fica armazenado na memória simultaneamente para permitir o compartilhamento do tempo de CPU e demais recursos de hardware, dando a impressão ao usuário de execução simultânea. Assim, podem existir diversas tarefas em um único núcleo de processador ou vários núcleos, arquitetura multicore. Unix e Windows são exemplos de sistemas multiprogramáveis.

No caso de um único processador, enquanto uma tarefa é executada, as outras ficam em modo de espera. São necessários mecanismos de trocas rápidas de processos, e existe um algoritmo de escalonamento que escolhe os processos que estarão em execução a cada instante.

Uma das vantagens desse tipo de sistema é uso mais otimizado dos recursos, pois os recursos de hardware serão compartilhados entre as diversas tarefas em execução.

Entre os sistemas multiprogramáveis, podemos destacar os seguintes grupos:

- sistemas em lote ou batch;
- sistemas de tempo compartilhado ou time-sharing;
- sistemas de tempo real.

1.5.3 Sistema operacional em lote ou batch

Os sistemas operacionais em lote ou batch consistem em submeter de uma só vez ao computador um lote de programas, que foram os primeiros sistemas operacionais multiprogramáveis a serem desenvolvidos na década de 1960. Todos os programas a executar eram colocados em uma fila.

Durante a década de 1960, os termos programa ou job eram os mais utilizados. Com a evolução dos sistemas, surgiu o conceito de processo e subprocesso e, posteriormente, o de thread. Os jobs, scripts com lote de programas, eram submetidos para execução através de cartões perfurados e armazenados em disco ou fita, onde aguardavam para ser processados. Os usuários são submetidos em ordem sequencial para execução.

Em sistema batch, não existe interação entre o usuário e o job durante a execução. O termo lote ainda é usado para definir um conjunto de comandos que rodam sem interferência do usuário. O processador recebia um programa após o outro, processando-os em sequência, o que permitia um alto grau de utilização do sistema.

Após a execução de um job, por conta da indisponibilidade de espaço na memória principal, aquele que é executado produz uma saída em disco ou fita. As entradas e saídas de dados do programa de

sistemas em lotes são implementadas através de memória secundária, normalmente em arquivos em disco, sem interação com usuário.

São exemplos de atividades processadas por esse tipo de sistema: cálculos numéricos para pesquisas acadêmicas, empresariais e militares, backups, ordenações e atividades que não necessitem de interação do usuário com o programa. Sistemas em lote, quando projetados adequadamente, podem ser muito eficientes, devido à melhor utilização do processador. Todavia, o tempo de resposta para uma atividade pode ser extremamente longo.

1.5.4 Sistemas de tempo compartilhado

Os sistemas de tempo compartilhado ou time-sharing são aqueles que possibilitam a execução de diversos programas a partir da divisão do tempo do processador em pequenos intervalos, denominados fatia de tempo ou time-slice, em inglês. Se a fatia de tempo não for suficiente para a finalização do programa, ele é interrompido pelo sistema operacional e ocorre a troca de processo; o programa que foi interrompido irá aguardar uma nova fatia de tempo.

Normalmente, sistemas de tempo compartilhado aceitam a interação dos usuários com o sistema através de terminais que incluem vídeo, teclado e mouse. Os diversos terminais se comunicam com o mainframe e dividem o tempo de processamento desse computador central. Esses sistemas possuem uma linguagem de controle que permite ao usuário comunicar-se diretamente com o sistema operacional através de comandos. Com isso, o usuário pode verificar arquivos armazenados em disco ou cancelar a execução de um programa – por exemplo, diversos sistemas bancários que são executados em computadores de grande porte, mainframes, após solicitações de consulta pelos terminais dos usuários.

Em condições normais, o sistema responde em poucos segundos à maioria desses comandos e também são denominados sistemas on-line.

1.5.5 Sistemas de tempo real

Sistemas de tempo real ou real-time operating systems (RTOS) são um tipo de sistema operacional nos quais o tempo é essencial, sendo caracterizados por um comportamento temporal previsível, isto é, seu tempo de resposta deve ser conhecido no melhor e pior caso de operação. Uma definição mais formal é:

> Sistemas operacionais de tempo real são uma subclasse de sistemas operacionais destinados à concepção de sistemas computacionais, geralmente embarcados, em que o tempo de resposta a um evento é fixo e deve ser respeitado sempre que possível (DENARDIN; BARRIQUELLO, 2019, p. 37).

Diferentemente da ideia-padrão, um sistema operacional de tempo real não precisa operar a velocidades extremas, pois a prioridade deve ser o cumprimento dos prazos de todos os eventos controlados por ele.

Por conta da restrição temporal, 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.

Esse sistema, normalmente, está presente em aplicações de controle de processos industriais, como no monitoramento de refinarias de petróleo, siderúrgicas, metalúrgicas, linhas de montagem, controle de tráfego aéreo, de usinas termoelétricas e nucleares, veículos autônomos ou em qualquer aplicação em que o tempo de processamento seja fator fundamental. Exemplos de sistemas operacionais de tempo real incluem o QNX, RT-Linux e VxWorks.

Há dois tipos de sistemas de tempo real: hard real-time systems ou sistemas de tempo real críticos e soft real-time ou sistemas de tempo real não críticos. Em hard, a perda de um prazo pelo sistema pode perturbar gravemente o sistema físico sob seu controle, com relevantes consequências humanas, econômicas ou ambientais. O controle de funcionamento de uma turbina de avião ou de um freio ABS são exemplos desse tipo de sistema.

Por outro lado, em sistemas de tempo real não críticos, a perda de um prazo é perceptível e degrada o serviço prestado, sem maiores consequências. Softwares de reprodução de mídia são exemplos. Quando acontecem atrasos, podem ocorrer falhas na música que está sendo tocada.

1.5.6 Sistemas embarcados

De acordo com Maziero (2019), um sistema operacional é classificado como embarcado ou embedded, em inglês, quando é construído para operar sobre um hardware com recursos limitados de processamento, armazenamento e energia. Em muitos casos, um sistema de tempo real é também um sistema embarcado.

Um sistema embarcado realiza uma série predefinida de tarefas, normalmente, com requisitos específicos. Por ser dedicado a tarefas específicas, o projeto desse sistema é otimizado, podendo reduzir as dimensões físicas, recursos computacionais e custo de produção.

Existem diversas aplicações de sistemas embarcados; por exemplo, em veículos o controle de injeção eletrônico e o controle de sistema de frenagem antibloqueio, freios ABS, são sistemas embarcados. Em eletrodomésticos, utiliza-se sistemas embarcados nas TVs, nos fornos de micro-ondas, nas centrais de alarme, nas lavadoras de roupa, entre outros. Na área de comunicação, existem sistemas embarcados nos telefones celulares, nos equipamentos de GPS, nas antenas de radiofrequência, nos roteadores de redes de computadores etc. Na robótica, existem drones e robôs industriais. Já nas áreas militar e aeroespacial, os sistemas de gerenciamento de voo e de controle de armas de fogo também são embarcados. LynxOS, TinyOS, Contiki e VxWorks são exemplos de sistemas operacionais embarcados.

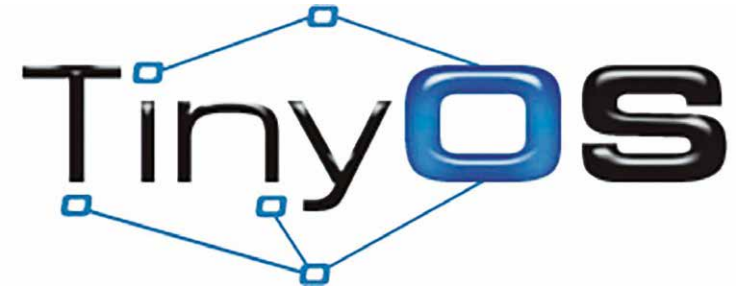


Figura 13

Disponível em: https://bit.ly/3YJgApb. Acesso em: 14 mar. 2023.

1.5.7 Sistemas operacionais distribuídos e de rede

De acordo com Deitel, Deitel e Choffnes (2005), um sistema operacional distribuído é aquele que gerencia recursos em mais de um sistema de computador mesmo estando instalado em um único computador. Ele gera a impressão de que diversos computadores compõem um único computador de grande capacidade, de forma que um processo poderá acessar qualquer recurso do sistema independentemente de sua localização.

Entretanto, poucos sistemas operacionais comerciais se enquadram nessa classificação, pois eles são extremamente complexos para implementação e requerem algoritmos complicados para habilitar os processos a se comunicar e a compartilhar dados. Os sistemas Chord do MIT e Amoeba da Vrije Universiteit de Amsterdã, na Holanda, são exemplos de sistemas operacionais distribuídos.

O avanço nas redes de telecomunicações e de computadores também impactou o sistema operacional. Um sistema operacional de rede permite que processos possam acessar recursos, tais como arquivos, que estejam em outros computadores de uma rede. Normalmente, esse sistema é baseado em uma arquitetura cliente/servidor. Os clientes dessa rede são os computadores que necessitam de recursos como tempo de processamento ou arquivos e utilizam um protocolo de rede adequado. Por outro lado, os servidores são os responsáveis pela resposta às requisições dos clientes e disponibilização dos recursos.

1.5.8 Sistemas operacionais móveis

No século XXI, houve uma expansão gigantesca da utilização de dispositivos móveis, tais como smartphones, tablets e relógios inteligentes (smartwatches). Frequentemente, os sistemas operacionais desse tipo incluem um kernel básico e outra estrutura de software denominada de middleware com objetivo de fornecer serviços adicionais para desenvolvedores de aplicativos.

Por exemplo, os dois sistemas operacionais de dispositivos móveis mais utilizados, Android da Google e iOS da Apple, utilizam um kernel básico e o middleware que possui capacidade de suportar operação em bancos de dados, elementos gráficos e ambientes multimídia.

1.5.9 Classificação de sistemas operacionais segundo utilização e funções

De acordo com Alves (2018), sistemas operacionais podem ser classificados com base em utilização e funções existentes. Por exemplo, a família Microsoft Windows pode ser separada em sistemas operacionais Home, Professional e Server. As versões Home ou Home Editions são compostas de softwares desenvolvidos para uso doméstico ou residencial, sendo simplificadas. Por exemplo, não há integração com recursos corporativos, mas dispõem de diversos recursos para entretenimento, como jogos, acesso a streaming de vídeo, ferramentas para edição de imagens e vídeos.

Como o próprio nome indica, a versão Professional está voltada ao mercado corporativo, sendo que uma diferença relevante é o tipo de licenciamento, geralmente feito por assinaturas anuais. Essa assinatura permite o upgrade do sistema conforme novas versões são lançadas. Adicionalmente, a versão Professional permite a integração com domínios, oferece recursos de criptografia e controle de acesso a discos, recursos de segurança e conexão remotas, como uma rede privada virtual (VPN) e ferramentas de controle e gestão para o software e seus usuários no ambiente corporativo.

Os servidores utilizam uma versão de sistema operacional específica que é conhecida como linha Server e são desenvolvidos para controlar toda a rede corporativa. A principal diferença para as outras versões é a capacidade de gerenciamento de recursos. Nela, pode-se estabelecer um domínio, em que os usuários, grupos, políticas de grupos e outros tipos de recurso corporativo são gerados para administrar todo um ambiente.

2 OPERAÇÃO DE SISTEMAS OPERACIONAIS

Neste capítulo serão apresentadas as principais arquiteturas do sistema operacional e as atividades de gerenciamento mais empregadas nele, tais como gerência do processador, gerência de memória, sistema de arquivos, gerência de dispositivos de entrada/saída e proteção para sistemas operacionais.

Como visto, na função de gerência de recursos, o sistema operacional é responsável por definir políticas para gerenciar o uso dos recursos de hardware pelos aplicativos e resolver eventuais disputas e conflitos em relação ao uso de processador, acesso a dispositivos de armazenamento, dispositivos de entrada/saída e alocação de memória.

2.1 Funções do sistema operacional

Visando cumprir as funções essenciais de abstração e gerência, o sistema operacional deve operar em vários campos. Os recursos de hardware possuem suas particularidades, impondo requisições específicas para o gerenciamento e abstração desses dispositivos. Por esse ponto de vista, as principais funcionalidades implementadas por um sistema operacional típico são:

- **Gerência do processador**: é também denominada gerência de processos, de tarefas ou de atividades. Tem como objetivo a distribuição da capacidade de processamento e a forma justa entre as aplicações, evitando que uma aplicação monopolize esse recurso e respeitando as prioridades definidas pelos usuários.

Lembrete

Para o sistema operacional, a divisão de um recurso de forma justa é diferente de divisão igual entre as partes.

O sistema operacional proporciona a visão de que existe um processador independente para cada tarefa, de modo a auxiliar a programação de aplicações e permitir a construção de sistemas mais interativos. Com isso, cada um dos processos poderá ser executado no tempo, sequência e velocidade adequados para cumprir suas funções sem prejudicar os outros. Outra tarefa dessa gerência é o fornecimento de abstrações para sincronizar tarefas interdependentes e prover formas de comunicação entre elas. Posteriormente detalharemos neste livro-texto a gerência de processos.

- **Gerência de memória**: através desta funcionalidade, cada aplicação terá um espaço selecionado de memória de forma isolada e independente de outras aplicações e do próprio sistema operacional. Com o isolamento dos espaços de memória, ocorre uma melhoria de estabilidade e da segurança do sistema computacional, pois programas com erros ou mesmo aplicações maliciosas não irão interferir na execução das demais aplicações. Adicionalmente, quando não há espaço suficiente na memória principal para as aplicações, o sistema operacional pode aumentar o tamanho da memória disponível utilizando o armazenamento em memória secundária, como um disco rígido. Isso ocorre de forma transparente para as aplicações, que desconhecem se estão executando na memória principal ou secundária. Outra abstração relevante implementada pela gerência de memória é o conceito de memória virtual, que permite o carregamento da aplicação em qualquer posição livre da memória, retirando a necessidade de estabelecer previamente na programação os endereços de memória onde a aplicação será executada.

- **Gerência de dispositivos**: é também conhecida como gerência de entrada/saída (E/S). Cada periférico do computador possui suas especificidades. Por exemplo, o procedimento de interação com uma placa de rede é bastante diferente da interação com um teclado USB ou disco rígido SSD. Entretanto, há muitas situações e abordagens semelhantes para acessar periféricos. Um desses casos é a criação de uma abstração única para a maioria dos dispositivos de armazenamento como, por exemplo, discos rígidos IDE, Sata ou SSD, pen drives, em um formato de vetor de blocos de dados. Sua função 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, mesmo que utilizem tecnologias distintas.

- **Gerência de arquivos**: esta funcionalidade é construída sobre a gerência de dispositivos, possibilita a criação de arquivos e de diretórios e estabelece uma interface de acesso, bem como as regras de utilização. 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 têm 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 informações internas do sistema operacional (como o diretório /proc do Unix).

- **Gerência de proteção**: considerando que os computadores estejam conectados em rede e os recursos sejam compartilhados por vários usuários, é essencial a definição clara de quais recursos cada pessoa pode acessar, as formas de acesso permitidas, isto é, a leitura, escrita ou execução, e as maneiras de garantir que essas permissões sejam dadas. Para proteger os recursos do sistema contra acessos indevidos, é necessário (MAZIERO, 2019):

  – a definição de usuários e grupos de usuários;

  – a identificação dos usuários que se conectam ao sistema, através de procedimentos de autenticação;

  – a descrição e aplicação das regras de controle de acesso aos recursos, relacionando todos os usuários, recursos e formas de acesso e as aplicando através de procedimentos de autorização;

  – o registro da utilização dos recursos pelos usuários, para fins de contabilização e auditoria.

2.2 Estrutura dos sistemas operacionais

Existem vários elementos que compõem o sistema e eles podem ser organizados de diversas maneiras, de forma a separar as suas funcionalidades e modular seu projeto. A estrutura do núcleo pode ser entendida como a maneira pela qual o código do sistema está organizado e o relacionamento entre seus diversos módulos, que podem variar em função da concepção do projeto.

Segundo Machado e Maia (2013), o sistema operacional é composto de uma coleção de rotinas que entregam serviços às aplicações dos usuários, denominada núcleo do sistema ou kernel. Afora o núcleo, a maior parte dos sistemas operacionais comtempla utilitários e linguagem de comandos, que são ferramentas de auxílio ao usuário, mas não fazem parte do núcleo do sistema. Uma aplicação convencional possui sua execução com o sequenciamento início, meio e fim. Por outro lado, as rotinas de um sistema de operação são processadas de forma concorrente e sem uma ordem predefinida, já que ocorre com base em eventos dissociados do tempo, isto é, assíncronos.

2.2.1 Núcleo do sistema operacional

O núcleo ou kernel do sistema operacional é considerado o coração do sistema e é responsável pela gerência dos recursos do hardware usados pelas aplicações. Adicionalmente, ele implementa as principais abstrações utilizadas pelos aplicativos e programas utilitários. Sua função principal é a alocação dos processos, pois o sistema operacional é um conjunto de processos inseridos a serem processados em uma CPU.

2.2.2 Modos de operação do sistema operacional

Para a operação do sistema operacional são definidos dois modos de acesso: a modalidade de usuário e a modalidade kernel, também denominada modalidade privilegiada ou modalidade de supervisor.

Ao operar na modalidade de usuário, seu processo não possui conexão direta ao hardware para acesso ao disco, por exemplo. Quando o processador trabalha no modo usuário, uma aplicação

somente poderá executar instruções denominadas **não privilegiadas** e há uma lista reduzida de instruções aplicáveis.

Quando necessário, a aplicação irá realizar a solicitação de um serviço ao sistema operacional, através de uma chamada de sistema ou system call. Com isso, o sistema passa da modalidade usuário para kernel e irá executar a solicitação. Após a execução da operação desejada, a aplicação voltará à modalidade usuário.

Quando o sistema operacional deseja obter controle sobre o hardware, ele deve selecionar a modalidade kernel. Nela, existe a permissão de execução de instruções privilegiadas. Exemplos: controle de entrada/saída, gerenciamento do timer e de interrupções, e instrução para alterar a modalidade de operação. Segundo Silberschatz, Galvin e Gagne (2015), quando ocorre a tentativa de realizar instruções privilegiadas no modo usuário, o hardware não irá executá-las e esses comandos serão tratados como ilegais, sendo interceptados pelo sistema operacional.

O modo de acesso é determinado por um conjunto de bits do registrador de status do processador, que indica o modo de acesso corrente. Através dele, o hardware verifica se a instrução pode ou não ser executada.

2.2.3 Chamadas de sistema ou system calls

O isolamento de cada aplicação em sua área de memória designado pela unidade de gerenciamento de memória ou *memory management unit* (MMU) proporciona robustez e confiabilidade ao sistema, pois garante que uma aplicação não poderá interferir nas áreas de memória. Um sistema é classificado como robusto quando a falha em um de seus componentes não implica no sistema como um todo.

Todavia essa proteção introduz um dilema (MAZIERO, 2019): como a aplicação atuará para invocar as rotinas disponíveis pelo núcleo para o acesso ao hardware e demais serviços do sistema operacional? Deve-se lembrar que o código do núcleo reside em uma área de memória inacessível à aplicação, então operações de desvio como jump e call não funcionariam. A ativação de uma rotina do núcleo utilizando esse mecanismo é conhecida como chamada de sistema ou system call, abreviada como syscall.

Os sistemas operacionais definem chamadas de sistema para todas as operações envolvendo o acesso a recursos de baixo nível (periféricos, arquivos, alocação de memória etc.) ou abstrações lógicas (criação e encerramento de tarefas, operadores de sincronização etc.). Geralmente as chamadas de sistema são oferecidas para as aplicações em modo usuário através de uma biblioteca do sistema (system library), que prepara os parâmetros, invoca a chamada e, no seu retorno, devolve à aplicação os resultados obtidos.

2.2.4 Interrupções

Segundo Maziero (2019), o processador e os dispositivos de entrada/saída se comunicam pelo acesso às portas de entrada/saída que podem ser lidas e/ou escritas pelo processador. Em algumas situações, o dispositivo deve informar rapidamente sobre um evento, como a finalização de operação no disco, por exemplo, ao clicar no mouse, digitar no teclado ou durante a chegada do pacote de rede.

O controlador pode esperar até a consulta do processador, o que poderá ser demorado caso o processador esteja ocupado com outras tarefas. Visando maior agilidade para tratar a interrupção, o controlador irá notificar o processador, enviando a ele uma requisição de interrupção ou *interrupt request* (IRQ) através do barramento de controle.

Após o recebimento da interrupção e depois de salvar o conteúdo do registrador na pilha de controle, um dos registradores a ter o conteúdo salvo é o contador de programa ou *program counter* (PC). Então, o processador suspende a execução de processo, desviando o fluxo de execução para um endereço definido previamente, onde está a rotina de tratamento de interrupção ou *interrupt handler.* Tal rotina será responsável pelo tratamento da interrupção, executando as atividades necessárias para o atendimento. Finalizando a rotina de tratamento da interrupção, o processador voltará a executar o código que estava sendo executado antes do recebimento da requisição, como se nada tivesse ocorrido.

O quadro a seguir descreve o mecanismo de interrupção, que é executado tanto no software quanto no hardware.

Quadro 2 – Mecanismo de interrupção

Por hardware	O processador recebe um sinal de interrupção
	O processador finaliza a execução da instrução corrente e identifica o pedido de interrupção
	Os conteúdos dos registradores PC e de status são armazenados
	O processador identifica qual a rotina de tratamento a ser executada e insere o PC com o endereço inicial dessa rotina
Por software	A rotina de tratamento salva o conteúdo dos demais registradores do processador na pilha de controle do programa
	A rotina de tratamento é executada
	Finalizada a execução da rotina de tratamento, são restaurados os valores dos registradores e ocorre o retorno da execução do programa interrompido

Fonte: Machado e Maia (2013, p. 39).

Outra rotina implementada pelo sistema operacional é a exceção, semelhante à interrupção. A principal diferença entre elas é que a exceção é originada por uma instrução do próprio programa, como a divisão de um número por zero ou a ocorrência de overflow em uma operação aritmética, e a interrupção é gerada por um dispositivo de entrada/saída.

2.3 Arquiteturas do sistema operacional

Os componentes do sistema operacional podem ser organizados de diversas maneiras, separando suas funcionalidades e modularizando seu projeto. Nesta seção, serão apresentadas as arquiteturas mais populares para a organização de sistemas operacionais.

2.3.1 Estrutura monolítica

Nesta estrutura, o sistema operacional consiste em um único módulo no qual um conjunto de programas é executado sobre o hardware, comportando-se como se fosse um único item. O bloco de código opera em modo núcleo, com acesso a todos os recursos do hardware e sem restrições de acesso à memória. Os módulos podem ser organizados de diversas maneiras, de forma a separar as suas funcionalidades e particionar o projeto do sistema operacional.

Os programas de usuário invocam rotinas do sistema operacional. São exemplos de sistemas monolíticos: MS-DOS, Unix e algumas distribuições do Linux. De acordo com Silberschatz, Galvin e Gagne (2015), o MS-DOS foi originalmente projetado e implementado por desenvolvedores que não imaginavam que o sistema se tornaria popular. Ele foi escrito para proporcionar o máximo de funcionalidade no menor espaço possível. Com isso, não foi dividido cuidadosamente em módulos. A figura a seguir apresenta uma das formas de organização do núcleo do sistema monolítico.

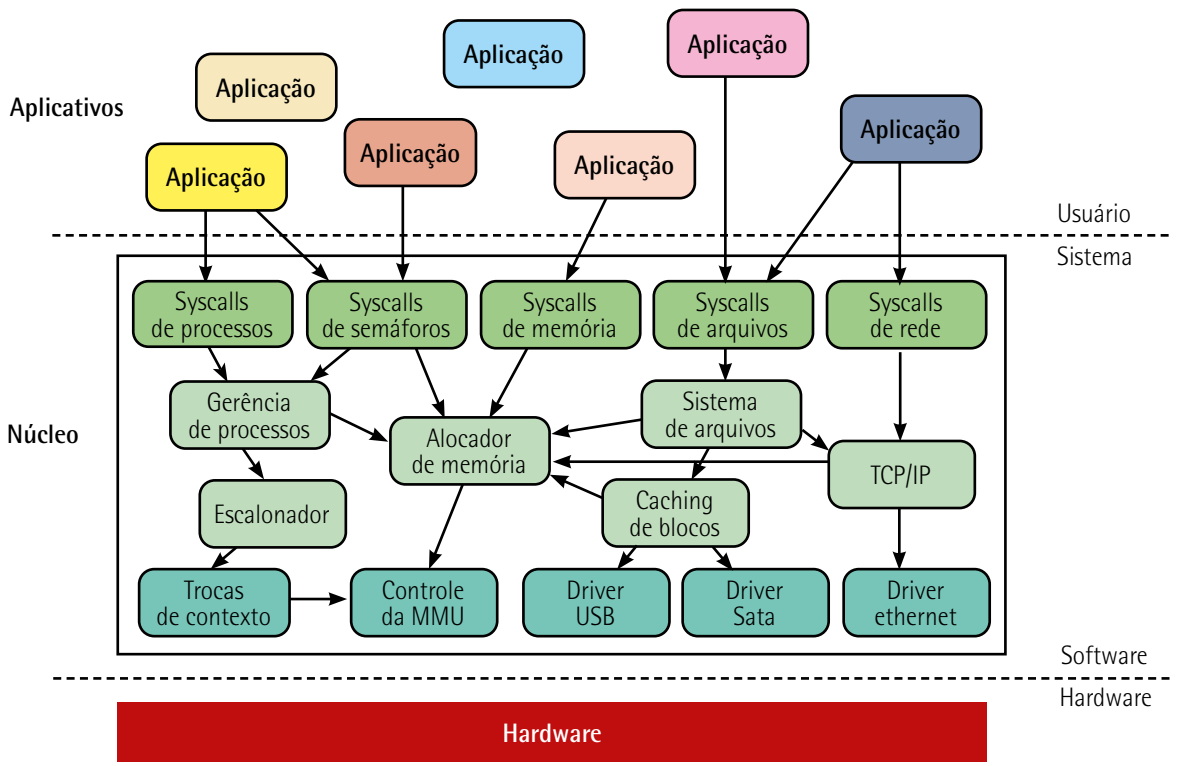


Figura 14 – Núcleo de sistema operacional monolítico

Fonte: Maziero (2019, p. 28).

A arquitetura monolítica possui seu desempenho como vantagem mais relevante, pois os componentes do núcleo acessam livre e diretamente a memória, os demais itens do núcleo e os dispositivos de entrada/saída (E/S). Essa interação entre os módulos proporciona sistemas compactos e de alta velocidade, além de dispensar mecanismos de comunicação entre as partes integrantes do núcleo.

Entretanto, a arquitetura monolítica impõe algumas desvantagens. Uma extremamente relevante é que a robustez do sistema operacional pode ficar comprometida. Como todos os componentes do sistema operacional possuem acesso privilegiado ao hardware, quando há uma falha em componente devido a algum erro, por exemplo acesso a um ponteiro inválido ou a uma posição inexistente em um vetor, ela pode se propagar rapidamente por todo o núcleo, levando o sistema ao colapso, isto é, ao travamento, à reinicialização ou ao funcionamento errático.

Outra desvantagem da arquitetura monolítica está ligada ao processo de desenvolvimento do sistema. Dado o acesso direto dos componentes do sistema operacional entre si, a existência de componentes fortemente interdependentes dificulta a construção e manutenção. Mesmo pequenas alterações na estrutura de dados de um componente podem impactar de forma inesperada outros componentes que a acessarem diretamente.

2.3.2 Estrutura microkernel

Com a expansão das funcionalidades do sistema operacional, o núcleo tornou-se cada vez mais complexo de gerenciar. Visando solucionar tal problema, a estrutura microkernel do sistema operacional remove todos os componentes não essenciais do kernel e implementa-os como programas de nível de sistema e de usuário, resultando em um kernel menor. Por reduzir o núcleo de sistema, essa abordagem foi denominada micronúcleo ou μ-kernel.

O núcleo fornece serviços de alocação de CPU, memória e comunicação entre processos (IPC). A figura a seguir ilustra a arquitetura de um microkernel típico.

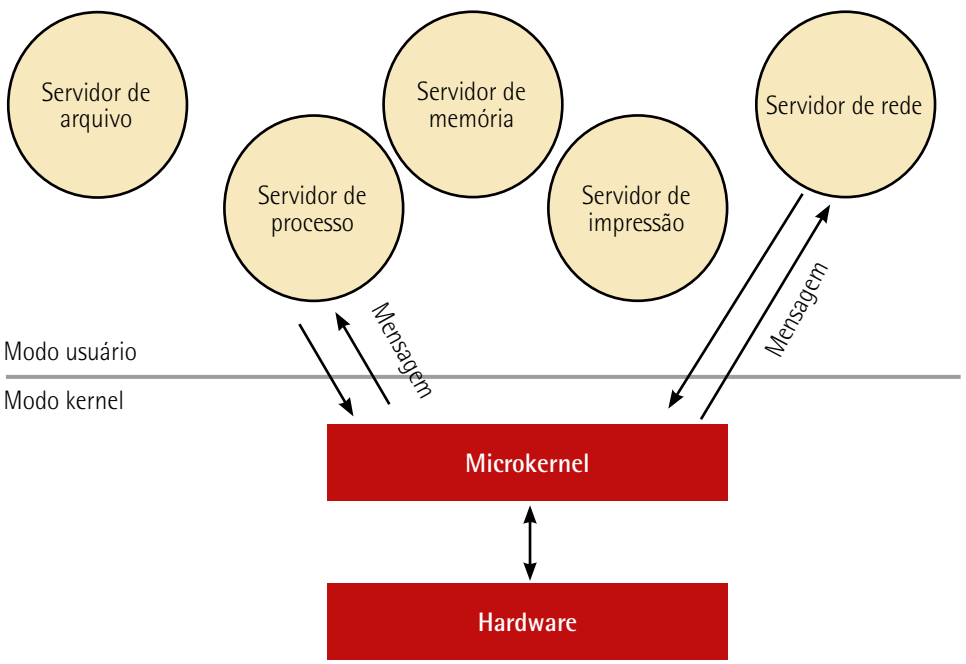


Figura 15 – Arquitetura de sistema operacional microkernel

Fonte: Machado e Maia (2013, p. 57).

Nesta arquitetura, quando o programa cliente buscar o acesso a um arquivo, ele irá interagir com o servidor de arquivos. O programa cliente e o serviço se comunicam indiretamente trocando mensagens com o microkernel; nunca o fazem por meio direto.

Uma vantagem da abordagem de microkernel é a maior modularidade, de forma a permitir que cada serviço seja desenvolvido de modo independente. Adicionalmente, há mais flexibilidade, já que os serviços podem ser ativados ou desativados conforme a demanda. Há melhoria da robustez do sistema, pois mesmo que um serviço falhe, somente ele será afetado e, devido ao confinamento de memória entre os serviços, não haverá uma falha geral e o resto do sistema operacional permanecerá intocado.

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 minimizadas, porque o microkernel é um kernel menor. O sistema operacional resultante é mais fácil de ser portado de um projeto de hardware para outro. O microkernel também fornece mais segurança e confiabilidade, já que a maioria dos serviços é executada como processo de usuário — e não do kernel.

Exemplos de sistemas operacionais com essa estrutura são Minix, um sistema operacional livre, baseado em Unix e criado por Andrew S. Tanenbaum, e Symbian, um sistema operacional móvel usado por diversas marcas de telefonia celular, principalmente a Nokia.

2.3.3 Estrutura de camadas

A ideia dessa estrutura é a criação de um sistema operacional modular e hierárquico. Com a modularização, divide-se um programa complexo, como é o próprio sistema operacional, em módulos de menor complexidade. A cada nível hierárquico, são ignorados os detalhes da operação de níveis inferiores.

As interfaces são definidas para facilitar a interação entre os módulos hierárquicos, a fim de estabelecer a comunicação com os níveis superior e inferior daquele módulo.

A estruturação em camadas foi muito bem-sucedida na área de redes de computadores, através do modelo de referência *open systems interconnection* (OSI). Inicialmente, espera-se que no desenvolvimento de sistemas operacionais tenha o mesmo sucesso.

Entretanto, inconvenientes limitariam a adoção do modelo em camadas. O desempenho do sistema é prejudicado, pois o empilhamento de várias camadas de software implica que cada pedido gaste mais tempo para chegar até o dispositivo periférico ou recurso a ser acessado. Adicionalmente, a divisão de funcionalidades do sistema em camadas não é uma tarefa trivial, porque muitas utilidades são interdependentes e não teriam como ser organizadas dessa forma.

São exemplos de sistemas operacionais com estrutura em camadas o Multics e o OpenVMS. A figura a seguir apresenta a estrutura do sistema OpenVMS.

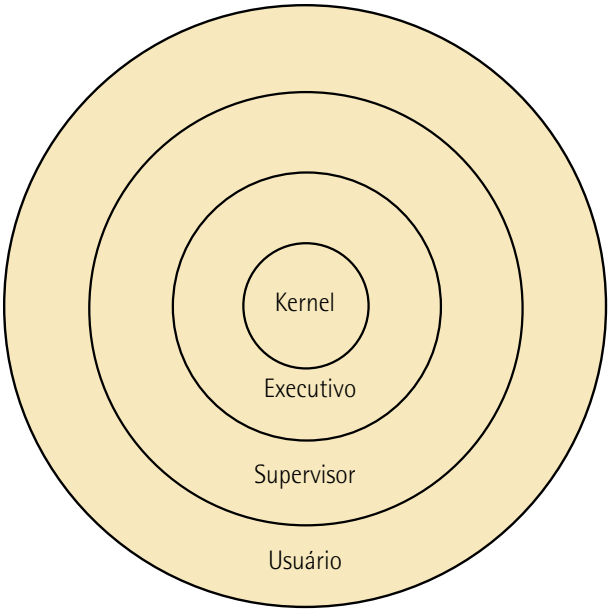


Figura 16 – Arquitetura de sistema operacional em camadas OpenVMS

Fonte: Machado e Maia (2013, p. 55).

A estrutura do OpenVMS é composta de quatro camadas concêntricas, sendo a mais privilegiada a kernel e a menos privilegiada usuário ou user. No modo kernel, existem o subsistema de E/S, subsistema de memória, gerenciamento de processos e escalonamento. No modo Executivo, há o Record Management Services (RMS), que provê suporte ao acesso a arquivos e registros. A camada Supervisor é composta por um interpretador de comandos (CLI) e pela *runtime library*, uma biblioteca de rotinas muito utilizada e previamente codificada que desempenha diversas funções. Por fim, o modo usuário contém utilitários, programas de usuário e comandos da linguagem DCL, utilizada para comunicação entre usuário e máquina.

2.3.4 Máquinas virtuais

Segundo Laureano (2006), o conceito de máquina virtual surgiu ainda no início da história dos computadores, no final da década de 1950 e início da década de 1960. Originalmente, as máquinas virtuais foram desenvolvidas visando à centralização dos sistemas computacionais utilizados no ambiente VM/370 da IBM. Para esse sistema, cada máquina virtual simula uma réplica física da máquina real e os usuários têm a impressão de possuírem o sistema de forma exclusiva.

Como existe um desenvolvimento independente entre os projetistas de hardware, sistemas operacionais e aplicações, que podem ocorrer em momentos e empresas diferentes, esses trabalhos resultam em diversas plataformas operacionais que podem não ser compatíveis entre si. Uma plataforma operacional representa a união entre hardware, sistema operacional e aplicações. Dessa forma, aplicações escritas para uma plataforma operacional não funcionam em outras.

A figura a seguir exibe distintas combinações de hardware, sistema operacional e aplicações. Nas três plataformas da parte superior, aplicações são compatíveis com o sistema operacional e com o hardware, e as duas plataformas na parte inferior apresentam problemas de compatibilidade entre aplicações, sistemas operacionais e plataforma de hardware.

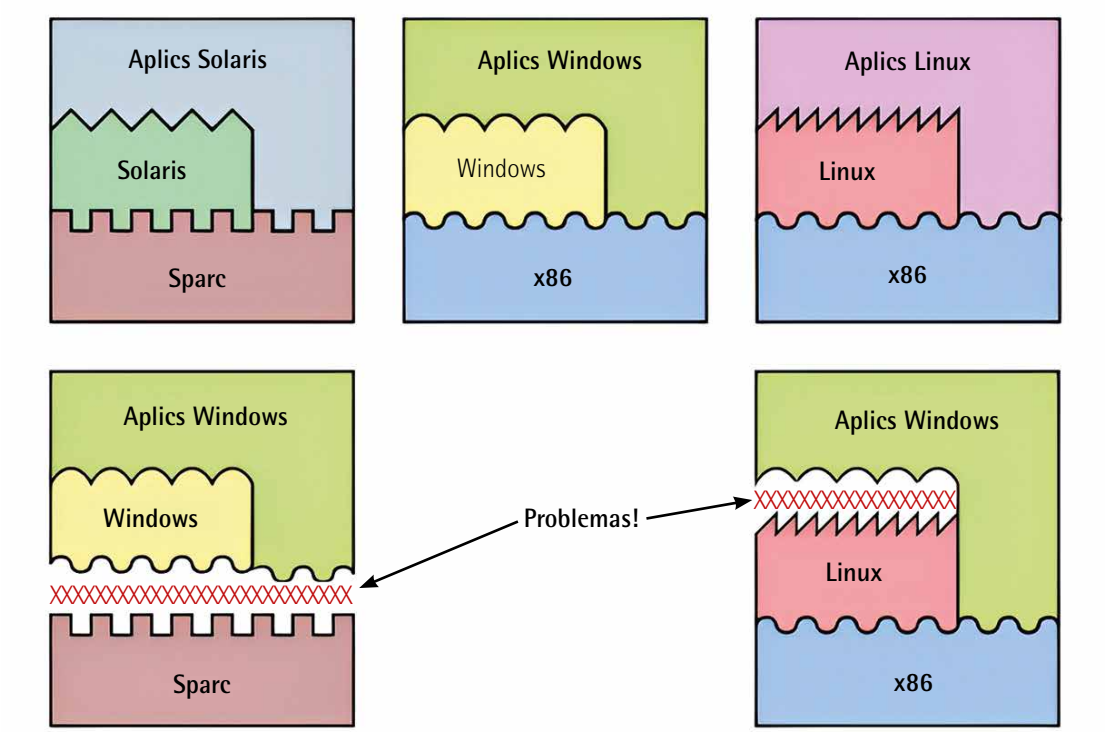


Figura 17 – Incompatibilidade entre plataformas operacionais

Fonte: Smith e Nair (2004, p. 36).

A arquitetura de processadores Sparc foi criada pela Sun Microsystems e é focada em servidores. Originalmente, o processador Sparc utilizava o sistema operacional Solaris, desenvolvido pela mesma empresa.

A família de processadores x86 foi desenvolvida pela Intel a partir do processador 8086, lançado em 1978. Os processadores 80286, 80386 e 80486 são exemplos da família x86 e foram utilizados na maior parte dos computadores pessoais das décadas de 1980 e início da década de 1990.

Com a introdução de máquinas virtuais, o problema de incompatibilidade é resolvido, porque elas introduzem uma camada intermediária entre hardware e sistema operacional, visando compatibilizar diferentes plataformas. O processo para criação dessa camada adicional é denominado virtualização, como mostra a figura a seguir.

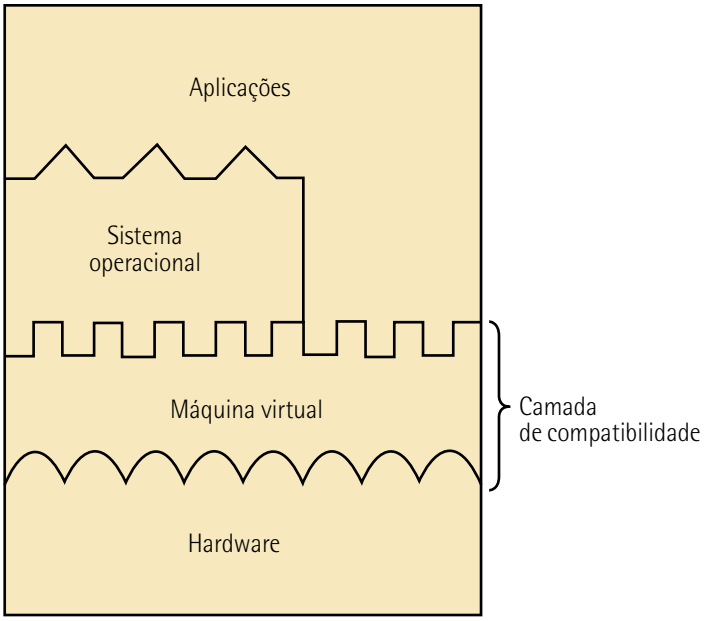


Figura 18 – Introdução da camada de virtualização

Fonte: Laureano (2006, p. 17).

Formalmente, uma máquina virtual ou *virtual machine* (VM) é definida como sendo uma duplicata eficiente e isolada de uma máquina real (POPEK; GOLDBERG, 1974). Por meio de máquinas virtuais, é possível a execução de um sistema operacional e de suas aplicações sobre outro sistema operacional, a ação de múltiplos sistemas operacionais e a flexibilização de uma plataforma complexa de trabalho. Por exemplo, conseguimos testar o software em diferentes sistemas operacionais e em diversas versões sem a necessidade de hardware dedicado.

Saiba mais

Um software utilizado para virtualização e que pode executar diferentes máquinas virtuais é o Oracle® VM VirtualBox®. Nele, o usuário pode instalar e rodar quantas máquinas virtuais desejar, mas existe a limitação do espaço em disco rígido e memória necessária para a virtualização. Baixe-o no site a seguir:

Disponível em: https://www.virtualbox.org/. Acesso em: 14 mar. 2023.

Outra aplicação de máquinas virtuais ocorre na execução de programas Java. A linguagem Java foi desenvolvida pela Sun Microsystems, que também criou uma máquina denominada Máquina Virtual Java ou Java Virtual Machine (JVM). Quando o compilador Java deseja rodar um código para a JVM, ele é executado por um interpretador JVM em software.

De acordo com Tanenbaum e Woodhull (2008), um benefício dessa implementação é que o código da JVM poderá ser executado por qualquer computador conectado à internet que possua JVM. Caso o compilador produzisse programas binários em Sparc ou Pentium, por exemplo, eles não poderiam ser enviados e executados em qualquer lugar de forma fácil. Adicionalmente, a segurança dos programas JVM recebidos pelo usuário poderá ser verificada e o software poderá ser executado em um ambiente seguro, evitando roubo de dados ou danos no computador do usuário.

2.4 Proteção para sistemas operacionais

Para Nieles, Dempsey e Pillitteri (2017), o termo segurança computacional pode ser definido como a proteção oferecida a um sistema de informações automatizado a fim de atingir os objetivos aplicáveis de preservar a integridade, disponibilidade, confidencialidade dos recursos do sistema de informações, o qual inclui hardware, software, firmware, informações/dados e telecomunicações.

Os sistemas de arquivos definidos pelo sistema operacional normalmente possuem informações extremamente valiosas para seus usuários. Portanto, a proteção delas contra uso não autorizado é uma preocupação mandatória em todos os sistemas de arquivos. Nesta seção, serão apresentados os conceitos de proteção, segurança e ameaças em sistemas operacionais.

É fundamental que os processos em um sistema operacional sejam protegidos das atividades de outros processos. Para tanto, podemos usar vários mecanismos a fim de assegurar que somente aqueles que tenham recebido previamente a autorização adequada possam realizar operações sobre arquivos, segmentos de memória, processador e outros recursos de um sistema.

De acordo com Silberschatz, Galvin e Gagne (2015, p. 18), "Proteção refere-se a um mecanismo para o controle do acesso de programas, processos ou usuários aos recursos definidos por um sistema de computação". Através desse mecanismo é possível especificar os controles a serem impostos com um meio de exigi-los.

2.4.1 Objetivos de proteção

Como diversos programas ocupam a memória de trabalho ao mesmo tempo, o usuário deve possuir uma área reservada onde seus dados e programas estão armazenados. O sistema operacional implementa mecanismos de proteção com objetivo de preservar essas informações de maneira reservada, de modo que, se um programa tentar o acesso a uma posição de memória fora de sua faixa, ocorrerá um erro no sistema operacional, indicando a violação de acesso.

Para que diferentes programas possam compartilhar uma mesma área de memória, o sistema operacional deve disponibilizar mecanismos para que ocorra comunicação sincronizada e controlada entre eles, a fim de evitar problemas de inconsistência. Analogamente, um mesmo disco rígido pode armazenar arquivos de diferentes usuários. Novamente, o sistema operacional deve garantir a integridade e a confidencialidade dos dados, permitindo que dois ou mais usuários tenham acesso simultâneo ao mesmo arquivo.

À medida que os sistemas de computação foram se tornando mais sofisticados e difusos em suas aplicações, a necessidade de proteger sua integridade também cresceu. A proteção foi concebida originalmente como um complemento dos sistemas operacionais com multiprogramação para que usuários não confiáveis pudessem compartilhar seguramente um espaço lógico de nomes comum, tal como um diretório de arquivos, ou um espaço físico de nomes comum, tal qual a memória. Os conceitos modernos de proteção evoluíram para aumentar a confiabilidade de qualquer sistema complexo que faça uso de recursos compartilhados.

O sistema operacional precisa de proteção por várias razões. A mais óbvia é a necessidade de impedir a violação maldosa e intencional de uma restrição de acesso por um usuário. De forma mais geral, a sua importância é a vontade de assegurar que cada componente de programa ativo em um sistema use os recursos do sistema somente de forma consistente com as políticas estabelecidas. Esse requisito é fundamental para um sistema confiável, posto que a proteção contribui com a melhoria da confiabilidade, ao detectar erros latentes nas interfaces entre subsistemas componentes. Com a detecção precoce de tais erros, é possível impedir a contaminação de um subsistema saudável por um subsistema defeituoso.

Além disso, um recurso desprotegido não pode se defender contra a utilização (ou má utilização) de um usuário não autorizado ou incompetente. Um sistema orientado à proteção fornece meios para diferenciar o uso autorizado do não autorizado. O papel da proteção em um sistema de computação é fornecer um mecanismo para a imposição das políticas que governam o uso de recursos. Essas políticas podem ser estabelecidas de várias maneiras. Algumas são fixadas no projeto do sistema, enquanto outras são formuladas pelo gerenciamento de um sistema. Há ainda aquelas definidas pelos usuários individuais para a proteção de seus próprios arquivos e programas.

Um sistema de proteção deve ter flexibilidade para impor uma variedade de políticas. As políticas de uso de recursos podem ser alteradas por aplicação e mudar com o tempo. Por essa razão, proteção não é mais preocupação apenas do projetista de um sistema operacional; o programador de aplicações também precisa usar mecanismos para proteger os recursos criados e suportados em um subsistema de aplicação contra a má utilização.

Lembrete

Mecanismos são diferentes de políticas. Os mecanismos determinam como algo será feito; as políticas decidem o que será feito. A separação entre ambos é importante para a flexibilidade. As políticas podem mudar de um local para outro ou de tempos em tempos. No pior caso, cada modificação na política demandaria uma transformação no mecanismo subjacente. O uso de mecanismos gerais habilita-nos a evitar tal situação.

2.4.2 Princípios de proteção

Frequentemente, um princípio geral pode ser usado em todo um projeto, como o projeto de um sistema operacional. Seguir esse princípio simplifica decisões de projeto e mantém o sistema consistente e fácil de entender. Há por exemplo o princípio do privilégio mínimo, que determina que programas, usuários e até sistemas recebam privilégios apenas suficientes para a execução de suas tarefas.

Considere a analogia entre um guarda de segurança e uma chave mestra. Se essa chave permitir que o guarda entre apenas nas áreas públicas que ele protege, então a sua má utilização resultará em um dano mínimo. Se, no entanto, ela possibilitar o acesso a todas as áreas, o prejuízo decorrente de sua perda, roubo, má utilização, cópia ou outro tipo de comprometimento será muito maior.

Um sistema operacional que siga o princípio do privilégio mínimo implementa seus recursos, programas, chamadas de sistema e estruturas de dados de modo que a falha ou comprometimento de um componente cause danos mínimos.

Daemon é um processo executado em segundo plano pelo sistema operacional que não está sob o controle do usuário. O estouro de um buffer em daemon pode causar a falha do processo, por exemplo, mas não deve permitir a execução do código da pilha do processo daemon que habilitaria um usuário remoto a obter privilégios máximos e acessar o sistema inteiro.

Adicionalmente, um sistema operacional fornece chamadas de sistema e serviços que possibilitam a implementação de aplicações com controles de acesso refinados. Outra função do sistema operacional é o fornecimento de mecanismos para habilitar privilégios quando esses são necessários e desabilitá-los quando não mais o são. Outra vantagem é a criação de trilhas de auditoria para todo acesso a funções privilegiadas.

A trilha de auditoria permite que o programador, o administrador do sistema ou o oficial responsável pelo cumprimento da lei rastreiem todas as atividades de proteção e segurança no sistema. O gerenciamento de usuários com o princípio do privilégio mínimo requer a criação de uma conta separada para cada pessoa, apenas com os privilégios de que o usuário precisa. Um operador que necessite montar fitas e fazer backup de arquivos no sistema tem acesso apenas aos comandos e arquivos para executar a tarefa. Alguns sistemas implementam o controle de acesso baseado em papéis (RBAC – *role-based access control*) para fornecer essa funcionalidade. Computadores implementados em uma instalação sob o princípio do privilégio mínimo podem ser limitados à execução de serviços específicos, ao acesso a determinados hospedeiros remotos por meio de serviços característicos e à realização dessas tarefas durante certos períodos. Normalmente, essas restrições são implementadas por meio da habilitação ou desabilitação de cada serviço e de listas de controle de acesso.

O princípio do privilégio mínimo pode ajudar a produzir um ambiente de computação mais seguro. Infelizmente, isso não costuma ocorrer. Por exemplo, o Windows 2000 possuía um esquema de proteção complexo em seu núcleo; mesmo assim, apresentava muitas falhas de segurança. Em comparação, o Solaris é considerado relativamente seguro, ainda que uma variante do Unix tenha sido historicamente projetada com pouca atenção à proteção. Uma razão para a diferença pode ser o fato de que o

Windows 2000 tem mais linhas de código e serviços do que o Solaris e, portanto, mais a guardar e proteger. Outra razão poderia ser que o esquema de proteção no Windows 2000 é incompleto ou protege os aspectos errados do sistema operacional, deixando outras áreas vulneráveis.

2.4.3 Domínio de proteção

Um sistema de computação é uma coleção de processos e objetos. Com objetos queremos nos referir tanto a objetos de hardware (tais como a unidade central de processamento ou CPU, segmentos de memória, impressoras, discos e drives de fita) quanto a objetos de software (tais como arquivos, programas e semáforos). Cada um deles tem um nome exclusivo no sistema e pode ser acessado só por operações bem definidas e significativas. Objetos são, essencialmente, tipos abstratos de dados. As operações possíveis podem depender do objeto.

Por exemplo, em uma CPU, podemos apenas executar. Segmentos de memória podem ser lidos e gravados, enquanto um CD-ROM ou DVD-ROM pode somente ser lido. Drives de fita podem ser lidos, gravados e rebobinados. Arquivos de dados podem ser criados, abertos, lidos, gravados, fechados e excluídos; arquivos de programas podem ser lidos, gravados, executados e excluídos.

Um processo deve ter permissão para acessar apenas os recursos para os quais ele tenha autorização. Além disso, a qualquer momento, um processo deve ser capaz de acessar somente os recursos de que precisa para executar sua tarefa. Esse segundo requisito, normalmente referenciado como princípio conhecer-o-necessário, é útil ao limitar o nível de dano que um processo incorreto pode causar ao sistema. Por exemplo, quando o processo p invoca o procedimento A( ), ele deve ser autorizado a acessar apenas suas próprias variáveis e os parâmetros formais passados a ele; não deve ser capaz de acessar todas as variáveis do processo p.

Da mesma forma, considere o caso em que o processo p invoca um compilador para um arquivo específico. Ele não deve ser capaz de acessar arquivos arbitrariamente, mas precisa acessar apenas um subconjunto bem definido de arquivos (tais como o arquivo-fonte, o arquivo de listagem, e assim por diante) relacionados com aquele a ser compilado. Inversamente, o compilador pode ter arquivos privados usados para fins de contabilização ou otimização que o processo p não deve ser capaz de acessar.

O princípio conhecer-o-necessário é semelhante ao princípio do privilégio mínimo, porque os objetivos de proteção destinam-se a minimizar os riscos de possíveis violações da segurança.

2.4.4 Estrutura de domínio

Para facilitar o esquema que acabamos de descrever, um processo opera em um domínio de proteção, que especifica os recursos que ele pode acessar. Cada domínio define um conjunto de objetos e os tipos de operações que podem ser invocadas sobre cada item. A capacidade de executar uma operação sobre um objeto é o direito de acesso. Um domínio é um conjunto de direitos de acesso, cada um deles sendo um par ordenado. Por exemplo, se o domínio D tem o direito de acesso, então um processo, sendo executado

no domínio D, tanto pode ler quanto gravar o arquivo F. Ele não pode, no entanto, executar qualquer outra operação sobre esse objeto. Domínios podem compartilhar direitos de acesso.

Por exemplo, na figura a seguir, existem três domínios: D_1, D_2 e D_3. O direito de acesso é compartilhado por D_2 e D_3, implicando que um processo em execução em um desses dois domínios pode imprimir o objeto O_4. Observe que um processo deve estar sendo executado no domínio D_1 para ler e gravar o objeto O_1, enquanto apenas processos no domínio D_3 podem executar o objeto O1. A associação entre um processo e um domínio pode ser estática se o conjunto de recursos disponíveis para o processo for fixo durante todo o tempo de vida do processo, ou dinâmica. Como era de se esperar, o estabelecimento de domínios de proteção dinâmicos é mais complicado do que aqueles de domínios de proteção estáticos.

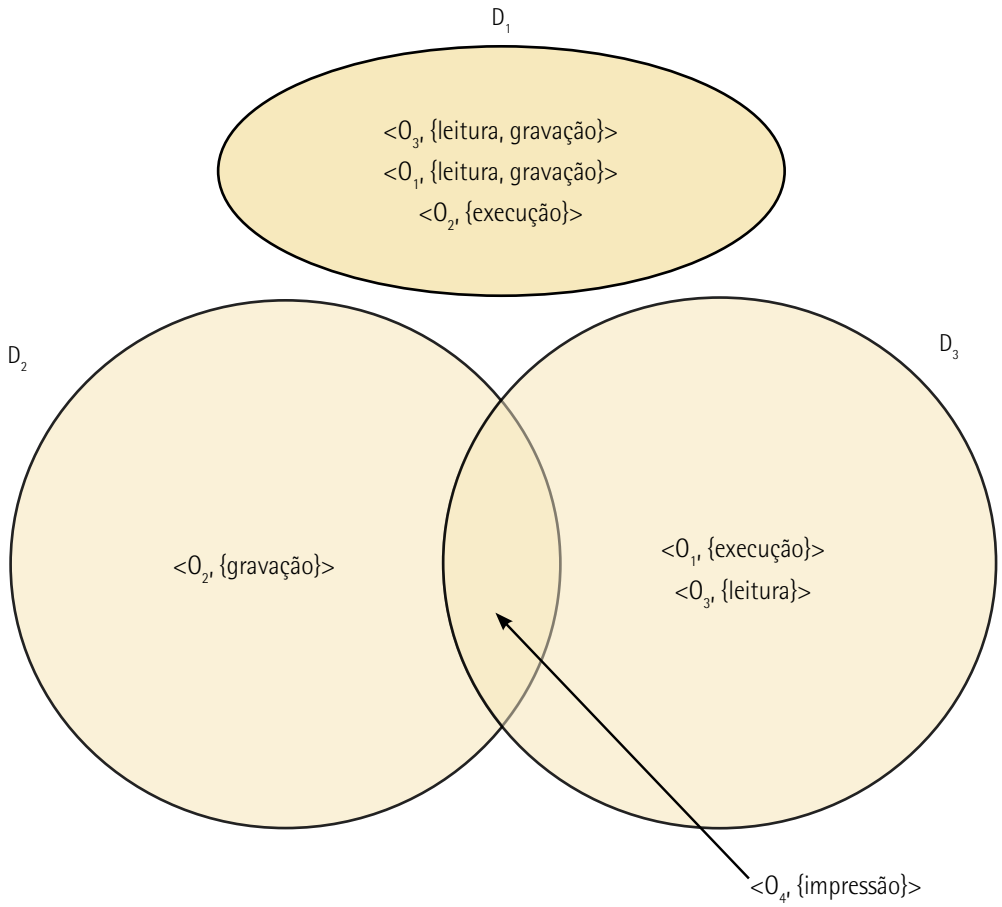


Figura 19 – Sistema com três domínios de proteção

Adaptada de: Silberschatz, Galvin e Gagne (2015, p. 344).

Se a associação entre processos e domínios é fixa e queremos aderir ao princípio conhecer-o-necessário, então deve estar disponível um mecanismo para alterar o conteúdo de um domínio. Isso provém do fato de que um processo pode ser executado em duas fases diferentes, por exemplo, precisar de acesso de leitura em uma fase e de acesso de gravação em outra. Se um domínio for estático, devemos defini-lo para que inclua tanto acesso de leitura quanto de gravação.

No entanto, esse esquema fornece mais direitos do que o necessário em cada uma das duas fases, já que temos acesso de leitura quando precisamos apenas de acesso de gravação, e vice-versa. Assim, o princípio conhecer-o-necessário é violado. Devemos permitir que o conteúdo de um domínio seja modificado para que ele sempre reflita os direitos de acesso mínimos necessários. Se a associação for dinâmica, disponibilizará um mecanismo para permitir a permuta de domínio, habilitando o processo a permutar de um domínio para outro. Também é desejado alterar o conteúdo de um domínio. Se não pudermos fazê-lo, obteremos o mesmo efeito ao criar um novo domínio com o conteúdo alterado, permutando para ele quando aspirarmos mudar o conteúdo do domínio.

Um domínio pode ser definido de várias maneiras:

- **Cada usuário pode ser um domínio**: o conjunto de objetos que pode ser acessado depende da identidade do usuário. A permuta de domínio ocorre quando o usuário muda — geralmente ele faz logoff, e outro login.

- **Cada processo pode ser um domínio**: o conjunto de objetos que pode ser acessado depende da identidade do processo. A permuta de domínio ocorre quando um processo envia uma mensagem a outro e, então, espera por uma resposta.

- **Cada procedimento pode ser um domínio**: o conjunto de objetos que pode ser acessado corresponde às variáveis locais definidas no procedimento. A permuta de domínio ocorre quando é feita uma chamada de procedimento.

Considere o modelo-padrão de execução do sistema operacional de modalidade dual (modalidade monitor-usuário). Quando um processo é executado em modalidade de monitor, ele pode executar instruções privilegiadas e, assim, obter o controle completo do sistema de computação. Por outro lado, quando executado em modalidade de usuário, pode invocar apenas instruções não privilegiadas. Consequentemente, ele pode ser executado somente no seu espaço de memória predefinido.

Essas duas modalidades protegem o sistema operacional (executando em domínio de monitor) dos processos de usuário (executando em domínio de usuário). Em um sistema operacional multiprogramado, dois domínios de proteção são insuficientes, já que os usuários também querem se proteger uns dos outros.

Exemplo de aplicação

No sistema operacional Unix, um domínio é associado a cada usuário. Assim, a permuta de domínio corresponde à alteração temporária da identificação do usuário. Essa mudança é feita através do sistema de arquivos. Uma identificação de proprietário é um bit de domínio, também conhecido como **bit setuid**, estando associado a cada arquivo. Quando o bit setuid está on (ligado) e um usuário executa esse arquivo, o userID é posicionado como o do proprietário do arquivo. Quando o bit está off (desligado), no entanto, o userID não se altera. Por exemplo, quando um usuário A (isto é, um usuário com userID = A) começa a executar um arquivo de propriedade de B, cujo bit de domínio associado está off, o userID é posicionado como A. Quando o bit setuid está on, o userID é posicionado como o do proprietário do arquivo: B. Quando o processo é encerrado, essa mudança temporária do userID termina.

São empregados outros métodos para alterar domínios nos sistemas operacionais em que userIDs forem usados na definição de domínios, porque quase todos os sistemas precisam fornecer um mecanismo assim. Esse mecanismo é usado quando um recurso, de outra forma privilegiado, precisa ser disponibilizado para a população geral de usuários. Por exemplo, pode ser desejável permitir que os indivíduos acessem uma rede sem deixá-los escrever seus próprios programas de rede. Nesse caso, em um sistema Unix, o bit setuid em um programa de rede seria posicionado, fazendo com que o userID seja alterado quando o programa for executado. O userID seria modificado para o de um usuário com privilégio de acesso à rede (tal como root, o userID mais poderoso). Um problema desse método é que, se uma pessoa conseguir criar um arquivo com userID igual a root e com seu bit setuid on, ela poderá se tornar o usuário root e fazer o que quiser no sistema.

Uma alternativa a esse método, usada em alguns sistemas operacionais, é a inserção de programas privilegiados em um diretório especial. O sistema operacional é projetado para alterar o userID de qualquer programa executado a partir desse diretório para o equivalente a root ou o userID do proprietário do diretório. Isso elimina um problema de segurança que ocorre quando intrusos criam programas para manipular o recurso setuid e os ocultam no sistema para uso posterior (utilizando nomes obscuros de arquivos ou diretórios). Esse método é menos flexível do que o usado no Unix, no entanto.

Ainda mais restritivos e, portanto, mais protegidos são os sistemas que simplesmente não permitem uma mudança do userID. Nesses casos, técnicas especiais devem ser usadas para possibilitar que os usuários acessem recursos privilegiados. Por exemplo, um **processo daemon** pode ser iniciado em tempo de inicialização e executado com um userID especial. Os usuários executam, então, um programa separado que envia solicitações a esse processo sempre que eles precisem usar o recurso. Esse método é usado pelo sistema operacional TOPS-20.

Em todos esses sistemas, deve-se ter muito cuidado ao escrever programas privilegiados. Qualquer descuido pode resultar na total falta de proteção no sistema. Geralmente, tais programas são os primeiros a serem atacados por pessoas que tentam invadir um sistema. Infelizmente, os invasores costumam ter sucesso. Por exemplo, a segurança tem sido violada em muitos sistemas Unix por causa do recurso setuid.

2.5 Segurança

Como apresentado anteriormente, a proteção é um problema estritamente **interno** que busca responder à seguinte questão: Como fornecer acesso controlado a programas e dados armazenados em um sistema de computação?

De acordo com Silva (2020), a segurança requer não apenas um sistema de proteção adequado, mas a consideração do ambiente ***externo*** no qual opera. Um sistema de proteção é ineficaz se a autenticação do usuário é comprometida ou se um programa é executado por usuário não autorizado.

Conseguimos considerar que a segurança de um sistema de informações pode começar pela segurança nos sistemas operacionais, mas existem muitos outros itens importantes para se alcançar um bom nível de segurança, como, por exemplo, os usuários internos da empresa, atacantes externos, entre outros.

Os recursos do computador devem ser protegidos contra acesso não autorizado, destruição ou alteração maliciosa e introdução acidental de inconsistências. Tais recursos incluem as informações armazenadas no sistema (tanto dados quanto código), assim como a CPU, memória, discos, fitas e rede que constituem o computador.

2.5.1 O problema da segurança

Em muitas aplicações, garantir a segurança do sistema de computação vale um esforço considerável. Grandes sistemas comerciais contendo folha de pagamentos ou outros dados financeiros são os alvos preferidos. Aqueles que contêm dados relacionados com operações empresariais podem ser do interesse de competidores inescrupulosos. Além disso, a perda de tais dados, acidental ou forjada, pode prejudicar seriamente a capacidade de funcionamento da empresa.

Esses mecanismos de proteção dos sistemas operacionais funcionam bem somente na medida em que os usuários se adaptem ao uso e ao acesso pretendidos para eles. Dizemos que um sistema é **seguro** quando seus recursos são usados e acessados como esperado sob todas as circunstâncias. Infelizmente, a segurança total não pode ser atingida, mas mesmo assim devemos possuir mecanismos que tornem as brechas de segurança uma ocorrência rara e não a norma.

Violações ou a má utilização da segurança do sistema podem ser categorizadas como intencionais (maliciosas) ou acidentais. É mais fácil se proteger contra a má utilização acidental do que contra a maliciosa. Geralmente, os mecanismos de proteção são a base da proteção contra acidentes. Constam a seguir vários tipos de violações acidentais e maliciosas da segurança. Devemos observar que, em nossa discussão sobre segurança, usamos os termos **invasor** e **cracker** para quem tenta violar a segurança. Além disso, **ameaça** é a possibilidade de uma violação de segurança, tal como a descoberta de uma vulnerabilidade, enquanto **ataque** é a tentativa de violar a segurança.

- **Brecha de sigilo**: envolve a leitura não autorizada de dados (ou roubo de informações). Normalmente, uma brecha de sigilo é o objetivo de um invasor. A captura de dados secretos de um sistema ou fluxo de dados, tais como informações de cartões de crédito ou informações de credenciais para roubo de identidade, pode resultar diretamente em dinheiro para o intruso.

- **Brecha de integridade**: envolve a modificação não autorizada de dados. Esses ataques podem, por exemplo, resultar na transferência de responsabilidade para terceiros inocentes ou na modificação do código-fonte de uma aplicação comercial importante.

- **Brecha de disponibilidade**: envolve a destruição não autorizada de dados. Alguns crackers preferem provocar destruição e ganhar status ou se vangloriar de direitos a obter ganhos financeiros. A desfiguração de sites é um exemplo comum desse tipo de brecha de segurança.

- **Roubo de serviço**: envolve o uso não autorizado de recursos. Por exemplo, um invasor (ou programa invasor) pode instalar um daemon em um sistema que atue como servidor de arquivos.

- **Recusa de serviço**: envolve o impedimento do uso legítimo do sistema. Ataques de recusa de serviço (DoS – denial-of-service) são, algumas vezes, acidentais. O verme original da internet transformou-se em um ataque DoS quando um bug não conseguiu retardar sua rápida disseminação.

Os agressores utilizam diversos métodos-padrão em suas tentativas de violar a segurança. O mais comum é o **mascaramento**, em que um participante de uma comunicação finge ser alguém que não é (outro hospedeiro ou outra pessoa). Por meio do mascaramento, os agressores violam a **autenticação**, a precisão da identidade; eles podem então obter acesso que, normalmente, não receberiam, ou aumentar seus privilégios – obter privilégios que, geralmente, não lhes seriam atribuídos. Outro ataque comum é a reexecução de uma troca de dados capturada. Um **ataque de reexecução** consiste na repetição maliciosa ou fraudulenta de uma transmissão de dados válida. Às vezes, a reexecução compõe o ataque inteiro – por exemplo, na repetição de uma solicitação para transferência de dinheiro. Mas, frequentemente, ela é feita com a **modificação de mensagens**, novamente para aumentar privilégios. Considere o dano que poderia ser causado se uma solicitação de autenticação tivesse as informações de um usuário legítimo substituídas pelas de um usuário não autorizado. Outro tipo de ataque é o **ataque do intermediário**, em que um invasor se instala no fluxo de dados de uma comunicação, mascarando-se como o emissor para o receptor, e vice-versa. Em uma comunicação de rede, um ataque do intermediário pode ser precedido de um **sequestro de sessão**, em que uma sessão de comunicação ativa é interceptada.

Deve-se atentar que a proteção absoluta do sistema contra abuso malicioso não é possível, mas o custo para o infrator pode se tornar suficientemente alto para deter a maioria dos invasores. Em alguns casos, tal como em um ataque de recusa de serviço, é preferível impedir o ataque, mas já é suficiente detectá-lo para que medidas defensivas possam ser tomadas.

Para proteger um sistema, devemos tomar medidas de segurança em quatro níveis:

- **Físico**: os locais onde estão instalados os sistemas computacionais e os equipamentos de rede devem ser fisicamente protegidos contra a entrada forçada ou furtiva de intrusos. Tanto as salas das máquinas quanto os terminais ou estações de trabalho que têm acesso remoto às máquinas devem ser protegidos.

- **Humano**: a autorização deve ser cuidadosa para assegurar que apenas usuários apropriados tenham acesso ao sistema. Até mesmo usuários autorizados, no entanto, podem ser encorajados a deixar outras pessoas utilizarem seu acesso (mediante suborno, por exemplo). Eles também podem ser levados a permitir o acesso pela engenharia social, como o phishing. Nesse caso, um e-mail ou página da web de aparência legítima engana um usuário, levando-o a inserir informações confidenciais. Outra técnica é o dumpster diving, um termo geral para a tentativa de coletar informações de modo a obter acesso não autorizado ao computador (examinando o conteúdo de lixeiras, bisbilhotando agendas telefônicas ou verificando lembretes contendo senhas, por exemplo). Esses problemas de segurança são questões pessoais e de gerenciamento, e não relacionados com os sistemas operacionais.

- **Sistema operacional**: o sistema deve proteger a si próprio contra brechas de segurança acidentais ou propositais. Um processo fora de controle constituiria um ataque acidental de recusa de serviço. Uma consulta a um serviço revelaria senhas. Um estouro de pilha permitiria o acionamento de um processo não autorizado. A lista de brechas possíveis é imensurável.

- **Rede**: muitos dados nos sistemas modernos viajam por linhas privadas dedicadas, linhas compartilhadas como a internet, conexões sem fio ou linhas dial-up. A interceptação desses dados seria tão danosa quanto uma invasão em um computador, e a interrupção de comunicações constituiria um ataque remoto de recusa de serviço, diminuindo o uso do sistema e a confiança dos usuários.

Deve-se sustentar a segurança nos dois primeiros níveis visando garantir segurança do sistema operacional. Uma vulnerabilidade em um nível alto de segurança (físico ou humano) permite que medidas de segurança estritamente de baixo nível (sistema operacional) sejam burladas. Portanto, o antigo adágio de que uma corrente é tão forte quanto seu elo mais fraco é particularmente verdadeiro quando se trata da segurança de sistemas. Todos esses aspectos devem ser abordados para que a segurança seja mantida.

Além disso, o sistema deve fornecer proteção para permitir a implementação de recursos de segurança. Sem a capacidade de autorizar usuários e processos, controlar seu acesso e registrar suas atividades, seria impossível para um sistema operacional implementar medidas de segurança, ou ser executado de forma segura. Recursos de proteção de hardware são necessários para suportar um esquema geral de proteção. Por exemplo, um sistema sem proteção de memória não pode ser seguro. Novos recursos de hardware permitem que os sistemas sejam mais seguros, como veremos posteriormente.

À medida que invasores exploram vulnerabilidades de segurança, medidas defensivas são criadas e implantadas. Isso faz com que eles se tornem mais sofisticados em seus ataques. Por exemplo, incidentes recentes de segurança incluem o uso de spyware para fornecer um canal de introdução de spam por intermédio de sistemas inocentes. São necessárias cada vez mais ferramentas de segurança para bloquear o número crescente de técnicas e atividades de invasores.

A segurança nos níveis físico e humano, embora importante, está em sua maior parte fora do escopo deste livro-texto. A segurança entre sistemas operacionais é implementada de várias maneiras, que vão desde o uso de senhas de autenticação até a proteção contra vírus e a detecção de invasões. Começamos explorando as ameaças à segurança.

Uma pesquisa do Instituto Nacional de Padrões e Tecnologias (NIST *apud* DEMARTINI, 2020), órgão do governo dos Estados Unidos, totalizou a quantidade de brechas e vulnerabilidades encontradas nas plataformas em si e não nos malwares disponíveis. Compilada a partir de dados do Banco de Dados Nacional de Vulnerabilidades (NVD) de 1999 a 2019, a distribuição Debian apresentou 3.067 vulnerabilidades a serem corrigidas por seus desenvolvedores nos últimos 10 anos. O android ficou em segundo lugar, com 2.563, no que os especialistas consideraram um problema bem mais grave devido à popularidade da plataforma e o perfil de seus usuários, conforme mostrado a seguir.

Tabela 1 – Quantidade de vulnerabilidades por sistema operacional nos períodos de 1999 a 2019 e no ano de 2019

1999-2019		2019	
Debian Linux	3.067	Android	414
Android	2.563	Debian Linux	360
Linux kernel	2.357	Windows Server 2016	357
Mac OS X	2.212	Windows 10	357
Ubuntu	2.007	Windows Server 2019	351
Mozilla Firefox	1.873	Adobe Acrobat Reader DC	342
Google Chrome	1.858	Adobe Acrobat DC	342
iPhone iOS	1.655	cPanel	321
Windows Server 2008	1.421	Windows 7	250
Windows 7	1.283	Windows Server 2008	248
Adobe Acrobat Reader DC	1.182	Windows Server 2012	246
Adobe Acrobat DC	1.182	Windows 8.1	242
Windows 10	1.111	Windows RT 8.1	235
Adobe Flash Player	1.078	Ubuntu	190
Windows Server 2012	1.050	Fedora	184

Fonte: NIST *apud* Demartini (2020).

De acordo com Tanenbaum e Woodhull (2008), um exemplo de situação para segurança pode ser a intromissão casual feita por usuários que não são técnicos, como no caso de muitas pessoas possuírem em suas mesas computadores pessoais conectados a um servidor de arquivos compartilhado e, sendo a natureza humana como ela é, algumas delas lerão o correio eletrônico e outros arquivos de outras pessoas se nenhuma barreira for colocada no caminho. A maioria dos sistemas Unix, por exemplo, tem o padrão de que todos os arquivos recentemente criados são legíveis publicamente.

Lembrete

Implementar um sistema com segurança total é um ideal inatingível, pois sempre pode ocorrer alguma situação imprevisível que comprometa a segurança, e a própria evolução tecnológica permite encontrar novas vulnerabilidades. No entanto, um parque computacional é considerado seguro quando acessado e utilizado sempre de acordo com o planejado.

2.5.2 Proteção contra vírus

De acordo com Silva (2023), vírus é um programa de computador ou parte dele, geralmente malicioso, que se propaga inserindo cópias dele mesmo. Para se multiplicar, depende da execução do programa ou arquivo hospedeiro, para se tornar ativo e continuar o processo de infecção. Dessa forma, os vírus podem causar destruição nos sistemas. O vírus é uma das categorias mais conhecidas de software malicioso ou malware.

Portanto, a proteção contra vírus e contra malware é uma preocupação de segurança importante. Programas antimalware costumam ser usados para fornecer essa proteção. Alguns deles são eficazes apenas contra vírus específicos conhecidos. Eles funcionam inspecionando todos os programas em um sistema em busca do padrão específico de instruções conhecido por compor o vírus. Quando encontram um padrão conhecido, removem as instruções, **desinfectando** o programa. Os programas antivírus podem ter catálogos com os milhares de vírus pelos quais eles procuram.

Tanto os vírus quanto os softwares antivírus continuam a se tornar mais sofisticados. Alguns vírus modificam a si mesmos à medida que infectam outros softwares para evitar a abordagem básica da comparação de padrões dos programas antivírus. Esses programas no entanto agora procuram por famílias de padrões em vez de um único padrão para identificar um vírus. Na verdade, alguns antivírus implementam uma variedade de algoritmos de detecção. Eles podem descomprimir vírus comprimidos antes de procurar por uma assinatura. Outros ainda buscam por anomalias em processos. Um processo abrindo um arquivo executável para gravação é suspeito, por exemplo, a menos que seja um compilador.

Outra técnica popular é a execução de um programa em **sandbox** ou uma **caixa de areia**, que é uma seção do sistema controlada ou emulada. O software antivírus analisa o comportamento do código na caixa de areia antes de deixá-lo ser executado sem monitoramento. Alguns antivírus também erguem um escudo completo em vez de apenas varrer arquivos em um sistema. Eles pesquisam setores de inicialização, memória, e-mails recebidos e enviados, arquivos quando são baixados, arquivos em dispositivos ou mídias removíveis, e assim por diante.

A melhor proteção contra os vírus de computador é a prevenção ou a prática da **computação segura**. Comprar softwares de código-fonte fechado com os fornecedores e evitar cópias livres ou piratas provenientes de fontes públicas ou troca de discos oferece a rota mais segura para evitar a infecção.

No entanto, até mesmo cópias novas de aplicações de software legítimas não estão imunes à infecção por vírus: em algumas situações, funcionários descontentes de uma empresa de software infectaram as cópias mestras de programas para causar danos econômicos à instituição. Para os vírus de macros, uma defesa é trocar documentos do Microsoft Word para um formato alternativo de arquivo chamado de **rich text format** (**RTF**). Diferentemente do formato nativo do Word, o RTF não inclui o recurso de anexação de macros.

Outra forma de defesa é evitar a abertura de qualquer anexo de e-mail enviado por usuários desconhecidos. Infelizmente, a história mostra que vulnerabilidades em e-mails aparecem tão rápido quanto são corrigidas. Por exemplo, em 2000, o vírus **love bug** foi amplamente propagado viajando em mensagens de e-mails que fingiam ser frases carinhosas enviadas por amigos dos receptores. Quando alguém abria o script em Visual Basic anexado, o vírus se propagava enviando a si mesmo aos primeiros endereços da lista de contatos do receptor. Felizmente, exceto pela obstrução de sistemas de e-mail e caixas de entrada dos usuários, ele era relativamente inofensivo. No entanto, desacreditou efetivamente a estratégia defensiva de abrir anexos apenas de pessoas

conhecidas do receptor. Um método de defesa mais eficaz é evitar abrir qualquer anexo de e-mail que contenha código executável. Algumas empresas já impõem isso como política removendo todos os anexos recebidos por e-mail.

Outra salvaguarda permite a detecção precoce, embora não impeça a infecção. Um usuário deve começar reformatando completamente o disco rígido, principalmente o setor de inicialização que, com frequência, é alvo de ataque viral. Apenas softwares seguros são carregados, e uma assinatura de cada programa é obtida por meio da computação segura de sínteses de mensagens. O nome de arquivo e a lista associada de sínteses de mensagens resultantes devem então ser mantidos livres de acesso não autorizado.

Periodicamente, ou sempre que um programa é executado, o sistema operacional computa novamente a assinatura e a compara com a assinatura na lista original; qualquer diferença serve como aviso de possível infecção. Essa técnica pode ser combinada com outras. Por exemplo, pode ser utilizada uma varredura antivírus de overhead alto, tal como uma caixa de areia; e, se um programa passar no teste, uma assinatura será criada para ele. Se as assinaturas coincidirem na próxima vez que o programa for executado, ele não precisará passar por uma varredura antivírus novamente.

2.5.3 Exemplos de furos de segurança famosos

Mesmo em sistemas operacionais mais antigos existiam preocupações em relação à segurança dos sistemas operacionais. Karger e Schell (1974) realizaram uma avaliação da segurança do sistema operacional Multics (**Mult**iplexed **I**nformation and **C**omputing **S**ervice). Ele foi criado em 1964 em um projeto conjunto entre General Electric e Massachusetts Institute of Technology (MIT). Foi o primeiro sistema operacional de tempo compartilhado, possibilitando a multiprogramação. A conclusão da avaliação de segurança Multics era mais segura do que dos outros sistemas operacionais comerciais da época. Entretanto, não se poderia confiar no sistema para proteger contra um ataque deliberado. Adicionalmente, correções ou restrições do sistema não podem fornecer nenhuma melhoria significativa em proteção.

Ele até provia segurança adequada para tempo compartilhado ou time-sharing. Todavia, a segurança era inexistente para operações em batch. Por exemplo: era possível modificar um programa de uso comum para obter informações, como em um programa que imprime "login:" e coleciona senhas.

Em relação à segurança do Windows, Kropiwiec e Geus (2004, p. 3) afirmaram que:

> O Windows apresentava originalmente uma segurança bem inferior a essa, por utilizar como sistema de suporte o DOS (e consequentemente, o sistema de arquivos FAT16/32, que não provia nenhum tipo de controle de acesso), e só posteriormente o Windows veio a utilizar um sistema de arquivos que permitia o controle de acesso dos usuários, o NTFS.

Os mesmos autores consideraram um paradigma de que o usuário dos sistemas era um agente limitador da própria segurança; para eles, outro paradigma seria a existência de um superusuário, isto é, o administrador do sistema, e o terceiro paradigma consistiria no uso da cifragem como forma de garantir a segurança.

Desta forma, Kropiwiec e Geus (2004) afirmaram que pode ocorrer a quebra da segurança considerando-se os paradigmas mencionados: com relação ao primeiro paradigma, existem dois pontos principais a serem analisados. É comum o usuário negligenciar a configuração correta das permissões sobre seus arquivos, seja por desconhecimento de como fazê-lo, seja por descuido ou descaso.

Consequentemente, caso o domínio de outro usuário seja comprometido, e as permissões do primeiro usuário assim o permitirem, a partir do outro será possível obter acesso aos arquivos, ou até destruí-los. O segundo ponto está relacionado com o domínio associado às aplicações. Elas possuem acesso ao mesmo domínio do usuário que as executou, e, caso sejam comprometidas por um ataque bem-sucedido, o invasor ingressará no mesmo domínio do usuário. Esse ponto tem maior implicação quando com o segundo paradigma. Em se tratando de um ataque a aplicações de usuários comuns, apenas os arquivos aos quais eles possuem acesso poderão ser copiados e/ou destruídos.

O risco maior encontra-se na possibilidade de que aplicações que necessitem de privilégios adicionais sejam comprometidas, pois, em função do segundo paradigma, uma vez que isso aconteça, todo o sistema estará comprometido. O problema, portanto, é a falta de granularidade na distribuição das operações privilegiadas do sistema operacional. Essa falta de granularidade decorre do fato de que todos os privilégios estão centralizados e isolados em um único domínio, o do superusuário, o que impossibilita restringir a aplicação apenas ao subconjunto mínimo de operações privilegiadas necessárias (KROPIWIEC; GEUS, 2004).

3 PROCESSOS E THREADS

Os sistemas computacionais atuais são capazes de desenvolver grande variedade de tarefas simultaneamente. Na sequência, serão apresentados conceitos de processos e threads. Um processo pode ser entendido como um programa em execução. Simplificadamente, escreve-se um arquivo de texto em uma linguagem de programação (código-fonte); quando se executa esse programa, ele se torna um processo que executa todas as tarefas mencionadas no código-fonte.

Os sistemas são compostos de um conjunto de processos do sistema operacional que executam código de sistema, e processos de usuário, os quais desempenham código de usuário, de forma que os processos são a unidade de trabalho do sistema operacional.

Além dos conceitos de processos, seus componentes e características, serão detalhadas as formas de comunicação entre processos e sistemas operacionais que suportam hardware com múltiplos processadores, como a maior parte dos dispositivos computacionais atuais.

O objetivo será o entendimento de como os sistemas operacionais realizam o gerenciamento de processos e threads visando garantir sua coexistência pacífica, cooperação e não colisão enquanto se ocupam de suas tarefas.

3.1 Conceito de processos

Na década de 1960, surgiu o conceito de sistemas operacionais multiprogramáveis e de tempo compartilhado ou time-sharing; nele se atende diversas tarefas dos usuários e se mantém informações a respeito de vários programas que estão sendo executados concorrentemente.

Neste sistema, o processador executa a tarefa durante um intervalo ou time-slice e, no instante seguinte, pode processar outra. A cada troca de tarefa é necessário salvar todas as informações daquela que foi interrompida para, quando for retornada, não lhe faltar qualquer informação para continuar seu processamento.

Segundo Machado e Maia (2013), pode-se definir um processo como conjunto necessário de informações para que o sistema operacional implemente a concorrência de programa, isto é, um programa em execução, abrangendo os dados que devem ser armazenados para continuação do processamento em período posterior.

O processo mantém todas essas informações para execução de um programa, como, por exemplo, conteúdo de registradores e espaço de memória. Ele ainda é o ambiente onde se executa um programa.

Nenhum programa é executado diretamente na memória principal, mas no processo, pois, nesse caso, o programa faria uso indiscriminado de qualquer área da memória principal, efetuando operações de E/S indevidas, que comprometem a integridade e a consistência dos dados.

Lembrete

Um programa por si só não é sinônimo de processo. Trata-se de uma entidade passiva, assim como um arquivo contendo uma lista de instruções armazenadas no disco, e normalmente é chamado de executável. Por sua vez, um processo é uma entidade ativa, com um contador de programa especificando a próxima instrução a ser realizada e um conjunto de recursos associados. Um programa se torna um processo no instante em que seu arquivo executável é carregado na memória principal (CÓRDOVA JUNIOR; LEDUR; MORAIS, 2019). Deitel, Deitel e Choffnes (2005) fazem a analogia de que programa é para o processo o mesmo que a partitura é para uma orquestra sinfônica.

Considere o cenário: usuários fazendo requisições simultaneamente, antivírus varrendo tudo que entra e sai, software de gerenciamento de redes monitorando e solicitando dados de desempenho do

servidor e diversos discos trabalhando em Raid (além de outras tarefas). É visível a necessidade de uma "**orquestração**" para todos esses processos.

De acordo com Maziero (2019), um processo pode ser entendido como uma unidade de contexto, isto é, um contêiner de recursos a serem utilizados por uma ou mais tarefas para sua execução. Constituem esses recursos: a área de memória como dados, código e pilha, as informações de contextos e descritores de recursos do núcleo como conexões de rede, arquivos abertos, entre outros.

Em função dos mecanismos de proteção providos pelo hardware, por exemplo, isolamento de áreas de memória, níveis de operação e chamadas de sistema, os processos são isolados entre si. Isso impossibilita que uma tarefa do processo p_a acesse um recurso atribuído ao processo p_b. A figura a seguir apresenta a visão do processo como unidade de contexto.

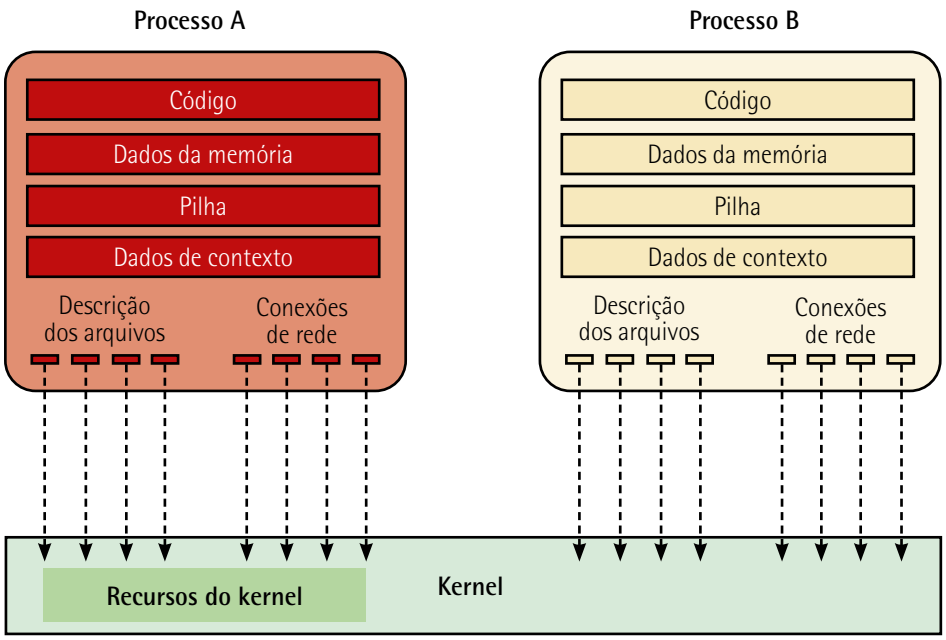


Figura 20 – Processo como um contêiner de recursos

Fonte: Maziero (2019, p. 55).

3.1.1 Componentes de um processo

Podemos entender um processo como o ambiente onde um programa é executado. Tal ambiente possui a quantidade de recursos do sistema que cada programa pode utilizar, como tempo de processador, espaço na memória principal e região em disco, além das informações sobre a sua própria execução. O resultado dessa execução pode variar, dependendo dos recursos disponibilizados para o programa.

A escassez de recursos pode impedir a execução bem-sucedida de um programa. Por exemplo, quando precisar utilizar uma área em disco acima de seu limite, o sistema operacional paralisará sua execução por falta de recursos disponíveis.

Podemos separar um processo em três partes: contexto de hardware, contexto de software e espaço de endereçamento. Juntas elas fornecem todas as informações necessárias à execução de um programa. A figura a seguir ilustra os componentes da estrutura do processo.

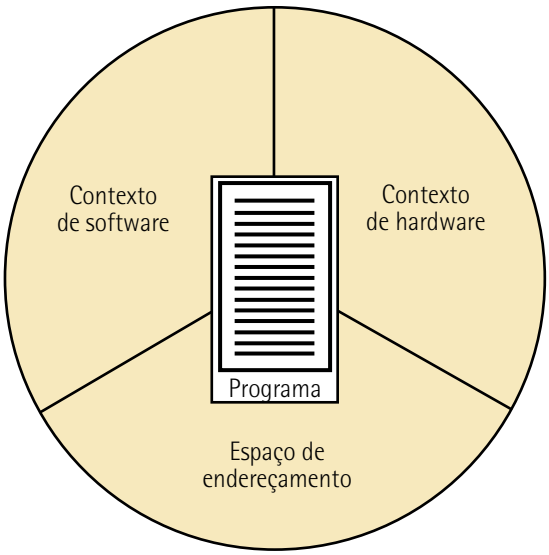


Figura 21 – Estrutura de um processo

Fonte: Machado e Maia (2013, p. 63).

No contexto de hardware, é armazenado o conteúdo dos registradores de uso gerais e específicos da CPU. São exemplos de registradores do processador: contador de programa ou *program counter* (PC), ponteiro da pilha ou *stack pointer* (SP) e bits de estado. O conteúdo deles é salvo durante a troca de contexto ou troca de processos para posterior recuperação.

O contexto de software é responsável por especificar características e limites dos recursos alocados ao processo. Por exemplo, o número máximo de arquivos abertos, a prioridade de execução, o tamanho de buffer de E/S, entre outros. A maior parte das informações localiza-se no chamado "arquivo de usuários", que é gerenciado pelo administrador do sistema operacional.

O contexto de software é composto de três grupos de informação: identificação, quotas e privilégio. No grupo de identificação, existem informações únicas para caracterizar cada processo e usuário, tais como:

- Identificação do processo ou *process identification* (PID)
- Identificação do usuário ou *user identification* (UID)

No grupo de quotas, são definidos limites de utilização de cada recurso para o processo. São seus exemplos: número máximo de arquivos abertos, tamanho máximo de alocação de memória, número de operações de E/S, buffer máximo para E/S, quantidade máxima de subprocessos, entre outros.

Em relação aos privilégios, eles apontam as ações permitidas ao processo. São divididos em três: privilégios que afetam o próprio processo, afetam demais processos e afetam o próprio sistema operacional (conta root).

O espaço de endereçamento do processo é a área da memória do processo em que o programa será executado, resultando em uma área isolada de memória para o processo.

No núcleo dos sistemas operacionais, existem os blocos de controle de processos ou *process control blocks* (PCBs), que armazenam diversas informações referentes aos processos ativos no ambiente, tais como:

- **Registradores da CPU**: incluem acumuladores, registradores de índice, ponteiros de pilhas e registradores de uso geral, além de qualquer informação do código de condição. Um dos registradores mais utilizados é o contador do programa, que indica o endereço da próxima instrução a ser executada para esse processo. Essas informações de estado devem ser salvas quando ocorre uma interrupção, para permitir que o processo seja retomado corretamente.

- **Informações de escalonamento da CPU**: incluem o número do processo (PID), a prioridade de um processo, ponteiros para filas de escalonamento e quaisquer outros parâmetros de escalonamento.

- **Informações de gerenciamento da memória**: incluem itens como o valor dos registradores base e limite e as tabelas de páginas, ou as tabelas de segmentos, dependendo do sistema de memória usado pelo sistema operacional.

- **Estado do processo**: condição de execução do processo, que pode ser novo, pronto, em execução, em espera, pronto e finalizado. Posteriormente eles serão detalhados.

3.1.2 Troca de contexto

Quando uma interrupção ocorre, é necessário que o sistema operacional salve o contexto corrente do processo em execução na CPU, para que possa retomá-lo. De forma geral, armazena o estado corrente da CPU, e então uma restauração do estado poderá retomar as operações.

A tarefa conhecida como **mudança de contexto** realiza a alocação da CPU a outro processo e requer a execução do armazenamento do estado do processo atual e a restauração do estado de um processo diferente. Durante o tempo gasto na mudança de contexto, o sistema não executa trabalho útil e representa perda no processamento. A velocidade típica para a mudança de contexto é a de alguns milissegundos e depende do suporte de hardware.

A figura a seguir apresenta as operações necessárias para a realização de uma troca de processos.

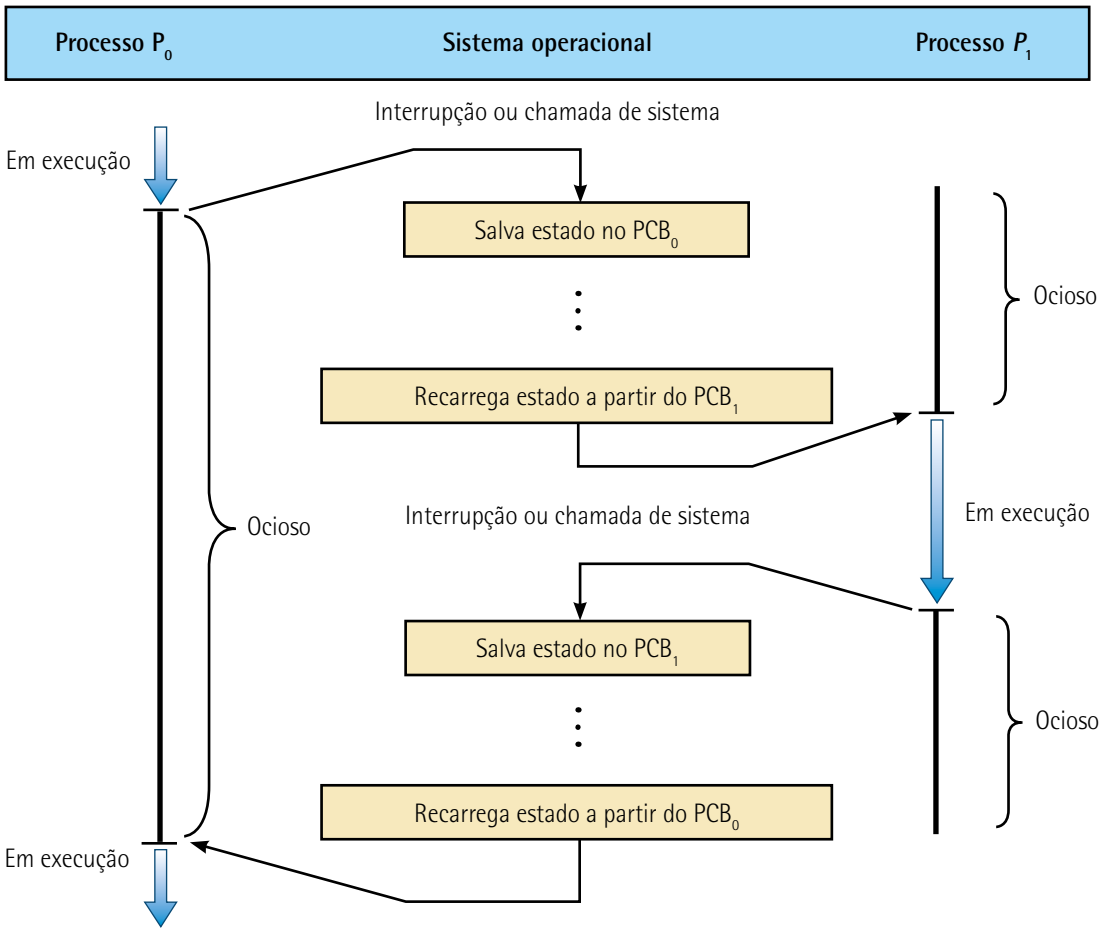


Figura 22 – Diagrama apresentando fluxo de atividades na CPU para troca de processos

Fonte: Silberschatz, Galvin e Gagne (2015, p. 30).

A mudança de contexto leva a um overhead de tempo, sendo considerada uma tarefa "cara". É preciso salvar as informações do processo cuja execução será interrompida na UCP, tais como os valores dos registradores e status do processo no seu PCB. Para o processo que será executado, será necessário realizar o carregamento das informações de contexto.

3.1.3 Estados de um processo

Como existe o compartilhamento do processador, os processos passam por diferentes estados ao longo de sua duração, devido aos eventos gerados pelo sistema operacional ou pelo próprio processo. Um processo pode estar em cinco diferentes estados: novo ou new, execução ou running, pronto ou ready, em espera ou wait e concluído ou exit.

Ele está no estado de **novo** ou no estado de criação quando o sistema operacional cria um novo PCB. Entretanto, ainda é possível colocá-lo na lista de processos do estado de pronto.

No estado de **execução**, o processo está efetivamente sendo processado pela UCP. Quando há apenas uma UCP, somente um processo fica em execução em um dado instante. Já quando há múltiplos processadores, é possível a execução de mais de um processo concomitantemente e um mesmo processo ser executado simultaneamente em mais de uma UCP, o que denominamos processamento paralelo.

O estado de **pronto** significa que o processo aguarda apenas para ser executado em uma fila de processos prontos, cuja ordem de execução é determinada pelo escalonador. O sistema operacional é responsável por determinar a ordem e os critérios pelos quais os processos passam.

Caso o processo precise aguardar um evento externo ou a liberação de um recurso para continuar seu processamento, ele estará no estado **em espera**, que também é conhecido com estado bloqueado ou blocked. O sistema pode esperar a finalização de uma leitura ou a gravação de um arquivo.

Um processo no estado **concluído** não poderá ter mais nenhum programa executado no seu contexto. Mas, o sistema operacional ainda armazena as informações desse processo para controle.

Observação

Lembre-se da diferença entre o processo pronto e o concluído. O estado de pronto significa que o processo está pronto para ser executado, e a escolha de qual processo irá ser escolhido é do sistema operacional. O processo concluído não será mais executado.

3.1.4 Mudança de estados de um processo

Um processo sofre alteração do seu estado em função de eventos voluntários, originados por ele mesmo, ou eventos involuntários, originados pelo sistema operacional. Podem ocorrer as seguintes transições de estado:

- **Novo → pronto**: acontece após o término do carregamento do processo na memória, com os dados e bibliotecas do processo, quando ele fica pronto para executar. O sistema o coloca em uma lista de processos de estado de pronto, onde aguarda sua escolha para ser executado.

- **Pronto → execução**: realizado quando o processo é escolhido para ser executado na lista de processos prontos. Cada sistema operacional possui critérios e algoritmos que o compõem na política de escalonamento do sistema.

- **Executando → espera**: nesta transição, o processo solicita acesso a recurso não disponível, como uma leitura de dados externos ou sincronização ou por eventos externos, tal como o sistema operacional suspender a execução do processo. O processo fica bloqueado até a liberação do recurso.

- **Espera → pronto**: quando um processo em espera tem sua solicitação de recurso atendida, ele voltará para lista de pronto e então aguardará ser escolhido para execução.

- **Execução → pronto**: esta transição é gerada pelo sistema normalmente pela finalização de tempo que o processo possui para execução. Na lista de pronto, o processo aguardará uma nova escolha para continuar seu processamento.

- **Execução → concluída**: ocorre com o encerramento do processamento ou quando ele é abortado devido a erro, como divisão por zero, acesso inválido à memória ou instrução ilegal.

A figura a seguir ilustra as transações de estados de processos.

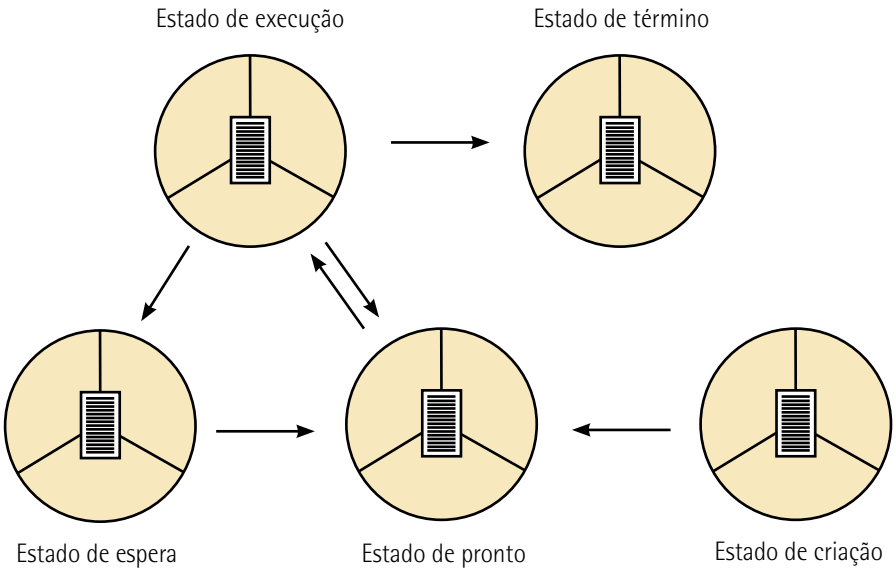


Figura 23 – Mudanças de estado processo

Fonte: Machado e Maia (2013, p. 70).

Lembrete

Quando ocorre a liberação do recurso necessário um processo no estado de espera deve obrigatoriamente ir para o estado de pronto antes de voltar a execução. Não existe a transição direta espera → execução.

3.1.5 Criação e finalização de um processo

A criação e finalização de processos ocorre constantemente no sistema operacional, e processos são criados e eliminados por diversas razões. A sua criação acontece quando o sistema operacional insere um novo PCB à sua estrutura e distribui um espaço de endereçamento na memória para utilização desse processo. Após criar o PCB, o sistema operacional identifica o processo e pode gerenciá-lo.

O momento em que são criados processos ocorre na inicialização do sistema operacional, quando são carregados inúmeros processos necessários para suportar tanto ele mesmo quanto as aplicações de usuário. Outra forma de inicialização é a requisição do usuário que pretende utilizar um software.

Outra forma é a execução de uma chamada de sistema para criação de um processo para ajudá-lo a fazer seu trabalho. No Unix, a chamada de sistema que cria processos é denominada fork, o processo-pai é aquele que o criou, e os processos-filhos são quaisquer que tenham sido criados pelo processo-pai.

Há ainda outro modo que se aplica apenas aos sistemas de lote encontrados nos computadores de grande porte. Neles, os usuários podem submeter tarefas de lote ou batch jobs para o sistema, normalmente de forma remota. Quando o sistema operacional confirma a existência de recursos suficientes para executar outra tarefa, ele cria um novo processo e executa a tarefa seguinte de sua fila de entrada.

De forma geral, um processo é encerrado quando finaliza a execução de sua última linha de comando e solicita ao sistema operacional que o exclua, usando a chamada de sistema exit ( ). Todos os recursos do processo, tais como memória física e virtual, arquivos abertos e buffers de E/S, são desalocados pelo sistema operacional, e o PCB é eliminado por ele. Por exemplo, quando um compilador finaliza a compilação de um programa recebido, ele realiza uma chamada de sistema para comunicar ao sistema operacional que sua tarefa terminou.

Outra forma de finalizar um processo é o encerramento de outro processo por meio de uma chamada de sistema apropriada, por exemplo, TerminateProcess ( ) no Windows. O quadro a seguir apresenta situações de encerramento de processos.

Quadro 3 – Formas de encerramento de processos

Evento	É voluntário do usuário?
Encerramento normal	Sim
Encerramento por erro ou ausência de recursos no sistema	Sim
Encerramento por erro fatal	Não
Cancelado por terceiros	Não

3.1.6 Tipos de processo

Existem processos que são executados em primeiro plano, os chamados **processos foreground**, e neles existe interação entre o usuário e processo para sua execução. No caso, os canais de entrada e saída estão associados a um dispositivo com teclado, mouse e monitor, para possibilitar a interação com o usuário.

Outros processos são executados em segundo plano, conhecidos como **processos background**; eles não estão associados a usuários específicos, mas possuem função característica. Não existe interação direta com o usuário, pois os canais de E/S não estão associados a dispositivo de E/S interativo, mas normalmente a arquivos de E/S.

Vejamos um exemplo desse tipo de processo, que foi projetado para aceitar pedidos de páginas web contidas nesta máquina, sendo acionado quando recebe um pedido para ser atendido. Os processos que ficam em segundo plano para executar alguma atividade, como buscar páginas web, impressão, entre outros, também são denominados **daemons**. A figura a seguir representa processos foreground e background.

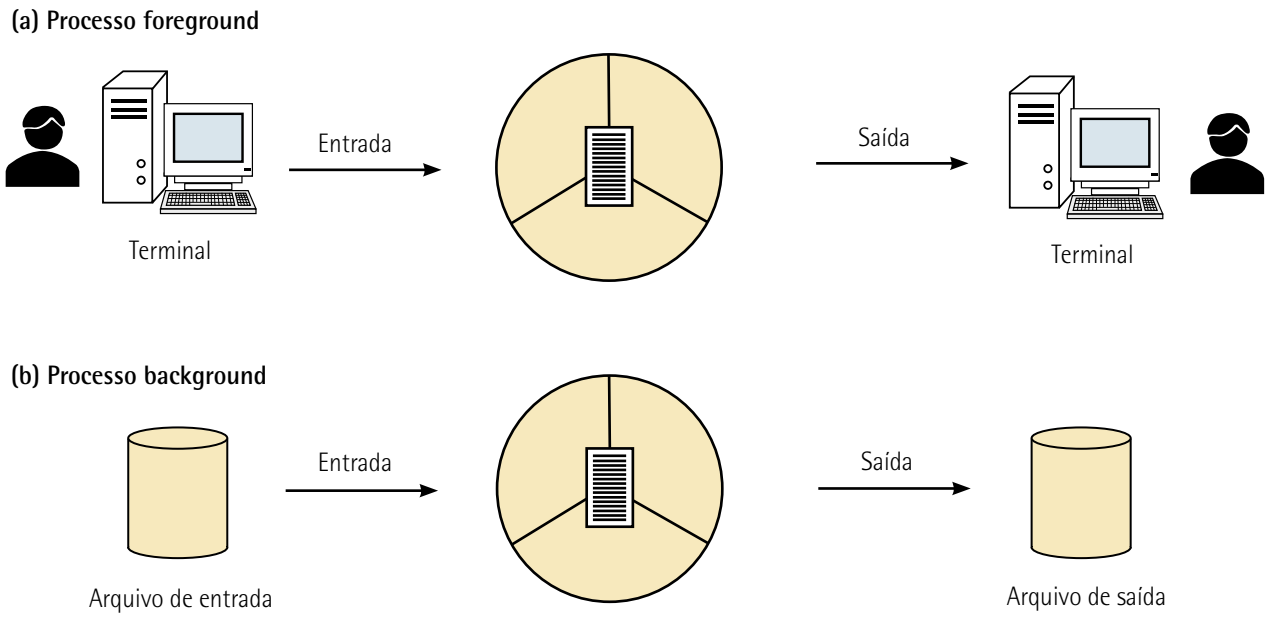


Figura 24 – Processos foreground e background

Fonte: Machado e Maia (2013, p. 72).

Outra forma de classificação de processos depende da utilização do processador e de dispositivos de E/S. Um processo é classificado como **CPU-bound** ou ligado à UCP quando fica a maior parte do tempo no estado de pronto ou de execução. Processos desse tipo são encontrados em aplicações cientificas que efetuam muitos cálculos e que realizam poucas operações de E/S.

Por outro lado, um processo é definido como **I/O-bound**, ligado à E/S, quando permanece a maior parte do tempo no estado de espera. Isto ocorre porque ele realiza uma elevada quantidade de operações de E/S. Esse tipo de processo é muito frequente em aplicações comerciais, que estão baseadas em leitura, processamento e gravação.

3.2 Threads

O conceito de threads, desenvolvido em 1979 durante a implementação do sistema operacional Toth, introduziu o conceito de processos lightweight, em tradução literal peso leve, onde o espaço de endereçamento de um processo era compartilhado por vários programas. O desenvolvimento do Mach, na Universidade de Carnegie Mellon, no final dos anos 1980, caracterizou as diferenças entre conceito de processo e thread.

Em processos com múltiplas threads ou multithread, as threads de um mesmo processo compartilham o espaço de endereçamento e são linhas de execução que compõem o processo. Com isso, a comunicação entre threads não envolve mecanismos lentos de intercomunicação entre processos, de forma a melhorar o desempenho da aplicação.

A maioria das aplicações de software executadas em computadores modernos é multithread. Uma aplicação é implementada, tipicamente, como um processo separado com diversos threads de controle. Com a introdução de múltiplas threads ou multithread, possibilitou-se o projeto e a implementação de aplicações com concorrência de forma eficiente, dado que um processo pode ter partes diferentes do seu código sendo executadas concorrentemente, com um menor overhead do que utilizando múltiplos processos.

Lembrete

Overhead é o tempo necessário para mudança de contexto. Durante o tempo de troca de processos, o sistema não realiza trabalho útil.

Segundo Córdova Junior, Ledur e Moraes (2018), a thread de execução pode ser definida como a menor sequência de instruções programadas capaz de ser gerenciada independentemente por um escalonador, que é parte integrante do sistema operacional.

A implementação de threads e processos difere entre sistemas operacionais mas, na maioria dos casos, a thread é um componente de um processo. Várias threads podem existir em um processo, executando simultaneamente e compartilhando recursos, como a memória, por exemplo, enquanto diferentes processos não os compartilham. Em particular, as threads de um processo compartilham seu código executável e os valores de suas variáveis a qualquer momento (SILBERSCHATZ; GALVIN; GAGNE, 2015).

No ambiente multithread, o processo tem pelo menos uma thread de execução, mas compartilha o seu espaço de endereçamento com inúmeras outras. A figura a seguir apresenta um comparativo entre um processo tradicional com uma única thread e um processo multithread.

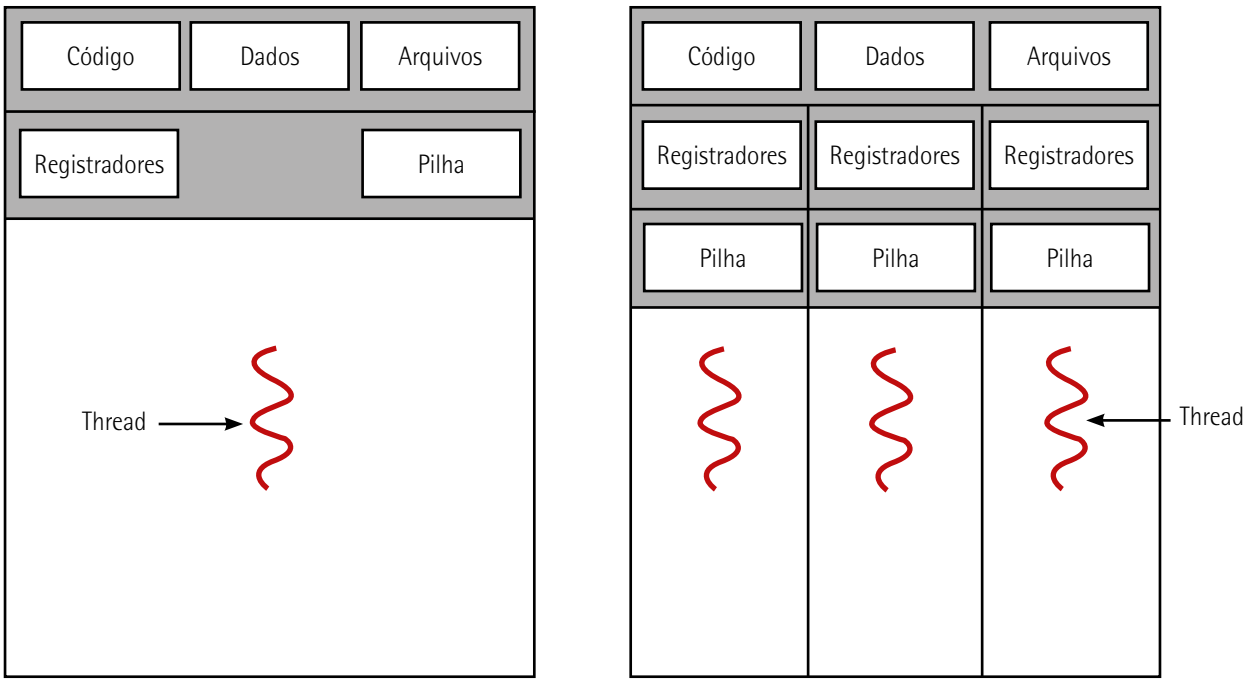


Figura 25 – Processo com uma única thread e com múltiplas threads

Fonte: Silberschatz, Galvin e Gagne (2015, p. 93).

3.2.1 Modelos de threads

O suporte às threads pode ser oferecido em função do nível de privilégio no sistema operacional; no nível do usuário existem as threads de usuário e no nível kernel as threads de kernel.

As threads de usuário são suportadas acima do kernel, gerenciadas sem o suporte do núcleo do sistema operacional e implementadas pela aplicação. A aplicação também deverá realizar tarefas como criação ou eliminação de threads, troca de mensagens entre threads e uma política de escalonamento e, visando este objetivo, deve existir uma biblioteca de rotinas do sistema operacional.

Uma grave limitação das threads de usuário é que o sistema operacional gerencia cada processo como se fosse composto de apenas uma thread. No momento em que uma thread realiza a chamada para uma rotina do sistema que a coloca em estado de espera ou rotina bloqueante, todo o processo fica no estado de espera, mesmo que existam outras threads prontas para execução. Visando tirar essa limitação, a biblioteca deve possuir rotinas que troquem as rotinas bloqueantes por outras que não resultem no bloqueio de uma thread, isto é, rotinas não bloqueantes.

Por sua vez, as threads de kernel são suportadas e gerenciadas diretamente pelo sistema operacional e incluem atividades internas do núcleo, como tarefas de gerência e sincronização. Quase a totalidade dos sistemas operacionais contemporâneos oferece suporte às threads de kernel, como Windows, Mac OS X, Solaris e Linux. O sistema operacional conhece as threads existentes e pode escaloná-las individualmente. Quando existem múltiplos processadores, as threads de um mesmo processo podem ser executadas simultaneamente.

É necessário estabelecer um relacionamento entre as threads de usuário e as threads de kernel. Há três maneiras comuns de estabelecer esse relacionamento: o modelo muitos-para-um, o modelo um-para-um e o modelo muitos-para-muitos.

O modelo muitos-para-um ou modelo N:1 realiza um mapeamento das muitas threads de nível de usuário para uma thread de kernel. O gerenciamento das threads é implementado pela biblioteca de threads no espaço do usuário. Contudo, o processo inteiro será bloqueado se uma thread fizer uma chamada de sistema bloqueadora. Um ponto contrário à utilização desse modelo é que apenas uma thread por vez pode acessar o kernel, e, com isso, muitas delas ficam impossibilitadas de executar em paralelo em sistemas multicore. Dessa forma, praticamente nenhum sistema operacional moderno utiliza esse modelo.

O modelo um-para-um mapeia cada thread de usuário para uma thread de kernel e também é representado por 1:1. Ele fornece mais concorrência do que o modelo muitos-para-um ao permitir que outra thread seja executada quando uma thread faz uma chamada de sistema bloqueadora. Permite também que múltiplas threads sejam executadas em paralelo em multiprocessadores. A única desvantagem desse modelo é que a criação de uma thread de usuário requer a criação da thread de kernel correspondente; já que o seu overhead de criação pode sobrecarregar o desempenho de uma aplicação, a maioria das implementações desse modelo restringe o número de threads suportadas pelo sistema. O Linux, com a família de sistemas operacionais Windows, implementa o modelo um-para-um.

Por fim, o modelo muitos-para-muitos, também conhecido como modelo threads N:M, implementa uma multiplexação de muitas threads de nível de usuário para um número menor ou igual de threads de kernel. O número de threads de kernel pode ser específico para determinada aplicação ou máquina, sendo que uma aplicação consegue receber mais threads de kernel em um ambiente multiprocessador do que em um ambiente monoprocessado.

A figura a seguir apresenta o modelo muitos-para-muitos.

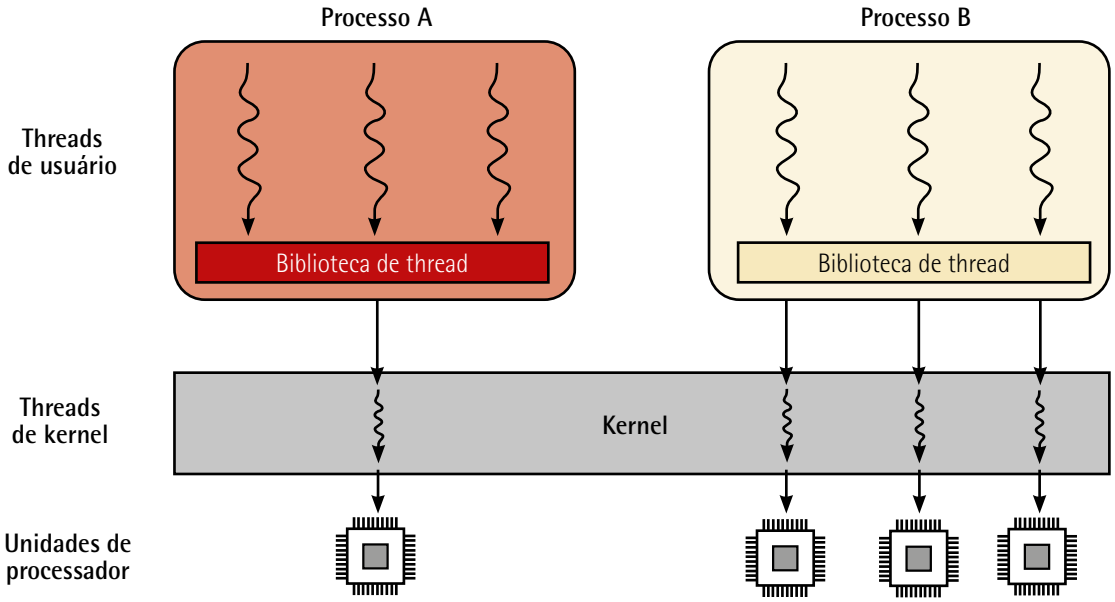


Figura 26 – Representação do modelo N:M

Fonte: Maziero (2019, p. 62).

O quadro a seguir compara as principais características dos três modelos e apresenta exemplos de sistemas operacionais que cada um deles aplica.

Quadro 4 – Comparação entre os três modelos de thread

Modelo	1:1	N:1	N:M
Característica	Cada thread do processo mapeado em uma thread de núcleo	N threads do processo mapeadas em uma thread de núcleo	N threads do processo mapeados em M<N threads de núcleo
Implementação	No núcleo	No processo (biblioteca)	Em ambos
Complexidade	Média	Baixa	Alta
Custo de gerência	Médio	Baixo	Alto
Escalabilidade	Baixa	Alta	Alta
Paralelismo entre threads do mesmo processo	Sim	Não	Sim
Troca de contexto entre threads do mesmo processo	Lenta	Rápida	Rápida
Divisão de recursos entre tarefas	Justa	Injusta	Variável, pois o mapeamento thread → processador é dinâmico
Exemplos	Windows, Linux	GNU Portable Threads, Microsoft UMS	Solaris, FreedBSD KDE

Adaptado de: Maziero (2019, p. 63).

3.2.2 Vantagens de utilização de threads

A utilização de threads possibilita a ocorrência de atividades simultâneas. Ao decompor as atividades de um processo, pode-se fazê-lo em atividades paralelas, realizadas por diferentes threads. Aplicando a um processo de navegador web, por exemplo, uma thread pode ser utilizada para exibir imagens ou texto, enquanto outra recupera dados da rede. Um processador de textos consegue ter uma thread para exibir elementos gráficos para formatação da fonte do texto, outra para responder ao uso do teclado pelo usuário, e uma terceira é responsável pela verificação ortográfica e gramatical, sendo realizada em background.

Como são compartilhados recursos de um processo para suas diferentes threads, é mais rápido e eficiente utilizar múltiplas threads em relação à troca de contextos entre processos, pois existe um tempo gasto pelo sistema para realizar tal troca. Esse tempo é praticamente eliminado se forem utilizadas múltiplas threads.

Em sistemas com múltiplas CPUs, cada thread de um processo pode ser alocada em diferentes processadores, implementando um paralelismo real entre eles, beneficiando-se da escalabilidade. De acordo com Deitel, Deitel e Choffnes (2005), um sistema operacional é escalável ao ser capaz de utilizar recursos à medida que esses são acrescentados.

Os benefícios da criação de múltiplas threads podem ser ainda maiores em uma arquitetura multiprocessadora em que as threads possam ser executadas em paralelo em diferentes núcleos de processamento. Um processo com uma única thread somente pode ser executado em um processador, independentemente de quantos estiverem disponíveis.

Uma dificuldade de threads é que, com o compartilhamento do espaço de endereçamento entre eles, muitas vezes é necessário sincronizar as informações dessa memória compartilhada.

Adicionalmente, o desenvolvimento de aplicações multithread é mais complexo do que com apenas uma thread, pois é obrigatório que o compartilhamento de recursos e a comunicação entre os diversas threads sejam realizados de forma sincronizada, visando impedir futuros problemas de inconsistências e deadlock. As dificuldades naturais no desenvolvimento de aplicações concorrentes são somadas ao complexo procedimento de depuração da aplicação.

3.3 Comunicação entre processos

Existem muitas implementações de sistemas que possuem diversos processos interdependentes e necessitam cooperar entre si para atingir os objetivos da aplicação, por exemplo, em um software para reprodução de vídeo ou áudio. Buscando a cooperação entre os processos de uma aplicação, eles precisam comunicar informações uns para os outros e coordenar suas ações, para garantir que os resultados obtidos sejam coerentes. Abordaremos os principais conceitos, problemas e soluções referentes à comunicação entre processos.

Processo cooperativo é aquele cuja execução pode afetar outros em execução no sistema, ou é afetado pelos outros processos. Os processos são executados de forma concorrente, mas o acesso concorrente a dados compartilhados pode causar inconsistência de dados.

Um exemplo de cooperação é o processo que gerencia os botões e menus de um navegador web, que precisa alertar o mais rápido possível para outros processos caso o usuário tenha clicado nos botões Voltar ou Recarregar.

Se as tarefas estão no mesmo processo, elas compartilham a mesma área de memória, e a comunicação pode então ser implementada facilmente, usando variáveis globais comuns na programação do processo. Entretanto, quando as threads forem provenientes de processos distintos, não há variáveis compartilhadas. Nesta situação, a comunicação deve ser realizada por intermédio do núcleo do sistema operacional. utilizando chamadas de sistema ou system calls.

Caso as threads estejam em computadores distintos, o núcleo deve implementar mecanismos de comunicação específicos, usando mecanismos de comunicação em rede. A figura seguir ilustra essas três situações.

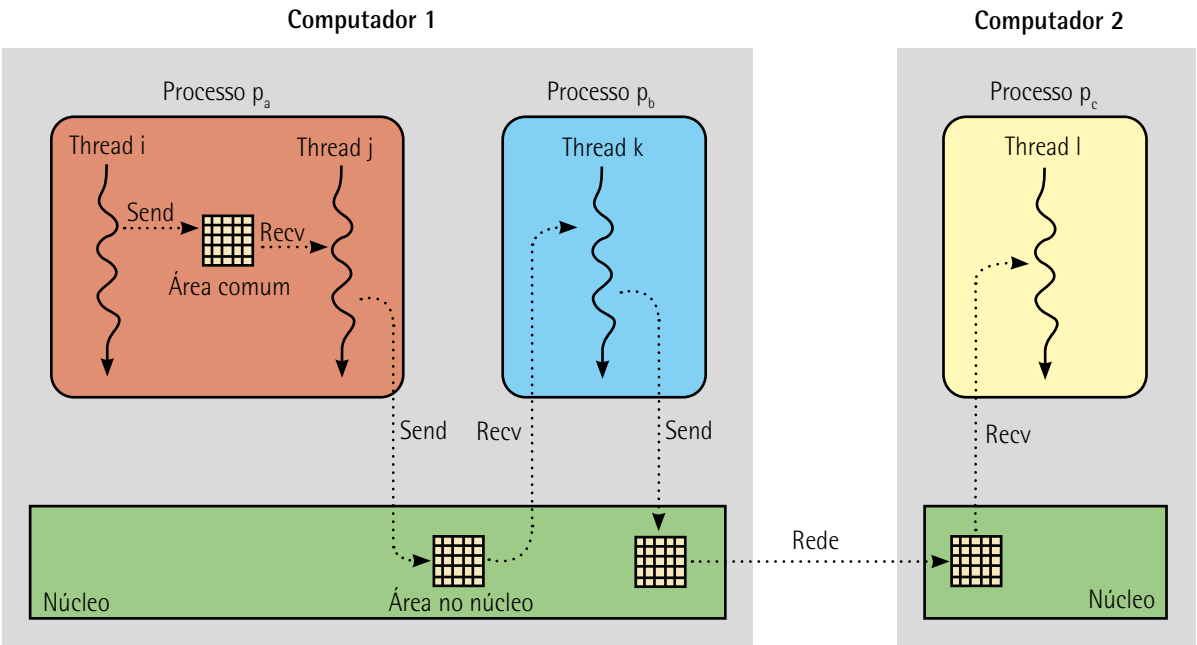


Figura 27 – Comunicação intraprocesso ($t_i \rightarrow t_j$), interprocessos ($t_j \rightarrow t_k$) e intersistemas ($t_k \rightarrow t_i$)

Fonte: Maziero (2019, p. 94).

Os mecanismos de comunicação entre processos são habitualmente denominados de forma genérica como mecanismos de *inter-process communication* (IPC). A comunicação entre processos é mais eficiente quando estruturada para gerar colaboração entre os processos, não utilizar interrupções e não entrar em conflito.

Existem várias formas para implementação da comunicação entre tarefas. Uma das etapas do projeto do sistema operacional é a definição dos mecanismos de comunicação oferecidos por ele e que consideram muitos aspectos, como o formato dos dados a transferir, o sincronismo exigido nas comunicações, a necessidade de buffers e o número de emissores/receptores envolvidos em cada ação de comunicação.

Lembrete

O buffer de dados é uma região de memória física com a função de armazenamento temporário de dados enquanto eles estão sendo deslocados de uma posição para outro local de armazenamento.

Na sequência, serão apresentados os seguintes aspectos de comunicação de processos: comunicação direta ou comunicação indireta, sincronismo do canal de comunicação, formato de envio e capacidade dos canais.

3.3.1 Comunicação direta ou indireta

A comunicação direta entre dois processos exige endereçamento explícito do nome do processo receptor ou transmissor quando ocorre envio ou recebimento de uma mensagem. Dessa forma, existem duas primitivas básicas: Enviar (dados, destino) e Receber (dados, origem).

Este tipo de comunicação é caracterizado por somente permitir a troca de mensagens entre dois processos, e o emissor identifica abertamente o receptor e vice-versa. Sua principal dificuldade é a necessidade do conhecimento do nome dos processos. A figura a seguir apresenta o fluxo de comunicação direta.

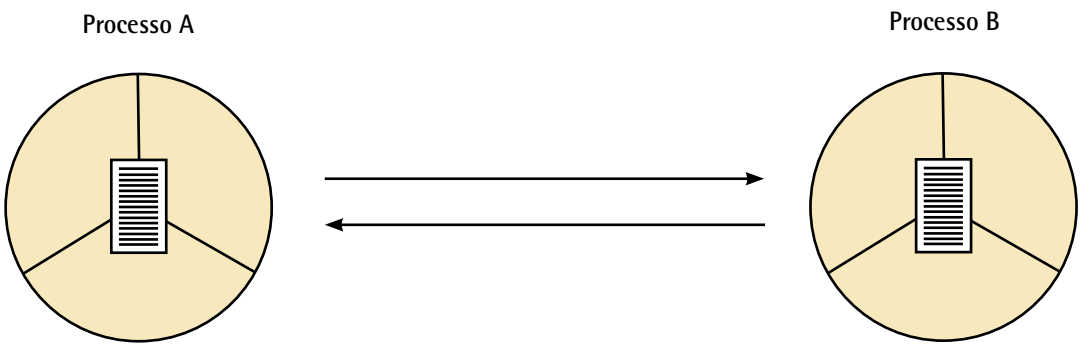


Figura 28 – Comunicação direta

Fonte: Machado e Maia (2013, p. 117).

Por outro lado, na comunicação indireta entre processos existe uma área compartilhada, na qual as mensagens podem ser inseridas pelo processo emissor e retiradas pelo receptor. Esse tipo de buffer é denominado mailbox ou port, e seus atributos de identificação e capacidade de armazenamento são estabelecidos no momento de criação. É possível associar diversos processos à mesma mailbox, e os parâmetros dos procedimentos Enviar e Receber são os nomes de mailboxes, retirando a necessidade de conhecer o nome do outro processo. A figura a seguir apresenta o fluxo de mensagens em uma comunicação indireta.

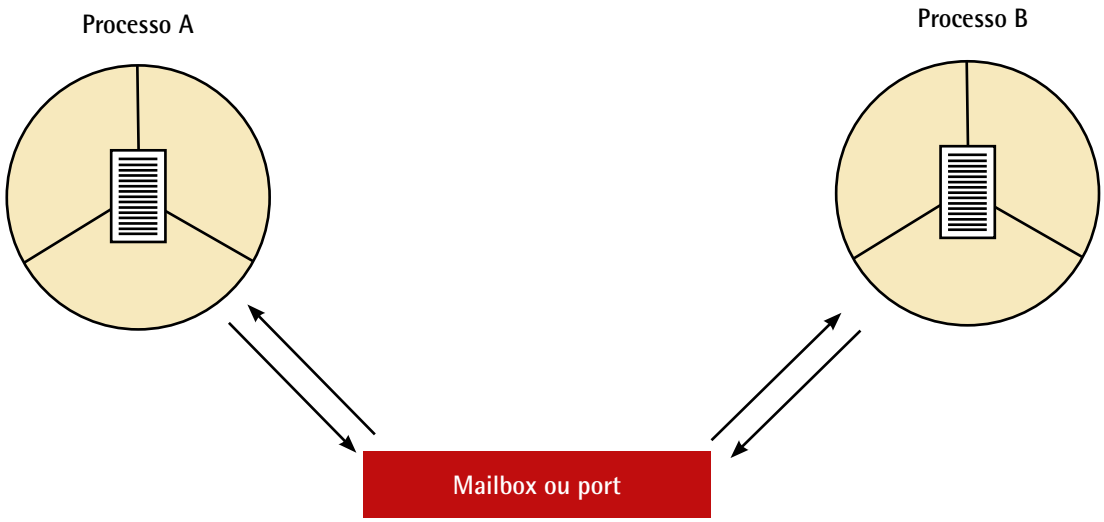


Figura 29 – Comunicação indireta

Fonte: Machado e Maia (2013, p. 118).

3.3.2 Sincronismo das mensagens

Seja qual for o mecanismo de comunicação utilizado, a troca de mensagens entre os processos deve ter suas execuções sincronizadas dependendo do fluxo delas. Situações de sincronização relevantes são o emissor não alterar a mensagem após o envio enquanto não ocorrer o recebimento da mensagem e a mesma mensagem não chegar duplicada ao receptor. Existem três diferentes esquemas para sincronizar os processos que se comunicam.

O esquema inicial de sincronização é assegurar que um processo, ao enviar uma mensagem, continue esperando até o momento da leitura do processo receptor. De forma análoga, um processo receptor, ao tentar receber uma mensagem ainda não enviada, deve aguardar até que o processo emissor finalize a transmissão. Esse tipo de comunicação é denominado síncrono ou bloqueante e implementa uma forma síncrona de comunicação entre os processos, sendo conhecido como *rendez-vous*, termo francês para encontro. A limitação deste método é que a execução dos processos está condiciona ao tempo de processamento no tratamento das mensagens.

A figura a seguir ilustra dois casos de *rendez-vous*.

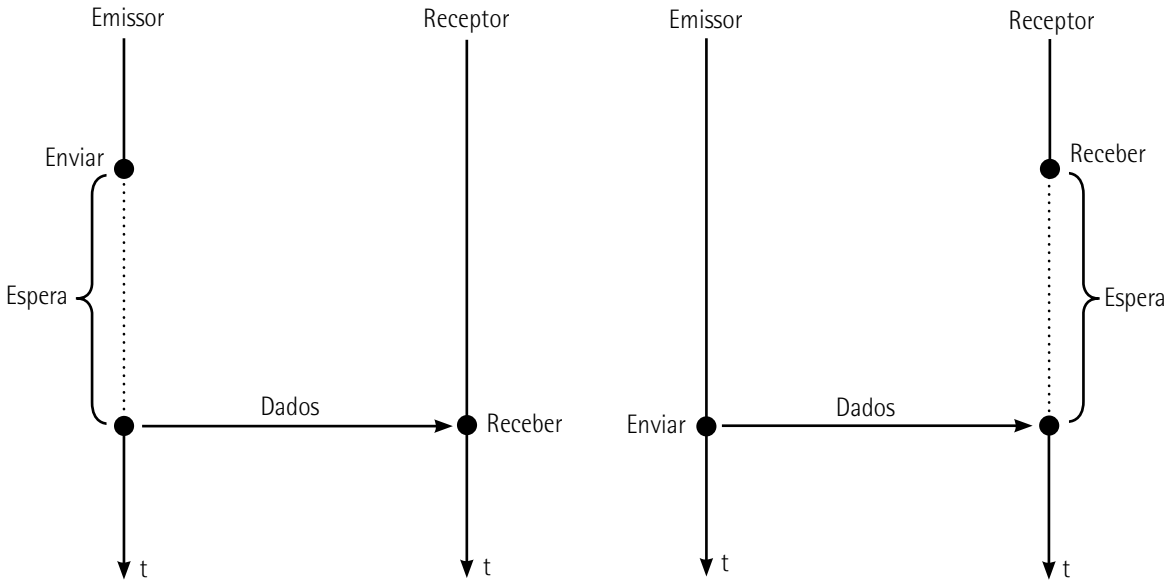


Figura 30 – Comunicação síncrona e capacidade de canal nula – *rendez-vous*

Fonte: Maziero (2019, p. 95).

Uma alteração desse método para torná-lo mais eficiente é permitir que o processo transmissor não se mantenha bloqueado aguardando a leitura da mensagem pelo processo receptor. Neste caso, um processo transmitirá mensagens para diferentes destinatários assim que possível.

O terceiro esquema aplica uma forma assíncrona de comunicação, em que nem o processo receptor nem o processo transmissor ficam aguardando o envio e o recebimento de mensagens. Além de necessitar de buffers para armazenar as mensagens, devem existir mecanismos de sincronização que o processo identifique se uma mensagem já foi transmitida ou recebida. Um ponto positivo para utilização deste mecanismo é o aumento da eficiência de aplicações concorrentes.

Uma razão para utilização dos buffers é a existência de diferença entre as taxas de transferência para recebimento e a taxa de processamento, ou quando elas são variáveis.

3.3.3 Capacidade dos canais

O comportamento síncrono ou assíncrono de um canal de comunicação pode ser afetado pela presença de buffers que permitam armazenar temporariamente os dados em trânsito, ou seja, as informações enviadas pelo emissor e que ainda não foram recebidas pelo receptor. Em relação à capacidade de buffering do canal de comunicação, três situações devem ser analisadas: capacidade nula, capacidade infinita e capacidade limitada.

Quando a capacidade é nula (n = 0), o canal não armazena dados e a comunicação é realizada diretamente do transmissor para o receptor, sem a criação de cópias intermediárias. Não é viável uma comunicação assíncrona para um canal com capacidade nula, pois é necessário existir um buffer para transmissão.

Se a comunicação é síncrona, o emissor permanece bloqueado até que o destinatário receba os dados, e o receptor fica bloqueado até o emissor enviá-los. A situação específica de comunicação síncrona com canais de capacidade nula resulta em uma forte sincronização entre as partes e é denominada *rendez-vous*.

Quando a capacidade do canal é infinita ($n \rightarrow \infty$) sempre é possível o transmissor enviar dados, os quais serão armazenados no buffer do canal até o receptor os capturar. Trata-se de uma situação hipotética, utilizada no estudo de algoritmos de comunicação, pois todos os sistemas de computação possuem capacidade de armazenamento e de memória finita.

A maior parte dos sistemas reais possui capacidade finita ($0 < n < \infty$) quando há um limite de dados que o emissor pode enviar sem que o receptor os capture. Entretanto, o emissor poderá ficar bloqueado ao tentar enviar dados em um canal já saturado. O bloqueio permanecerá até o surgimento de espaço no buffer do canal.

3.4 Como evitar problemas de comunicação

Para analisar a comunicação entre os processos, é necessário observar alguns critérios, por exemplo:

- Como um processo passa informação para outro?
- Como garantir que processos não invadam espaços uns dos outros?
- Existe dependência entre os processos? Se houver, qual a ordem de execução adequada?

A seguir, serão apresentados o problema de condições de corridas, o conceito de exclusão mútua e as formas de solucioná-la.

3.4.1 Condições de corrida ou *race conditions*

Pode-se definir condições de corrida ou *race conditions* como situações nas quais dois os mais processos estão lendo ou escrevendo algum dado compartilhado de forma concorrente e o resultado final depende da ordem de processamento. Existe uma corrida por recurso, que pode ser memória, arquivos, impressoras, discos ou variáveis que resultam em erros e inconsistências provenientes de acessos concorrentes a dados compartilhados.

Um exemplo prático é a fila de impressão de uma impressora ligada em uma rede de computadores. Quando um usuário deseja imprimir um arquivo, um dos processos irá colocar o arquivo em um local especial denominado spooler, uma fila de impressão. Outro processo, conhecido como printer spooler, verifica se existe algum novo arquivo a ser impresso. Caso exista, ele é impresso e retirado do spooler após a finalização. Imagine a seguinte situação: dois processos desejam ao mesmo tempo imprimir um arquivo. O que o sistema operacional deve fazer?

Daremos agora outro exemplo simples que empregava variáveis compartilhadas na memória. Um processo A utilizava uma variável compartilhada *a* por meio da equação a = b + c. Um processo X atualiza a variável x através de x = a + y. Se o processo X fizer esse cálculo antes de o processo A executar, resultará em um valor de variável *x* incorreto.

Assim, as condições de corrida devem ser evitadas, pois tiram o comportamento previsível desejado para o sistema operacional. Adicionalmente, a depuração de programas que contêm condições de corrida não é simples, já que não é possível prever quando o processo será suspenso e os erros não aparecem no código-fonte do programa, mas se manifestam na durante a execução.

3.4.2 Regiões críticas e exclusão mútua

Agora que entendemos o conceito de condições de corrida e suas consequências, como evitá-las? Uma forma simples de resolver os problemas de compartilhamento é impedir que dois ou mais processos tenham acesso a um mesmo recurso de forma simultânea.

A região crítica ou *critical region* é a parte do código do programa na qual é implementado o acesso ao recurso, que será compartilhado. Quando existe a garantia de que dois processos não estão em regiões críticas simultaneamente, os problemas de compartilhamento são resolvidos.

Quando um processo estiver com acesso ao recurso, todos os outros que necessitem do mesmo recurso devem esperar até que ele termine de utilizá-lo e saia da região crítica. Essa exclusividade de acesso à região crítica em cada instante é denominada exclusão mútua ou *mutual exclusion*. O sistema operacional deve assegurar a exclusão mútua recorrendo aos mecanismos de sincronização.

A figura a seguir apresenta a situação de dois processos (A e B) que possuem um recurso compartilhado. Quando o processo A entra na região crítica em T_1, o recurso fica bloqueado para o processo B até que o processo A saia da região crítica em T_3. O processo B tentou acessar a região crítica em T_2, mas foi obrigado a esperar a liberação do processo A ocorrida em T_3. Caso o processo A ou outro tente acessar a região crítica depois de T_3, ele não terá acesso até o tempo T_4, quando B libera a região crítica.

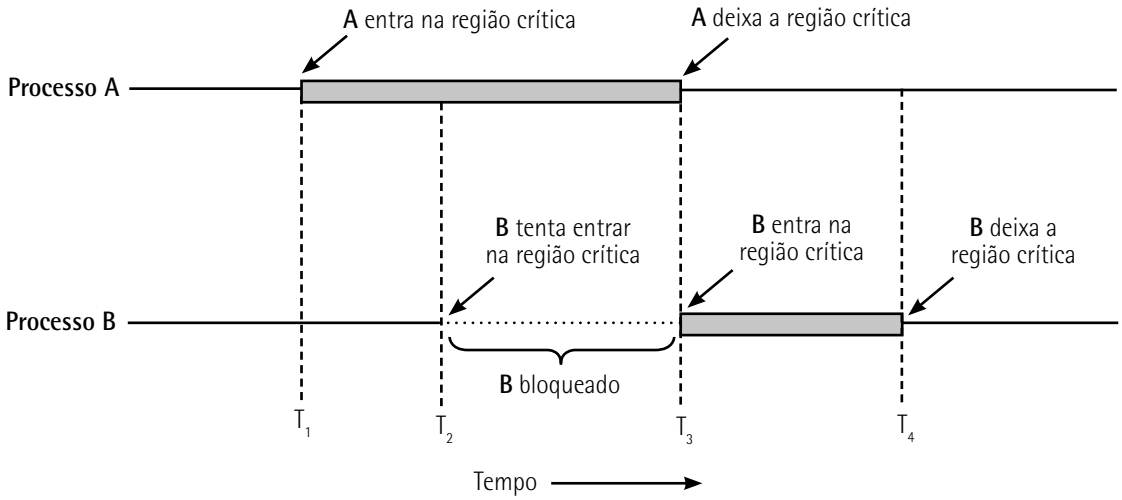


Figura 31 – Exclusão mútua utilizando regiões críticas

Fonte: Tanenbaum e Bos (2016, p. 83).

Segundo Tanenbaum e Bos (2016), existem quatro condições para que os processos cooperem de modo correto e eficiente com dados compartilhados. A primeira é que dois ou mais processos não podem estar simultaneamente em uma região crítica. A segunda condição é não fazer suposições em relação à velocidade e ao número de CPUs; a solução deve operar com um ou múltiplos processadores.

Na terceira condição, nenhum processo fora da região crítica pode causar bloqueio a outro. A quarta e última condição é que um processo não pode esperar infinitamente para ter acesso à região crítica; a situação do processo é conhecida como starvation ou espera indefinida.

Foram desenvolvidas diversas soluções para implementação da exclusão mútua, tais como espera ocupada, primitivas sleep/wakeup, semáforos, monitores e troca de mensagens.

3.4.3 Soluções para o problema da exclusão mútua – espera ocupada

Na implementação da exclusão mútua com **espera ocupada**, quando um processo está na região crítica atualizando a memória compartilhada, nenhum outro poderá acessá-la. Uma implementação simples é a desabilitação de interrupções. Ao inibir interrupções, não há troca de tarefas com a ocorrência de interrupções de tempo ou de eventos externos e o processo poderá atualizar a memória compartilhada sem influência de outros processos.

Entretanto, esta solução é aplicável somente ao sistema de processador único, pois apenas uma CPU, a afetada pela instrução bloqueante, terá as interrupções inibidas. Adicionalmente, não é conveniente atribuir ao usuário a decisão de habilitar as interrupções. Consequentemente, é necessário utilizar formas mais complexas para implementar a exclusão mútua.

Outra tentativa para exclusão com espera ocupada é o uso de variáveis do tipo trava, que é implementada por software. Cria-se uma única variável compartilhada do tipo trava ou lock, com valor

inicial igual a 0. Quando a variável trava é 0, a região crítica está livre e, quando for 1, isso significa que algum processo está acessando a região crítica.

Quando um processo deseja entrar para região crítica, é realizado um teste na variável trava. Caso o valor dela seja 0, o processo entrará na região crítica e o valor da trava será alterado para 1. Se o valor de lock for 1, o processo precisa aguardar para acessar a região crítica. Um problema do método é que ele pode apresentar condições de corrida, pois se um processo lê que a trava é igual a 0 e, antes de alterar o valor da trava para 1, outro é escalonado para execução e altera o valor para 1, e ambos estarão em regiões críticas simultaneamente.

Uma terceira abordagem para a exclusão mútua é a alternância explícita, na qual processos se intercalaram no acesso à região crítica. Nela, define-se uma variável turno, que representa qual processo entrará na seção crítica. A variável deve ser ajustada cada vez que um processo sai da seção crítica, para indicar a próxima tarefa a usá-la.

Suas desvantagens são o teste contínuo do valor da variável turno e o desperdício do processador quando as tarefas não precisam da mesma frequência de utilização para acesso à região crítica.

3.4.4 Sincronização condicional

A situação em que o acesso ao recurso compartilhado demanda a sincronização de processos ligada a uma condição de acesso é conhecida como sincronização condicional. Quando um recurso não estiver pronto para utilização, o processo que deseja acessá-lo permanecerá em espera até que o recurso fique disponível.

Uma aplicação típica desse tipo de sincronização é a comunicação entre dois processos através de operações de gravação e leitura em um buffer, como mostra a figura a seguir. Nesse caso, os processos produtores geram informações utilizadas pelos processos consumidores, por isso é conhecido como problema do produtor-consumidor ou do buffer limitado.

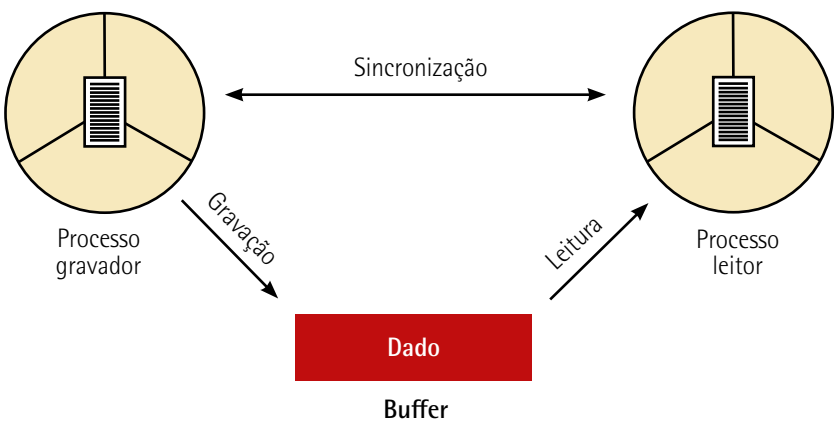


Figura 32 – Comunicação condicional

Fonte: Machado e Maia (2013, p. 94).

Enquanto um processo escreve os dados em um buffer, outro lê os dados de forma concorrente. Os processos envolvidos devem estar sincronizados a uma variável de condição, de maneira que o processo produtor não tente escrever mais dados quando o buffer estiver cheio ou o processo consumidor não ler um buffer vazio.

3.4.5 Semáforos

Outra implementação aplica o conceito de semáforos, que foi proposto pelo holandês E. W. Dijkstra, em 1965, como um mecanismo de sincronização que permitia implementar, de forma simples, a exclusão mútua e a sincronização condicional entre processos.

Um **semáforo** é uma variável inteira e não negativa, manipulada por somente duas instruções: down e up. Elas foram chamadas originalmente de instruções P, de **proberen** (teste), e V, **verhogen** (incremento), em holandês. As instruções down e up são atômicas e sua execução não pode ser interrompida.

Segundo Maziero (2019), a instrução **up** incrementa o contador interno e realiza um teste: um contador nulo indica que há tarefas suspensas naquele semáforo. A primeira tarefa da fila é então devolvida à fila de tarefas prontas, para retomar sua execução assim que possível. Por sua vez, a instrução down decrementa o contador interno e realiza o teste: se o contador estiver nulo, a tarefa solicitante é adicionada à fila do semáforo e suspensa. Caso contrário, ele retorna e a tarefa pode continuar sua execução. A figura a seguir ilustra a operação com as funções up e down do semáforo.

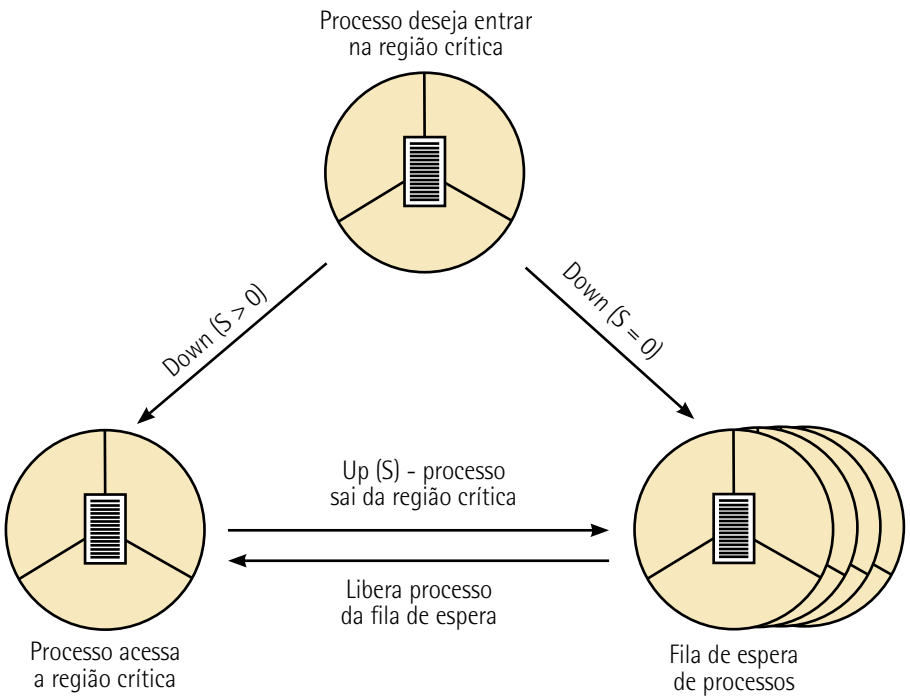


Figura 33 – Utilização de semáforo na exclusão mútua

Fonte: Machado e Maia (2013, p. 108).

Saiba mais

Um problema clássico no mundo da computação é o Jantar dos Filósofos, cuja solução é aplicação de semáforos. Veja nos links a seguir:

O FAMOSO problema do jantar dos filósofos! – Sistemas Operacionais. 2018. 1 vídeo (11 min). Publicado pelo canal Luís Paulo Bravin. Disponível em: https://bit.ly/3YSqFQD. Acesso em: 14 mar. 2023.

O PROBLEMA dos 5 filósofos. 2015. 1 vídeo (9 min). Publicado pelo canal Valquiria Huttner. Disponível em: https://bit.ly/3YUjJCA. Acesso em: 14 mar. 2023.

No artigo a seguir, é apresentada uma ferramenta que contribui com a aprendizagem de programação concorrente utilizando o conceito de semáforos. Ela é aplicada na simulação do caso "crianças brincando", um problema que compreende sincronização e exclusão mútua entre processos concorrentes.

BRAGA, S. A.; BEZERRA, H. C.; GARCIA, F. P. Crianças brincando: uma ferramenta para o auxílio na aprendizagem de programação concorrente. *In:* CONGRESSO BRASILEIRO DE EDUCAÇÃO EM ENGENHARIA (COBENGE), 43., 2015, São Bernardo do Campo. *Anais* [...]. São Bernardo do Campo: Cobenge, 2015. Disponível em: https://bit.ly/3LoHcsC. Acesso em: 14 mar. 2023.

3.4.6 Monitores

Um monitor é uma estrutura de sincronização de alto nível que ocupa ou libera a seção crítica associada a um recurso e implementa de forma automática a exclusão mútua entre os procedimentos declarados. Somente um processo pode executar um dos procedimentos do monitor em um determinado instante. De acordo com Maziero (2019), o monitor consiste nos seguintes elementos:

- um recurso compartilhado, representado como um conjunto de variáveis internas ao monitor;
- um conjunto de procedimentos e funções que permitem o acesso a essas variáveis;
- um mutex ou semáforo para controle de exclusão mútua, pois cada procedimento de acesso ao recurso deve obter o mutex antes de iniciar e liberá-lo ao concluir;
- um invariante sobre o estado interno do recurso.

A figura a seguir apresenta a estrutura de um monitor.

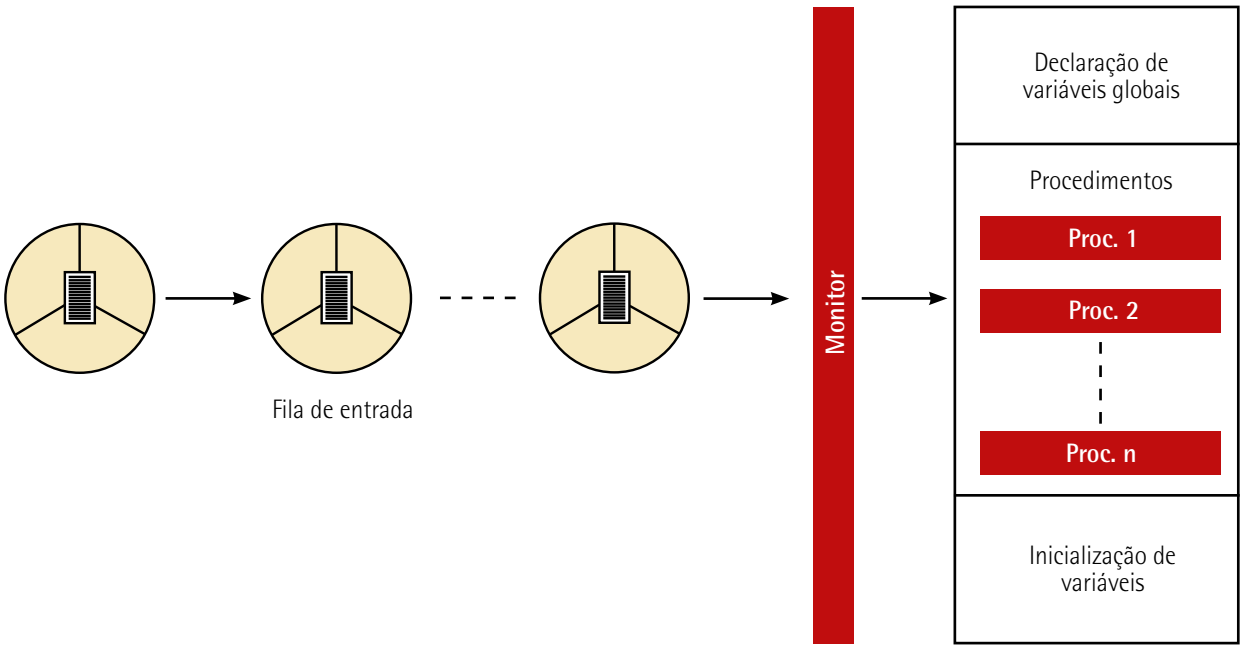


Figura 34 – Estrutura do monitor

Fonte: Machado e Maia (2013, p. 113).

3.5 Deadlocks

Como visto anteriormente, a fim de compartilhar recursos, deve-se suspender algumas tarefas para que outras acessem esses recursos e garantam a sua consistência. Neste caso, cada recurso é associado a um semáforo ou outro mecanismo equivalente. Dessa forma, as tarefas solicitam e aguardam liberação de cada semáforo. Entretanto, a utilização desses semáforos ou mutexes pode ocasionar a condição denominada impasse ou deadlock.

Impasses ou deadlocks são situações nas quais o processo espera por um recurso que nunca estará disponível ou um evento que não ocorrerá. Com isso, as tarefas envolvidas ficam bloqueadas, esperando a liberação de semáforos e nenhuma outra será realizada. Esses impasses ocorrem como consequência do compartilhamento de recursos entre processos em que se exige a exclusão mútua.

Trazendo para um exemplo real, o congestionamento em um cruzamento de uma cidade onde tudo está parado é uma situação de impasse ou deadlock. A figura a seguir apresenta um situação de deadlock de tráfego.

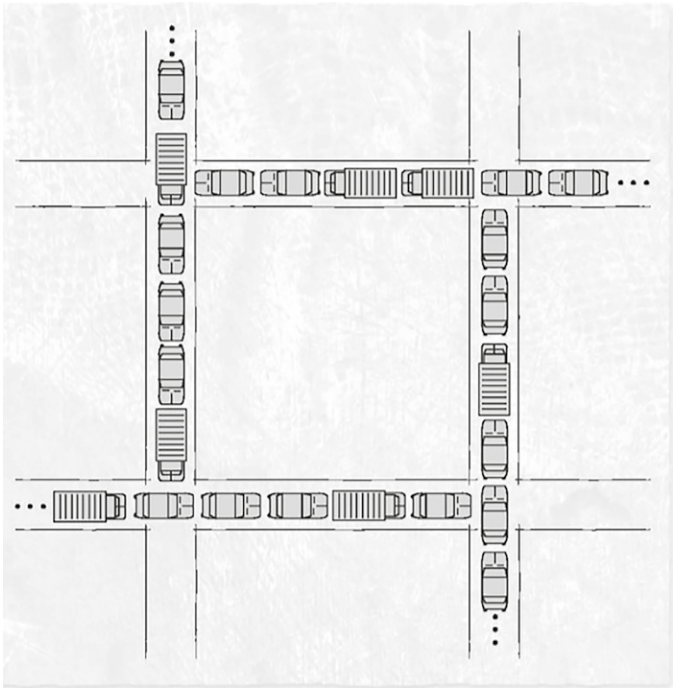


Figura 35 – Impasse ou deadlock

Fonte: Deitel, Deitel e Choffnes (2005, p. 180).

Nessa analogia, os carros seriam os processos e os recursos seriam a parte da rua que ocupam. Cada carro em movimento utiliza a parte da rua diretamente abaixo e requisita a parte à frente do veículo.

Para ocorrência da situação de deadlock, são indispensáveis quatro condições simultâneas:

- **Exclusão mútua**: cada recurso pode ter alocação a somente um processo em um determinado instante.
- **Espera por recursos**: um processo, além dos recursos já alocados, ainda espera a disponibilidade de outros.
- **Não preempção**: um recurso não será liberado de um processo somente porque outros deles desejam o mesmo recurso.
- **Espera circular**: um processo pode ter que aguardar por um recurso alocado a outro processo e vice-versa.

Para representar as situações de deadlock, utilizaremos o **grafo de alocação de recursos**, no qual as tarefas serão representadas por círculos (○) e os recursos por retângulos (□). Para identificar que um recurso está alocado à tarefa, é utilizada a representação □→○ e, para representar que a tarefa requer um recurso, utiliza-se □→○.

A figura a seguir apresenta um deadlock, no qual dois processos (P_1 e P_2) necessitam de dois recursos para serem executados (R_1 e R_2) e somente um deles está disponível. Verifique que a situação descrita atende às quatro condições de deadlock simultaneamente (exclusão mútua, espera por recursos, não preempção e espera circular). Cada processo retém um recurso requisitado por outro e nenhum processo possui disposição para liberação dos recursos sob sua alocação.

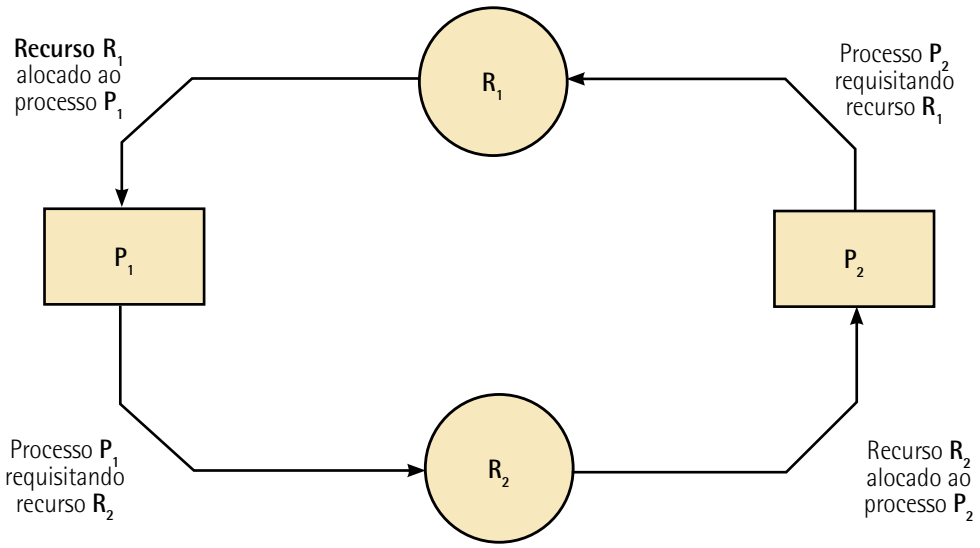


Figura 36 – Impasse ou deadlock

Fonte: Deitel, Deitel e Choffnes (2005, p. 181).

4 ESCALONAMENTO DO PROCESSADOR

O escalonamento do processador é um dos principais pilares dos sistemas operacionais multiprogramados. Ao realizar a alternância de processos na UCP, o sistema operacional pode tornar o computador mais eficiente. Neste capítulo, serão apresentados os conceitos envolvidos no escalonamento de processos ou scheduling, isto é, na escolha de qual processo será executado em cada instante de tempo, que é um dos componentes mais relevantes da gerência de tarefas. Existem diferentes algoritmos e parâmetros para realização do processo de escalonamento, e o algoritmo definirá o comportamento do sistema operacional para tratar de forma rápida e eficiente os processos a serem executados.

O objetivo da seção é apresentar algoritmos de escalonamento utilizados pelos sistemas operacionais.

4.1 Escalonamento em sistema monotarefa

As duas primeiras gerações de computadores executavam apenas uma tarefa por vez. Nelas, cada programa binário era carregado do disco para a memória e executado até seu término. Os dados de entrada da tarefa eram carregados na memória e os resultados obtidos no processamento eram descarregados de volta no disco após a finalização da tarefa.

Todas as operações de transferência de código e dados entre o disco e a memória eram coordenadas por um operador humano. Esses sistemas primitivos eram usados sobretudo para aplicações de cálculo numérico, tais como problemas de balística, trigonometria, cálculos de equações diferenciais, mecânica dos fluidos, entre outros. A figura a seguir ilustra um sistema desse tipo.

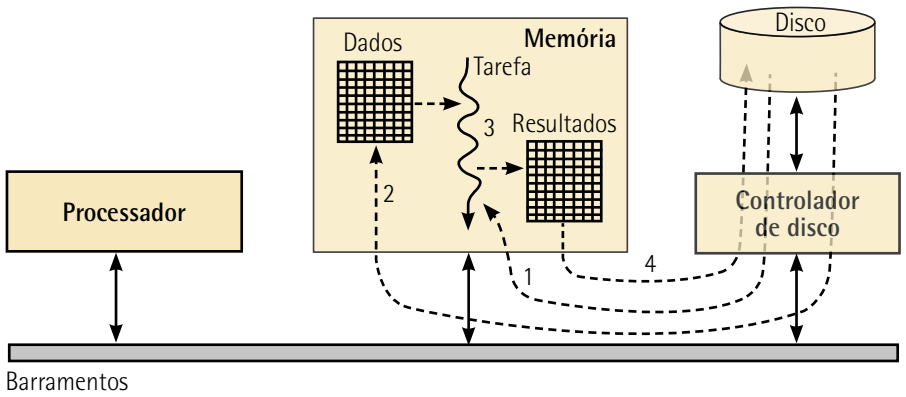


Figura 37 – Sistema monotarefa

Fonte: Maziero (2019, p. 30).

Na etapa 1, ocorre a carga do código, que originalmente está armazenado no disco, na memória de trabalho. Posteriormente, há o carregamento na memória dos dados de entrada a serem utilizados para execução do programa. Na etapa 3, é realizado o processamento das instruções do programa, que irá utilizar os dados de entrada e produzir resultados na saída. Por fim, na etapa 4, é finalizada a execução com a gravação dos resultados obtidos pelo programa no disco.

Nesse tipo de sistema, existem três estados para os processos: processo novo, processo em execução e processo finalizado. A figura a seguir apresenta o diagrama de estados para os processos em sistemas monotarefa.

Figura 38 – Diagrama de estados de sistema monotarefa

4.2 Escalonamento em sistema multitarefa

Aplicações também podem ser projetadas para alavancar capacidades de processamento em sistemas multicore. Elas podem executar diversas tarefas CPU-intensivas em paralelo, em múltiplos núcleos de computação.

A vantagem da abordagem multiprocesso é que os websites são executados isoladamente. Se um website falha, apenas o seu processo renderizado é afetado; todos os outros permanecem inalterados.

Além disso, processos renderizadores são executados em uma sandbox, o que significa que o acesso ao I/O de disco e de rede é restrito, minimizando os efeitos de quaisquer invasões na segurança.

A figura a seguir ilustra a concorrência de três programas, PROG_1, PROG_2 e PROG_3, que são integrantes dos processos X, Y e Z. Em sua parte inferior, existe uma representação temporal que mostra quais processos estão sendo executados a cada instante.

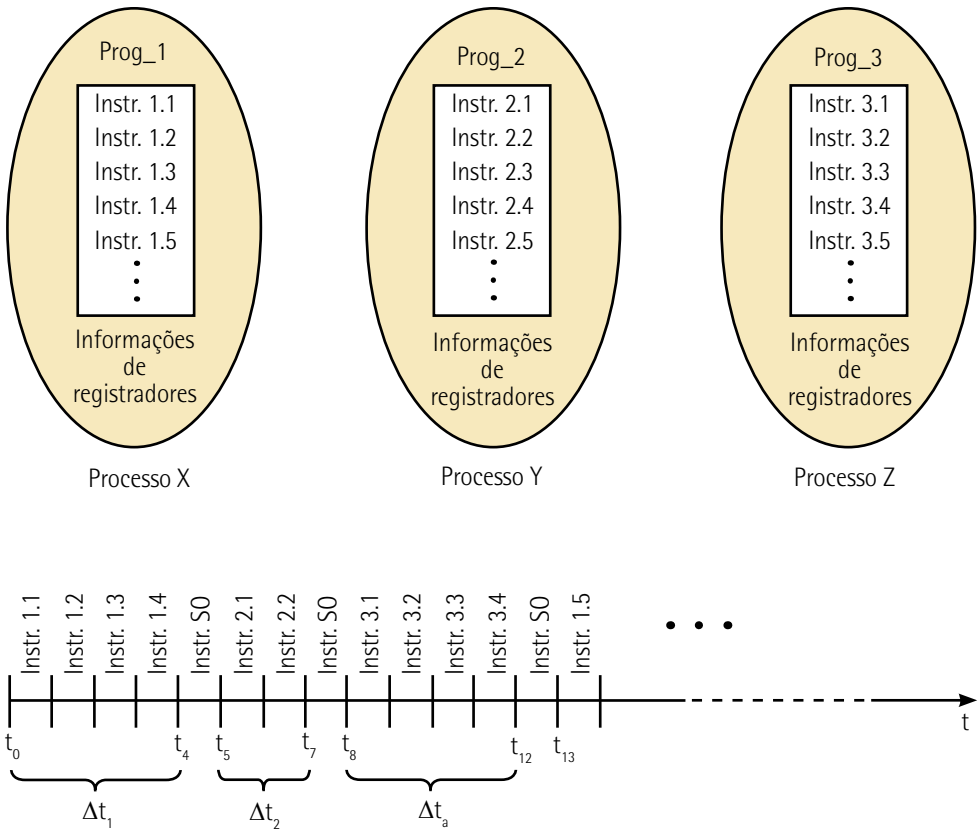


Figura 39 – Concorrência de programas e processo

Fonte: Machado e Maia (2013, p. 62).

Inicialmente, o processador executa instruções do PROG_1 por um intervalo de tempo Δt_1. No instante de tempo t_4, o sistema operacional interrompe temporariamente a execução do PROG_1 e salva o conteúdo dos registradores do processador, guardando seu conteúdo no processo X para voltar a executar a partir da última instrução executada.

A seguir, o PROG_2 do processo Y é iniciado e executado durante o intervalo de tempo Δt_2. No instante t_7, o sistema operacional decide interromper o PROG_2 e salva o conteúdo dos registradores no processo Y. Neste momento o PROG_3 do processo Z é iniciado e executado durante o intervalo de tempo Δt_3. A execução desse programa ocorre até que o sistema operacional decide interrompê-lo, armazena os valores dos registradores no processo Z e retoma a execução de PROG_1 do processo X.

Para voltar a PROG_1, os valores dos registradores do processo X são carregados no processador durante t_{12}, permitindo que PROG_1 continue sua execução como se não tivesse sido interrompido. A troca de um processo por outro no processador, comandada pelo sistema operacional, é chamada de mudança de contexto e através dela o sistema implementa e gerencia um ambiente multiprogramável.

Exemplo de aplicação

Um exemplo de aplicação ocorre no navegador Chrome da Google. Muitos websites apresentam conteúdos ativos em JavaScript, Flash e HTML5 para oferecer uma experiência mais dinâmica na página web para o usuário. Infelizmente, essas aplicações web podem gerar bugs de software, tornando o tempo de resposta maior, conseguindo, até mesmo, deixar a página indisponível no browser. Como os navegadores web modernos realizam a navegação através de guias, é possível abrir em uma única instância de uma aplicação vários websites ao mesmo tempo, sendo que cada um deles fica em uma guia separada.

Uma desvantagem dessa abordagem é que, caso uma aplicação web em uma guia tenha um bug, o processo inteiro, incluindo todas as outras guias que estiverem exibindo websites adicionais, também cairá.

O Chrome foi projetado para abordar esse dilema utilizando uma arquitetura multiprocesso. São identificados três tipos de processo: navegador, renderizadores e plug-ins. O processo navegador gerencia a interface de usuário, assim como os processos de E/S do disco e de rede. Um novo processo navegador é criado quando o Chrome é iniciado e apenas um processo navegador é feito.

Através dos processos renderizadores, são manipulados os códigos de HTML, JavaScript, imagens, e assim por diante. De forma geral, um novo processo renderizador é criado para cada website aberto em uma nova guia e, desse modo, diversos processos renderizadores podem estar na lista de processos PCB ao mesmo tempo.

Outro tipo de processo é o plug-in, que são ferramentas que adicionam funções ao navegador web, como Flash ou QuickTime. Para cada plug-in é inicializado um processo que contém o código para o plug-in, assim como um código adicional que permite à ferramenta se comunicar com os processos renderizadores associados e com o processo navegador.

4.3 Escalonamento de processos

Podemos considerar que escalonamento é a troca de processos, em que se escolhe o próximo item a ser executado.

Os componentes envolvidos no escalonamento de processos são o despachante ou dispatcher e o escalonador ou scheduler. O despachante é responsável por armazenar na memória principal e recuperar as informações do contexto do processo e atualizar as informações no PCB. A rotina do sistema operacional, que tem como principal função implementar os critérios da política de escalonamento, é

denominada escalonador ou scheduler. A função do escalonador é a escolha da próxima tarefa a ser executada no processador.

A ação do despachante é relativamente rápida, em torno de 0,1 ms, e a parte mais demorada é a do escalonador, pois exige mais processamento. Será necessário verificar a fila de processos prontos, atualizar o status do processo na fila e escolher qual processo estará no topo.

Existem diferentes situações nas quais o escalonador de processos recebe uma chamada de sistema:

- Um novo processo é criado, e o escalonador irá definir a posição desse novo processo na fila de pronto.
- Um processo-pai cria um processo-filho, sendo necessário que o escalonador realize a escolha para executar o processo-pai ou o processo-filho.
- Quando um processo é finalizado, um processo da fila de pronto deve ser executado.
- Quando um processo entra em estado de espera, por dependência de uma entrada/saída, o escalonador é chamado para escolher outro processo a ser executado.

No caso de ocorrência de um evento de E/S, o escalonador deve selecionar uma das três opções: executar o processo que estava aguardando esse evento, continuar executando o processo que já está em andamento, ignorando a E/S; ou executar outro processo existente na lista de processos prontos, a ser escolhido pelo escalonador.

Em um sistema multiprogramável, o escalonador é fundamental, pois todo o compartilhamento do processador é dependente desta rotina.

4.3.1 Categorias de escalonador

O escalonador de um sistema operacional pode ser classificado como preemptivo ou não preemptivo. Nos sistemas preemptivos, com a chegada de um novo evento, é gerada a interrupção forçada de um processo para que outro utilize a CPU. A cada interrupção, exceção ou chamada de sistema (system call), o escalonador verifica todos os processos da fila de pronto e decide se continua ou substitui aquele atualmente em execução.

Por sua vez, sistema não preemptivo exige a cooperação dos processos entre si na gestão do processador, para que todos possam executar. Logo, ela também é chamada de cooperativo. Mesmo quando chega um novo processo, aquele em execução permanece no processador tanto quanto possível e o processo somente será liberado caso termine de executar, libere explicitamente o processador, voltando à fila de tarefas prontas, ou solicite uma operação de E/S.

Os sistemas operacionais modernos de uso geral são preemptivos. Já sistemas mais antigos, como o Windows 3.*, MacOS 8 e 9 e Palm OS 3, operavam de forma cooperativa ou não preemptiva.

4.3.2 Política de escalonamento

A política de escalonamento define regras de escolha e existem diversas técnicas e algoritmos para escalonamento de processos. Ela considera uma série de critérios, tais como: utilização do processador, throughput, tempo de espera, tempo de turnaround e tempo de resposta.

A **utilização do processador** representa a relação que o processador está executando no que se refere ao tempo total. De acordo com Machado e Maia (2013), uma utilização próxima a 30% indica um sistema com baixa carga de processamento. Já uma utilização próxima a 90% indica um sistema próximo da sua capacidade máxima e bastante carregado.

Throughput representa a quantidade de processos executados em um determinado intervalo de tempo. Na maior parte dos sistemas operacionais, deseja-se maximizar o throughput.

O tempo total que um processo continua na fila de pronto e aguardando para ser executado é denominado **tempo de espera**. A diminuição desse tempo é um dos objetivos da maior parte das políticas de escalonamento.

Tempo de turnaround ou tempo de vida é a duração que um processo leva desde a sua criação até sua finalização, considerando todo o tempo dispendido na espera para alocação de memória, o tempo de espera, o tempo de processamento na UCP e o tempo que o processo estava bloqueado, como em operações de E/S. As políticas de escalonamento buscam a minimização do tempo de turnaround.

O tempo decorrido entre uma solicitação ao sistema ou à aplicação e o momento em que a resposta é apresentada é conhecido como **tempo de resposta**. Normalmente, ele não é limitado pela capacidade de processamento do sistema computacional, mas pela velocidade dos dispositivos de E/S. Em sistemas interativos, como sites de comércio eletrônico e mobile banking, os tempos de resposta devem ser da ordem de poucos segundos.

Pelo critério de **justiça**, caso haja dois processos com comportamentos e prioridades similares, eles devem ter durações de execução idênticas.

Lembrete

As políticas de escalonamento não possuem influência sobre o tempo de processador de um processo e este tempo depende somente do código da aplicação e da entrada de dados.

A política de escalonamento de um sistema operacional possui várias funções com o objetivo de manter o processador ocupado a maior parte do tempo, privilegiar a execução de aplicações críticas, balancear a utilização da UCP entre processos, maximizar o throughput do sistema e buscar diminuir os tempos de turnaround, espera e resposta. Apesar disso, as funções que uma política de escalonamento possuem são muitas vezes conflitantes.

Cada sitema operacional tem uma política de escalonamento específica para se adequar ao seu propósito e às suas características. Sistemas de tempo compartilhado, por exemplo, possuem requisitos de escalonamento distintos daqueles de tempo real.

Em ambientes que implementam apenas processos, o escalonamento é realizado com base nos processos prontos para execução. Em sistemas que implementam threads, o escalonamento é feito considerando as threads no estado de pronto, independentemente do processo. Logo, podemos considerar o processo como a unidade de alocação de recursos, enquanto a thread é a unidade de escalonamento.

4.4 Algoritmos de escalonamento

Nesta seção, serão descritos os algoritmos de escalonamento de processos mais simples existentes na literatura. Em ambientes que implementam apenas a visão de processos, o escalonamento é realizado com base nos processos prontos para execução. Em sistemas que implementam threads, ele considera as threads no estado de pronto.

Para simplificar a análise dos algoritmos, as tarefas t_1, t_2, t_5 são orientadas a processamento, ou seja, não param para realizar operações de E/S. Cada tarefa tem uma data de ingresso (instante em que entrou no sistema), uma duração (tempo de processamento que necessita para realizar sua execução) e uma prioridade.

Na maioria dos sistemas atuais, o espaço de endereços virtuais de cada processo é organizado da seguinte forma: a parte inicial dos endereços virtuais é reservada para uso do processo, enquanto a parte final é reservada para o núcleo do sistema operacional.

4.4.1 Escalonamento *first come, first served* (FCFS)

A forma de escalonamento mais básica simplesmente atende aos processos sequencialmente à medida que eles estão em estado de pronto, ou seja, conforme a ordem de chegada na fila de processos prontos.

Esse algoritmo é conhecido como *first come, first served* (FCFS), isto é, o primeiro processo que vier na lista de pronto será o primeiro a ser servido ou executado. Também é denominado *first in, first out* (Fifo). Ele é não preemptivo e possui como principal vantagem sua simplicidade.

A figura a seguir ilustra o algoritmo do FCFS.

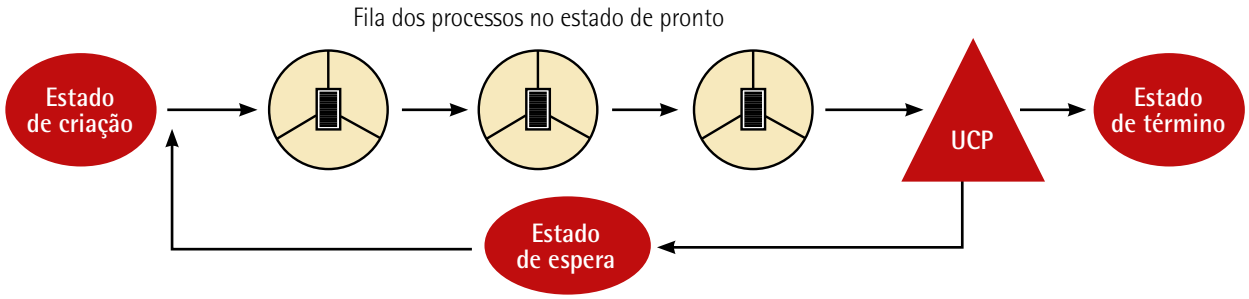


Figura 40 – Escalonamento FCFS

Fonte: Machado e Maia (2013, p. 130).

Entretanto, sua principal desvantagem é a impossibilidade de previsão de quando um processo começará a ser processado, pois depende do tempo de execução dos demais processos posicionados à sua frente na fila de pronto. Adicionalmente, esse algoritmo de escalonamento não busca reduzir o tempo médio de espera dos processos. Esse problema pode ser melhor percebido nos tempos de turnaround que demandam menor tempo de UCP.

Nesse tipo de escalonamento, os processos CPU-bound possuem vantagem no uso do processador sobre aqueles com I/O-bound.

4.4.2 Escalonamento *shortest job first* (SJF)

Este algoritmo de escalonamento escolhe o processo que tiver o menor tempo de processador ainda por executar. Dessa forma, o processo em estado de pronto que necessitar de menos tempo de UCP para terminar seu processamento é selecionado.

Inicialmente, essa implementação foi aplicada nos primeiros sistemas operacionais do tipo batch ou em lote. Para cada novo processo admitido no sistema, era associado um tempo de processador ao seu contexto de software. Entretanto, não é possível conhecer previamente o tempo para cada processo, então são feitas estimativas com base em análises estatísticas de execuções anteriores dos programas.

Assim, o principal problema nesta implementação é a impossibilidade de estimar o tempo de processador para processos interativos, uma vez que a entrada de dados é uma ação imprevisível.

4.4.3 Escalonamento circular

Partindo-se do algoritmo FCFS e adicionando-se a preempção por tempo, chega-se ao algoritmo de escalonamento circular ou round-robin. Assim, quando um processo passa para o estado de execução, existe um limite de tempo para o uso do processador de forma exclusiva por um processo que é denominado fatia de tempo, time-slice ou quantum. Esse escalonamento é do tipo preemptivo e foi projetado principalmente para sistemas de tempo compartilhado.

No escalonamento circular, cada vez que um processo é escalonado para execução, é concedida uma nova fatia de tempo. Caso a fatia de tempo seja consumida, o sistema operacional irá paralisar o processo em execução, gravar seu contexto e direcioná-lo para o final da fila de pronto. Esse mecanismo é conhecido como preempção por tempo.

A figura a seguir apresenta o escalonamento circular, no qual a fila de processos em estado de pronto é abordada como uma fila circular. Inicialmente, a UCP será alocada no primeiro processo da fila de pronto. Este processo continuará no estado de execução até que finalize seu processamento, voluntariamente vá para o estado de espera ou que sua fatia de tempo expire, sofrendo então uma preempção pelo sistema operacional. Depois disso, um novo processo é escalonado com base na política de FCFS.

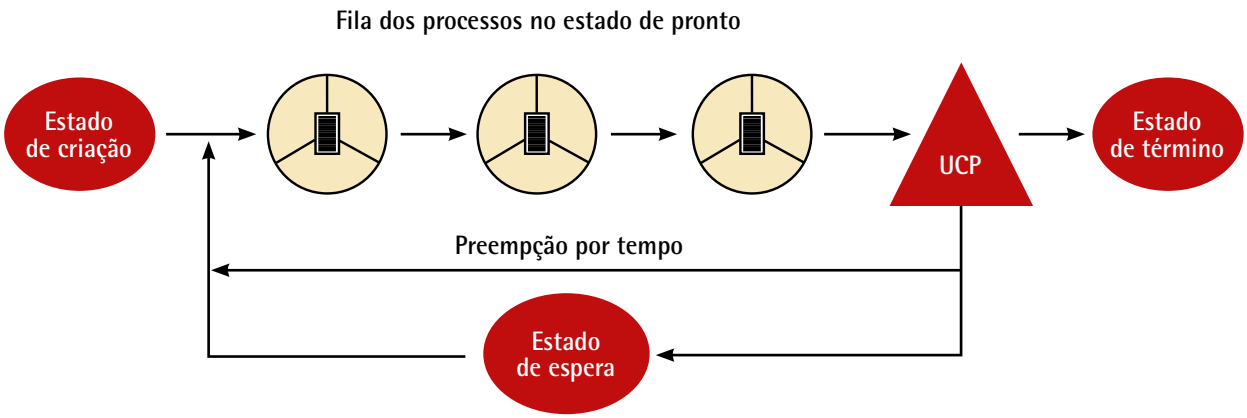


Figura 41 – Escalonamento circular

Fonte: Machado e Maia (2013, p. 130).

Nele, um processo em execução pode liberar o processador de forma voluntária, voltando à fila de processos prontos. Com isso, um novo processo será escalonado, possibilitando a melhor distribuição na utilização do processador e de forma cooperativa nos processos. O processo em execução realiza uma verificação periódica na fila de mensagens para determinar se existem outros processos na fila de pronto.

Neste tipo de escalonamento, a principal vantagem é que nenhum processo irá monopolizar o tempo de processamento da UCP, pois o tempo máximo alocado é igual à fatia de tempo definida no sistema. O escalonamento circular é muito adequado para sistemas de tempo compartilhados, quando existem muitos processos interativos concorrendo pela utilização da UCP.

Uma dificuldade para essa política é que processos CPU-bound se tornam beneficiados no uso do processador comparando com processos I/O-bound, o que pode ocasionar um desbalanceamento no uso do processador entre os processos.

4.4.4 Escalonamento por prioridades

No escalonamento por prioridade, a cada processo é associada uma prioridade, normalmente por um número inteiro, o qual representa sua importância no sistema. Esses valores de prioridade são então utilizados para definição da ordem de execução dos processos.

Ordenando-se os processos na lista de processos prontos por ordem de prioridade, aquele com maior prioridade é sempre o escolhido para execução, e processos com valores iguais são escalonados seguindo a ordem de chegada, segundo o critério FCFS.

A figura 42 apresenta o escalonamento por prioridades e o escalonamento é aplicado alocando o processador ao primeiro processo da fila de prioridade mais alta. Para cada prioridade, existe uma fila de processos em estado de pronto que é tratada como uma fila circular. O processo fica no estado de execução até que termine seu processamento, voluntariamente passe para o estado de espera ou sofra uma preempção devido a um processo de maior prioridade.

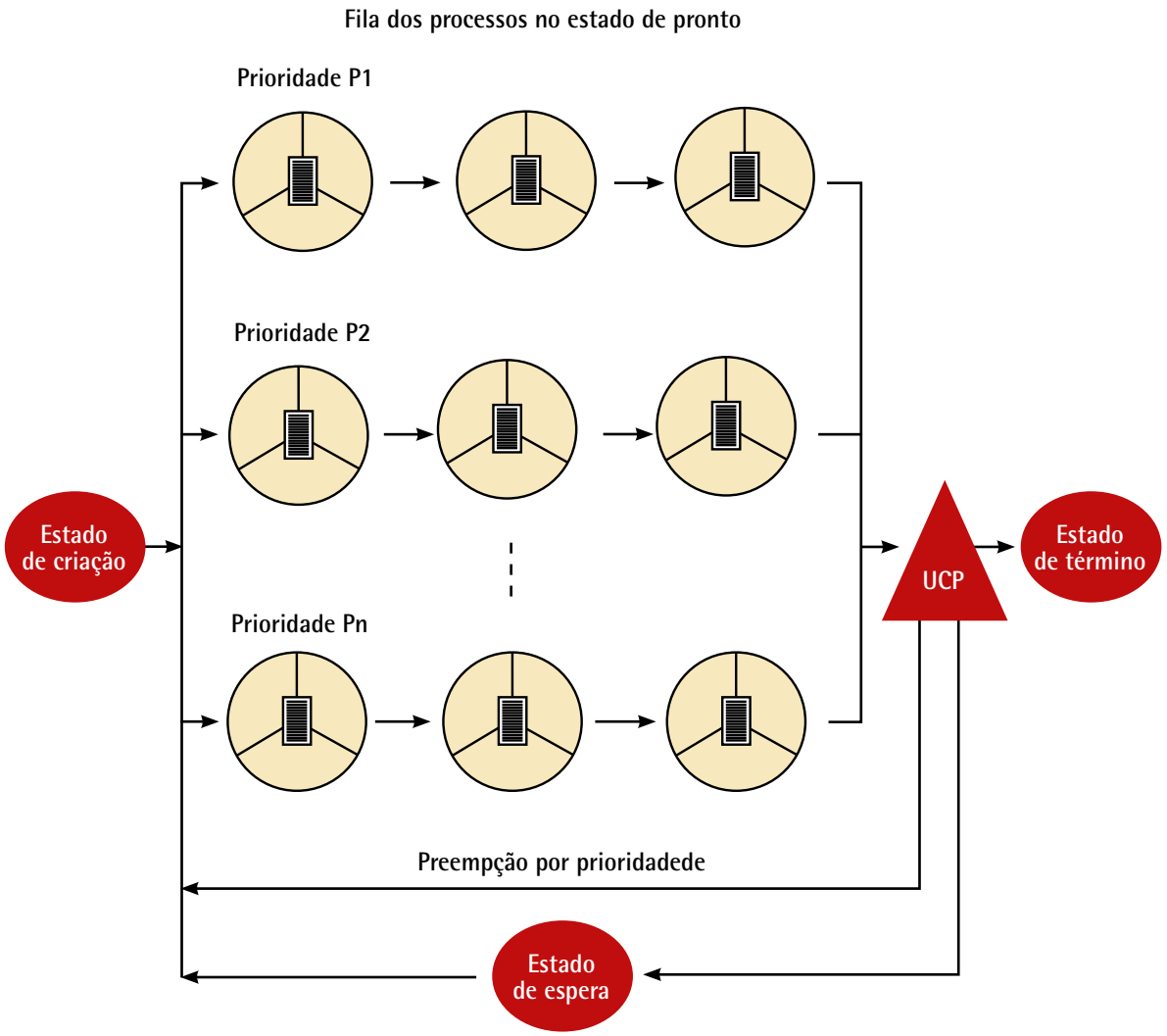


Figura 42 – Escalonamento por prioridades

Fonte: Machado e Maia (2013, p. 135).

Um dos principais problemas no escalonamento por prioridades é a espera indefinida ou starvation, pois processos de baixa prioridade podem não ser escolhidos pelo escalonamento, permanecendo indefinidamente na fila de pronto.

4.4.5 Escalonamento circular com prioridades

O escalonamento circular com prioridades utiliza tanto a fatia de tempo quanto a prioridade de execução associada a cada processo. Um processo permanece no estado de execução enquanto seu processamento não termina, voluntariamente passa para o estado de espera ou sofre uma preempção por tempo ou prioridade. A figura a seguir representa esse tipo de escalonamento.

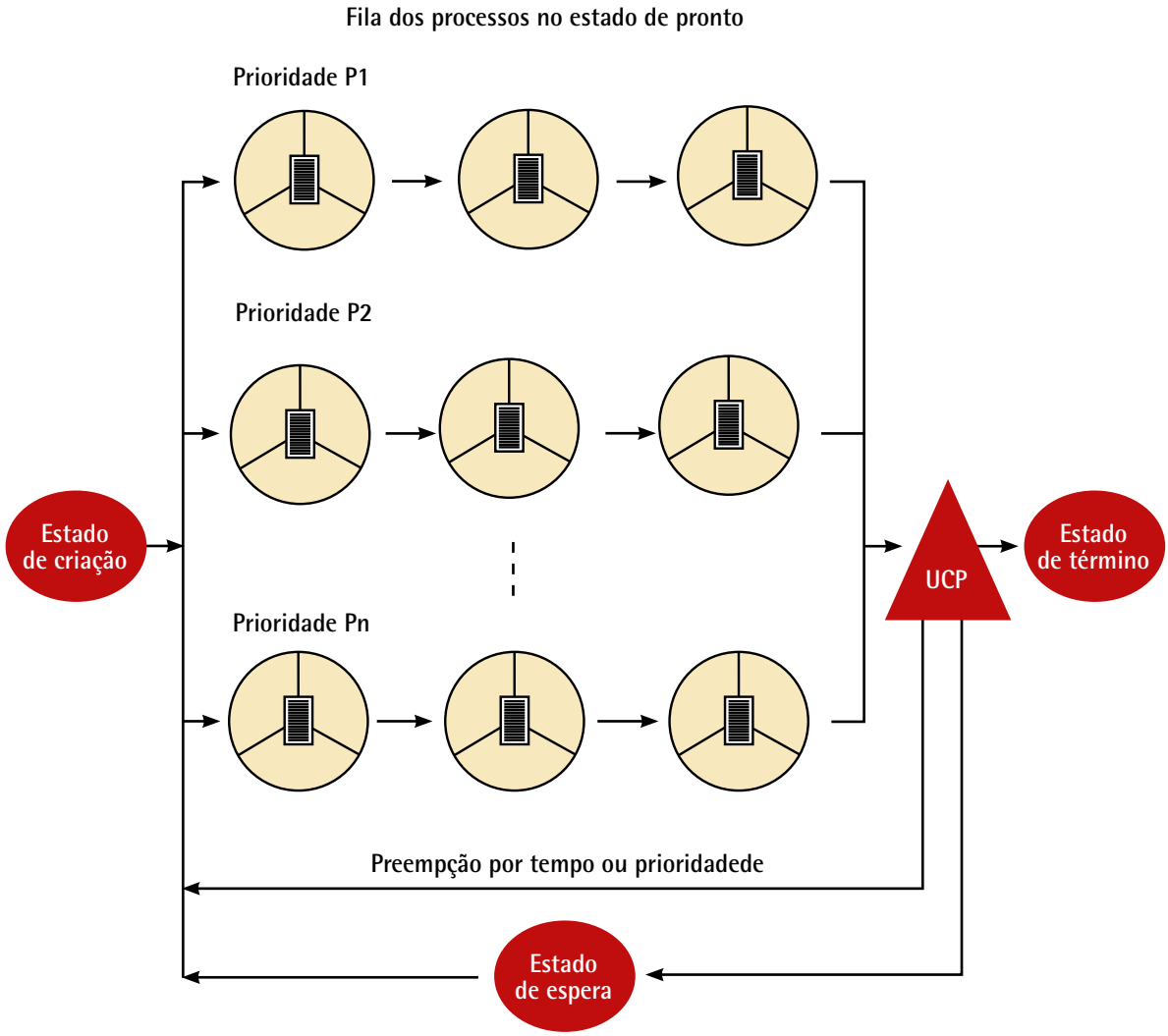


Figura 43 – Escalonamento circular por prioridades

Fonte: Machado e Maia (2013, p. 137).

Nesse escalonamento, sua principal vantagem é obter um melhor balanceamento no uso do processador em sistemas de tempo compartilhado. Tanto processos I/O-bound como CPU-bound compartilharão o processador de forma mais igualitária. Esse tipo de escalonamento é amplamente utilizado em sistemas de tempo compartilhado, como o Windows e o Unix.

4.5 Sistemas operacionais com suporte a múltiplos processadores

Sistemas computacionais modernos utilizam múltiplos processadores, isto é, duas ou mais unidades centrais de processamento (UCPs) interligadas e trabalhando conjuntamente. Sua vantagem é permitir que vários programas sejam executados ao mesmo tempo ou que um mesmo programa seja subdividido em partes para serem executadas simultaneamente em mais de um processador.

Somente nos casos de sistemas com múltiplos processadores é que realmente haverá diversos programas sendo atendidos no mesmo instante. O controle de múltiplas atividades em paralelo é uma prática que está sendo desenvolvida e aprimorada com base em um modelo conceitual de processos sequenciais que facilita o paralelismo.

Com múltiplos processadores foi possível a criação de sistemas computacionais voltados, principalmente, para processamento científico, aplicado, por exemplo, no desenvolvimento aeroespacial, simulações com foco na segurança de sistemas críticos, inteligência artificial, processamento de imagens e vídeos, entre outras aplicações.

De forma geral, qualquer software com uso intensivo da UCP terá benefícios com o aumento de processadores ao sistema. A evolução de sistemas com múltiplos processadores deve-se majoritariamente ao elevado custo de desenvolvimento de processadores de alto desempenho.

Além dos benefícios de uso otimizado dos recursos computacionais alcançado com a multiprogramação, os múltiplos processadores possuem vantagens adicionais como escalabilidade, disponibilidade e balanceamento de carga.

A escalabilidade possibilita ampliar o poder computacional do sistema apenas adicionando novos recursos, no caso, processadores. Em ambientes com um único processador, caso haja problemas de desempenho, seria necessário substituir todo o sistema por outra configuração com maior poder de processamento. Em arquiteturas com múltiplos processadores, ao acrescentar novos processadores à configuração, a capacidade do sistema será ampliada.

Disponibilidade é a capacidade de manter o sistema em operação mesmo em situações de falha de seus componentes. Neste caso, se um dos processadores falhar, os demais podem assumir suas funções de maneira transparente aos usuários e suas aplicações, embora com menor capacidade de computação. Balanceamento de carga é a possibilidade de distribuir o processamento entre os diversos processadores da configuração a partir da carga de trabalho de cada processador, melhorando, assim, o desempenho do sistema como um todo.

São fatores-chave no desenvolvimento de sistemas operacionais com múltiplos processadores a forma de comunicação entre as UCPs e o grau de compartilhamento da memória e dos dispositivos de E/S. Dependendo desses fatores, os sistemas com múltiplos processadores podem ser classificados em fortemente ou fracamente acoplados.

Resumo

Nesta unidade, foi apresentado o conceito de sistema operacional como uma camada intermediária entre o hardware e o software do usuário. As principais funções do sistema operacional são a gerência de recursos, tais como processador, memória, dispositivos de E/S, sistema de arquivos, e a abstração de recursos, permitindo que a aplicação do usuário não precise entender as complexidades do hardware. Assim, como o hardware, o sistema operacional evoluiu em funções realizadas, em complexidade e em interação com usuário final. Para atender diferentes necessidades, foram desenvolvidos tipos de sistemas operacionais a fim de se adequar aos diversos tipos de hardware existentes e com finalidades específicas.

Foram exibidas a evolução histórica dos sistemas operacionais desde os primeiros computadores até os atuais e algumas classificações de sistemas operacionais, como sistemas monotarefa, multitarefa, em lote ou batch, sistemas de tempo compartilhado, entre outros.

Vimos que os principais tipos de arquiteturas do sistema operacional são: monolítico, em camadas, micronúcleo e máquinas virtuais. Eles foram apresentados, e houve um comparativo para destacar as vantagens e desvantagens de cada arquitetura. As principais funções de gerenciamento do sistema operacional foram brevemente descritas: gerência de processador, da memória, de dispositivos de E/S, do sistema de arquivos e da gerência de proteção.

Demonstramos os processos, que são conceitos fundamentais para sistemas operacionais, compostos de um código-fonte, um espaço de endereçamento e um contexto de hardware e software. São estados de um processo: novo, pronto para ser executado, em espera, executando e terminado. Os processos são criados e finalizados continuamente e precisam se comunicar entre si, em especial quando acessam dados compartilhados.

Foram criadas formas para evitar o compartilhamento inconsistente no sistema operacional como a exclusão mútua, semáforo e monitores.

Por fim, entendemos que através do processo de escalonamento o sistema operacional determina o processo a estar em execução por meio da política de escalonamento, implementada pelo escalonador.

Exercícios

Questão 1. (Enade 2021, adaptada) Durante parte do tempo, um processo está ocupado realizando computações internas e outras coisas que não levam a condições de corrida. No entanto, às vezes, ele tem de acessar uma memória compartilhada ou arquivos, ou realizar outras tarefas críticas que podem levar a corridas. É a parte do programa onde a memória compartilhada recebe o nome de região crítica ou seção crítica. Se conseguíssemos arranjar as coisas de maneira que dois processos jamais estivessem em suas regiões críticas ao mesmo tempo, poderíamos evitar as corridas. Embora essa exigência evite as condições de corrida, ela não é suficiente para garantir que processos em paralelo cooperem de modo correto e eficiente usando dados compartilhados. Precisamos que quatro condições se mantenham para chegar a uma boa solução.

1 – Dois processos jamais podem simultaneamente estar em suas regiões críticas.

2 – Nenhuma suposição pode ser feita a respeito de velocidades ou de número de CPUs.

3 – Nenhum processo executando fora de sua região crítica pode bloquear qualquer processo.

4 – Nenhum processo deve ser obrigado a esperar de forma indeterminada para entrar em sua região crítica. Em um sentido abstrato, o comportamento que queremos é mostrado na figura a seguir.

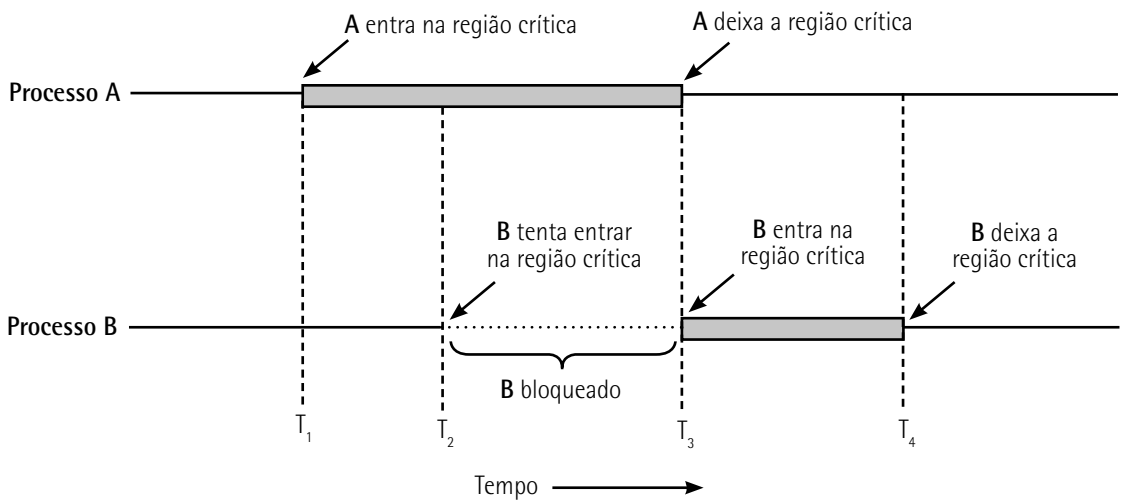


Figura 44 – Exclusão mútua usando regiões críticas

Adaptada de: TANENBAUM, A. S. *Sistemas operacionais modernos*. 4. ed. São Paulo: Pearson Education do Brasil (2016, p. 83).

Considerando o texto e a figura, avalie as asserções a seguir e a relação proposta entre elas.

I – Em algumas situações, a exclusão mútua pode ser obtida por meio da desabilitação da interrupção controlada pelo sistema operacional, não sendo permitido que o seu controle seja feito pelo usuário.

porque

II – A desabilitação da interrupção é uma técnica que pode impedir que o processador que está executando um processo em sua região crítica seja interrompido para executar outro código, sendo mais eficiente em sistemas de multiprocessadores devido à quantidade de processos concorrentes.

A respeito dessas asserções, assinale a opção correta.

A) As asserções I e II são proposições verdadeiras, e a asserção II é uma justificativa correta da I.

B) As asserções I e II são proposições verdadeiras, e a asserção II não é uma justificativa correta da I.

C) A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.

D) A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.

E) As asserções I e II são proposições falsas.

Resposta correta: alternativa C.

Análise das asserções

I – Asserção verdadeira.

Justificativa: região crítica é a parte do código do programa na qual é implementado o acesso ao recurso que será compartilhado por processos. Quando há a garantia de que dois processos não estão em regiões críticas simultaneamente, os problemas de compartilhamento são resolvidos. Quando um processo estiver com acesso ao recurso, todos os outros que necessitem desse mesmo recurso devem aguardar até que o processo finalize a utilização do recurso e saia da região crítica. Essa exclusividade de acesso à região crítica em cada instante é denominada exclusão mútua. O sistema operacional deve assegurar a exclusão mútua, recorrendo aos mecanismos de sincronização.

Uma das soluções de implementação da exclusão mútua é a desabilitação de interrupções. Ao inibir interrupções, não há troca de tarefas com a ocorrência de interrupções, e o processo poderá atualizar a memória compartilhada sem influência de outros processos. Além disso, neste caso, não é conveniente atribuir ao usuário a decisão de habilitar as interrupções.

II – Asserção falsa.

Justificativa: a solução de desabilitar interrupções é aplicável somente a sistemas de processador único, pois apenas uma CPU, a afetada pela instrução bloqueante, terá suas interrupções inibidas. Logo, em sistemas de multiprocessadores, outras soluções de exclusão mútua devem ser aplicadas.

Questão 2. (Enade 2021, adaptada) Leia o texto a seguir, a respeito do escalonador de um sistema operacional.

Quando um computador é multiprogramado, ele geralmente tem múltiplos processos ou threads que competem pela CPU ao mesmo tempo. Essa situação ocorre sempre que dois ou mais processos estão simultaneamente no estado pronto. Se somente uma CPU se encontrar disponível, deverá ser feita uma escolha de qual processo executar em seguida. A parte do sistema operacional que faz a escolha é chamada de **escalonador**, e o algoritmo que ele usa é o **algoritmo de escalonamento**.

Adaptado de: TANENBAUM, A. S. *Sistemas operacionais modernos*. 3. ed. São Paulo: Pearson, 2010.

Considerando que, em ambientes diferentes, são necessários algoritmos diversos de escalonamento, garantindo assim que seja maximizado o uso de seus recursos, assinale a opção que apresenta um algoritmo de escalonamento seguido do tipo de ambiente no qual deva ser implementado.

A) Primeiro a chegar, primeiro a sair (Fifo); propício para sistemas de tempo real.

B) Escalonamento por taxas monotônicas (RMS); propício para sistemas em lote.

C) Tarefa mais curta primeiro; propício para sistemas interativos.

D) Escalonamento por chave circular (round-robin); propício para sistemas de tempo real.

E) Escalonamento por prioridades; propício para sistemas interativos.

Resposta correta: alternativa E.

Análise das alternativas

A) Alternativa incorreta.

Justificativa: o algoritmo FCFS, também conhecido como Fifo, é de escalonamento mais básico, atendendo aos processos sequencialmente à medida que eles chegam à fila de processos prontos. Esse algoritmo é não preemptivo e tem como principal vantagem sua simplicidade. No entanto, sua principal desvantagem é a impossibilidade de previsão de quando um processo começará a ser processado, pois isso depende do tempo de execução dos demais processos posicionados à sua frente na fila de pronto. Por isso, esse algoritmo não é adequado para sistemas operacionais de tempo real, que necessitam manter prazos de tempo de resposta e previsibilidade.

B) Alternativa incorreta.

Justificativa: o escalonamento por taxas monotônicas (RMS) é um algoritmo preemptivo, que atribui prioridades aos processos de acordo com o número de vezes que eles serão executados por unidade de tempo. Quanto maior a frequência de execução, maior a prioridade. Seu uso é adequado para sistemas operacionais de tempo real.

C) Alternativa incorreta.

Justificativa: o algoritmo de tarefa mais curta primeiro (SJF) escolhe o processo que tiver o menor tempo de processador ainda por executar. Dessa forma, o processo em estado de pronto que necessitar de menos tempo de CPU para terminar seu processamento é selecionado para execução. Inicialmente, essa implementação foi aplicada nos primeiros sistemas operacionais em lote. Para cada novo processo admitido no sistema, era associado um tempo de processador ao seu contexto de software. O principal problema nessa implementação é a impossibilidade de estimarmos o tempo de processador para processos interativos, já que a entrada de dados é uma ação imprevisível. Logo, esse não é um algoritmo adequado para sistemas interativos.

D) Alternativa incorreta.

Justificativa: o escalonamento por chave circular (round-robin) parte do princípio dos algoritmos FCFS, mas com a adição do recurso de preempção por tempo. Quando um processo passa para o estado de execução, existe um limite de tempo para o uso do processador de forma exclusiva por um processo, que é denominado time-slice. Esse escalonamento é do tipo preemptivo e foi projetado principalmente para sistemas de tempo compartilhado.

E) Alternativa correta.

Justificativa: no escalonamento por prioridade, a cada processo é associada uma prioridade, normalmente representada por um número inteiro, que indica seu grau de importância. Esses valores de prioridade são então utilizados para definição da ordem de execução dos processos, sendo que o processo com maior prioridade é sempre o escolhido para execução, e processos com valores iguais são escalonados seguindo a ordem de chegada, segundo o FCFS. Um algoritmo para sistemas interativos deve ter como um de seus principais objetivos a minimização do tempo de resposta para usuários, o que pode ser atingido por meio do escalonamento por prioridade, já que o algoritmo se baseia na ideia de que os processos não são igualmente importantes para o sistema ou para o usuário.