Universidade do Sul de Santa Catarina

Introdução ao Ambiente Operacional

UnisulVirtual
Palhoça, 2015

Créditos

Universidade do Sul de Santa Catarina – Unisul

Reitor
Sebastião Salésio Herdt
Vice-Reitor
Mauri Luiz Heerdt

Pró-Reitor de Ensino, de Pesquisa e de Extensão
Mauri Luiz Heerdt
Pró-Reitor de Desenvolvimento Institucional
Luciano Rodrigues Marcelino
Pró-Reitor de Operações e Serviços Acadêmicos
Valter Alves Schmitz Neto

Diretor do Campus Universitário de Tubarão
Heitor Wensing Júnior
Diretor do Campus Universitário da Grande Florianópolis
Hércules Nunes de Araújo
Diretor do Campus Universitário UnisulVirtual
Fabiano Ceretta

Campus Universitário UnisulVirtual

Diretor
Fabiano Ceretta

Unidade de Articulação Acadêmica (UnA) – Ciências Sociais, Direito, Negócios e Serviços
Amanda Pizzolo *(coordenadora)*
Unidade de Articulação Acadêmica (UnA) – Educação, Humanidades e Artes
Felipe Felisbino *(coordenador)*
Unidade de Articulação Acadêmica (UnA) – Produção, Construção e Agroindústria
Anelise Leal Vieira Cubas *(coordenadora)*
Unidade de Articulação Acadêmica (UnA) – Saúde e Bem-estar Social
Aureo dos Santos *(coordenador)*

Gerente de Operações e Serviços Acadêmicos
Moacir Heerdt
Gerente de Ensino, Pesquisa e Extensão
Roberto Iunskovski
Gerente de Desenho, Desenvolvimento e Produção de Recursos Didáticos
Márcia Loch
Gerente de Prospecção Mercadológica
Eliza Bianchini Dallanhol

Márcia Cargnin Martins Giraldi

Introdução ao Ambiente Operacional

Livro didático

Designer instrucional
Marina Cabeda Egger Moellwald

UnisulVirtual
Palhoça, 2015

Livro Didático

Professora conteudista
Márcia Cargnin Martins Giraldi

Designer instrucional
Marina Cabeda Egger Moellwald

Projeto gráfico e capa
Equipe UnisulVirtual

Diagramador(a)
Pedro Teixeira

Revisor(a)
Letra de Forma

005.43
G43 Giraldi, Márcia Cargnin Martins
Introdução ao ambiente operacional : livro didático / Márcia Cargnin Martins Giraldi ; design instrucional Marina Cabeda Egger Moellwald. – Palhoça : UnisulVirtual, 2015.
98 p. : il. ; 28 cm.

Inclui bibliografia.

1. Sistemas operacionais (computadores). I. Moellwald, Marina Cabeda Egger. II. Título.

Ficha catalográfica elaborada pela Biblioteca Universitária da Unisul

Sumário

Introdução

Este livro foi escrito de forma simples e objetiva para que você entre no universo dos ambientes operacionais.

Com o desenvolvimento da área de tecnologia da informação, muitos avanços ocorreram tanto no hardware quanto no software. Durante sua leitura, você poderá acompanhar essa transformação, estudando a evolução das funcionalidades do sistema operacional.

Ao término da leitura, é esperado que você compreenda como um sistema operacional pode ser organizado, como é feita a gerência do processador, como os processos são alocados na memória, como um sistema de arquivos é organizado e como acontece a comunicação entre os dispositivos de entrada e saída.

Dessa forma, tenho como objetivo proporcionar a você fundamentos que o habilitem a reconhecer e a compreender características básicas de um sistema operacional.

Sinta-se extremamente bem-vindo!

Abraço,
Márcia Cargnin Martins Giraldi

Capítulo 1

O sistema operacional: conceitos e comandos básicos

Habilidades

Este capítulo foi desenvolvido para que você conheça as características básicas de um sistema operacional, compreenda sua estrutura básica e aprenda a identificar os sistemas de código aberto e os proprietários, bem como os conceitos básicos para gerenciar um sistema operacional.

Seções de estudo

Seção 1: Conceitos básicos

Seção 2: Estrutura do sistema operacional

Seção 3: Sistemas operacionais: proprietário e de código aberto

Seção 4: Gerenciamento de contas

Seção 1
Conceitos básicos

Neste capítulo, você terá a oportunidade de conhecer os aspectos básicos de um sistema operacional, que permitirá que você faça distinções entre os vários tipos e estruturas dos sistemas operacionais.

Embora as funções de um sistema operacional sejam sempre as mesmas, a forma como ele as desempenha pode ser diferente. Também temos sistemas operacionais que permitem que seu código seja alterado, adaptado, enquanto outros não permitem nem sua visualização.

Alguns conceitos que serão estudados neste capítulo poderão auxiliar você a compreender por que existe tanta variedade de sistemas operacionais no mercado e que a escolha por um deles depende da área onde ele será utilizado.

Vamos iniciar conhecendo um pouco mais sobre os sistemas operacionais, suas funções, estrutura e principais conceitos. Um sistema operacional nada mais é do que um conjunto de rotinas (programas) executado pelo processador, da mesma forma que outros programas são executados.

Na figura 1.1, podemos observar que o sistema operacional pode ser visto como a interface entre o usuário e o hardware:

Figura 1.1 - Visão de um sistema operacional

Fonte: Adaptação de Machado e Maia (2007, p. 4).

Podemos categorizar as inúmeras funções de um sistema operacional em duas:

- facilitar o acesso aos recursos do sistema; e
- compartilhar os recursos de forma organizada e protegida.

Sistemas multiusuários onde vários usuários podem utilizar o mesmo recurso de forma concorrente.

Quando pensamos em **sistemas multiusuários**, é necessário que todos tenham chance de utilizar o recurso, de forma que um usuário não interfira no trabalho do outro. O sistema operacional permite que diversos usuários acessem, concorrentemente, os vários recursos do sistema, de forma organizada e segura, dando a impressão ao usuário de estar trabalhando sozinho.

Podemos visualizar o computador como sendo uma máquina de dois níveis, contendo:

- hardware (parte física); e
- software (programas).

No seguinte quadro, visualizamos o que consta em cada um desses níveis:

Quadro 1.1 - Máquinas de camadas

Software	Aplicação
	Utilitários
	Sistema operacional
Hardware	Linguagem de máquina
	Microprogramação
	Dispositivos físicos

Fonte: Adaptação de Machado e Maia (2007, p. 6).

Sabemos que o hardware sozinho não passa de um gabinete cheio de fios e circuitos eletrônicos sem utilidade. É por meio do software que conseguimos dar vida ao computador e realizar todas as tarefas de que necessitamos.

Nos primeiros computadores, a programação era feita em painéis de fios, exigindo dos programadores um grande conhecimento do hardware e de sua linguagem de máquina.

Com o surgimento do sistema operacional, a complexidade do hardware não tem importância para o programador, ou seja, a parte física ficou transparente para o usuário. Na realidade, esses dois níveis podem estar divididos em quantos forem necessários para atender às necessidades do usuário (TANENBAUM, 2009).

Agora, vamos rever alguns aspectos sobre o hardware do computador, que nos auxiliarão a compreender melhor o sistema operacional e suas funções.

1.1 Hardware

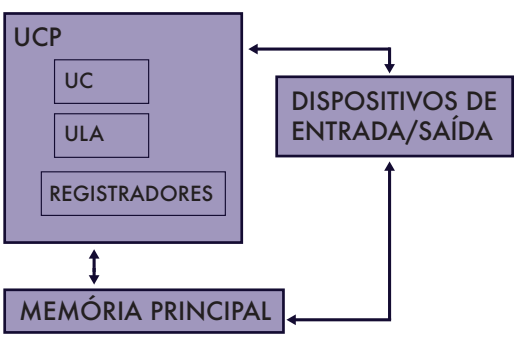
Um **computador digital** é constituído por um conjunto de componentes interligados, composto por processadores, memória principal, registradores, terminais, impressoras e discos, entre outros dispositivos. Esses dispositivos manipulam os dados de forma digital, o que proporciona uma maneira confiável de representação (MACHADO; MAIA, 2007).

Todo sistema computacional pode ser visto como o conjunto de três unidades funcionais:

- processador ou unidade central de processamento;
- memória principal; e
- dispositivos de entrada/saída.

Vejamos essa disposição na seguinte figura:

Figura 1.2 - Organização básica de um computador



Fonte: Adaptação de Machado e Maia (2007, p. 25).

O **processador** ou a **unidade central de processamento (UCP)** tem como função principal unificar todo o sistema, controlando todas as funções realizadas por cada componente do computador (unidade funcional); é responsável pela

execução de todos os programas do sistema que, obrigatoriamente, terão que estar carregados na memória principal.

Um **programa** é composto por uma série de instruções que são executadas sequencialmente pela UCP, por meio de operações básicas, como: somar, subtrair, comparar e mover. Dessa forma, a UCP busca a instrução na memória principal e a interpreta para sua execução.

São componentes da UCP:

- Unidade de controle (UC), responsável pelo controle das atividades de todos os componentes do computador, por meio de pulsos elétricos.
- Unidade lógica e aritmética (ULA), responsável pelas operações lógicas (comparações) e aritméticas (soma e subtração).

A velocidade de processamento da UCP é determinada pelo número de instruções que o processador executa por unidade de tempo (HENNESSY, 2000).

- Clock é um dispositivo localizado na UCP que gera pulsos elétricos constantes em um mesmo intervalo de tempo (sinal do clock). O **sinal do clock** é utilizado pela UC para execução das instruções. Uma instrução pode demorar vários ciclos do clock para ser executada. O ciclo do clock é o intervalo de tempo entre os sinais do clock. Para calcular o tempo de execução, utilizamos a seguinte fórmula:

  tempo_execução = nº de estados x ciclo do clock
- Registradores são dispositivos de alta velocidade, localizados fisicamente na UCP, para armazenamento temporário dos dados. O número de registradores varia em função da arquitetura de cada processador. Alguns registradores são de uso específico e têm propósitos especiais; outros são de uso geral.

Os três principais registradores de uso específico são:

- Contador de instruções (CI) ou program counter (PC).
- Apontador da pilha (AP) ou stack pointer (SP).
- Registrador de estado (RST) ou program status word (PSW).

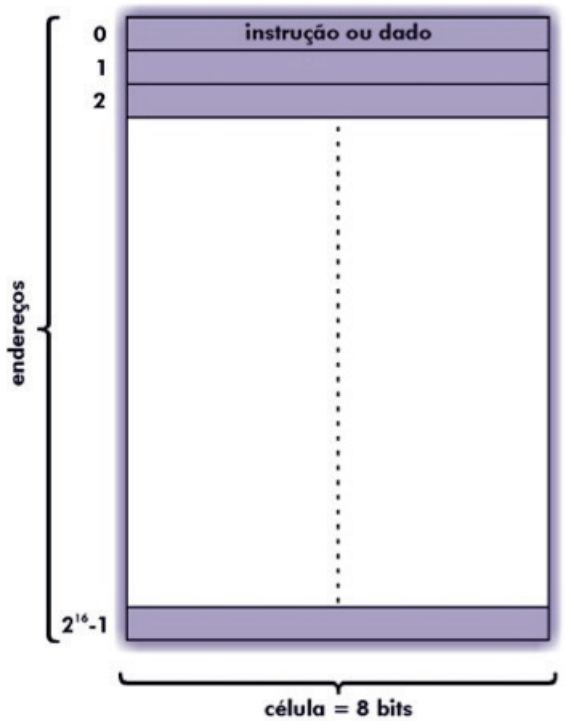
O CI (ou PC) contém o endereço da próxima instrução a ser executada pela UCP. O AP (ou SP) contém o endereço de memória do topo da **pilha**, ou seja, da estrutura onde o sistema mantém informações sobre os processos que foram interrompidos por algum motivo. A cada instrução executada, o RST (ou PSW) é alterado conforme o resultado gerado pela instrução, como a ocorrência de **carry** ou **overflow**.

> Em uma operação aritmética, pode acontecer um "vai um", denominado carry. Overflow ocorre quando o resultado da operação aritmética é maior do que o espaço reservado para ele (estouro de campo).

Estudamos a primeira unidade funcional do sistema computacional; agora, vamos à segunda.

A **memória principal** (primária ou real) é o local do computador onde são armazenados dados e instruções. Ela é composta por unidades (células) com um número determinado de bits (bynary digit), podendo chegar, hoje, a 64 bits (MACHADO; MAIA, 2007), conforme a seguinte figura:

Figura 1.3 - Exemplo de uma memória principal com 64 Kbytes



Fonte: Machado e Maia (2013, p. 25).

Além da memória principal, existem outras memórias de extrema importância.

A **memória cache** é uma memória de alta velocidade. O tempo de acesso ao dado contido nela é muito menor do que se ele estivesse na memória principal. Toda vez que o processador faz referência a um dado armazenado na memória principal, ele "olha" primeiro na cache. Se o dado não estiver na cache, ele terá que acessar a memória principal e transferir um bloco de dados da memória principal para a cache. O tempo de transferência entre as memórias é pequeno, se comparado ao aumento de desempenho obtido com a utilização desse tipo de memória. Seu uso é limitado por seu alto custo.

A **memória secundária** é um meio permanente, ou seja, não volátil de armazenamento de programas e dados. O acesso a esse tipo de memória é lento, porém, seu custo é baixo e sua capacidade de armazenamento é bem superior à da memória principal.

Dispositivo de **armazenamento não volátil** é aquele que, mesmo sem fonte de alimentação, mantém os dados gravados. Isso não acontece com as memórias principal e cache, pois são voláteis e só mantêm os dados enquanto estiverem ligadas a uma fonte de energia.

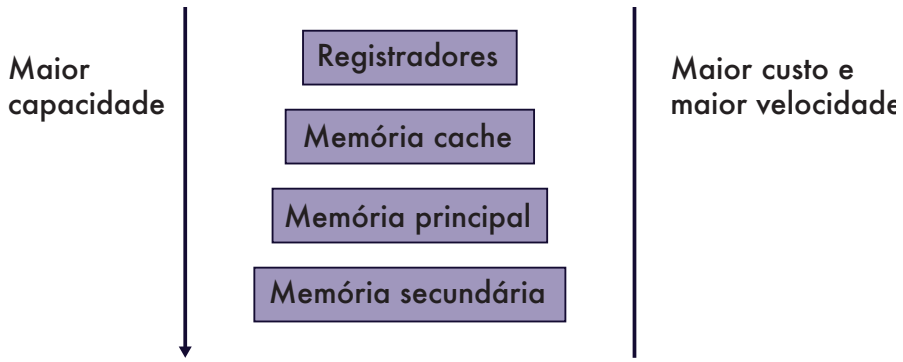
A terceira unidade funcional do sistema computacional são os **dispositivos de entrada e saída (e/s)**, utilizados para permitir a comunicação entre o computador e o mundo externo.

De acordo com Machado e Maia (2007), os dispositivos de e/s podem ser classificados como:

- Memória secundária (discos, fitas magnéticas, discos óticos, entre outros).
- Interface homem-máquina (teclado, monitor de vídeo, impressoras, plotter, scanner, caneta ótica, mouse, entre outros).

Vejamos a relação entre esses dispositivos na seguinte figura:

Figura 1.4 - Relação entre os diversos tipos de armazenamento



Fonte: Adaptação de Machado e Maia (2007, p. 29).

Os componentes anteriores (UCP, memória principal e dispositivos de e/s) são interligados por meio de linhas de comunicação denominadas de **barramentos**. É um conjunto de condutores pelos quais trafegam informações, como dados, endereços ou sinais de controle.

A inicialização do sistema (boot) remete ao processo em que o sistema operacional é carregado da memória secundária para a principal. É realizado por um programa localizado em uma posição específica do disco (boot block) – geralmente, o primeiro bloco.

1.2 Software

> Para que o hardware tenha utilidade prática, deve existir um conjunto de programas utilizado como interface entre as necessidades do usuário e as capacidades do hardware. A utilização de softwares adequados às diversas tarefas e aplicações torna o trabalho dos usuários muito mais simples e eficiente. (MACHADO; MAIA, 2007, p. 34).

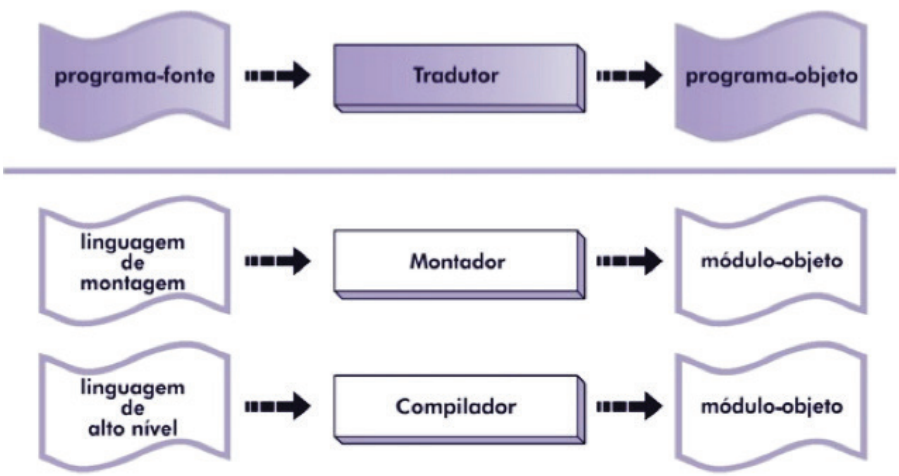
Os softwares relacionados mais diretamente com serviços do sistema operacional, como compiladores, linkers e depuradores, podem ser chamados de utilitários. Os softwares desenvolvidos pelos usuários são chamados de softwares aplicativos ou aplicações.

Vamos conhecer alguns dos **softwares utilitários**:

- Tradutor (interpretador, compilador e montador).
- Linker.
- Loader.
- Depurador.

Tradutor é o utilitário que faz a conversão de uma linguagem que o computador não entende (linguagem de montagem e de alto nível) para uma que ele conhece (linguagem de máquina). Apesar da tradução, o módulo-objeto, como é chamado o resultado da tradução, ainda não pode ser executado.

Figura 1.5 - Tradutor



Fonte: Machado e Maia (2013, p. 32).

O **montador**, também conhecido como assembler, é o utilitário responsável por gerar, a partir de um programa escrito em linguagem de montagem, um programa em linguagem de máquina não executável (módulo-objeto). A linguagem de montagem está diretamente ligada às características da arquitetura do processador (MACHADO; MAIA, 2007).

Linguagens de alto nível Pascal, Fortran, Cobol, Delphi, C, entre outras.

O **compilador** é o utilitário responsável por gerar, a partir de um programa escrito em linguagem de alto nível, um programa em linguagem de máquina não executável (módulo-objeto). As **linguagens de alto nível** não têm nenhuma relação direta com a máquina, ficando essa preocupação exclusivamente com o compilador.

O **interpretador** é um tradutor que não gera código-objeto. A partir de um programa-fonte, escrito em linguagem de alto nível, o interpretador, no momento da execução do programa, traduz cada instrução e a executa em seguida.

Todo sistema operacional possui um conjunto de comandos com o objetivo de permitir a comunicação entre ele e o usuário. Esses comandos, após digitados pelos usuários, são interpretados por um programa denominado **interpretador de comandos** ou **shell**. O interpretador reconhece a linha de comando, verifica sua sintaxe, envia mensagens de erro e faz chamada a rotinas do sistema. Dessa forma, o usuário dispõe de uma interface interativa com o sistema operacional para realizar tarefas como acessar um arquivo em disco ou consultar um diretório.

O **linker** (ligador), também chamado de linkage editor (editor de ligação), é o utilitário responsável por gerar, a partir de um ou mais módulos-objeto, um único programa executável. O **loader**, também chamado de carregador, é o responsável por colocar fisicamente na memória um programa para execução. O **depurador** permite ao usuário controlar toda a execução de um programa, a fim de detectar erros em sua estrutura.

Detalhes como: registradores, modos de endereçamento e tipos de dados.

Linguagem de máquina é a linguagem de programação que o computador realmente consegue entender. Cada processador possui um conjunto único de instruções de máquina, definido pelo próprio fabricante. As instruções especificam **detalhes** que caracterizam um processador e sua potencialidade.

A camada chamada de **microprogramação** é formada por microinstruções que interpretam a linguagem de máquina de cada processador. Normalmente, esses microprogramas estão gravados numa memória do tipo read-only (ROM). Esse microcódigo ou conjunto de microprogramas, na verdade, é um interpretador, que busca as **instruções de máquina** na memória principal (ADD, MOVE, JUMP, entre outras), gerando o conjunto de sinais de controle necessários à execução de tais instruções pelo hardware. Para cada instrução de linguagem de máquina, existe um microprograma associado.

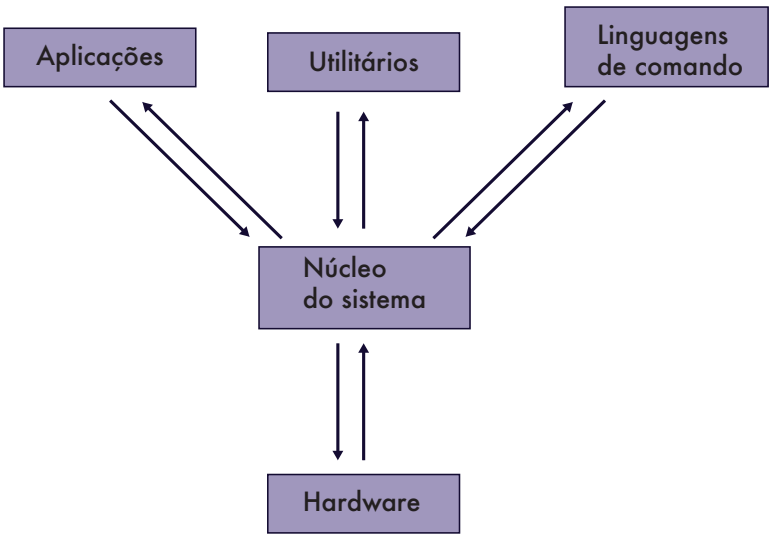
Seção 2
Estrutura do sistema peracional

Como pudemos perceber, o sistema operacional é formado por um conjunto de rotinas (programas) que oferecem serviços:

- aos usuários;
- às aplicações dos usuários; e
- ao próprio sistema operacional.

A estrutura do sistema operacional, ou seja, a forma como essas rotinas estão organizadas e como atendem aos serviços solicitados, varia conforme o projeto do sistema. Chamamos esse conjunto de rotinas de **núcleo do sistema** ou **kernel**. As demais aplicações, como os utilitários, interpretadores de comando (shell), não fazem parte do núcleo do sistema, apenas acompanham o sistema operacional, como mostra a seguinte figura:

Figura 1.6 - Estrutura do sistema operacional



Fonte: Adaptação de Machado e Maia (2007, p. 51).

Segundo Machado e Maia (2007), as principais funções do núcleo do sistema são:

- Tratamento de interrupções e exceções.
- Criação e eliminação de processos e threads.
- Sincronização e comunicação entre processos e threads.
- Escalonamento e controle de processos e threads.
- Gerência de memória.
- Suporte a redes locais e distribuída.
- Contabilização do uso do sistema.
- Auditoria e segurança do sistema.
- Gerência do sistema de arquivos.
- Gerência de dispositivos de e/s.

Antes de estudarmos os principais tipos de estruturas de sistemas operacionais, vamos ver alguns conceitos que envolvem a proteção do sistema:

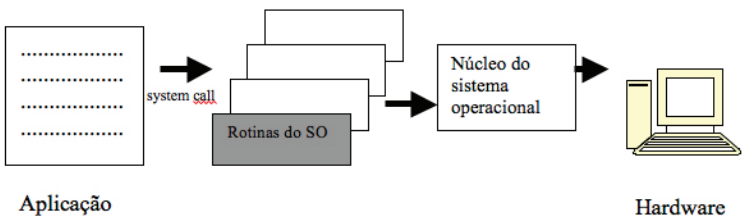
- system calls; e
- modos de acesso.

A chamada ao sistema, **system call**, é um mecanismo criado para proteger o núcleo do sistema operacional. É a porta de entrada para o núcleo do sistema. Sempre que um usuário ou aplicação necessita de um serviço do sistema operacional, é realizada uma chamada a uma de suas rotinas por meio de uma system call.

O termo system call é tipicamente utilizado em sistemas Unix, mas outros termos são utilizados com a mesma função, como system services, no Open VMS, e Application Program Interface (API), no Windows da Microsoft.

Para cada serviço de um determinado sistema operacional, existe uma system call com parâmetros e formas de ativação específicos:

Figura 1.7 - System call



Fonte: Machado e Maia (2007, p. 54).

Por isso, as aplicações que são desenvolvidas para serem executadas em um determinado sistema não são portáveis para outros.

Um simples comando de leitura a um arquivo, utilizando uma linguagem de alto nível depois de compilada, envolve uma chamada de sistema especifica que, quando executada, verifica a ocorrência de erros e retorna os dados ao programa de forma transparente ao usuário.

Como as system calls, os **modos de acesso** também fazem parte da política de proteção do sistema. Algumas operações, quando realizadas indevidamente, podem comprometer o sistema como um todo.

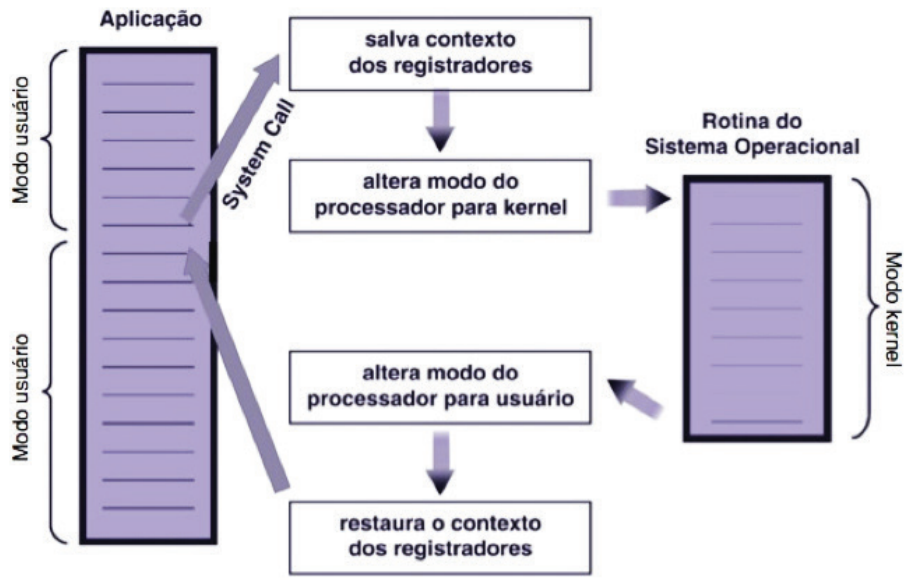
> Por exemplo: uma alteração realizada em um arquivo no disco de uso compartilhado. Essa atualização é realizada pelo sistema operacional para evitar que o programa do usuário faça a atualização de forma errada.

Há dois tipos de instruções ao sistema:

- as que podem comprometer o sistema, chamadas de instruções privilegiadas; e
- as que não oferecem risco ao sistema, chamadas de instruções não-privilegiadas.

Para que uma **instrução privilegiada** possa ser executada, o sistema deve estar em modo supervisor ou kernel. Sempre acontece uma chamada de sistema, o sistema é alterado de modo usuário para modo kernel, conforme a seguinte figura:

Figura 1.8 - Chamada a uma rotina do sistema



Fonte: Machado e Maia (2013, p. 49).

Os modos de acesso são determinados por meio de um bit localizado no registrador de estados (PSW). Por meio desse registrador, o hardware verifica se a instrução privilegiada pode ou não ser executada. Isso evita que programadores mal intencionados possam executar instruções que venham a prejudicar o sistema.

Para melhor proteger o sistema, o ideal é que apenas o sistema operacional execute as operações privilegiadas e só ele possa trabalhar em modo kernel.

A **linguagem de controle** é um conjunto de comandos oferecidos pelo sistema operacional para servir de comunicação entre ele e o usuário. Esses comandos são digitados pelos usuários e interpretados por um programa denominado interpretador de comandos ou **shell**.

O interpretador reconhece a linha de comando, verifica sua sintaxe, envia mensagens de erro e faz chamada a rotinas do sistema. Dessa forma, o usuário dispõe de uma interface interativa com o sistema operacional para realizar tarefas como acessar um arquivo em disco ou consultar um diretório.

A seguir vamos abordar os principais tipos de arquitetura dos sistemas operacionais:

- Arquitetura monolítica.
- Arquitetura em camadas.
- Máquina virtual.
- Arquitetura microkernel.

A **arquitetura monolítica** foi utilizada no sistema operacional MS-DOS e nos primeiros sistemas Unix. As rotinas do sistema são desenvolvidas e depois compiladas e linkadas, formando um único executável. A grande desvantagem desse sistema é que é de difícil manutenção e depuração; por outro lado, apresenta ótimo desempenho e muita simplicidade.

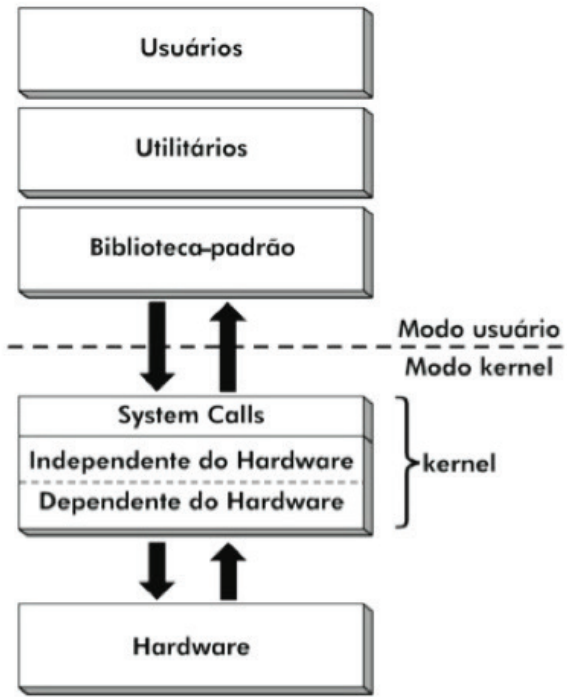
Com o aumento das funcionalidades, complexidade e tamanho dos códigos dos sistemas operacionais, as técnicas de análise e programação estruturadas passaram a ser utilizadas no desenvolvimento dos sistemas operacionais.

Na **arquitetura em camadas**, o sistema é dividido em camadas sobrepostas, onde cada camada possui funções que atendem às camadas superiores. A ideia dessa abordagem é isolar as funções do sistema operacional, facilitando a depuração e a manutenção, além de criar uma hierarquia de acesso ao kernel, possibilitando maior segurança.

Exemplos de sistemas com este tipo de projeto: THE (seis camadas, 1968), MULTICS e Open VMS (kernel, executivo, supervisor, usuário).

Hoje em dia, a maioria dos sistemas Unix é desenvolvida em camadas, conforme a seguinte figura, assim como o Windows 2000, possuindo apenas duas camadas (modo usuário e modo kernel).

Figura 1.9 - Estrutura do sistema operacional Unix



Fonte: Machado; Maia (2013, p. 288).

A **máquina virtual** é um tipo de estrutura que apresenta uma camada intermediária entre o hardware e o sistema operacional, chamada de **gerência de máquinas virtuais**. Esse nível cria diversas máquinas virtuais independentes, onde cada uma oferece uma cópia virtual do hardware, incluindo os modos de acesso, interrupções, dispositivos de e/s etc.

Diferentes como o sistema VM/370 da IBM (década de 1960).

Como as máquinas virtuais são independentes, é possível que cada uma delas possua seu próprio sistema operacional, podendo, inclusive, ser **diferentes**.

Um exemplo atual desse tipo de arquitetura é a máquina virtual JAVA (JVM). Para executar um programa JAVA, é necessária uma máquina virtual JAVA. Qualquer sistema operacional pode suportar uma aplicação JAVA, desde que possua uma JVM desenvolvida para ele. Essas aplicações têm a facilidade de serem executadas em qualquer máquina com diferentes sistemas operacionais que possuam JVM.

Quanto à **arquitetura microkernel**, de acordo com Machado e Maia (2007), a tendência nos sistemas operacionais modernos é tornar o núcleo do sistema menor e mais simples. Para isso, os serviços do sistema operacional são

oferecidos por meio de processos que recebem pedidos das aplicações, executam o pedido e devolvem a resposta à aplicação.

O Kernel fica encarregado apenas de controlar a comunicação entre o cliente (aplicação) e o servidor (rotinas do sistema operacional). “A utilização deste modelo permite que os servidores executem em modo usuário, ou seja, não tenham acesso direto a certos componentes do sistema.” (MACHADO; MAIA, 2007, p. 61).

O núcleo do sistema, que é responsável pela comunicação entre os clientes e os servidores, executa as instruções em modo kernel.

Uma tendência no **projeto de sistemas operacionais** é a utilização de técnicas de orientação por objetos.

Segundo Machado e Maia (2002), esse tipo de projeto permite:

- melhoria na organização das funções e nos recursos do sistema;
- redução no tempo de desenvolvimento;
- maior facilidade na manutenção e na extensão do sistema; e
- facilidade de implementação do modelo de computação distribuída.

A utilização de linguagens de alto nível no desenvolvimento de sistemas operacionais permite que ele seja portável para outras plataformas de hardware. Uma das desvantagens é a perda de desempenho em relação às rotinas desenvolvidas em assembly (linguagem de baixo nível). Por isso, partes críticas do sistema – como o device drivers, o escalonador e as rotinas de tratamento de interrupções – são desenvolvidas em assembly.

Periféricos tais como: discos rígidos, teclados, mouse, monitores, interfaces de redes.

Device drivers são compostos por um conjunto de funções e de estruturas de dados que controlam um ou mais **periféricos**. O **escalonador** é a rotina do sistema operacional responsável por selecionar o processo que irá utilizar o processador.

Seção 3
Sistemas operacionais: proprietário e de código aberto

Nesta seção, vamos conhecer um pouco mais sobre dois tipos de software:

- proprietário; e
- de código aberto.

Assim como qualquer outro software, um sistema operacional pode ser concebido com licença de código aberto ou como um software proprietário.

Qual é a diferença entre essas duas concepções?

Podemos resumir a resposta dizendo que **software proprietário** é aquele que tem um "dono", ou seja, detentor dos direitos autorais e que não disponibiliza seu código-fonte. O **software livre** é uma expressão utilizada para designar qualquer programa de computador que possui liberdades de poder ser executado, copiado, modificado e redistribuído.

É importante, também, diferenciar software livre e código aberto.

> O Software Livre é um tipo de software de código aberto. As duas coisas não são sinônimas. Na prática, Software Livre é um movimento, é o tipo de software produzido sob a licença GPL e derivadas. Já o produto de código aberto é aquele em que existe, à disposição do usuário, o código do produto em linguagem de programação, de forma gratuita ou a custos razoáveis. (MADEIRA, 2011, p. 19).

No início da era da computação, considerava-se que o maior valor estava no hardware, mas, com o passar o tempo, essa situação se inverteu. Agora, o maior valor está no software, pois é ele que faz a diferença para os usuários.

Importante lembrar que o fato de um sistema operacional ser livre não significa que o usuário não tenha que pagar pelo uso, e sim que ele pode modifica-lo e adequá-lo de acordo com suas necessidades. Portanto, a grande diferença, entre um software/sistema operacional proprietário e um livre se refere à **propriedade intelectual**.

Um exemplo de sistema operacional proprietário é o Windows (que você compra o “pacote” e não pode modificá-lo em sua codificação). Um exemplo de sistema operacional livre é o Linux em suas várias distribuições – algumas pagas e outras gratuitas.

Num sistema operacional opensource (código aberto), todos podem ter acesso ao seu código-fonte. Essa é uma das grandes vantagens desse tipo de software, pois muitas pessoas estão detectando erros e falhas de segurança, fazendo o software evoluir mais rapidamente e ser mais seguro. Além disso, qualquer pessoa pode modificá-lo e adaptá-lo às suas necessidades, bem como copiá-lo e distribuí-lo.

Nos sistemas operacionais proprietários, o código-fonte do sistema está restrito aos programadores do software, ou seja, aos programadores da empresa proprietária.

Seção 4
Gerenciamento de contas

Nesta seção, vamos conhecer alguns conceitos importantes para o gerenciamento de contas de usuários e grupos.

Os conceitos de conta de usuário e grupo são fundamentais para entender e implementar um servidor.

Uma **conta de usuário** é um conjunto (nome de acesso e senha) que possibilita ao usuário acessar sua área do sistema. Sendo assim, a mesma pessoa pode possuir várias contas; basta que sejam criadas com nomes de acesso diferentes.

O usuário deve ter sido previamente cadastrado pelo administrador do sistema. Por uma questão de segurança, o sistema diferencia o que cada pessoa faz e o recurso que a pessoa está usando. Quando solicitado, o usuário deve informar seu **username** (login) logo após sua senha.

Basicamente, existem dois tipos de contas:

- a **conta do usuário comum**, que utiliza o sistema e suas ferramentas; e
- a **conta de superusuário** ou conta de root, onde é possível realizar as configurações do sistema.

A senha de root é muito importante, pois algumas configurações só são possíveis por meio dela.

O **superusuário** é, portanto, aquele usuário que tem plenos poderes dentro do Linux. Ele é quem pode criar novos usuários, alterar direitos, configurar e fazer a atualização do sistema. É recomendado utilizar essa conta o mínimo possível para evitar que algum erro danifique o sistema.

Por não possuir restrições de segurança, a **conta de root** só deve ser usada para manutenções no sistema e, de preferência, o menor tempo possível. Para evitar permissões desnecessárias no sistema, deve-se usar uma conta normal de usuário em operações comuns. Se um arquivo for executado pelo usuário root e contiver comandos ou códigos que causem danos ao sistema, esses danos poderão ser irreparáveis.

Esse mesmo superusuário no Windows é chamado de **administrador**.

Em relação ao **grupo**, no Linux é possível fazer o agrupamento de vários usuários que devem compartilhar algumas características comuns, como o acesso a arquivos e dispositivos. Um grupo é basicamente um conjunto de usuários. Geralmente ele é criado quando se deseja que usuários tenham permissão restrita a arquivos em comum. Isso é útil para garantir privacidade e segurança dos dados.

Uma vez identificado pelo sistema, o usuário recebe a linha de comando (prompt) onde poderá executar seus comandos. Quando a conexão for feita pelo administrador do sistema (username = root) o prompt a ser exibido é o #. O sistema UNIX e todas as distribuições Linux são sensíveis a letras maiúsculas e minúsculas, tratando-as como letras diferentes. Por questão de padronização, o nome do usuário e a senha serão sempre minúsculos.

Para encerrar a sessão, digita-se o comando exit na linha de comando ou as teclas Ctrl-D ou logout. Para evitar danos ao sistema de arquivos, é necessário que o superusuário pare o sistema antes de desligar o computador.

Vejamos algumas das formas de desligar/reiniciar o sistema:

- Shutdown –h now (desliga agora).
- Shutdown –h –t 30 “O sistema será desligado em 30 segundos” (desliga em 30 segundos).
- Shutdown –r now (reinicia o sistema).
- Halt – desliga o sistema.
- Reboot – reinicia o sistema.

Quando solicitar que o sistema seja desligado, aguarde até que ele envie a mensagem “O sistema está parado”.

Capítulo 2

Tipos de sistemas operacionais

Habilidades

Pela leitura deste capítulo, você conhecerá os tipos de sistemas operacionais e aprenderá a classificá-los entre monoprogramável, multiprogramável e os que administram múltiplos processadores. Também aprenderá a distinguir os tipos de sistemas multiprogramáveis e suas aplicações.

Seções de estudo

Seção 1: Evolução dos sistemas operacionais

Seção 2: Sistemas operacionais monoprogramáveis e multiprogramáveis

Seção 3: Sistemas operacionais com múltiplos processadores

Seção 1
Evolução dos sistemas operacionais

Antes de iniciarmos o estudo sobre os tipos de sistemas operacionais, vamos acompanhar a evolução desse software, observando que ela está relacionada diretamente à evolução do hardware.

1.1 Breve histórico

De acordo com Machado e Maia (2013, p.6), "a evolução dos sistemas operacionais está, em grande parte, relacionada ao desenvolvimento dos computadores". Esse desenvolvimento proporcionou a fabricação de equipamentos cada vez mais velozes, compactos e de baixo custo.

Em seu livro "Sistemas Operacionais Modernos", Tanenbaum (2009) organiza a história dos sistemas operacionais em **gerações**. Na sequência, apontamos as principais inovações e acontecimentos das seguintes fases:

- Primeira geração (1945-1955): válvulas.
- Segunda geração (1956-1965): transistores e sistemas em lote (batch).
- Terceira geração (1966-1980): circuitos integrados e multiprogramação.
- Quarta geração (1981 - presente): computadores pessoais.

Esquema de hora blocada a máquina ficava disponível para um usuário durante um certo período de tempo.

Os primeiros computadores digitais surgiram no início da Segunda Guerra e eram formados por milhares de válvulas, ocupando enormes áreas, e de funcionamento lento e duvidoso. Trata-se da **primeira geração dos sistemas operacionais**, entre os anos de 1945 e 1955. Os computadores eram construídos com válvulas e exigiam do operador/programador conhecimento profundo do hardware; a programação era feita em linguagem de máquina, diretamente nos painéis, por meio de fios; não existia o conceito de sistema operacional e havia o **esquema de hora blocada**.

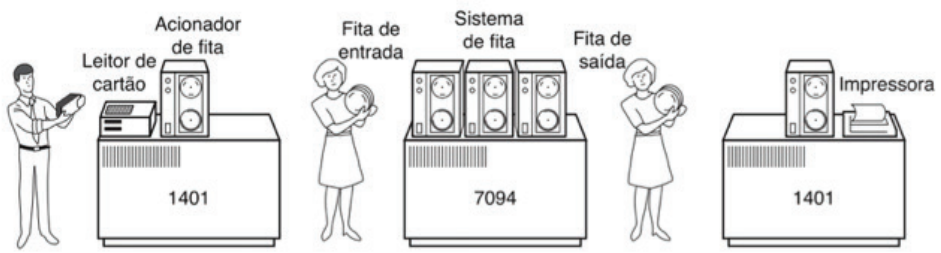
O surgimento do transistor e das memórias magnéticas contribuiu para o enorme avanço dos computadores da época. Trata-se da **segunda geração**, entre 1956 e 1965. O transistor permitiu o aumento da velocidade e da confiabilidade do processamento e as memórias magnéticas permitiram acesso mais rápido

Processamento batch lotes de programas eram submetidos ao computador.

aos dados, maior capacidade de armazenamento e computadores menores. Nessa segunda geração, o computador tinha transistor e memória magnética, linguagens de programação (Assembly e Fortran), **processamento batch** e sistema operacional com um conjunto próprio de rotinas para operações de entrada e saída.

A evolução do hardware no final dessa fase é reflexo do surgimento do conceito de canal, que permitiu a transferência dos dados entre dispositivos de e/s e memória principal sem a intervenção da UCP.

Figura 2.1 - Um sistema em lote (batch) antigo



Fonte: Tanenbaum (2009, p.5).

Multiprogramação compartilhamento da memória principal e do processador entre os vários processos.

Substituição da fita por disco possibilidade de alteração na ordem de execução das tarefas, chamada de técnica de spooling (Simultaneous Peripheral Operation OnLine).

Compartilhamento de tempo fatias de tempo do processador para cada programa em execução.

Por meio dos circuitos integrados (CIs) e, posteriormente, dos microprocessadores, foi possível viabilizar e difundir o uso de sistemas operacionais por empresas, devido à diminuição de seus custos de aquisição e utilização. Trata-se, aqui, da **terceira geração**, entre os anos de 1996 e 1980. Nessa geração, houve grande aumento do poder de processamento e diminuição do tamanho dos equipamentos.

Houve, também, evolução dos processadores de e/s, **multiprogramação**, divisão da memória em partições, **substituição da fita por disco**, interação online entre o sistema operacional e o usuário (por meio de terminais de vídeo e teclado), sistemas com **compartilhamento de tempo** (timesharing), sistema operacional UNIX (1969, desenvolvido utilizando principalmente a linguagem C) e microcomputadores com sistema operacional de 8 bits.

A **quarta geração**, que iniciou em 1981, mantém-se até os dias atuais. A integração em larga escala

(LSI) e a integração em muito larga escala (VLSI) levaram adiante o projeto de miniaturização e barateamento dos equipamentos. Os mini e os superminicomputadores se firmaram no mercado e os microcomputadores ganharam grande impulso. Essa geração contempla microcomputadores PC 16 bits, sistema operacional DOS, sistemas operacionais multiusuários (UNIX, VMS, DEC), multiprocessamento, paralelismo, sistemas operacionais para redes e sistemas operacionais distribuídos.

De acordo com Machado e Maia (2013, p. 11), **multiprocessamento** significa "execução de mais de um programa simultaneamente ou até de um mesmo programa por mais de um processador". O **paralelismo** se refere à execução de um programa, simultaneamente, por dois ou mais processador.

A partir de 2000, tivemos muito mais ênfase no desenvolvimento de técnicas de:

- inteligência artificial;
- multimídia;
- processamento distribuído; e
- sistemas operacionais embarcados.

Podemos ver sistemas operacionais embarcados, por exemplo, em fornos micro-ondas, aparelhos de TV, carros, aparelhos DVD e telefones celulares, entre outros.

Seção 2
Sistemas operacionais monoprogramáveis e multiprogramáveis

A seguir, conheceremos os vários tipos de sistemas operacionais considerando sua capacidade de gerenciar a execução de uma ou mais tarefas/programas.

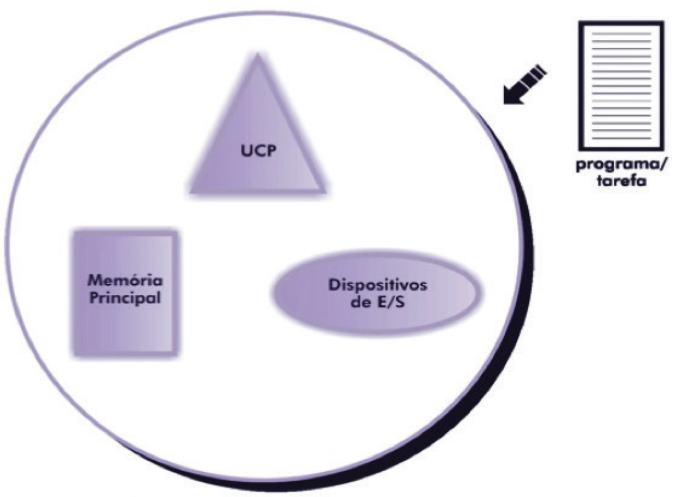
Figura 2.2 - Tipos de sistemas operacionais

Fonte: Machado e Maia (2013, p.15).

2.1 Sistema operacional monoprogramável

Um sistema operacional monoprogramável se caracteriza por permitir a execução de um programa/tarefa por vez. Nesse caso, o sistema fica inteiramente dedicado a um único usuário e a uma tarefa, conforme ilustrado na seguinte figura:

Figura 2.3 - Sistemas monoprogramáveis/monotarefas



Fonte: Machado e Maia (2013, p.16).

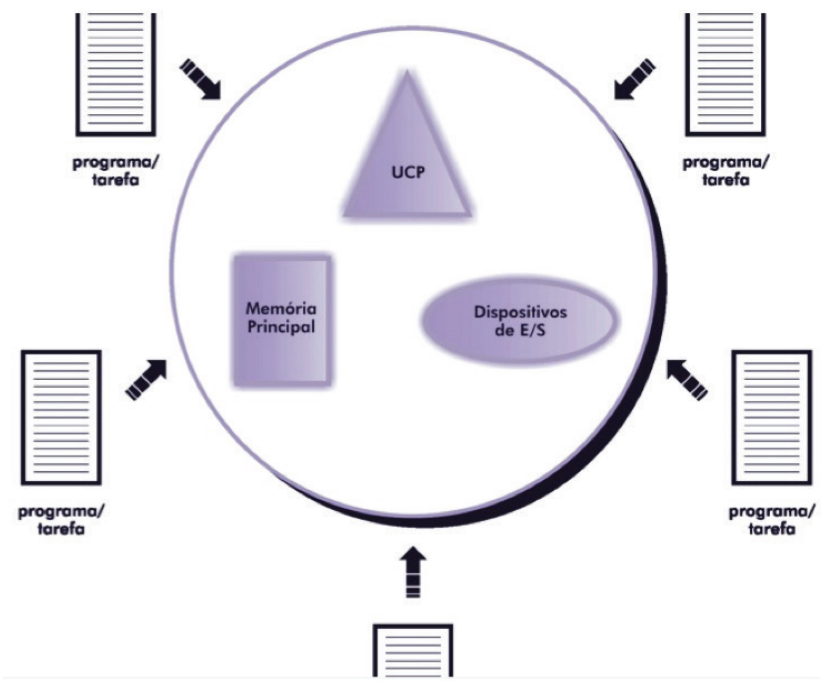
Nesses sistemas, enquanto o programa aguarda por um evento, como a digitação de um dado, o processador fica ocioso, sem realizar qualquer outra tarefa. A memória é subutilizada, caso o programa não a preencha totalmente, e os periféricos, como discos e impressoras, estão dedicados a um único usuário, nem sempre utilizados de forma integral.

São de simples implementação por não apresentarem problemas de proteção, pois o sistema está todo dedicado a um único programa/tarefa.

2.2 Sistema operacional multiprogramável

O sistema operacional multiprogramável, ao contrário do monoprogramável, é capaz de gerenciar vários programas/tarefas simultaneamente. Por permitirem a execução de vários programas ao mesmo tempo, a implementação desse tipo de sistema é mais complexa e inteligentes. Vários programas/tarefas dividem os diversos recursos do sistema, conforme a seguinte figura:

Figura 2.4 - Sistemas multiprogramáveis/multitarefa



Fonte: Machado e Maia (2013, p.17).

Com o surgimento dos sistemas multiprogramáveis, houve aumento da produtividade dos usuários e redução dos custos de utilização do sistema.

Os sistemas multiprogramáveis podem ser classificados pelo número de usuários que interagem com o sistema como **monousuário** ou **multiusuário**. Nos sistemas multiprogramáveis, é possível que um usuário possa executar vários programas/tarefas ao mesmo tempo, como edição de texto, impressão de documento e acesso à internet. Os **sistemas multiprogramáveis multiusuário** são sistemas interativos que permitem que vários usuários tenham acesso ao sistema e executem diversas tarefas (MACHADO; MAIA, 2013).

Outra classificação dos sistemas operacionais multiprogramáveis é dada pela forma como interagem com os usuários, podendo ser divididos em:

- sistemas batch;
- de tempo compartilhado; ou
- de tempo real.

Um sistema operacional pode suportar um ou mais desses tipos de processamento, conforme vemos na seguinte figura:

Figura 2.5 - Tipos de sistemas multiprogramáveis/multitarefa

Fonte: Machado e Maia (2013, p.18).

De acordo com Machado e Maia (2013, p. 10), “os sistemas batch foram os primeiros tipos de sistemas multiprogramáveis a serem implementados na década de 1960”. Nos **sistemas batch**, os programas são colocados em uma fila (disco ou fita) para serem executados sequencialmente. Os programas chamados jobs (tarefas) não exigem interação com o usuário. Podem ser bem eficientes quanto ao uso do processador, mas também podem oferecer tempos de resposta longos, em face do processamento puramente sequencial.

São exemplos de programas que executam no formato batch: compilações, linkedições, sorts, balance-line.

Os **sistemas de tempo compartilhado** ou time-sharing são assim denominados por alocarem fatia de tempo (time-slice) para que os programas/tarefas utilizem o processador. Permitem que o usuário interaja com o sistema, de forma online, por meio de uma **linguagem de controle.** Normalmente, essa interação com o usuário acontece utilizando o teclado e terminal de vídeo (monitor).

Todos os recursos do sistema são compartilhados pelos usuários e suas tarefas, que trabalham em ambiente próprio criado pelo sistema operacional, dando a impressão de estarem sozinhos. É um sistema complexo, que aumenta a produtividade dos usuários, reduzindo seus custos de utilização.

> A maioria das aplicações comerciais atualmente é processada em sistemas de tempo compartilhado, pois elas oferecem tempos de respostas razoáveis a seus usuários e custos mais baixos, em função da utilização compartilhada dos diversos recursos do sistema. (MACHADO; MAIA, 2013, p. 18).

A implementação do sistema de tempo real ou real-time é semelhante ao **sistema de tempo compartilhado**; a diferença está no tempo de resposta exigido na execução das tarefas. Esse tipo de sistema exige tempo de resposta dentro de limites rígidos. Não existe fatia de tempo: o programa executa o tempo que for necessário ou até aparecer outro com maior prioridade em função de sua importância no sistema.

A prioridade de execução é controlada pela própria aplicação (programa) e não pelo sistema operacional, como no de tempo compartilhado. Isso quer dizer que a prioridade dos programas é definida pela própria aplicação e não pelo sistema operacional, como acontece nos sistemas de tempo compartilhado.

Esse tipo de sistema operacional é utilizado em aplicações onde tempo e segurança são fatores fundamentais. De acordo com Tanenbaum (2009), os sistemas operacionais de tempo real podem ser classificados como:

- crítico; e
- não crítico.

No sistema de **tempo real crítico**, caso a ação não ocorra no tempo certo, poderá ocasionar danos ao processo.

Machado e Maia (2013), assim como Tanenbaum (2013), citam como exemplos: refinaria de petróleo, controle de tráfego aéreo, usinas termoelétricas e nucleares e linha de montagem automatizada com robôs.

No sistema de **tempo real não crítico**, não é desejável que haja atraso no tempo de resposta, mas, caso isso venha a ocorrer, não gera danos irreparáveis.

Como exemplos, Tanenbaum (2009) cita sistema de áudio digital, multimídia e telefones digitais, entre outros.

2.3 Concorrência

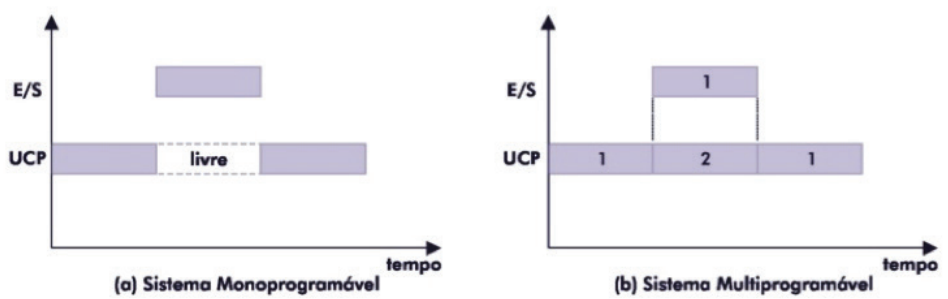
Processos podemos denominar processo todo programa/tarefa em execução.

O conceito de concorrência é fundamental para a compreensão do funcionamento de um sistema operacional multiprogramável. Nesse contexto, concorrência é a execução de tarefas concorrentes, ou seja, vários **processos** concorrendo entre si pela utilização dos diversos periféricos e dispositivos do sistema.

A execução de tarefas, de forma concorrente, proporciona várias vantagens como:

- aumento da utilização da CPU;
- utilização mais eficiente da memória principal;
- compartilhamento dos dispositivos de e/s pelos vários usuários; e
- melhor tempo de resposta na execução das tarefas.

Figura 2.6 - Sistema monoprogramável x sistema multiprogramável



Fonte: Machado e Maia (2013, p.37).

Existem alguns mecanismos e dispositivos que possibilitam a implementação de sistema operacional multiprogramável. São eles:

- interrupção;
- operações de e/s;
- buffering;
- spooling;
- reentrância; e
- proteção do sistema.

Interrupção é o mecanismo que tornou possível a implementação da concorrência nos computadores. É por meio dele que o sistema operacional sincroniza a execução das suas rotinas e dos programas dos usuários, além de controlar periféricos e dispositivos do sistema.

Durante a execução de um programa, podem ocorrer eventos que necessitem da intervenção do sistema operacional, ou seja, de interrupção. Quando ocorre uma interrupção, o fluxo da execução do programa é desviado para uma rotina de tratamento (TANENBAUM, 2009).

De acordo com Machado e Maia (2013), as interrupções podem ser:

- internas; e
- externas.

As **internas,** também chamadas de exceção, são geradas por uma instrução do próprio processo (traps ou armadilhas). Acontecem de forma síncrona, ou seja, sempre que a instrução que gerou a interrupção for executada.

Exemplos: divisão por zero, overflow.

Dispositivo disco rígido, teclado, mouse, entre outros.

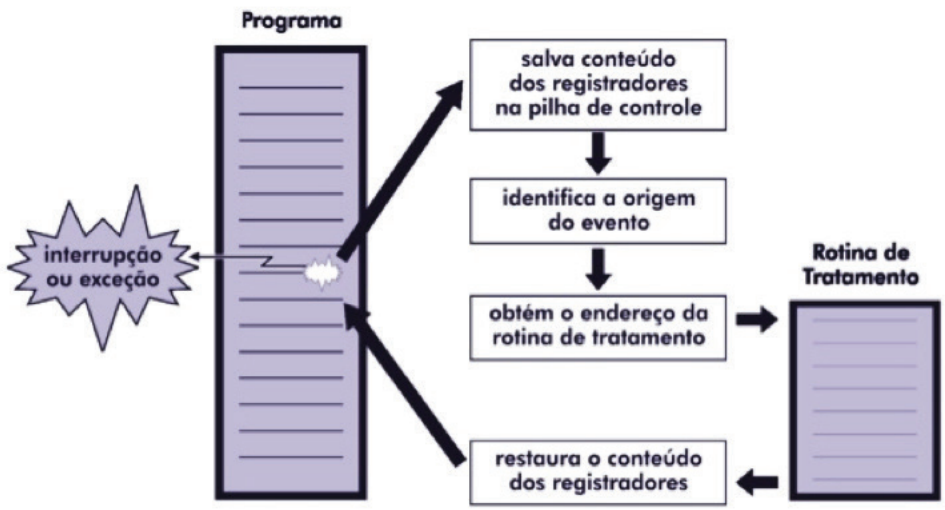
As **externas** são geradas pelo sistema operacional ou por algum **dispositivo**. A UCP interrompe o programa que está sendo executado para atender o dispositivo que gerou a interrupção. Isso acontece de forma assíncrona. Por exemplo: o disco rígido avisa a UCP que está pronto para transmitir algum dado; a impressora avisa a UCP que a impressão foi concluída.

Existem vários tipos de interrupções e, para cada uma delas, uma rotina de tratamento específica. Essas informações estão em uma estrutura do sistema

operacional chamada vetor de interrupções (TANENBAUM, 2009). As rotinas para tratamento das interrupções internas podem ser escritas, muitas vezes, pelo próprio programador.

A seguinte figura apresenta o fluxo de atendimento de ocorrência de uma interrupção:

Figura 2.7 - Mecanismo de interrupção e exceção



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

Como muitas vezes o fluxo da execução tem que voltar para o programa, um conjunto de informações sobre sua execução (contexto do programa) tem que ser preservado. Essas informações consistem no conteúdo de alguns registradores, que deverão ser restaurados, posteriormente, para a continuação do programa.

As **interrupções externas** são classificadas em:

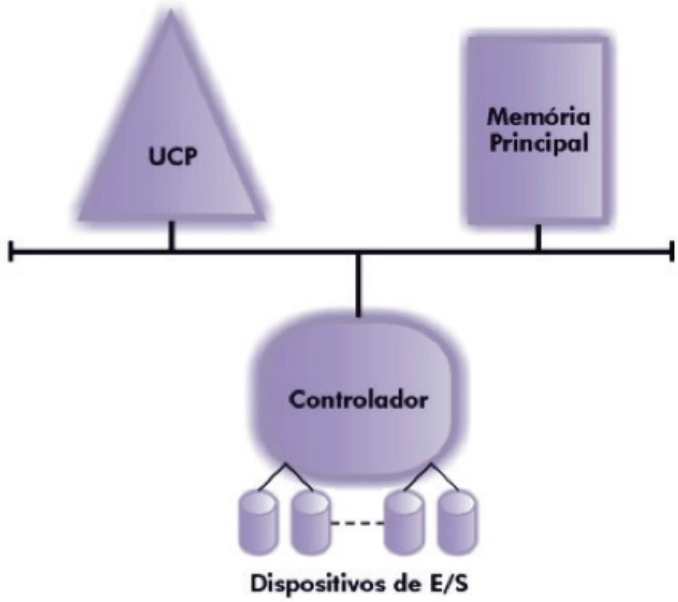
- **mascaráveis**, por não precisarem de tratamento pelo processador; e
- **não mascaráveis**, quando, obrigatoriamente, precisam ser tratadas.

Normalmente, o hardware dos computadores possui um dispositivo responsável por avaliar as interrupções geradas e suas prioridades de atendimento, chamado de **controlador de interrupções**.

Nos primeiros sistemas computacionais, as **operações de e/s** eram realizadas pelo processador, que ficava dedicado a essa tarefa até sua conclusão. Com a evolução do hardware, surgiram os **controladores** ou **interfaces**, que passaram a

executar a comunicação entre UCP e periféricos (dispositivos de e/s), permitindo que o processador ficasse livre para executar outras tarefas.

Figura 2.8 - Controlador



Fonte: Machado e Maia (2013, p. 41).

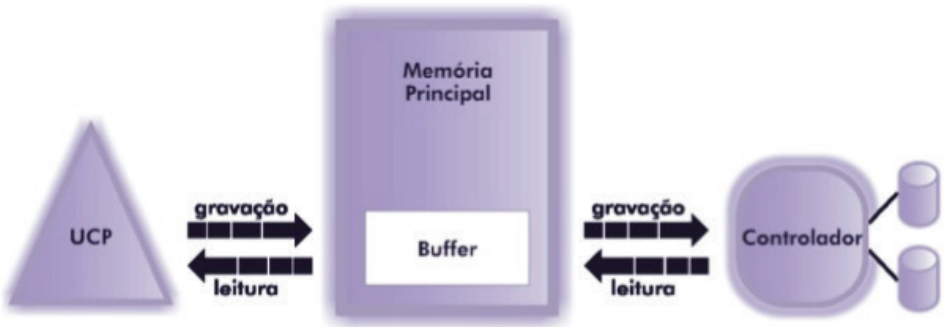
A técnica de **Direct Memory Access** (**DMA**) permite que um bloco de dados seja transferido entre memória e periféricos, sem a intervenção da UCP, exceto no início e no final da transferência. A área de memória principal utilizada pelo controlador na técnica de DMA é chamada buffer, sendo reservada exclusivamente para esse fim.

Buffering consiste na utilização de uma área de memória (buffer) para a transferência de dados entre memória e periféricos. Enquanto um bloco de dados que está no buffer é manipulado pela UCP, outro bloco pode estar sendo lido ou gravado por outro dispositivo.

O buffering é uma das implementações para minimizar o problema da disparidade da velocidade de processamento existente entre a UCP e os dispositivos de e/s. O objetivo do buffering é manter a UCP e os dispositivos de e/s ocupados a maior parte do tempo.

A seguinte figura ilustra a técnica de buffering:

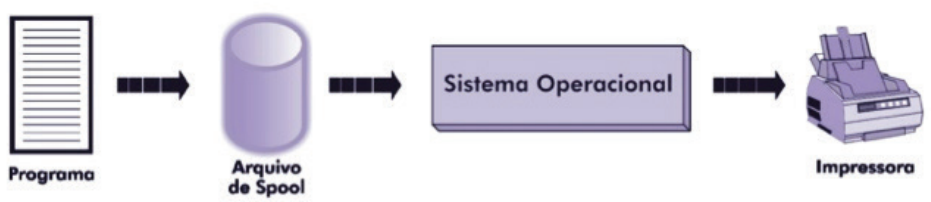
Figura 2.9 - Operações de e/s utilizando buffer



Fonte: Machado e Maia (2013, p. 43).

A técnica de **spooling** consiste na utilização de uma área em disco rígido (HD) para armazenamento temporário de dados. Ou seja, é a utilização do disco como uma grande área de buffer, permitindo que dados sejam gravados e lidos em disco, enquanto outras tarefas são processadas, conforme a seguinte figura:

Figura 2.10 - Técnica de spooling

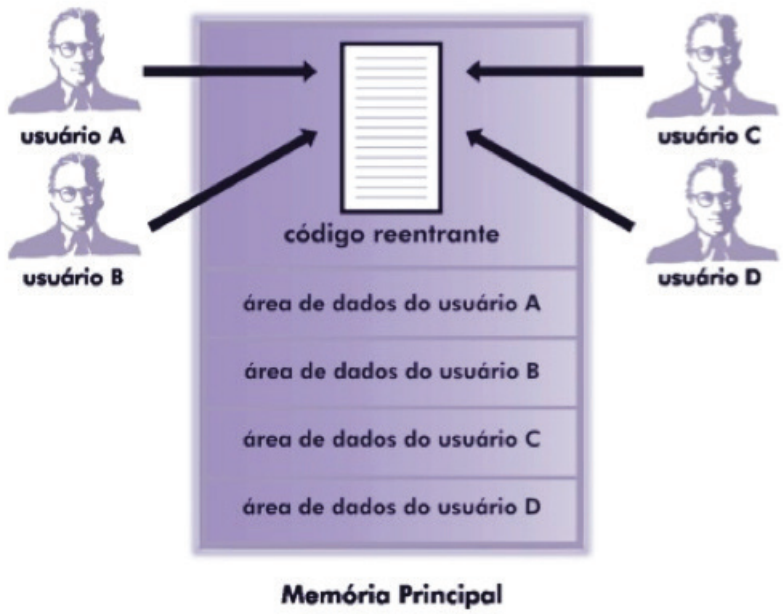


Fonte: Machado e Maia (2013, p. 44).

Essa implementação permite maior grau de compartilhamento e utilização de impressoras.

Reentrância acontece quando o mesmo código de programa residente em memória é utilizado por vários usuários ao mesmo tempo, como editores de texto e compiladores. Proporciona grande economia de memória. O código reentrante não pode ser modificado pelo usuário.

Figura 2.11 - Reentrância



Fonte: Machado e Maia (2013, p. 45).

Na maioria dos sistemas operacionais, a **proteção do sistema** é implementada por meio do **estado de execução**. O estado de execução é uma característica associada ao programa em execução que determina se ele pode ou não executar certas instruções ou rotinas. Há, basicamente, dois estados de execução:

1. **Estado usuário -** só pode executar instruções que não afetem outros programas.
2. **Estado supervisor (monitor ou kernel) -** pode executar instruções privilegiadas, ou seja, qualquer instrução do sistema operacional. É o estado permanente do sistema operacional.

O estado de execução está sinalizado em um **registrador especial**.

Seção 3
Sistemas operacionais com múltiplos processadores

Como o título já especifica, são sistemas de computação que possuem mais de um processador, que podem ou não compartilhar o mesmo sistema operacional.

Esses sistemas mantêm o mesmo conceito de multiprogramação, só que aplicado a vários processadores. É uma arquitetura que permite a reconfiguração e o balanceamento do sistema. Isso quer dizer que o sistema pode continuar funcionando mesmo que um dos processadores falhe e possibilita balancear a carga de processamento e das operações de e/s entre os diversos processadores; permite que vários programas sejam executados ao mesmo tempo ou que um programa seja dividido em subprogramas para execução simultânea em mais de um processador (paralelismo).

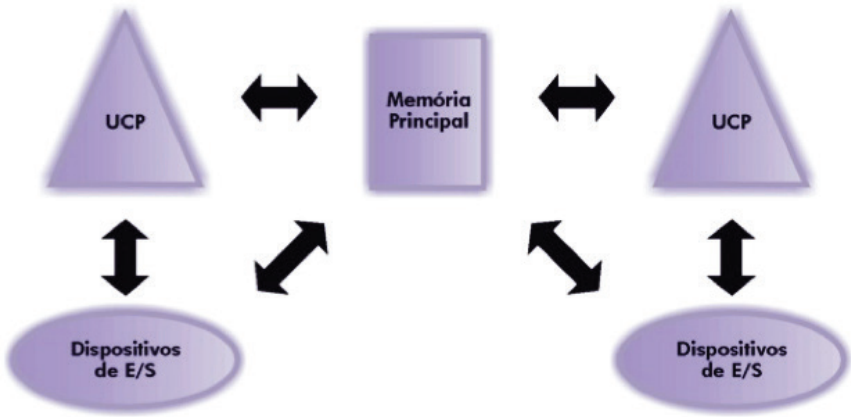
Figura 2.12 - Tipos de sistemas com múltiplos processadores

Fonte: Machado e Maia (2013, p. 20).

Conforme essa figura, os sistemas multiprocessáveis podem ser divididos em dois grupos, dependendo da forma de comunicação entre os processadores e o grau de compartilhamento da memória e dos dispositivos de e/s:

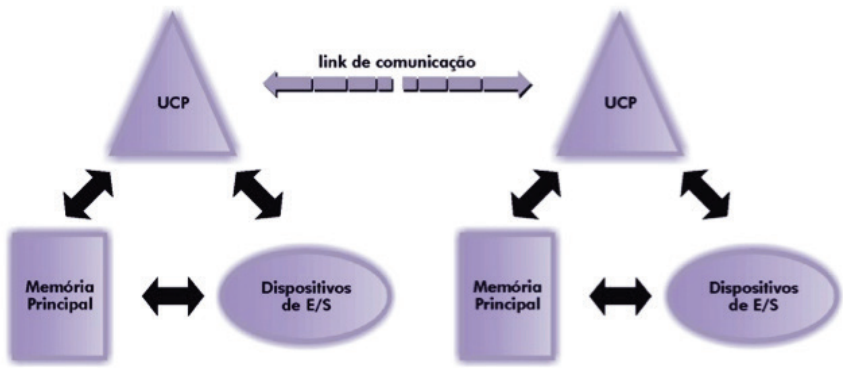
- fracamente acoplados; e
- fortemente acoplados.

Figura 2.13 - Sistemas fortemente acoplados



Fonte: Machado e Maia (2013, p. 20).

Figura 2.14 - Sistemas fracamente acoplados



Fonte: Machado e Maia (2013, p. 21).

Os primeiros dizem respeito às redes locais e os últimos remetem a vários processadores que compartilham a mesma memória e são controlados por um único sistema operacional. Os **fortemente acoplados** são subdivididos em:

- assimétricos ou mestre/escravo; e
- simétricos.

Em relação aos **assimétricos ou mestre/escravo**, apenas um processador pode executar funções do sistema operacional. Sempre que um escravo necessitar de uma operação de e/s, terá que solicitar ao mestre. O processador mestre é aquele que concentra a execução de todas as instruções privilegiadas, aquelas que executam apenas no **modo kernel**. Todos os demais processadores dessa

arquitetura são chamados escravos, pois executam instruções não privilegiadas e são gerenciados pelo processador mestre. Esse modelo apresenta ineficiência pelo número de interrupções, pois são canalizadas para o mestre. Se o mestre falhar, há uma parada geral no sistema, sendo necessário fazer uma reconfiguração de um escravo para mestre.

Quanto aos **simétricos**, todos os processadores podem executar os serviços do sistema operacional. Apenas algumas funções ficam a cargo de um processador, como o *boot* de sistema (inicialização do sistema). Não há problema se algum processador falhar, a não ser a diminuição da capacidade de computação. Nesse tipo de sistema, um programa pode ser executado por qualquer processador ou por vários ao mesmo tempo (paralelismo).

Capítulo 3

Gerenciando processos e memória

Habilidades

Neste capítulo, você conhecerá a estrutura que o sistema operacional cria para cada programa em execução (processo), bem como os estados pelos quais o processo passa durante sua execução. Será capaz de compreender a gerência do processador, os mecanismos de escalonamento e suas características. Conhecerá e compreenderá a forma como o sistema operacional gerencia o espaço de memória principal, para atender aos vários processos em execução.

Seções de estudo

Seção 1: Processo e sua execução

Seção 2: Gerência do processador

Seção 3: Gerência de memória

Seção 1
Processo e sua execução

O conceito de processo é a base para a implementação de um sistema multiprogramável. O processador foi projetado apenas para executar instruções, não sendo capaz de identificar qual processo está em execução. A gerência de um ambiente multiprogramável é função exclusiva do sistema operacional, o qual deve controlar a execução dos diversos programas e o uso do processador.

> Um programa, para ser executado, deve estar sempre associado a um processo.

A gerência de processos é uma das principais funções do sistema operacional. Por meio de processos, um programa pode:

- alocar recursos;
- compartilhar dados;
- trocar informações; e
- sincronizar sua execução.

Nos sistemas multiprogramáveis, os processos são executados concorrentemente, compartilhando, entre outros recursos, o uso do processador, da memória principal e dos dispositivos de e/s.

Nos sistemas com múltiplos processadores, não só existe a concorrência de processos pelo uso do processador, como também a execução simultânea de processos nos diferentes processadores.

1.1 Estrutura do processo

O conceito de processo é mais abrangente do que apenas um programa em execução. Isso fica mais claro quando pensamos em como os sistemas multiprogramáveis atendem aos diversos usuários e mantêm informações sobre os diversos programas que estão em execução.

Em um **sistema multiusuário**, ao executar um programa, o usuário tem a impressão de que está sozinho na máquina, ou seja, de que todos os recursos, inclusive a UCP, estão à sua disposição, mas isso não é verdade. O que acontece, como já sabemos, é que os recursos são compartilhados entre os diversos programas em execução. Nesse caso, o processador executa um programa em

um intervalo de tempo e, no intervalo seguinte, pode estar processando outro programa.

Para que essa troca de programas aconteça sem problemas, é necessário que todas as informações do programa interrompido sejam guardadas para que, quando retornar o processamento, nenhuma informação falte. Todas as informações necessárias para a execução de um programa fazem parte do processo.

Um processo é formado por três partes:

- contexto de hardware;
- contexto de software; e
- espaço de endereçamento.

Visualizamos essa relação na seguinte figura:

Figura 3.1 - Estrutura do processo

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

Contexto de hardware armazena o conteúdo dos registradores gerais e de uso especifico. O quadro seguinte apresenta as funções dos principais registradores de uso específico:

Quadro 3.1 - Funções dos registradores de uso específico

- o *contador de instruções* (CI) ou program counter (PC) contém o endereço da próxima instrução que o processador deve buscar e executar. Toda vez que o processador busca uma nova instrução, este registrador é atualizado com o endereço de memória da instrução seguinte a ser executada;
- *apontador da pilha* (AP) ou stack pointer (SP) contém o endereço de memória do topo da pilha, que é a estrutura de dados onde o sistema mantém informações sobre programas que estão sendo executados;
- o *registrador de instruções* (RI) é responsável por armazenar a instrução que será decodificada e executada pelo processador;
- o *registrador de status* ou *program status Word* (PSW) é responsável por armazenar informações sobre a execução de instruções, como a ocorrência de overflow. A maioria das instruções, quando executadas, altera o registrador de status conforme o resultado.

Fonte: Machado e Maia (2013, p. 24).

Contexto de software registra as características e limites dos recursos que podem ser alocados para o processo.

É composto por três grupos de informações:

- identificação;
- quotas; e
- privilégios.

A identificação ocorre de duas maneiras, pelo **PID**, que identifica um processo e pelo **UID**, que identifica o dono do processo, ou seja, o usuário que o criou. As **quotas** representam:

- número máximo de arquivos abertos;
- tamanho máximo de memória principal e secundária que pode alocar;
- número máximo de operações de e/s;
- tamanho máximo do buffer para as operações de e/s; e
- número máximo de processos, subprocessos e threads que podem ser criados.

Por fim, **privilégios** identificam o tipo de instrução que o processo pode executar.

Espaço de endereçamento é a área de memória onde as instruções e os dados são armazenados.

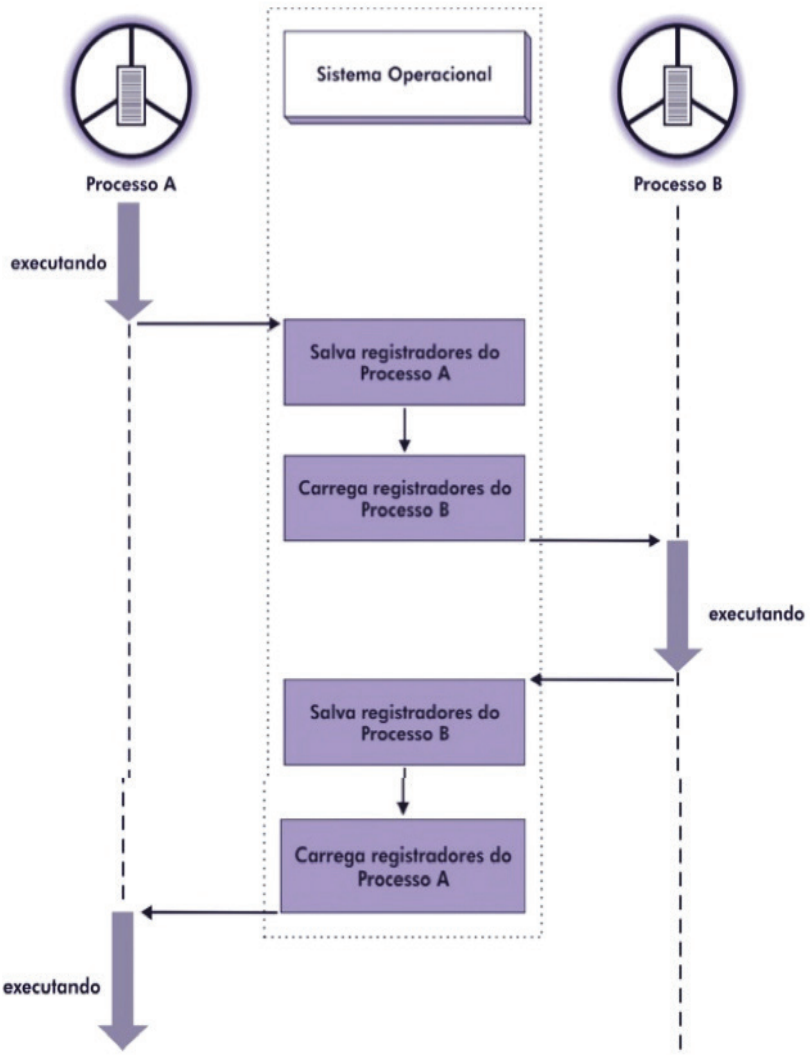
Na seguinte figura, podemos ter a visão geral das características de um processo.

Figura 3.2 - Características da estrutura de um processo

Fonte: Machado e Maia (2013, p. 66).

Um processo também pode ser definido como o ambiente onde o programa é executado. A execução de um mesmo programa pode variar dependendo do processo no qual ele está sendo executado. Essa troca de processo na UCP é chamada de **mudança de contexto**.

Figura 3.3 - Mudança de contexto

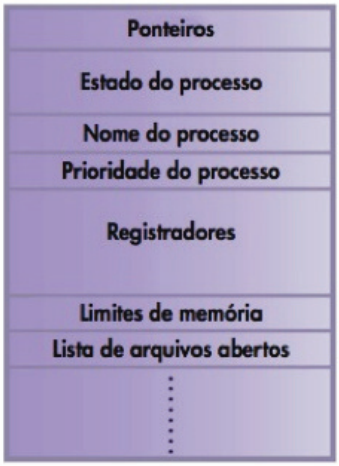


Fonte: Machado e Maia (2013, p. 64).

O **bloco de controle do processo** (process control block – PCB) é a estrutura criada pelo sistema operacional que mantém as informações sobre o contexto de hardware, de software e espaço de endereçamento de cada processo.

Essa estrutura está apresentada na figura a seguir:

Figura 3.4 - Bloco de controle de processo



Fonte: Machado e Maia (2013, p. 66).

1.2 Estados do processo

Os processos passam pelos seguintes **estados** ao longo do seu processamento:

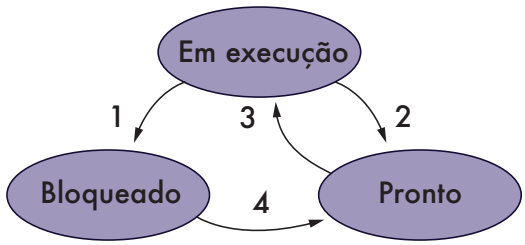
- execução (running);
- pronto (ready); e
- espera ou bloqueado (wait).

No estado de **execução**, o processo se encontra alocado no processador, ou seja, efetivamente executando. Quando o processo está em estado de **pronto**, significa que ele está apenas aguardando para utilizar o processador, pois já está com todos os recursos necessários alocados para sua execução. Um processo está no estado de **bloqueado** (espera) quando necessita de recursos que ainda não foram disponibilizados ou aguarda alguma resposta do sistema para poder continuar a execução.

São possíveis várias mudanças de estado durante a execução do processo:

- pronto → execução;
- execução → bloqueado;
- bloqueado → pronto; e
- execução → pronto.

Figura 3.5 - Transições entre estados de um processo



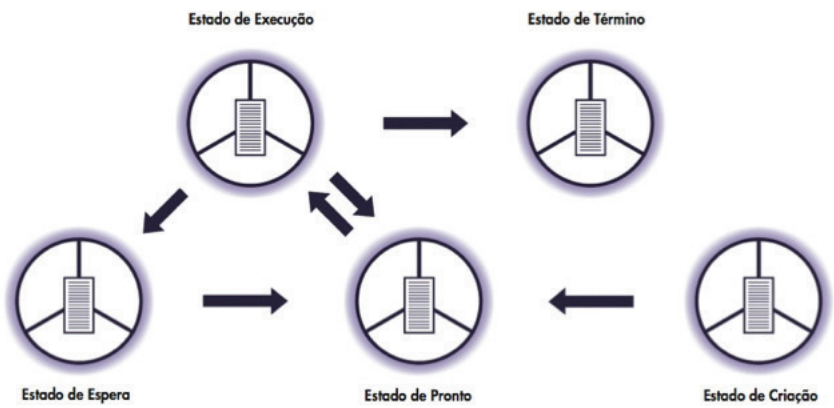
1. O processo bloqueia aguardando uma entrada
2. O escalonador seleciona outro processo
3. O escalonador seleciona esse processo
4. A entrada torna-se disponível

Fonte: Tanenbaum (2009, p. 54).

Conforme observamos na figura, há quatro processos enumerados. O **escalonador** é o componente do sistema operacional responsável por "escolher" o processo que irá utilizar o processador. A criação de um processo ocorre quando o sistema operacional cria a estrutura PCB (bloco de controle de processo). Além dos três estados citados, a maioria dos sistemas estabelece mais dois estados:

- criação; e
- terminado.

Figura 3.6 - Transição entre os estados processo, incluindo criação e término



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

1.3 Processos independentes, subprocessos e threads

Sabemos agora que para todo programa que entra em execução, o sistema cria um ambiente, o qual chamamos de **processo**.

Dependendo do tipo de ambiente criado e do comportamento desse **processo**, ele pode ser classificado como:

- processo independente;
- subprocesso; ou
- thread.

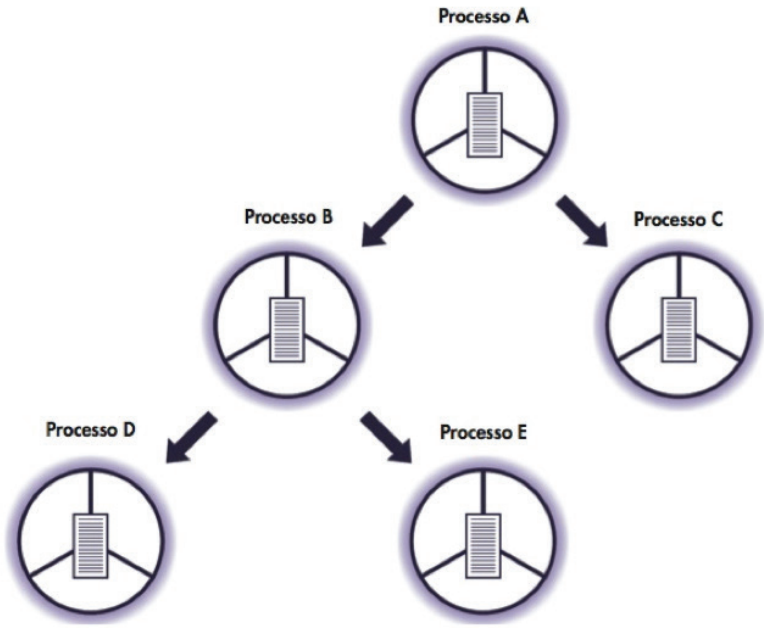
Os processos **independentes** não têm vínculo com o processo criador, exigem alocação de um PCB, possuindo contexto de hardware, contexto de software e espaço de endereçamento próprios.

Estrutura hierárquica o processo criador é denominado processo pai e o novo processo é chamado de processo filho ou subprocesso.

Os **subprocessos** são criados dentro de uma **estrutura hierárquica**. Cada subprocesso possui seu próprio PCB, mas é dependente do processo pai, ou seja, sua existência é determinada pelo processo pai. O subprocesso pode criar novos processos e compartilhar cotas de seu pai, ou seja, o pai cede parte de sua quota ao filho.

A seguinte figura mostra a estrutura de processos e subprocessos, sendo que cada processo filho possui contextos e espaço de endereçamentos independentes.

Figura 3.7 - Estrutura de processos e subprocessos



Fonte: Machado e Maia (2013, p. 75).

Cabe ressaltar que quando o processo pai é terminado, todos os processos filhos também são terminados.

As **threads** foram desenvolvidas para tentar reduzir o tempo gasto na criação, eliminação e troca de contexto de processos, bem como economizar recursos como um todo. Cada thread possui seu próprio contexto de hardware, mas compartilha o contexto de software e o espaço de endereçamento com as demais threads do processo (MACHADO; MAIA, 2013; TANENBAUM, 2009).

1.4 Processos foreground e background

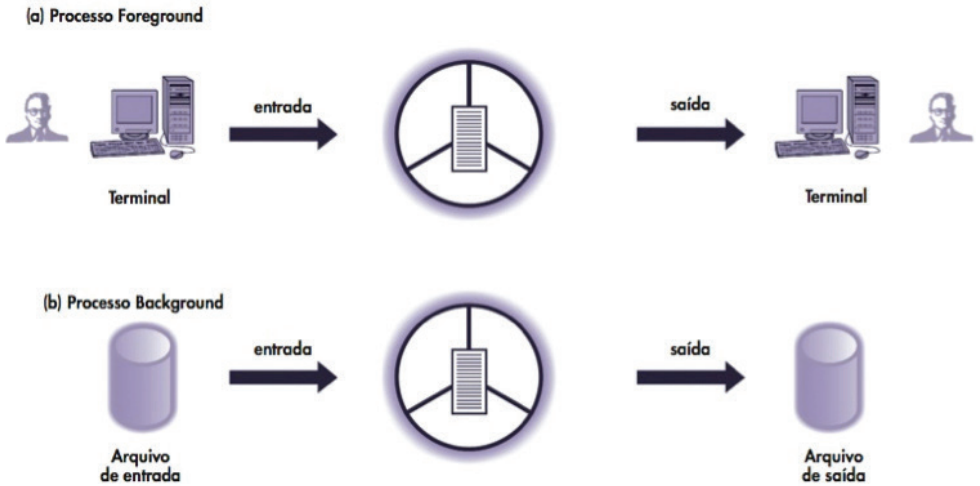
Durante a execução das aplicações/programas em nosso computador, podemos observar que alguns necessitam da nossa interação, enquanto outros executam "sozinhos", sem nossa intervenção.

Assim, podemos ainda classificar os processos quanto ao grau de interação com o usuário:

- processo **foreground** permite a interação direta com o usuário; e
- processo **background**, no qual não existe interação com o usuário durante o processamento.

A seguinte figura ilustra essa relação de interação com o usuário:

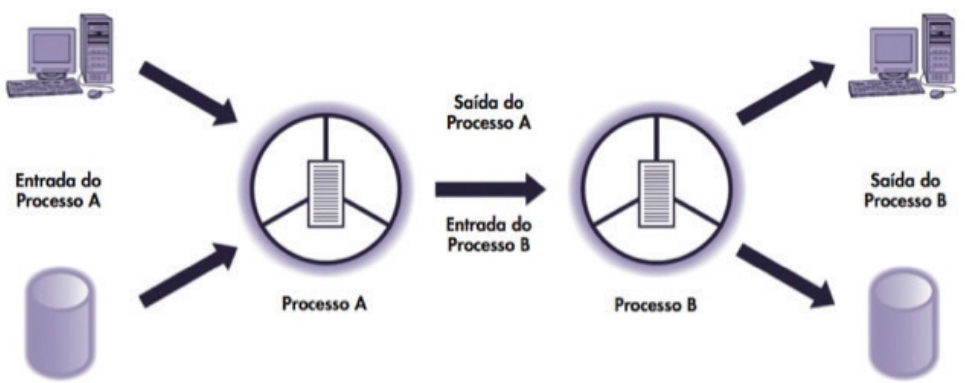
Figura 3.8 - Processos foreground e background



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

É possível associar o canal de saída de um processo ao canal de entrada de outro. Nesse caso, dizemos que há um pipe (|).

Figura 3.9 - Pipe.



Fonte: Machado e Maia (2013, p. 73).

1.5 Processos do sistema operacional

“O conceito de processo, além de estar associado a aplicações de usuários, pode também ser implementado na própria arquitetura do sistema operacional.” (MACHADO; MAIA, 2013, p.76).

Esses processos são implementados para atender serviços do próprio sistema operacional, como:

- auditoria e segurança;
- serviços de rede;
- contabilização do uso de recursos;
- contabilização de erros;
- gerência de impressão; e
- gerência de jobs batch.

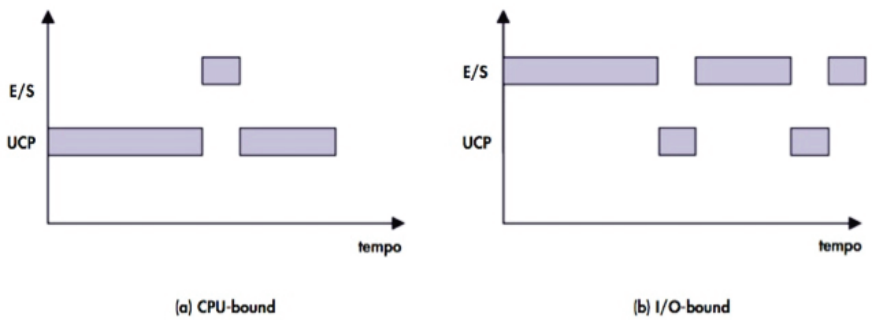
1.6 Processos CPU-bound e I/O-bound

Os processos são classificados pelo tempo que utilizam o processador, podem ser:

- processos CPU-bound, que ficam a maior parte do tempo em estado de execução; ou
- processos I/O-bound, que ficam a maior parte do tempo em estado de espera (bloqueado).

Na seguinte figura, podemos ver a execução de cada um desses tipos de processos:

Figura 3.10 - Processos cpu-bound x processos i/o-bound



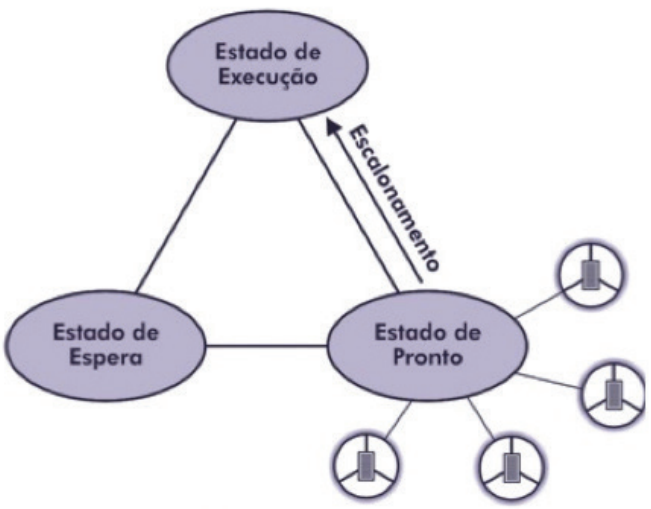
Fonte: Machado e Maia (2013, p. 71).

Nesta seção, conhecemos as principais características dos processos que são executados pelo computador. Na seção seguinte, você terá a oportunidade de estudar sobre como o sistema operacional organiza a execução dos diversos processos, escolhendo o processo que irá utilizar o processador.

Seção 2
Gerência do processador

Uma das principais funções de um sistema operacional multiprogramável é gerenciar a utilização do processador, entre os vários processos que concorrem pela sua utilização. Esse procedimento de escolha de qual processo irá utilizar o processador é chamado de **escalonamento** (scheduling). A seguinte figura apresenta essa função do sistema operacional:

Figura 3.11 - Escalonamento



Fonte: Machado e Maia (2013, p. 127).

Podemos verificar que existem vários processos, em estado de pronto, aguardando a vez para utilizar o processador, ou seja, mudar para o estado de execução. O escalonador, então, conforme o algoritmo implementado, identifica o processador que irá para o estado de execução. “Os critérios utilizados para esta seleção compõem a chamada política de escalonamento, que é a base da gerência do processador e da multiprogramação em um sistema operacional.” (MACHADO; MAIA, 2013, p. 127).

Observe que os processos em estado de espera precisam passar primeiro para o estado de pronto, para, então, poder utilizar o processador.

2.1 Políticas de escalonamento

Vamos iniciar diferenciando os tipos de escalonamento de acordo com a possibilidade, ou não, de o sistema operacional interromper o processo em execução:

- não preemptivo; e
- preemptivo.

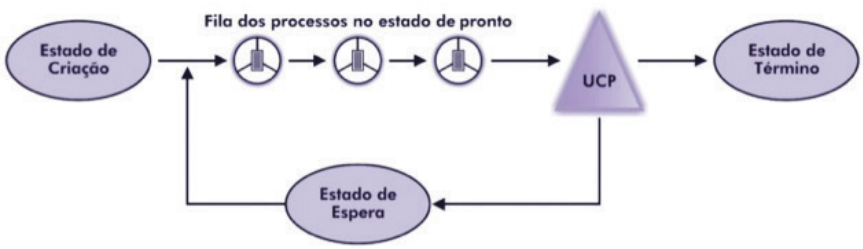
Nos **escalonamentos não preemptivos,** quando um processo ganha o direito de utilizar o processador, nenhum outro processo pode lhe tirar esse direito. Foi utilizado nos primeiros sistemas multiprogramáveis (tipicamente batch). Existem algumas políticas de escalonamento que seguem esse princípio:

- FIRST-IN-FIRST-OUT (FIFO); e
- SHORTEST-JOB-FIRST (SJF).

O escalonamento tipo **FIFO** é implementado utilizando uma fila, onde os processos que passam para o estado de pronto entram no final da fila e são escalonados quando chegam ao início da fila. Nesse tipo de escalonamento, a utilização do processador é feita sem interrupção, a não ser que o próprio processo necessite de uma operação de entrada/saída e entre em estado de espera espontaneamente.

Veja, seguinte figura, como acontece esse processo:

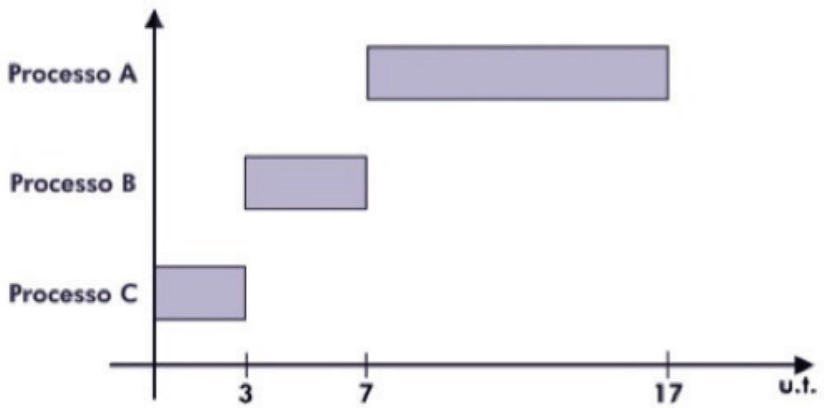
Figura 3.12 - Escalonamento FIFO



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

No escalonamento tipo **SJF**, o processo em estado de pronto que necessitar de menos tempo de processador é escalonado (selecionado). Essa política favorece os programas menores, reduzindo o tempo de espera em relação ao FIFO. O problema dessa política consiste em precisar exatamente o tempo que o processo utilizará do processador.

Figura 3.13 - Escalonamento SJF



Fonte: Machado e Maia (2013, p. 131).

Essa figura mostra a sequência de execução de três processos:

- o **processo C**, o qual consome 3 ut (unidades de tempo) é escalonado primeiro, executando até o final;
- depois entra em estado de execução o **processo B**, com 4 ut; e, em seguida,
- o **processo A**, o mais longo, com 10 ut.

Essas técnicas não são aplicadas em sistemas operacionais de tempo compartilhado, os quais necessitam de um tempo de resposta razoável.

Nas políticas de **escalonamento preemptivas**, o sistema operacional pode interromper um programa em execução, para que um outro processo utilize o processador. Esse tipo de escalonamento permite que o sistema operacional dê atenção imediata a processos de maior prioridade (tempo real) e proporcione melhor tempo de resposta (tempo compartilhado).

> **!** O compartilhamento de processador é mais uniforme nos escalonamentos preemptivos.

Existem algumas políticas de escalonamento preemptivas, como:

- circular (round robin);
- por prioridades;
- por múltiplas filas; e
- por múltiplas filas com realimentação.

O **escalonamento preemptivo circular** é implementado em sistemas operacionais de tempo compartilhado. O algoritmo e implementado com base em uma fila circular, na qual os processos têm um tempo limite para utilização contínua do processador (time-slice). “O valor da fatia de tempo depende da arquitetura de cada sistema operacional e, em geral, varia entre 10 e 100 milissegundos.” (MACHADO; MAIA, 2013, p.133).

Figura 3.14 - Escalonamento circular

Fonte: Machado e Maia (2013, p. 133).

No **escalonamento por prioridades,** o sistema operacional gerencia a fila de processos em estado de pronto por prioridade associada a cada processo. Nesse tipo de escalonamento, os processos I/O-bound recebem uma prioridade mais alta que os CPU-bound.

A prioridade associada ao processo pode ser:

- estática; ou
- dinâmica.

Isso significa que a prioridade do processo pode (**dinâmica**) ou não (**estática**) se alterar durante a sua execução.

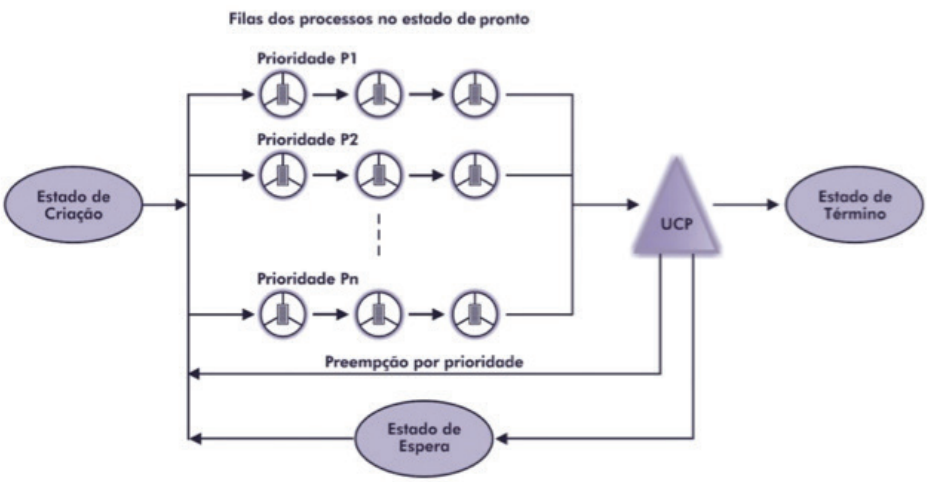
Uma das formas de alterar, dinamicamente, a prioridade do processo, é considerar cada vez que um processo sai da fila de estado de espera e elevar a sua prioridade. Em certos intervalos de tempo, o sistema operacional varre a lista de processos em estado de pronto, para identificar a existência de algum processo com maior prioridade do que o processo em execução. Caso haja, o processo em execução é interrompido e colocado na lista de processos em estado pronto, associada à sua prioridade. O sistema operacional escalona, então, o processo de mais alta prioridade que estiver na lista de pronto.

No escalonamento por prioridades, não existe a ideia de fatia de tempo!

A figura seguinte demonstra que, após a criação do processo, ele é colocado na fila correspondente à sua prioridade. Quando acaba o seu tempo de utilização do processador (preempção por tempo), ele é recolocado no final da fila. Caso ele esteja executando e solicite uma operação de e/s, entrará em estado de espera e, assim, ficará até que sua solicitação seja atendida e colocado no final da fila associada à sua prioridade.

Quando o sistema implementa o mecanismo de prioridade dinâmica, o processo pode voltar do estado de espera em uma prioridade maior do que a inicial.

Figura 3.15 - Escalonamento por prioridade

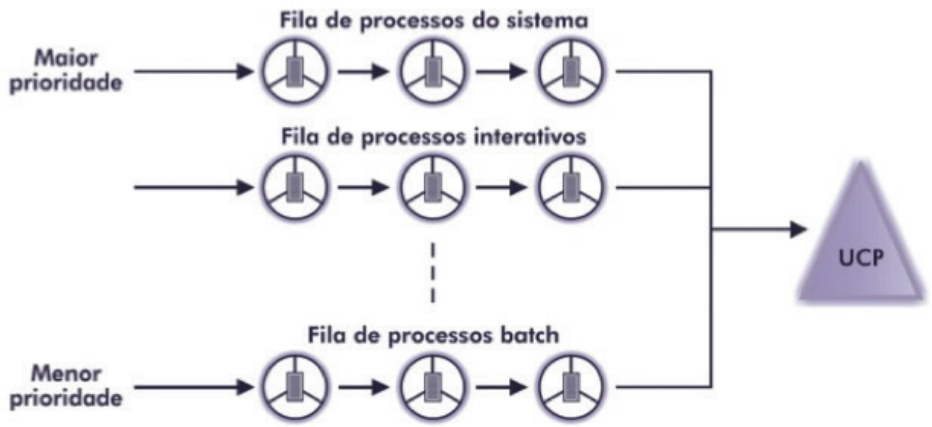


Fonte: Machado e Maia (2013, p. 135).

No **escalonamento por múltiplas filas**, existem várias filas de pronto, onde cada uma possui um mecanismo próprio de escalonamento em função das características do processo. Um processo só pode estar associado a uma fila. Os processos são previamente classificados, de forma estática, para serem

encaminhados a uma determinada fila. As filas também têm prioridade umas sobre as outras. O sistema operacional só pode escalonar um processo de uma fila, se todas as filas de prioridade mais alta estiverem vazias (MACHADO; MAIA, 2013).

Figura 3.16 - Escalonamento por múltiplas filas



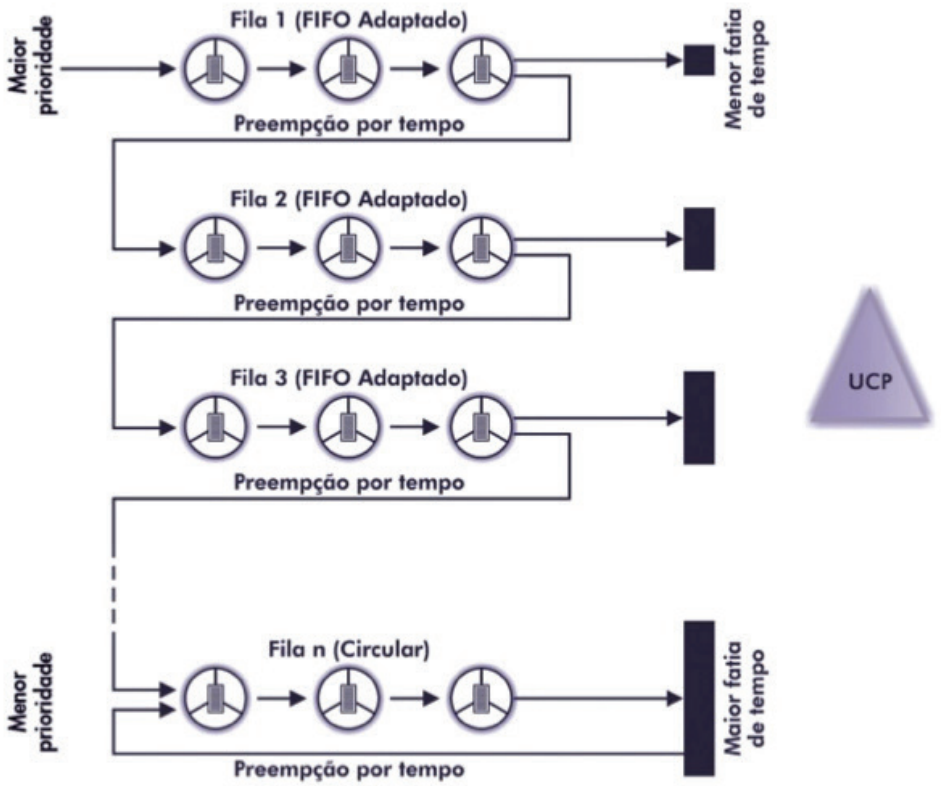
Fonte: Machado e Maia (2013, p. 138).

Há também o **escalonamento por múltiplas filas com realimentação**. Nesse tipo de escalonamento, a classificação dos processos é dinâmica. O processo, quando é criado, é colocado no final da fila de maior prioridade, sendo que todas adotam o mecanismo FIFO.

Caso um processo esgote o tempo para utilizar o processador, ele é colocado no final da fila com menor prioridade do que a anterior, onde se encontrava. Assim, cada vez que um processo atinge o limite da sua fatia de tempo, ele é "penalizado", voltando imediatamente para uma fila de prioridade mais baixa do que a anterior. Por outro lado, quanto menor a prioridade da fila, maior a fatia de tempo associada aos processos dessa fila.

Pode-se perceber que os processos I/O - bound têm preferência nesse esquema, por se tratar de um mecanismo mais generalista, complexo e eficiente, conforme seguinte figura:

Figura 3.17 - Escalonamento por múltiplas filas com realimentação



Fonte: Machado e Maia (2013, p. 139).

Seção 3
Gerência de memória

Na memória principal residem todos os programas e dados que serão executados ou referenciados pelo processador. Toda vez que um programa residente em memória secundária tiver que ser executado, ele tem que ser carregado para a memória principal. A forma de carregar ou retirar informações da memória principal é outra das principais funções do sistema operacional, a qual chamamos de gerência de memória. A seguir, vamos conhecer desde as primeiras estratégias de gerenciamento de memória, até chegar aos métodos utilizados atualmente.

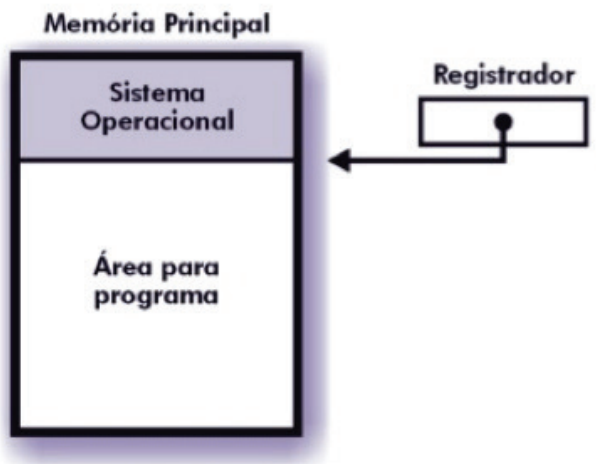
3.1 Alocação contígua simples

Essa estratégia foi utilizada na implementação dos primeiros sistemas operacionais, os quais eram monoprogramáveis. Nesse método, a memória principal era dividida em duas partes:

- uma utilizada para armazenar o sistema operacional; e
- outra para armazenar o programa do usuário.

Veja essa disposição na seguinte figura:

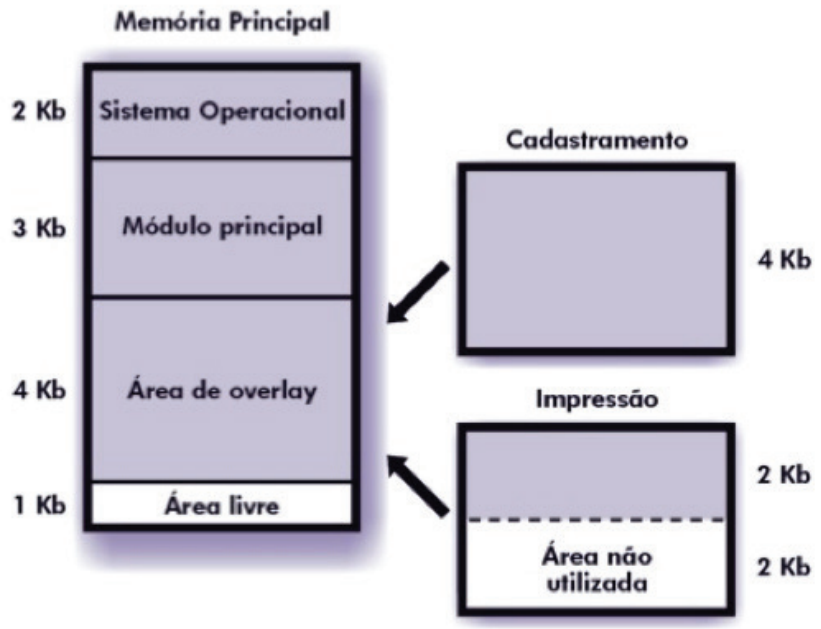
Figura 3.18 - Alocação contígua simples e o esquema de proteção



Fonte: Machado e Maia (2013, p. 147).

Nesse tipo de alocação, o programa do usuário não poderia ser maior do que a área total da memória principal menos a área ocupada pelo SO. A proteção era feita por um registrador que delimitava as áreas do sistema operacional e do usuário. Com o passar do tempo, os programas de usuário começaram a não caber na área disponível de memória. Nesse momento, foi desenvolvida a **técnica de overlay** (sobreposição). Por meio dessa técnica, o programa do usuário é divido em módulos independentes que utilizam a mesma área de memória, sendo que o tamanho da área de overlay tem o tamanho do maior módulo.

Figura 3.19 - Técnica de overlay



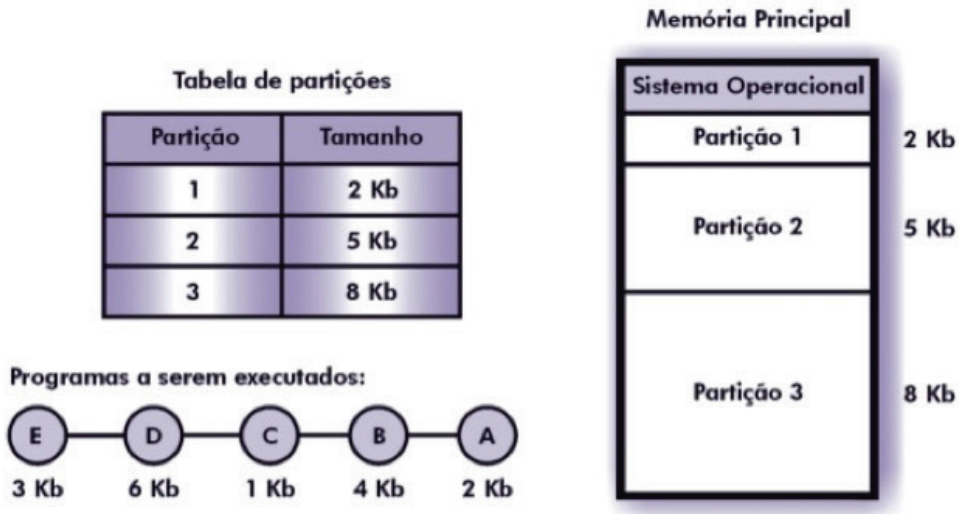
Fonte: Machado e Maia (2013, p. 149).

3.2 Alocação particionada estática

A implementação de sistemas operacionais multiprogramáveis só foi possível quando o hardware evoluiu permitindo a alocação de vários programas na memória principal, simultaneamente. A alocação particionada inicialmente foi desenvolvida de forma estática – alocação particionada estática, evoluindo para alocação particionada dinâmica.

Na **alocação particionada estática**, a memória era dividida em blocos de tamanhos fixos chamados de **partições**. O tamanho das partições era estabelecido na inicialização do sistema operacional (boot), de acordo com o tamanho dos programas que iriam ser executados.

Figura 3.20 - Alocação particionada estática



Fonte: Machado e Maia (2013, p. 149).

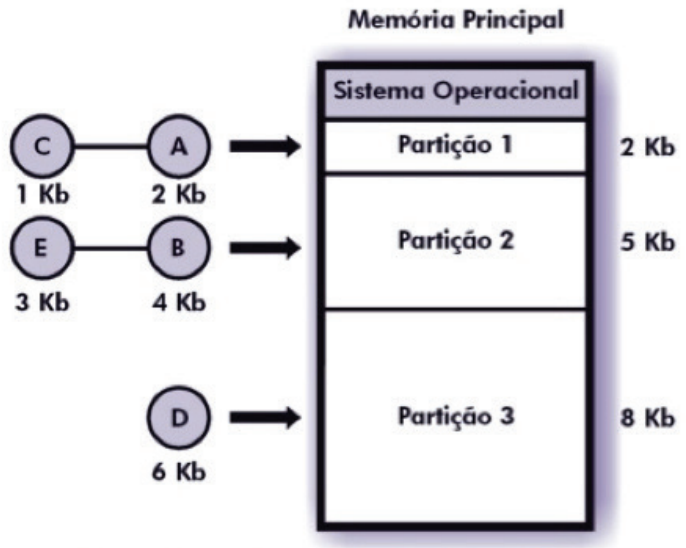
Para a alteração dos tamanhos das partições, era necessária uma nova reinicialização do sistema. A alocação particionada estática é classificada em:

- absoluta (fixa); ou
- relocável.

Código absoluto significa que o programa deve ser carregado sempre no mesmo espaço de memória.

Na **alocação particionada estática absoluta**, um programa só pode ser carregado em uma das partições. Isso devido à limitação dos compiladores/montadores, que geravam **código absoluto**.

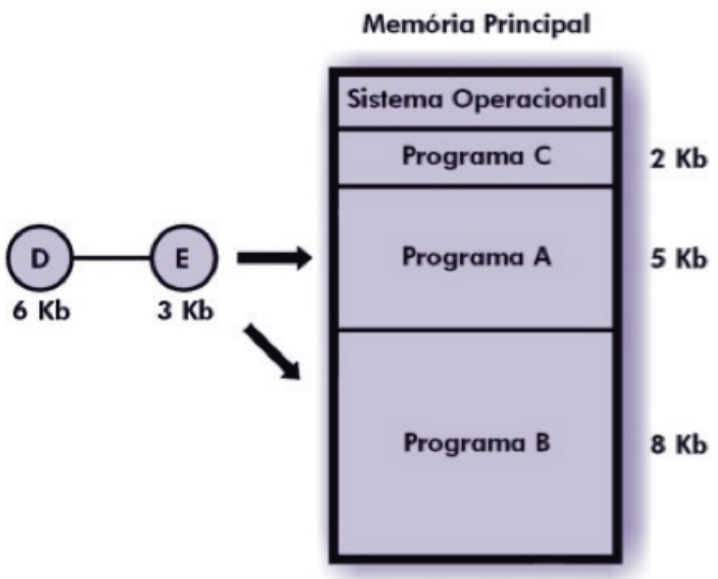
Figura 3.21 - Alocação particionada estática absoluta



Fonte: Machado e Maia (2013, p. 150).

Na **alocação particionada estática relocável**, um programa pode ser carregado em qualquer partição livre com tamanho suficiente. Veja a diferença do mecanismo anterior na seguinte figura:

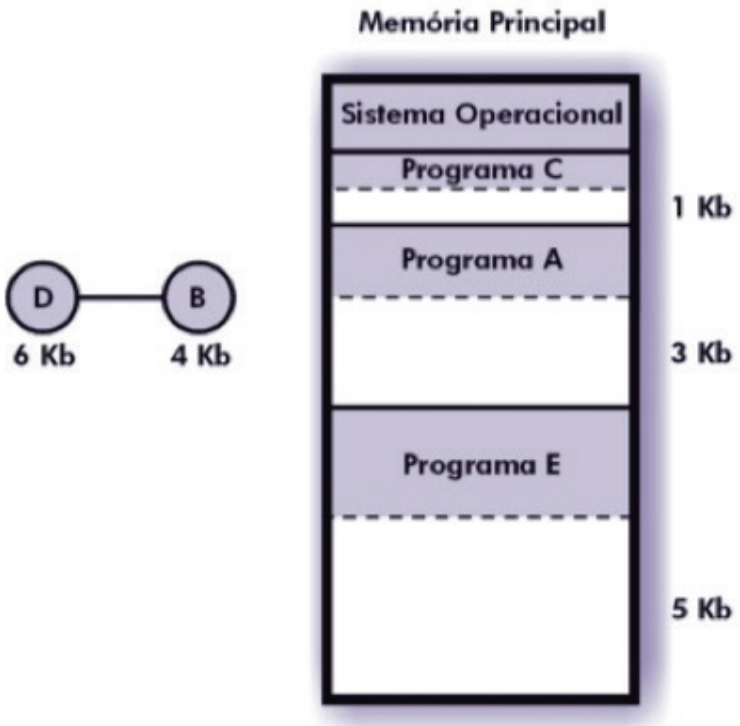
Figura 3.22 - Alocação particionada estática relocável



Fonte: Machado e Maia (2013, p. 150).

Esse tipo de técnica gera muita **fragmentação**, ou seja, áreas livres de memória que não são utilizadas por serem muito pequenas.

Figura 3.23 - Fragmentação interna gerada pela alocação particionada

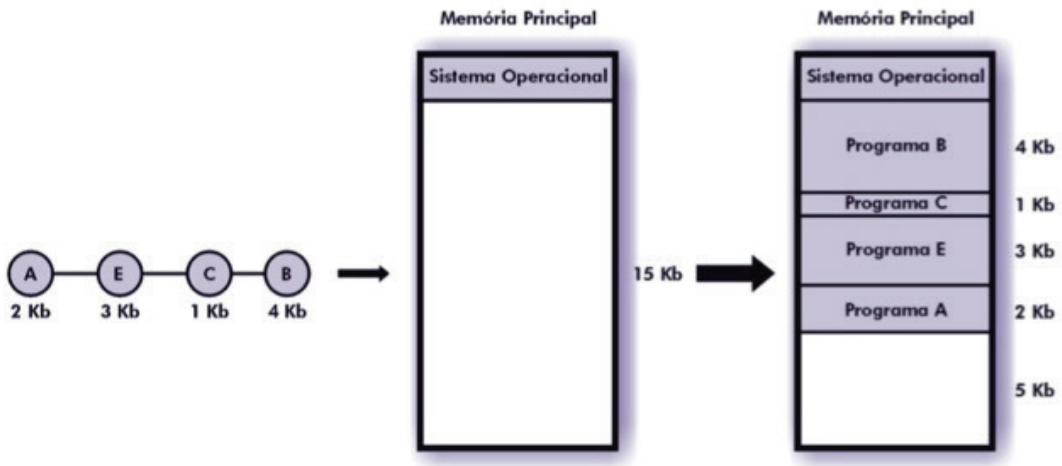


Fonte: Machado e Maia (2013, p. 152).

3.3 Alocação particionada dinâmica

Na estratégia de **alocação particionada dinâmica** não existe partição de tamanho fixo. Cada programa ocupa apenas o espaço de que necessita. Esse mecanismo, embora seja mais eficiente do que os anteriores, também gera problemas de fragmentação, e mais sérios, quando os programas vão acabando sua execução.

Figura 3.24 - Alocação particionada dinâmica



Fonte: Machado e Maia (2013, p. 152).

O problema da fragmentação é o mais sério no que se refere à gerência de memória. De acordo com Machado e Maia (2013), existem algumas soluções para esses casos, como:

- reunir somente os espaços adjacentes (mais simples), tornando maior a partição livre; ou
- fazer a relocação de todas as partições ocupadas, o que gera um maior consumo dos recursos do sistema.

Uma vez que os programas podem ser alocados em qualquer espaço livre e com tamanho suficiente, não sendo necessário espaços pré-definidos, deve existir uma estratégia para melhor alocar os espaços, fazendo com que a utilização da memória seja otimizada e evitando a fragmentação. O sistema operacional mantém uma lista de áreas livres e seu respectivo tamanho.

De acordo com Machado e Maia (2013) as principais **estratégias de escolha da participação** são:

- best-fit;
- worst-fit; e
- first-fit.

A primeira, **best-fit**, diz respeito ao melhor espaço – é escolhida a “melhor” partição, ou seja, sobre o menor espaço. Nesse caso, a fragmentação gerada é de espaços bem pequenos, aumentando o problema.

A segunda, **worst-fit**, diz respeito ao pior espaço. Nesse caso, é escolhida a pior partição, ou seja, a que deixa o maior espaço livre. Os espaços livres que sobram são maiores, permitindo que um maior número de programas possa ser carregado. Diminui significativamente o problema da fragmentação.

Por fim, a **first-fit**, diz respeito ao primeiro espaço. Nesse caso, é escolhida a primeira partição livre de tamanho suficiente. A lista de espaços livres está em ordem de endereço de memória. Esse modelo é o mais rápido e acaba sendo mais eficiente, uma vez que existe a chance de encontrar espaços livres maiores concentrados nos endereços maiores de memória.

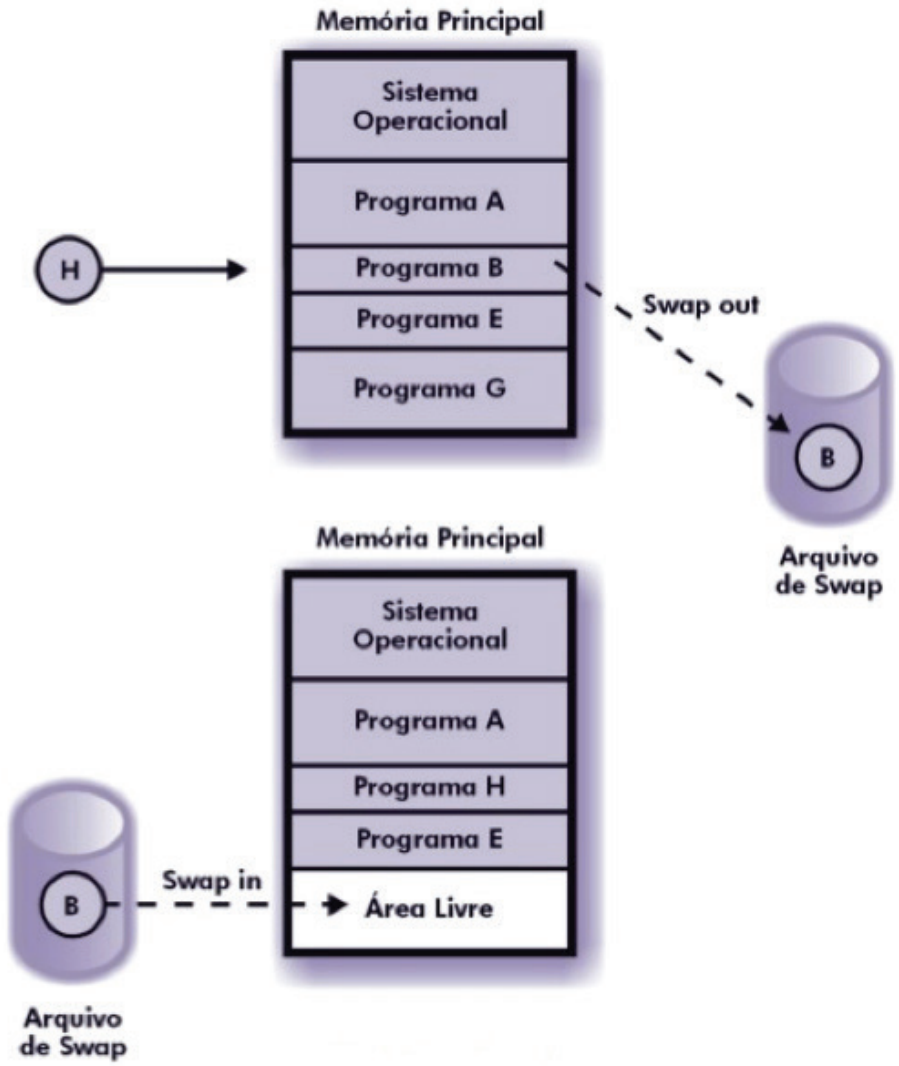
3.4 Swapping

A técnica de swapping foi desenvolvida com o objetivo de minimizar o problema de falta de espaço em memória para a execução dos programas.

Qualquer programa, para ser executado, deve estar carregado na memória principal.

Com a técnica de swapping, é possível que um programa que não está utilizando o processador seja levado para o disco (swapped-out), dando lugar a outro que necessite de área em memória, e depois volta (swapped-in), como se nada tivesse acontecido. Existe um espaço no disco dedicado a essa função. Para utilização dessa técnica é necessário que haja relocação dinâmica.

Figura 3.25 - Swapping



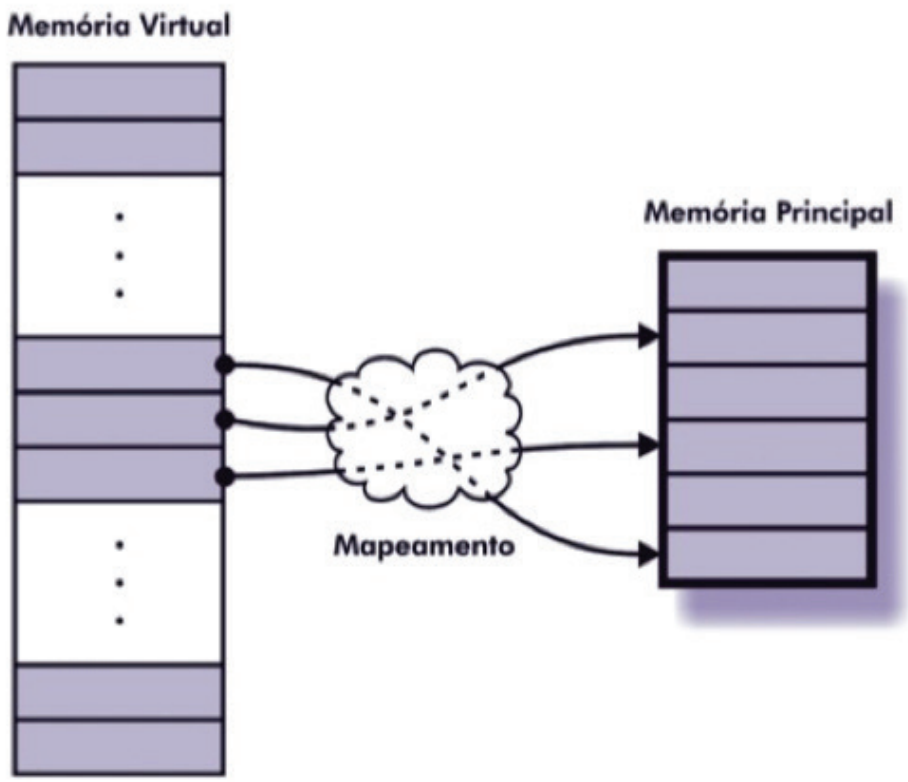
Fonte: Machado e Maia (2013, p. 156).

3.5 Memória virtual

É uma técnica que combina a memória principal e secundária (normalmente, para isso, é utilizado o disco rígido), dando ao usuário a ilusão de uma memória muito maior do que a principal. O tamanho da memória física disponível não é mais problema. Essa técnica, atualmente implementada em praticamente todos os sistemas operacionais de uso comum, também tem como objetivo minimizar o problema da fragmentação.

Ela trabalha com um **espaço de endereçamento virtual**, que é um conjunto de endereços virtuais que os processos podem endereçar. O compilador e o linkeditor geram código em função desses endereços e o sistema operacional executa. O mecanismo que traduz o endereço virtual para o real é chamado de **mapeamento**. Somente a parte do programa que está em execução é carregada fisicamente na memória principal. Com o mecanismo de **mapeamento**, o programa em execução não precisa estar em áreas contíguas na memória principal para ser executado. Cada processo tem a sua **tabela de endereçamento virtual**.

Figura 3.26 - Mapeamento



Fonte: Machado e Maia (2013, p. 162).

O mapeamento pode ser feito com blocos de:

- mesmo tamanho (páginas); ou
- diferentes tamanhos (segmentos).

Isso quer dizer que a memória pode ser organizada em espaços de mesmo tamanho ou organizada em espaços de diferentes tamanhos, os quais podem variar conforme a estrutura do programa.

No esquema de **paginação,** o espaço de endereçamento virtual e o espaço de endereçamento real são divididos em blocos de mesmo tamanho (**páginas**). O tamanho da página pode variar de 512 a 16M, conforme a arquitetura do computador (MACHADO; MAIA, 2013).

No mecanismo de **segmentação**, o programa é dividido logicamente em subrotinas e estrutura de dados, colocados em blocos de informações na memória. Os blocos têm tamanho diferente e são chamados de **segmentos,** onde cada um tem seu próprio espaço de endereçamento. Permite a relação entre a lógica do programa e a sua divisão na memória; assim, quanto mais modular o programa, melhor a performance.

Existe também a combinação entre os dois mecanismos: **segmentação com paginação**. Nesse tipo de mecanismo, o programa é dividido logicamente em segmentos e eles fisicamente em páginas.

Capítulo 4

O sistema de arquivos e a gerência de dispositivos

Habilidades

Com a leitura deste capítulo, você conhecerá as características básicas de um sistema de arquivos e poderá compreender sua estrutura, identificar os sistemas de arquivos atuais e conhecer os conceitos básicos da gerência de dispositivos e suas funcionalidades.

Seções de estudo

Seção 1: Sistema de arquivos

Seção 2: Gerência de dispositivos

Seção 1
Sistema de arquivos

Sob o ponto de vista do usuário, um arquivo é constituído de uma coleção de registros, que podem ser de tamanho fixo ou variável, cuja finalidade é guardar informações em dispositivos de armazenamento permanente.

Os arquivos podem conter programas-fontes, relatórios, textos e dados numéricos, entre outras informações. Observe alguns exemplos de arquivos com diferentes extensões – o que, normalmente, indica o tipo de arquivo:

Quadro 4.1 - Extensão de arquivos

Extensão	Significado
.bak	Cópia de segurança
.c	Código-fonte de programa C
.gif	Imagem no formato Graphical Interchange Format
.hlp	Arquivo de ajuda
.html	Documento em HTML
.jpg	Imagem codificada segundo padrões JPEG
.mp3	Música codificada no formato MPEG (camada 3)
.mpg	Filme codificado no padrão MPEG
.o	Arquivo objeto (gerado por compilador, ainda não ligado)
.pdf	Arquivo no formato PDF (Portable Document File)
.ps	Arquivo PostScript
.tex	Entrada para o programa de formatação TEX
.txt	Arquivo de texto
.zip	Arquivo compactado

Fonte: Tanenbaum (2009, p. 160).

Com o objetivo de permitir um gerenciamento eficiente dessas informações, um conjunto de programas é suportado pelos sistemas operacionais, formando o que é chamado de **sistema de arquivos**.

O sistema de arquivos é o componente do sistema operacional, com o qual a grande maioria dos usuários de um computador tem contato direto. No entanto,

essa aproximação faz com que esses usuários utilizem as facilidades do sistema de arquivos sem levar em consideração sua complexidade, que não resulta da natureza das facilidades oferecidas ao usuário, mas da necessidade de garantir a segurança e a integridade do sistema de arquivos.

Com a finalidade de oferecer uma interface amigável ao usuário, um sistema de arquivos deve suportar um **conjunto de funcionalidades**, como:

- ser capaz de criar e eliminar arquivos;
- ser capaz de controlar o acesso aos arquivos;
- permitir referências aos arquivos pelo nome simbólico e não sobre a localização precisa dos arquivos na memória secundária;
- permitir o compartilhamento de arquivos;
- listar os arquivos de cada usuário;
- ter arquivos protegidos contra falhas de software e de hardware;
- introduzir novos usuários ou retirar usuários e seus arquivos do sistema; e
- introduzir memória adicional (expansão) como unidade de disco e fita magnética a ser usada pelo sistema de arquivos.

Podemos perceber que essa função do sistema operacional para o usuário final é, de fato, uma das mais palpáveis.

1.1 Diretórios e direitos de acesso

O diretório é a forma como os arquivos de um determinado sistema de computação estão organizados. Ele deve conter as seguintes informações:

- o nome simbólico do arquivo;
- o tipo do arquivo;
- a posição e o tamanho do arquivo;
- informações para contabilização; e
- o tipo de acesso permitido ao arquivo.

A estrutura dos diretórios e o relacionamento entre eles é o principal ponto onde os sistemas de arquivo tendem a se diferenciar, apesar da semelhança entre o conjunto de facilidades oferecidas por eles.

Existem algumas operações que podem ser realizadas sobre a estrutura de um diretório, como:

- pesquisa;
- criação de arquivos; e
- eliminação de arquivos.

Por **pesquisa**, entendemos que deve ser possível percorrer a estrutura de um diretório com o objetivo de encontrar um determinado arquivo. Outra operação essencial é a possibilidade de **criação** e inclusão de novos arquivos no diretório. Também deve existir a possibilidade de **eliminação** de arquivos existentes em um diretório, bem como as informações a respeito deles.

A facilidade de compartilhar arquivos é um tópico muito complexo e depende, em grande parte, da relação entre os diretórios e da informação de permissão que cada diretório mantém. Essa informação indica como outros usuários podem acessar os arquivos.

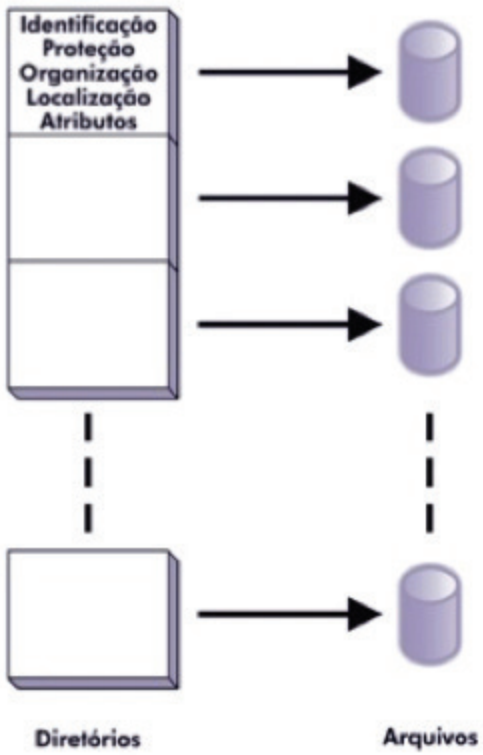
A associação de informação de permissão de acesso para cada arquivo tem a finalidade de proteger o arquivo de acesso ilegal. Assim, por exemplo, um arquivo texto poderá ter acesso para leitura e escrita associado a ele.

Várias estruturas de diretórios têm sido propostas, como as que seguem:

- diretório de um único nível;
- diretório de dois níveis; e
- diretório hierárquico.

O **diretório de um único nível** é a forma mais simples de organizar arquivos em um sistema de arquivos. Nesse caso, todos os arquivos estão em um mesmo diretório, como mostra a figura seguinte:

Figura 4.1 - Estrutura de diretórios de nível único

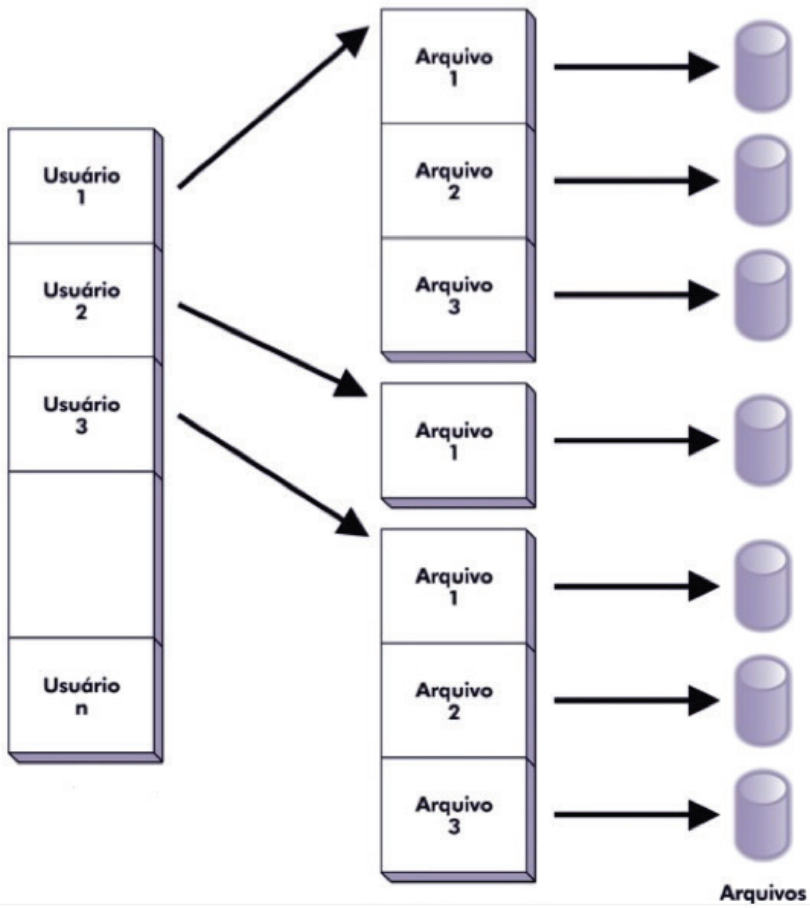


Fonte: Machado e Maia (2013, p. 198).

Esse tipo de diretório possui limitações; o nome do arquivo, por exemplo, deve ser único, mesmo que vários usuários tenham acesso ao sistema.

No caso do **diretório de dois níveis**, o sistema mantém um bloco mestre que tem uma entrada para cada usuário do computador, como ilustrado na figura seguinte:

Figura 4.2 - Estrutura de diretório em dois níveis



Fonte: Machado e Maia (2013, p. 199).

O bloco mestre contém os endereços dos diretórios individuais, todos os diretórios estão no mesmo nível e todos os usuários são considerados igualmente.

Quando um usuário entra no sistema, o diretório e todos os arquivos associados a ele são disponibilizados para seu uso. Ainda, pode-se associar uma senha ao arquivo para obter uma proteção adicional.

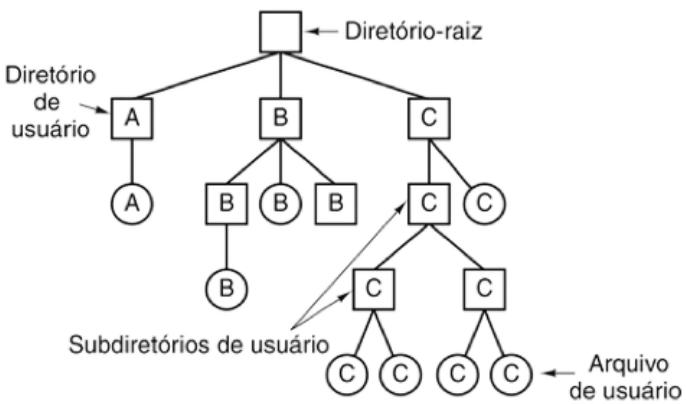
Esse tipo de diretório pode ser adequado à situação onde todos os usuários possuem o mesmo status. Caso contrário, é necessário um diretório cuja estrutura reflita a natureza da organização dos usuários do computador.

Na figura anterior, pudemos observar que a estrutura de dois níveis é formada por um nível que contém as informações dos usuários e outro que contém as informações do conjunto de arquivos de cada usuário. O segundo nível possui o endereço onde o arquivo se encontra gravado.

O **diretório hierárquico** é uma extensão do diretório com dois níveis – que pode ser visto como uma árvore de dois níveis – para uma árvore com vários níveis. Essa organização permite aos usuários criarem seus próprios subdiretórios, organizando seus arquivos dentro deles. A árvore assim formada possui um diretório raiz, a partir do qual são criados os subdiretórios. Um caminho (path) é definido a partir do diretório raiz, passando por todos os subdiretórios, até chegar a um determinado arquivo.

A principal vantagem de ter um diretório hierárquico é a facilidade de estruturar os usuários no sistema, facilitando o controle sobre sua utilização de recursos. A figura seguinte é um exemplo de um diretório hierárquico:

Figura 4.3 - Estrutura de diretório de múltiplos níveis



Fonte: Tanenbaum (2013, p. 167).

Novamente, o acesso aos arquivos é feito por meio do diretório apropriado. Pode-se definir a estrutura de um diretório hierárquico de tal forma que os usuários de um subdiretório sejam capazes de acessar somente os arquivos que se encontram sob aquele diretório. Acessos entre diretórios só são permitidos considerando a hierarquia dos mesmos. Nesse exemplo, o dono dos diretórios e arquivos está indicado por uma letra (A, B ou C).

1.2 Compartilhamento e segurança

O problema da segurança de arquivos surge como uma consequência imediata do desejo de compartilhá-los entre os vários usuários.

Em um sistema que não permite compartilhamento, onde a única pessoa que pode acessar um arquivo é seu proprietário, a segurança pode ser obtida checando a identidade do usuário.

Quando os arquivos podem ser compartilhados, seu proprietário necessita de um meio de especificar os usuários que poderão acessá-los. Além disso, é conveniente ao proprietário ser capaz de especificar o tipo de acesso que será permitido. Pode-se permitir que alguns usuários, por exemplo, atualizem um arquivo, que outros somente o leiam e que outros somente o carreguem para execução.

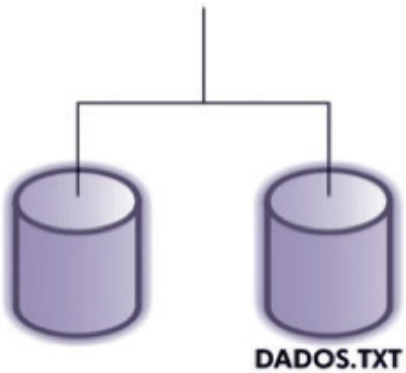
O proprietário deve ser capaz de determinar os **direitos de acesso**. Alguns exemplos típicos de direitos de acesso que podem ser atribuídos a arquivos são:

- nenhum acesso (N);
- executar somente (X);
- ler somente (R);
- alterar (A);
- eliminar (E); e
- gravar (G).

A partir desses direitos de acesso, pode-se criar uma chave de proteção de arquivos, composta de um ou mais tipos, podendo ser atribuída a cada arquivo ou a cada usuário de um arquivo.

A figura seguinte mostra um exemplo de compartilhamento do arquivo DADOS. TXT, apenas para leitura, com os usuários que fazem parte do grupo ao qual o proprietário pertence:

Figura 4.4 - Proteção por grupos de grupos de usuários



Nível de Proteção	Tipo de Acesso
Owner	Leitura Escrita Execução Eliminação
Group	Leitura
All	—

Fonte: Machado e Maia (2013, p. 206).

Seção 2
Gerência de dispositivos

A gerência de dispositivos é uma função do sistema operacional exercida por meio de um conjunto de rotinas e dispositivos.

> É a parte do sistema operacional que cuida do compartilhamento e do acesso aos dispositivos de entrada e saída do sistema.

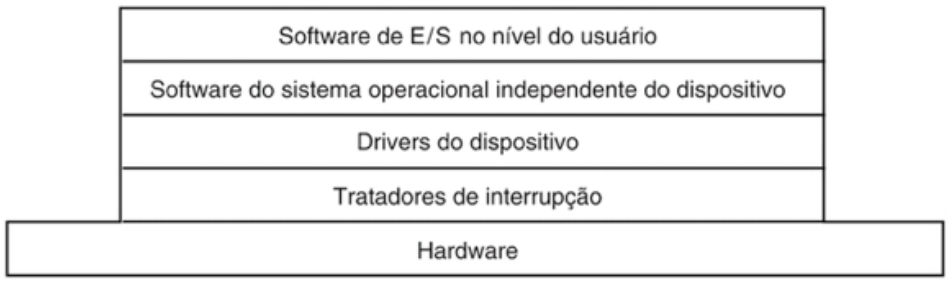
Para facilitar essa gerência, as rotinas foram divididas em diversas camadas: as camadas de baixo nível tratam mais da parte do hardware do dispositivo, ficando escondidas das camadas de mais alto nível (MACHADO; MAIA, 2013).

Basicamente, existem dois grupos de camadas:

- o primeiro visualiza os diversos dispositivos do sistema; e
- o segundo é específico para cada dispositivo.

A maioria das rotinas do sistema operacional manipula os dispositivos da mesma maneira, ou seja, o sistema operacional pode se comunicar com qualquer periférico. Esta característica é chamada de **independência de dispositivos**.

Figura 4.5 - Camadas do software de e/s

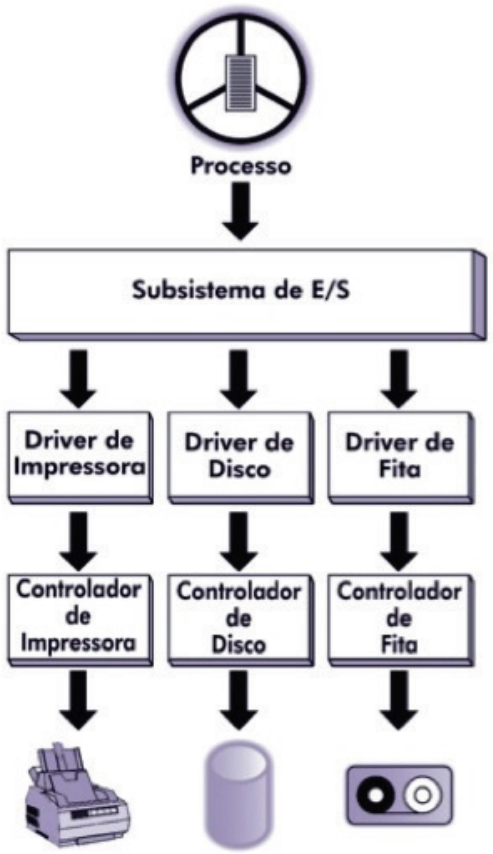


Fonte: Tanenbaum (2009, p. 214).

As **rotinas de e/s** são responsáveis pela comunicação entre o programa do usuário e os dispositivos de e/s, evitando que o programador tenha que se preocupar com o dispositivo envolvido na operação. Elas são responsáveis por implementar o mecanismo de proteção de acesso aos dispositivos e aos arquivos e, principalmente, abstrair do programador características específicas de cada dispositivo.

Os **device drivers** são rotinas do sistema operacional que têm como função fazer a comunicação com os dispositivos de e/s em nível de hardware, geralmente por controladores. Nessa camada são especificadas as características físicas de um dispositivo. Sua principal função é receber instruções abstratas, geralmente das **rotinas de e/s** e, traduzi-las para comandos que o **controlador** do dispositivo entenda e execute.

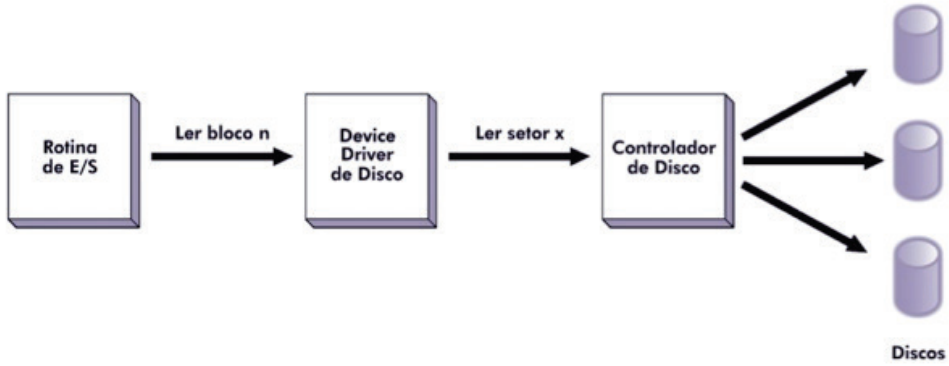
Figura 4.6 - Device drivers



Fonte: Machado e Maia (2013, p. 211).

A figura seguinte apresenta um exemplo de uma operação de e/s, durante a qual a rotina de e/s informa ao device driver do disco a requisição de leitura de um bloco (n). O device driver interpreta a requisição e informa ao controlador o local no disco onde se encontra o bloco (n). O controlador, então, realiza a leitura do bloco. Durante todo esse tempo, o processo que fez a solicitação de leitura fica em **estado de espera**.

Figura 4.7 - Driver de disco



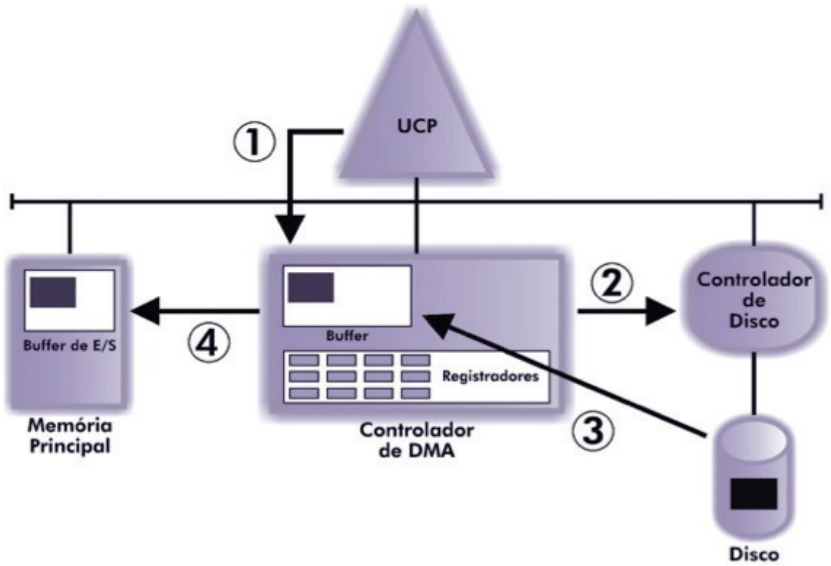
Fonte: Machado e Maia (2013, p. 212).

Os **controladores** são componentes eletrônicos responsáveis por manipular diretamente os dispositivos de e/s. O sistema operacional quase sempre se comunica com o controlador e não com o dispositivo. Geralmente, possuem memória e registradores próprios para poder executar as instruções enviadas pelo device driver.

A maioria dos controladores implementa a técnica de Directory Memory Access (DMA), sendo que, ao término da transferência, o controlador gera uma interrupção na UCP, avisando a conclusão da transferência. O sistema operacional testa, então, os resultados por meio dos registradores do controlador.

Observe que o conteúdo lido do disco fica no buffer de memória do controlador e depois no buffer da memória principal:

Figura 4.8 - Técnica de DMA



Fonte: Machado e Maia (2013, p. 214).

Os **dispositivos de e/s** são os responsáveis pela comunicação entre o computador e o mundo externo. De acordo com Machado e Maia (2013), esses dispositivos podem ser classificados em:

- estruturados; e
- não estruturados.

Os dispositivos **estruturados** armazenam informações em blocos de tamanho fixo, onde cada um possui um endereço fixo, como, por exemplo, o disco magnético. Os dispositivos **não estruturado**s fazem a transferência das informações por meio de uma sequência de caracteres não endereçáveis, não existindo a possibilidade de acesso após a transmissão. Exemplos são monitores, impressoras, mouse e teclado.

Considerações Finais

Ao longo da leitura deste livro, pudemos entender um pouco mais a respeito da funcionalidade de um sistema operacional e de suas características principais.

Agora, você é capaz de identificar o sistema mais adequado, conforme sua necessidade. Pode, também, reconhecer a origem de falhas e é capaz de corrigi-las.

Espero poder tê-lo auxiliado em busca de mais conhecimento e coloco-me à disposição ao longo de sua jornada acadêmica.

Meus sinceros agradecimentos,
Márcia Cargnin Martins Giraldi

Referências

HENNESSY, J. L.; PATTERSON, D. A.; LARUS, J. R. **Organização e Projeto de Computadores**: A Interface Hardware/Software. 2. ed. Rio de Janeiro: LTC, 2000.

MACHADO, F. B.; MAIA, L. P. **Arquitetura de Sistemas Operacionais**. 3. ed. Rio de Janeiro: LTC, 2002.

_____. **Arquitetura de Sistemas Operacionais**. 4. ed. Rio de Janeiro: LTC, 2007.

_____. **Arquitetura de Sistemas Operacionais**. 5. ed. Rio de Janeiro: LTC, 2013. Disponível em: <http://integrada.minhabiblioteca.com.br/books/978-85-216-2288-8/page/15>. Acesso em: 14 dez. 2014. (Acesso restrito MinhaUnisul).

MADEIRA, M. N. **Software Livre**: Livro Didático. Palhoça: UnisulVirtual, 2011.

TANENBAUM, A.S.; **Sistemas Operacionais Modernos**. 3. ed. Rio de Janeiro: Prentice-Hall do Brasil, 2009.

Sobre a Professora Conteudista

Márcia Cargnin Martins Giraldi

Formou-se em Ciência da Computação pela Universidade Federal de Santa Catarina (UFSC) no final de 1985 e, nesse mesmo ano, iniciou sua vida profissional na Telesc/SA (Telecomunicações de Santa Catarina), permanecendo até o final de 1989, exercendo o cargo de Analista de Sistemas. Em 1998, cursou Especialização em Administração Pública na Escola Superior de Administração e Gerência (Esag) da Universidade do Estado de Santa Catarina (UDESC) e em 1995, cursou Especialização em Informática na UFSC, tendo concluído o mestrado em Ciência da Computação pela mesma universidade no ano 2000. Trabalhou também no Banco Bamerindus (Curitiba, PR) na área de desenvolvimento de sistemas, equipe Conta Corrente e Base de Restritivos. Em 1993, passou a trabalhar na Chapecó Cia. Industrial de Alimentos, onde participou e liderou projetos de implantação de softwares integrados. Na Unisul desde 1996, atua como professora nos cursos de Ciência da Computação e Sistemas de Informação, sendo que, paralelamente, fez parte da equipe de desenvolvimento da Assessoria de Tecnologia da Informação (ATI) de 1997 a 2002, quando passou a integrar a equipe da Coordenação de Ensino/Diretoria de Ensino Pesquisa e Extensão. Em 2008, passou a coordenar o curso de Ciência da Computação, no campus de Tubarão. Atua, também, como professora nos cursos presenciais e a distância.