e-Tec Brasil
Escola Técnica Aberta do Brasil

Sistemas Operacionais

Bruno Cardoso Coutinho

Curso Técnico em Informática

INSTITUTO FEDERAL
ESPÍRITO SANTO

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Ministério da Educação

e-Tec Brasil
Escola Técnica Aberta do Brasil

Sistemas Operacionais

Bruno Cardoso Coutinho

INSTITUTO FEDERAL
ESPÍRITO SANTO

Colatina - ES
2010

Presidência da República Federativa do Brasil

Ministério da Educação

Secretaria de Educação a Distância

Este Caderno foi elaborado em parceria entre o Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo e a Universidade Federal de Santa Catarina para o Sistema Escola Técnica Aberta do Brasil – e-Tec Brasil.

Equipe de Elaboração
Instituto Federal do Espírito Santo – IFES

Coordenação do Curso
Joao Henrique Caminhas Ferreira/IFES

Professores-autores
Bruno Cardoso Coutinho/IFES

Comissão de Acompanhamento e Validação
Universidade Federal de Santa Catarina – UFSC

Coordenação Institucional
Araci Hack Catapan/UFSC

Coordenação do Projeto
Silvia Modesto Nassar/UFSC

Coordenação de Design Instrucional
Beatriz Helena Dal Molin/UNIOESTE e UFSC

Coordenação de Design Gráfico
Carlos Antônio Ramirez Righi/UFSC

Design Instrucional
Alessandro Poleto Oliveira/IFES

Web Master
Rafaela Lunardi Comarella/UFSC

Web Design
CEAD/IFES

Diagramação
Edison Patto/UFSC
Guilherme Ataide Costa/UFSC
Juliana Tonietto/UFSC

Revisão
Luciane Ferreira Lacerda/IFES

Projeto Gráfico
e-Tec/MEC

C871s Coutinho, Bruno Cardoso

Sistemas operacionais : Curso Técnico em Informática / Bruno Cardoso Coutinho. – Colatina: CEAD / Ifes, 2010.
78 p. : il.

1. Sistemas operacionais (Computadores). 2. Sistemas de computação. 3. Material didático. I. Instituto Federal do Espírito Santo. II. Título.

CDD: 005.43

Apresentação e-Tec Brasil

Prezado estudante,

Bem-vindo ao e-Tec Brasil!

Você faz parte de uma rede nacional pública de ensino, a Escola Técnica Aberta do Brasil, instituída pelo Decreto nº 6.301, de 12 de dezembro 2007, com o objetivo de democratizar o acesso ao ensino técnico público, na modalidade a distância. O programa é resultado de uma parceria entre o Ministério da Educação, por meio das Secretarias de Educação a Distancia (SEED) e de Educação Profissional e Tecnológica (SETEC), as universidades e escolas técnicas estaduais e federais.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade, e promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

O e-Tec Brasil leva os cursos técnicos a locais distantes das instituições de ensino e para a periferia das grandes cidades, incentivando os jovens a concluir o ensino médio. Os cursos são ofertados pelas instituições públicas de ensino e o atendimento ao estudante é realizado em escolas-polo integrantes das redes públicas municipais e estaduais.

O Ministério da Educação, as instituições públicas de ensino técnico, seus servidores técnicos e professores acreditam que uma educação profissional qualificada – integradora do ensino médio e educação técnica, – é capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!

Desejamos sucesso na sua formação profissional!

Ministério da Educação
Janeiro de 2010

Nosso contato
etecbrasil@mec.gov.br

Indicação de ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.

Atenção: indica pontos de maior relevância no texto.

Saiba mais: oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.

Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.

Mídias integradas: sempre que se desejar que os estudantes desenvolvam atividades empregando diferentes mídias: vídeos, filmes, jornais, ambiente AVEA e outras.

Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Sumário

Palavra do professor-autor

Olá,

Meu nome é Bruno Cardoso Coutinho, formado em Ciência da Computação com mestrado em Informática pela UFES. Sou professor do IFES campus Colatina desde janeiro de 2009, onde tenho ministrado disciplinas como Sistemas Operacionais Locais e de Rede. Ultimamente, além das minhas atividades de professor, tenho me aventurado em pesquisas científicas nas áreas de redes de computadores, ambientes distribuídos e otimização.

A disciplina de Sistemas Operacionais é de extrema importância para seu curso e para sua carreira profissional, já que se trata do sistema gestor de qualquer computador. O Sistema Operacional é que organiza a execução dos aplicativos, aloca espaço em memória para uma execução mais rápida, envia e recebe dados de dispositivos e os trata para serem utilizados pelos aplicativos dos usuários, além de muitas outras atividades. Em suma, o Sistema Operacional protege a máquina do usuário e protege o usuário da máquina.

Por ser uma matéria essencial para seu curso, empenhe-se nos estudos, leia o material com calma e releia se for o caso. Tire as suas dúvidas com seus tutores e utilize os materiais indicados como apoio a seus estudos. Não se prenda apenas à apostila do curso.

Sucesso na sua carreira!

Um grande abraço!
Prof. Bruno.

Apresentação da disciplina

Nesta disciplina você terá uma visão geral dos sistemas operacionais, bem como aprenderá conceitos fundamentais e como é feito o gerenciamento de recursos de *hardware* e *software* do computador.

O Sistema Operacional é o grande gestor do computador, com muitas responsabilidades como: alocar recursos, gerenciar usuários e processos, controlar a execução de programas de usuários e muito mais. Ao passar pelas aulas deste curso você poderá perceber o quão difícil e árduo é construir um sistema como *Windows* ou *Linux*.

Você algumas vezes deve ter passado por esta situação: estar navegando na internet, conversando no MSN e editando um trabalho de escola no *Word*. Parece estar tudo executando ao mesmo tempo correto? Mas provavelmente não. Apesar de as máquinas mais novas conseguirem processar algumas instruções realmente em paralelo, ainda sim, esses aplicativos disputam recursos sob a gerência do Sistema Operacional.

Nas primeiras aulas do curso, faremos uma caracterização dos sistemas operacionais levando em consideração a evolução do *hardware* ao longo dos anos. Conhecer o histórico do desenvolvimento destes sistemas também é importante para analisar a motivação de cada nova tecnologia e sua relação com o *software* gerenciador. Depois, precisaremos rever alguns conceitos de *hardware* e *software*, caracterizar alguns componentes básicos e sua importância em um sistema computacional. Por fim, analisaremos as principais funções de gerência de um sistema computacional, abordando seus tópicos principais.

Vamos estudar, também, como funciona a gerência de aplicações, de recursos e a estrutura interna de um sistema operacional. São muitos conceitos novos, não deixe acumular!

Projeto instrucional

Disciplina: Sistemas Operacionais (carga horária: 60h)**.**

Ementa: Visão geral dos Sistemas Operacionais. Conceitos e gerenciamento de recursos de *hardware* e *software* do computador.

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
1. Visão geral de Sistemas Operacionais	Compreender os conceitos básicos de sistemas operacionais. Conhecer suas funções principais. Analisar o Sistema Operacional como uma máquina de níveis.	Caderno e Ambiente Virtual de Ensino-Aprendizagem. www.cead.ifes.edu.br	10
2. Histórico e classificação	Conhecer o histórico de sistemas operacionais. Compreender como as inovações de hardware colaboraram com o desenvolvimento dos sistemas operacionais. Saber classificar os sistemas conforme suas características principais.	Caderno e Ambiente Virtual de Ensino-Aprendizagem. www.cead.ifes.edu.br	10
3. Elementos de *hardware* e *software* – Parte I	Conhecer a arquitetura básica de computadores. Descrever os principais dispositivos de entrada e saída. Compreender conceitos de *software* utilitário.	Caderno e Ambiente Virtual de Ensino-Aprendizagem. www.cead.ifes.edu.br	10
4. Elementos de *hardware* e *software* – Parte II	Conhecer conceitos mais específicos sobre programas de sistema. Analisar o mecanismo de interrupção na concorrência entre processos. Descrever as operações de entrada e saída. Conhecer as características dos sistemas em lote e sistemas de tempo compartilhado. Compreender a funcionalidade de alguns serviços do sistema operacional. Analisar as características básicas de uma arquitetura de sistema operacional.	Caderno e Ambiente Virtual de Ensino-Aprendizagem. www.cead.ifes.edu.br	10

continua

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
5. Arquitetura do Sistema Operacional	Conhecer os modos de acesso ao processador como forma de proteção do sistema. Compreender a estrutura das chamadas de sistema utilizadas para a comunicação com o *kernel* do sistema. Analisar as características de uma arquitetura de sistema operacional dividido em camadas ou não. Verificar o funcionamento e importância de um interpretador de comandos.	Caderno e Ambiente Virtual de Ensino- Aprendizagem. www.cead.ifes.edu.br	10
6. Introdução à gerência de processos, memória e arquivos	Conhecer as funções de gerência principais de um sistema operacional. Analisar as funções do elemento processo dentro de um sistema operacional. Verificar a administração da utilização de recursos pelo sistema operacional.	Caderno e Ambiente Virtual de Ensino-Aprendizagem. www.cead.ifes.edu.br	10
			conclusão

Aula 1 – Visão geral de Sistemas Operacionais

Objetivos

Compreender os conceitos básicos de Sistemas Operacionais.

Conhecer suas funções principais.

Analisar o Sistema Operacional como uma máquina de níveis.

1.1 Conceitos básicos

Diferentemente do que muitas pessoas imaginam, o computador não faz nada sozinho. Ele apenas processa uma série de informações inseridas pelo usuário para então fornecer os resultados. As informações inseridas e os resultados que recebemos precisam estar num formato que nós humanos conseguimos entender. Para facilitar essa comunicação entre homem e computador, foram criados os *softwares* ou programas de computador. Na realidade, tudo que fazemos com um computador é pela execução desses programas.

De acordo com um dos principais autores da área,

> Um sistema operacional é um programa que atua como intermediário entre o usuário e o *hardware* de um computador. O propósito de um sistema operacional é propiciar um ambiente no qual o usuário possa executar outros programas de forma conveniente, por esconder detalhes internos de funcionamento e eficiência, por procurar gerenciar de forma justa os recursos do sistema (Silberschatz, Galvin e Gagne, 2000, p.22].

Vamos estender o conceito de **sistema operacional** ao longo do curso, mas podemos definí-lo, de forma simples, como um conjunto de rotinas executadas pelo processador com a principal função de controlar o funcionamento do computador, gerenciando os diversos recursos disponíveis no sistema. Na Figura 1.1 vemos a posição que um Sistema Operacional ou simplesmente "SO" ocupa dentre os vários elementos que compõem um sistema de computação. Você deve observar que a palavra "Usuários" está sendo usada com dois sentidos diferentes: para as pessoas que utilizam o computador e para os programas e utilitários instalados no computador.

A-Z

Sistema Operacional
Sistema Operacional: segundo o Aurélio (verbete sistema), sistema operacional é um conjunto integrado de programas básicos, projetado para supervisionar e controlar a execução de programas de aplicação em um computador.

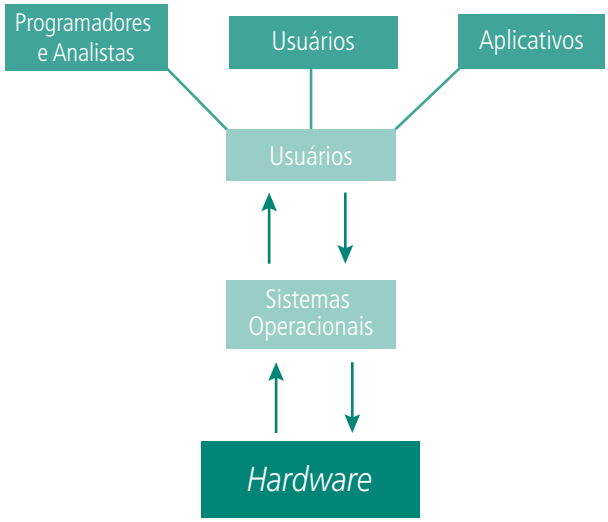

Figura 1.1: Visão do Sistema Operacional
Fonte: Adaptado de Machado, 2004

Resumidamente, o sistema operacional tem a função de proteger a máquina do usuário e proteger o usuário da máquina.

1.2 Funções principais

Na Figura 1.1 foi destacado o controle de *hardware.* Esta é uma das funções básicas do SO e pode ser desmembrada em:

a) Facilidade de acesso aos recursos do sistema

Um sistema de computação possui, normalmente, diversos componentes, como monitores, impressoras e discos rígidos. Quando utilizamos um desses dispositivos, não nos preocupamos com a maneira como é realizada esta comunicação e os inúmeros detalhes envolvidos.

Você pode obter mais informações sobre "setor" e "trilha" no livro Organização Estruturada de Computadores, de Andrew S. Tanembaum, 5ª Edição, Editora Prentice-Hall, ou na própria internet em *sites* especializados como o "Clube do *Hardware*" em http://www.clubedohardware.com.br

Uma operação frequente como, por exemplo, a leitura de um arquivo em um CD ou disco pode parecer simples. Existe um conjunto de rotinas específicas, controladas pelo sistema operacional, que são responsáveis por acionar a cabeça de leitura e gravação da unidade de disco, posicionar na trilha e setor onde estão os dados, transferir os dados do disco para a memória e, finalmente, informar ao programa a chegada dos dados.

O sistema operacional, então serve de interface entre o usuário e os recursos de *hardware*, tornando esta comunicação transparente (ou imperceptível) e permitindo ao usuário um trabalho mais eficiente e com menos possibilidades de erros.

b) Compartilhamento de recursos de forma organizada e protegida

Se imaginarmos, por exemplo, que uma impressora pode ser utilizada por vários usuários do sistema, então deverá existir algum controle para impedir que a impressão de um usuário interrompa a impressão de outro. O sistema operacional é o responsável por permitir o acesso organizado a esse e a outros recursos disponíveis no computador.

O compartilhamento de recursos permite a diminuição de custos, na medida em que mais de um usuário pode utilizar as mesmas facilidades concorrentemente, tais como discos, impressoras, linhas de comunicação, etc. Com isto, uma mesma impressora (ou linha de comunicação ou outro recurso) pode atender a vários usuários.

Não é só no controle do acesso a hardware compartilhado que o sistema operacional atua, ele nos permite executar várias tarefas, como imprimir um documento, copiar um arquivo pela internet ou processar uma planilha, entre outros. O SO deve ser capaz de controlar a execução concorrente de todas essas tarefas. Ainda podemos dizer que, embora alguns programas sejam escritos baseados nas instruções de um determinado processador, será responsabilidade do sistema operacional executar tarefas básicas do micro, ou seja, ensinar ao processador como desenhar uma janela ou imprimir um documento.

> De um modo geral, os programas que os usuários executam não são escritos para um processador, mas sim para um SO. Isto facilita a comunicação do programa com o *hardware* do computador. As tarefas são executadas pelo SO, tornando os programas menores e mais fáceis de serem programados (Machado e Maia, 2004. p.1-3).

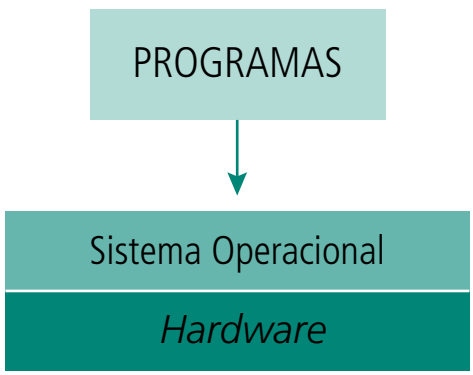

Figura 1.2: O Sistema Operacional funciona como uma interface entre o hardware e os programas de usuários
Fonte: Adaptado de Machado, 2004

Conforme mostra a Figura 1.2, o SO é o intermediário entre *hardware* e programas utilizados pelos usuários.

Em uma situação ideal, somente o sistema operacional deve ter acesso ao hardware do computador. Um programa que desejasse, por exemplo, fazer um desenho no monitor, obrigatoriamente teria de repassar esta tarefa ao sistema operacional. Este, por sua vez, iria analisar o pedido do programa e, considerando o pedido válido, o executaria. Caso um determinado programa resolvesse fazer um pedido estranho (por exemplo, apagar todos os dados do disco rígido), o SO simplesmente poderia ignorar tal pedido, terminar a execução do programa e informar a ocorrência ao usuário.

Um programa de usuário não deve acessar recursos do computador diretamente, deve antes passar pela intermediação e autorização do sistema operacional.

Essa é a condição ideal de um sistema operacional ESTÁVEL e SEGURO. Isto acontece, sobretudo, em sistemas operacionais para gerenciamento de rede local *(Windows Server, Unix e Linux)* e entre os sistemas operacionais para PCs que não foram desenvolvidos para serem servidores de rede como o *MacOS e Windows* nas suas versões XP, *Vista e Windows 7.*

O antigo DOS não trabalhava nessas condições. Na época em que foi criado, o PC tinha pouquíssima memória RAM (1 MB) e o sistema operacional, como ficava residente em memória, tinha de ser o menor possível. Uma solução para diminuir o tamanho do SO foi permitir aos programas que acessassem diretamente o *hardware* do micro para tarefas especiais, como desenhar gráficos ou enviar dados à impressora. A Figura 1.3 ilustra isto.

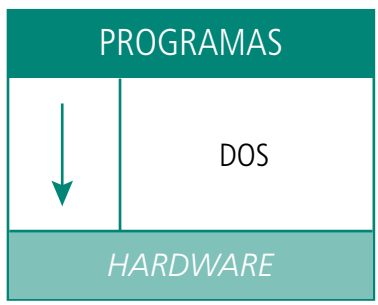

Figura 1.3: O Sistema DOS permitia acesso direto ao *hardware* pelos programas de usuário
Fonte: Adaptado de Machado, 2004

No entanto, isso acabava gerando um problema maior: se um programa fizesse um acesso indevido diretamente ao *hardware* do computador ou se o programa não estivesse bem escrito, isso inevitavelmente era refletido no hardware, fazendo com que o programa parasse por travamento. Esse problema continuou em versões do *Windows* 3.x, 95, 98 e ME, por utilizarem o mesmo núcleo do DOS, permitindo acessos direto ao *hardware.*

1.3 Máquina de níveis

A linguagem entendida pelo computador é uma linguagem binária de difícil entendimento pelos seres humanos, sendo chamada de linguagem de "baixo nível" ou "de máquina". As linguagens mais próximas aos seres humanos são classificadas como linguagens de "alto nível". Os computadores entendem apenas programas feitos em sua linguagem binária. Os seres humanos, no entanto, elaboram programas em linguagens de alto nível.

Existem muitas linguagens de alto nível utilizadas para os mais diversos fins, como C, C++ e *Java*. Os códigos-fontes escritos nessas linguagens precisam ser convertidos em linguagem binária. Por exemplo, a linguagem C utiliza uma forma de conversão diferente da utilizada pela linguagem Java.

Um computador, visto somente como um gabinete composto de circuitos eletrônicos, cabos e fontes de alimentação *(hardware)*, não tem nenhuma utilidade. É por meio de programas *(software)* que o computador consegue armazenar dados em discos, imprimir relatórios, gerar gráficos, realizar cálculos, entre outras funções. O *hardware* é o responsável pela execução das instruções de um programa, com a finalidade de se realizar alguma tarefa.

Nos primeiros computadores, a programação era realizada em painéis, através de fios, exigindo um grande conhecimento do *hardware* e de linguagem de máquina. Isso trazia uma grande dificuldade para os programadores da época, que normalmente eram os próprios engenheiros projetistas e construtores desses computadores.

A solução para esse problema foi o surgimento do Sistema Operacional, que tornou a interação entre usuário e computador mais simples, confiável e eficiente. A partir desse acontecimento, não existia mais a necessidade de o programador se envolver com a complexidade do *hardware* para poder trabalhar; ou seja, a parte física do computador tornou-se transparente para o usuário.

Podemos considerar o computador como uma máquina de níveis ou camadas, em que inicialmente existem dois níveis: o nível 0 *(hardware)* e o nível 1 (sistema operacional). Desta forma, o usuário pode enxergar a máquina como sendo apenas o sistema operacional, ou seja, como se o *hardware* não existisse. Esta visão modular e abstrata é chamada **máquina virtual.**

Para o sistema operacional, o programador e os programas também são usuários, pois usam recursos disponibilizados pelo SO. Em vários pontos deste texto, você poderá ver que a palavra usuário se aplica ao programador ou ao programa.

Entretanto, um computador não possui apenas dois níveis, e sim tantos níveis quantos forem necessários para adequar o usuário às suas diversas aplicações.

Quando o usuário está trabalhando em um desses níveis, não necessita saber da existência das outras camadas, acima ou abaixo de sua máquina virtual.

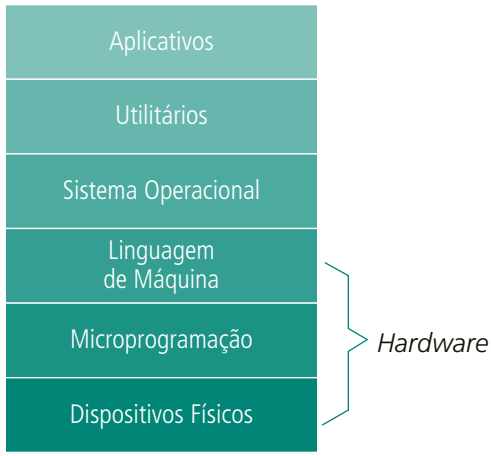

Figura 1.4: O computador como máquina de níveis
Fonte: Adaptado de Machado, 2004

Atualmente, a maioria dos computadores possui a estrutura mostrada na Figura 1.4, podendo conter mais ou menos camadas. A linguagem utilizada em cada um desses níveis é diferente, variando da mais elementar (baixo nível) à mais sofisticada (alto nível). Os aplicativos são programas executados pelo usuário. Os utilitários são programas de uso genérico e frequente, geralmente fornecidos junto com o SO. Cada um desses níveis será abordado com mais detalhes nas próximas aulas.

Um sistema operacional pode então ser definido sob dois aspectos: como uma máquina estendida ou máquina virtual e como um gerenciador de recursos.

O sistema operacional, como uma máquina virtual, esconde do programador detalhes do *hardware,* apresentando uma visão simples, mais conveniente e mais fácil de utilizar.

O sistema operacional, como um gerenciador de recursos, fornece uma alocação controlada e ordenada dos recursos do computador entre os vários programas que competem por esses recursos. Os recursos incluem processadores, memórias, dispositivos de E/S (unidades de disco, impressoras, *mouse*, etc.), interfaces de rede, dentre outros.

Quando um computador tem vários usuários, existe a necessidade de se proteger a memória , os dispositivos de E/S e os outros recursos. O sistema operacional então mantém informação sobre quem está usando qual recur-

so (para garantir os recursos a quem precisa deles), contabilizar o uso (para evitar que um usuário use por um período injustamente longo) e mediar quando há pedidos conflitantes sobre um mesmo recurso.

Resumo

Nesta aula você aprendeu alguns conceitos básicos sobre sistemas operacionais, algumas questões importantes sobre seu funcionamento e funções principais. Viu também como a estruturação de um sistema em camadas pode ser vantajosa em termos de eficiência de todo o ambiente computacional.

Atividades de aprendizagem

1. Quais seriam as principais dificuldades que um programador teria no desenvolvimento de uma aplicação em um ambiente sem um sistema operacional?

2. Explique o conceito de máquina virtual. Qual a grande vantagem em utilizar esta metodologia?

3. Defina o conceito de uma máquina de camadas.

4. Explique a seguinte frase: "O Sistema Operacional protege o usuário da máquina e a máquina do usuário".

Aula 2 – Histórico e classificação

Objetivos

Conhecer o histórico de Sistemas Operacionais.

Compreender como as inovações de *hardware* colaboraram com o desenvolvimento dos Sistemas Operacionais.

Classificar os sistemas conforme suas características principais.

2.1 Histórico

Vimos que o sistema operacional interage diretamente com o *hardware* e, com isso, é influenciado diretamente pela evolução do mesmo. Portanto, a evolução dos sistemas operacionais está, em grande parte, relacionada ao desenvolvimento de equipamentos cada vez mais velozes, compactos e de custos baixos e à necessidade de aproveitamento e controle destes recursos. Assim, ao falar sobre o histórico dos sistemas operacionais, estaremos recordando um pouco a evolução do *hardware*. Devemos lembrar que as datas das fases da evolução são aproximadas.

Desde os tempos do computador programado por chaves e cabos até o surgimento do teclado e impressora de caracteres, procurou-se ao longo do processo evolutivo do computador tornar a sua utilização mais amigável, precisa, rápida e eficaz.

O conjunto de equipamentos e recursos utilizados para que o homem possa controlar o computador é genericamente denominado interface. O aprimoramento da interface atingiu o ponto em que o usuário passou a interagir com pequenos desenhos ou símbolos de objetos comuns ao seu trabalho. Pensou-se em representar, por exemplo, a tarefa de impressão de documentos pelo desenho de uma pequena impressora e a eliminação de um documento por uma pequena lixeira. Surgiram os ícones. O histórico da evolução foi dividido em fases, cada uma marcada pela evolução significativa do *hardware*, do *software,* da interação com o sistema ou por aspectos de conectividade. Primeiramente, devemos ressaltar que o mapeamento das datas de evoluções

e gerações dos Sistemas Operacionais e das Arquiteturas de Computadores são, de certa forma, vagas e imprecisas, mas com certa estrutura.

a) Primeira fase (1945-1955) - Válvulas e Painéis de Programação

No início da Segunda Guerra Mundial, surgiram os primeiros computadores digitais, formados por milhares de válvulas, que ocupavam áreas enormes, sendo de funcionamento lento e duvidoso.

O ENIAC (*Electronic Numerical Integrator and Computer*) foi o primeiro computador digital de propósito geral. Criado para a realização de cálculos balísticos, sua estrutura possuía 17.468 válvulas, 10 mil capacitores, 70 mil resistores e pesava 32 toneladas. Quando em operação era capaz de realizar cinco mil adições por segundo.

A programação era feita em painéis, através de fios, utilizando linguagem de máquina. Não existia o conceito de sistema operacional. Outros computadores foram construídos nessa época, mas eram utilizados apenas em universidades e órgãos militares.

Muitas empresas foram fundadas ou investiram no setor, como, por exemplo, a IBM, o que levou à criação dos primeiros computadores para aplicações comerciais.

b) Segunda fase (1956-1965) - Transistores e Sistemas em Lote (*batch*)

A criação do transistor e das memórias magnéticas contribuiu para o enorme avanço dos computadores da época. O transistor permitiu o aumento da velocidade e da confiabilidade do processamento; as memórias magnéticas permitiram o acesso mais rápido aos dados, maior capacidade de armazenamento e computadores menores.

Surgiram os primeiros sistemas operacionais, para tentar automatizar as tarefas manuais até então realizadas e as primeiras linguagens de programação, como *Assembly* e *Fortran.* Os programas deixaram de ser feitos diretamente no *hardware*, o que facilitou enormemente o processo de desenvolvimento de programas. Surgiu o processamento em *batch*, em que um lote (*batch*) de programas e de dados era submetido ao computador por vez.

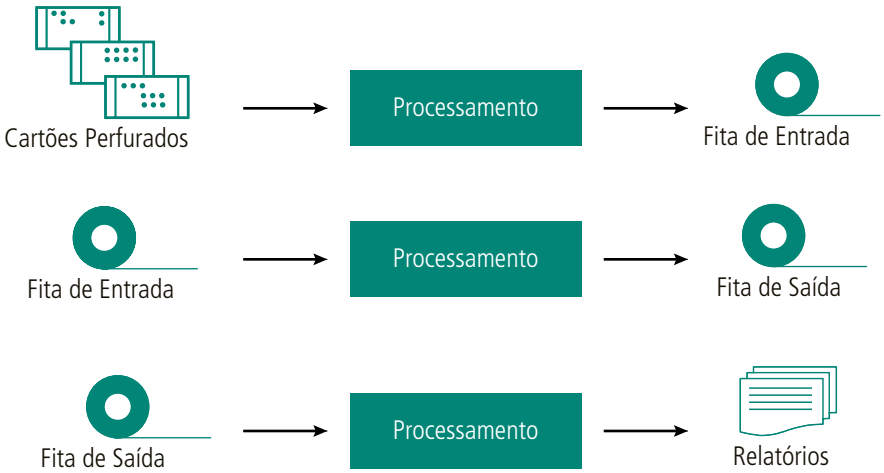

Figura 2.1: Ciclos de processamento na segunda fase
Fonte: Adaptado de Machado, 2004

Os programas passaram a ser perfurados em cartões que, submetidos a uma leitora, eram processados e gravados em uma fita de entrada, conforme Figura 2.1. A fita de entrada, então, era lida pelo computador, que executava um programa de cada vez, processando e gravando o resultado em uma fita de saída. Ao término de todos os programas, as fitas de saída eram lidas e processadas novamente para serem impressas, gerando assim os relatórios.

Com o processamento em *batch*, um grupo de programas era submetido de uma só vez, o que diminuía o tempo existente entre a execução dos programas, permitindo, assim, melhor uso do computador.

c) Terceira fase (1966-1980) - Circuitos Integrados e Multiprogramação

Por meio dos circuitos integrados e, posteriormente, dos microprocessadores, foi possível viabilizar e difundir o uso de sistemas computacionais por empresas, devido à diminuição de seus custos de aquisição. Houve um aumento no poder de processamento e diminuição no tamanho dos equipamentos.

A evolução dos processadores de entrada/saída permitiu que, enquanto um programa esperasse por uma operação de leitura/gravação, o processador executasse um outro programa. Para tal, a memória foi dividida em partições, em que um programa esperava sua vez para ser processado. A essa técnica de compartilhamento da memória principal e processador deu-se o nome de **multiprogramação**.

A-Z

Multiprogramação
Multiprogramação é a execução simultânea de dois ou mais programas.

Com a substituição das fitas por discos no processo de submissão dos programas, o processamento *batch* tornou-se mais eficiente, pois permitia a alteração na ordem de execução das tarefas, até então somente sequencial. A essa técnica de submissão de programas chamou-se *spooling*, que, mais tarde, também viria a ser utilizada no processo de impressão.

Duas inovações de *hardware* foram fundamentais para o surgimento da multiprogramação: os discos magnéticos e as interrupções de *hardware*. Os discos magnéticos compõem praticamente todas as máquinas atuais, com vários *Gigas* e até *Terabytes* de capacidade de armazenamento. As interrupções de hardware são sinais que as controladoras de dispositivos enviam à CPU para avisar que as operações de entrada ou saída foram finalizadas ou tiveram algum problema.

Os sistemas operacionais, mesmo com a evolução do processamento *batch* e a multiprogramação, ainda estavam limitados a processamentos que não exigiam comunicação com o usuário. Para permitir a interação rápida entre o usuário e o computador, foram adicionados terminais de vídeo e teclado (interação *on-line*).

A multiprogramação evoluiu, preocupada em oferecer aos usuários tempos de resposta razoáveis e uma interface cada vez mais amigável. Para tal, cada programa na memória utilizaria o processador em pequenos intervalos de tempo. A esse sistema de divisão de tempo do processador chamou-se *time-sharing* (tempo compartilhado).

Nos sistemas *time-sharing*, os usuários possuíam um terminal que podia interagir com o programa em execução. Esses usuários tinham a ilusão de possuir a máquina dedicada à execução de seu programa. O que não era verdade! Essa ilusão vinha da divisão de tempo de processamento de CPU entre os usuários

Outro fato importante nessa fase foi o surgimento do sistema operacional UNIX.

Ao final dessa fase, com a evolução dos microprocessadores, surgiram os primeiros microcomputadores, muito mais baratos que qualquer um dos computadores até então comercializados.

d) Quarta fase (1981-1990) - Computadores Pessoais

Existem na internet muitas informações sobre o sistema UNIX, sistema pai de muitos sistemas operacionais atuais, como o *Linux*, Solaris e outros. Procure o vídeo na internet chamado de "Revolução dos Sistemas Operacionais", do Inglês *"Revolution OS"*, que conta um pouco dessa história.

Os mini e superminicomputadores se firmaram no mercado e os microcomputadores ganharam um grande impulso. Surgem as estações de trabalho (*workstations*) que, apesar de monousuárias, permitem que se executem diversas tarefas concorrentemente, criando o conceito de multitarefa.

No final dos anos 80 os computadores tiveram um grande avanço, decorrente de aplicações que exigiam um enorme volume de cálculos. Para acelerar o processamento, foram adicionados outros processadores, exigindo dos sistemas operacionais novos mecanismos de controle e sincronismo. Com o multiprocessamento, foi possível a execução de mais de um programa simultaneamente, ou até de um mesmo programa por mais de um processador. Foram introduzidos processadores vetoriais e técnicas de paralelismo de processamento, fazendo com que os computadores se tornassem ainda mais poderosos.

O uso das redes distribuídas se difundiu por todo o mundo, permitindo o acesso a outros sistemas de computação, independentemente de cidade, país e, até mesmo, fabricante. Os *softwares* de rede passaram a estar intimamente relacionados com o sistema operacional de cada máquina e surgem os sistemas operacionais de rede.

e) Quinta fase (1991-2000)

Houve grandes avanços em termos de *hardware*, *software* e telecomunicações como consequência da evolução das aplicações, que necessitavam cada vez mais de capacidade de processamento e armazenamento de dados. Sistemas especialistas, sistemas multimídia, bancos de dados distribuídos, inteligência artificial e redes neurais são apenas alguns exemplos da necessidade cada vez maior de informação e de capacidade de processamento.

O conceito de processamento distribuído é explorado nos sistemas operacionais, de forma que suas funções estejam espalhadas por vários processadores através de redes de computadores.

> A década de 90, foi definitiva para a consolidação dos sistemas operacionais baseados em interfaces gráficas (TANENBAUM, 2000, p. 4 a 12).

Atualmente temos as plataformas *multicore* (vários núcleos) disponíveis em computadores pessoais e até *notebooks,* possibilitando cada vez mais a paralelização de aplicações e elevando, em muito, o tempo de execução dos programas. Os fanáticos pelos jogos de computador que o digam!

Vamos relembrar as principais características e evoluções de cada fase? Faça um quadro, sumarizando cada uma das cinco fases, para ajudá-lo no estudo e entendimento da evolução de *hardware* necessária para o desenvolvimento dos sistemas operacionais. Para cada fase, anote os acontecimentos mais importantes.

2.2 Tipos de Sistemas Operacionais

Os tipos de sistemas operacionais e sua evolução estão relacionados diretamente com a evolução do *hardware* e das aplicações por ele suportadas e podem ser classificados conforme Figura 2.2.

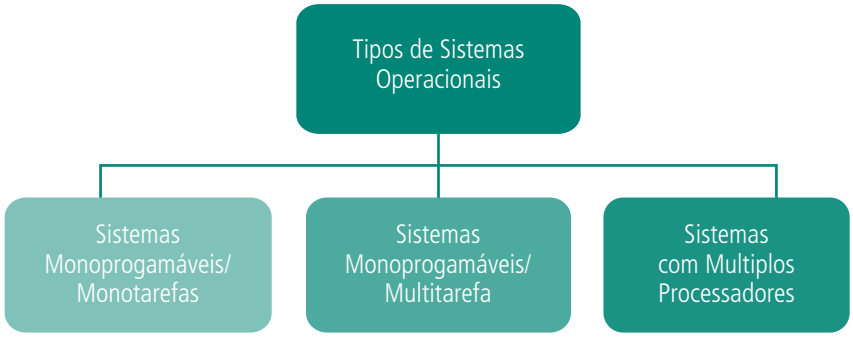

Figura 2.2: Tipos de sistemas operacionais
Fonte: Adaptado de Macha, 2004

Considerando o processamento, podemos classificar os sistemas operacionais de acordo com a quantidade de tarefas que podem ser executadas simultaneamente.

A-Z

Monoprogramáveis ou Monotarefa
Sistemas monoprogramáveis ou monotarefa são aqueles em que é executado, por vez, um único programa ou uma única tarefa.

Monoprogramáveis ou **Monotarefa** podem executar apenas um programa por vez. Para que um usuário possa executar outro programa, deverá aguardar a finalização do programa corrente. Esta era uma característica dos primeiros sistemas operacionais que estavam relacionados ao surgimento dos primeiros computadores na década de 60.

Caracterizavam-se por permitir que todos os recursos de *hardware* ficassem exclusivamente dedicados a um único programa. Em consequência, sua principal desvantagem residia no fato de que enquanto um programa aguardava por um evento externo, como a digitação de um caractere do teclado, o processador permanecia ocioso. Além disso, tanto a memória principal quanto os recursos de E/S (Entrada e Saída) como impressoras e discos eram subutilizados, uma vez que todos estariam dedicados a um único programa como mostra a Figura 2.3.

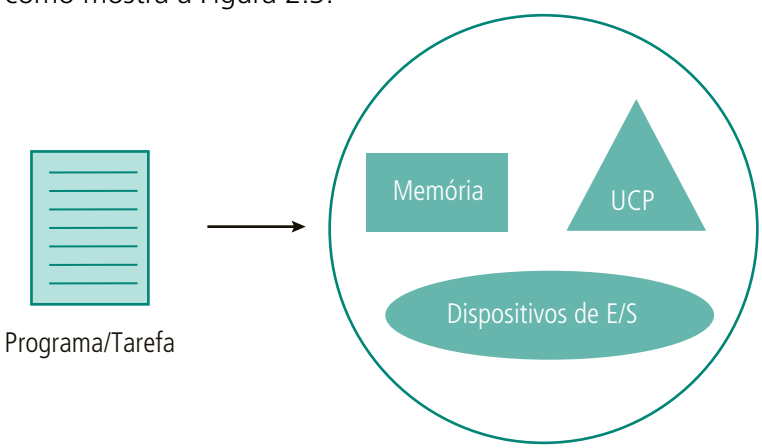

Figura 2.3: Sistema monoprogramável ou monotarefa
Fonte: Adaptado de Machado, 2004

Sistemas Multiprogramáveis ou Multitarefa: os recursos computacionais são compartilhados entre os diversos usuários e suas aplicações. Podemos observar o compartilhamento de memória e do processador. Nesse caso, o compartilhamento de tempo no processador é distribuído. Assim, o usuário tem a impressão que vários processos estão sendo executados simultaneamente. Um dos processos ocupa o processador enquanto os outros ficam enfileirados, aguardando a sua vez de entrar em execução. Cabe ao sistema operacional o papel de gerenciar de forma ordenada e protegida o acesso concorrente aos recursos disponíveis.

Sistemas multiprogramáveis ou multitarefa permitem o compartilhamento dos recursos computacionais entre diversos usuários e aplicações, permitindo sua execução concorrente.

A vantagem desse tipo de sistema é uma melhor utilização dos recursos disponíveis, o que resulta em menor tempo de resposta das aplicações. Além de um custo reduzido, uma vez que haverá o compartilhamento dos recursos entre as diferentes aplicações e aumento da produção do usuário.

Graças aos sistemas multiprogramáveis é possível editar um documento no *MS Word,* navegar na internet, ouvir música, utilizar os mensageiros instantâneos (MSN), tudo ao mesmo tempo!

Sistemas com múltiplos processadores: o sistema operacional distribui as tarefas entre dois ou mais processadores. A vantagem desse tipo de sistema é permitir que mais de um programa possa ser executado simultaneamente ou que um mesmo programa seja dividido em várias partes e executado simultaneamente nos vários processadores, aumentando o desempenho.

Esse tipo de sistema surgiu da necessidade de aplicações que requeriam um grande poder computacional, como sistemas de previsão do tempo, modelagens, simulações, desenvolvimento aeroespacial, entre outros. Com múltiplos processadores, é possível reduzir drasticamente o tempo de processamento destas aplicações. Inicialmente, as configurações limitavam-se a poucos processadores, mas, atualmente existem sistemas com milhares de processadores.

Sistemas com múltiplos processadores
Sistemas com múltiplos processadores caracterizam-se por possuir duas ou mais CPU's interligadas e trabalhando de forma conjunta na solução de um problema.

Os sistemas com múltiplos processadores podem ser classificados em fortemente acoplados e fracamente acoplados, em função da comunicação entre CPU's e o grau de compartilhamento da memória. Em sistemas fortemente acoplados, há uma única memória principal compartilhada por todos os processadores, enquanto em sistemas fracamente acoplados cada sistema tem sua própria memória. Com isso, a taxa de transferência entre processadores em sistemas fortemente acoplados é bem maior que em sistemas fracamente acoplados.

Resumo

Nesta aula você pôde perceber como o desenvolvimento de inovações de *hardware* foi importante para a evolução dos sistemas operacionais ao longo dos anos. Viu também como podemos classificar os sistemas operacionais por suas características principais. Alguns conceitos básicos importantes foram vistos e que serão de fundamental relevância para as aulas seguintes. Então se ficou alguma dúvida, revise esta aula!

Atividades de aprendizagem

1. Por que dizemos que existe uma subutilização de recursos em sistemas monoprogramáveis?

2. Quais as vantagens dos sistemas multiprogramáveis?

3. O que caracteriza o processamento *batch*? Que aplicações podem ser processadas neste tipo de ambiente?

4. Qual a grande diferença entre sistemas fortemente acoplados e fracamente acoplados?

Aula 3 – Elementos de hardware e software – Parte I

Objetivos

Conhecer a arquitetura básica de computadores.

Descrever os principais dispositivos de entrada e saída.

Compreender conceitos de *software* utilitário.

3.1 *Hardware*

O *hardware* do computador é constituído por um conjunto de componentes interligados: processadores, memória principal, registradores, terminais, impressoras e discos magnéticos, além de outros dispositivos físicos.

Os componentes físicos do computador são agrupados em três subsistemas básicos:

- Unidade Central de Processamento (CPU);
- Memória;
- Dispositivos de Entrada e Saída.

3.1.1 Unidade Central de Processamento (CPU)

A CPU tem como função principal unificar todo o sistema, controlando as funções realizadas em cada unidade funcional. É responsável pela execução de todos os programas, que obrigatoriamente deverão estar armazenados na memória principal.

A unidade central de processamento é dividida em **dois** componentes básicos:

- Unidade de controle (UC);
- Unidade lógica e aritmética (ULA);

A UC é responsável por controlar as atividades de todos os componentes do computador, mediante a emissão de pulsos elétricos (sinais de controle) gerados por um dispositivo chamado *clock*. Esse controle pode ser exercido, por exemplo, sobre a gravação de um dado no disco ou a busca

de uma instrução na memória.

A ULA é responsável pela realização de operações lógicas (testes e comparações) e aritméticas (somas e subtrações).

Atualmente os nossos processadores conseguem executar bilhões de instruções por segundo!

A velocidade de processamento de uma unidade central de processamento (CPU) é determinada pelo número de instruções que o processador executa por unidade de tempo, normalmente em segundos. A unidade de medida é MIPS (milhões de instruções por segundo).

A transmissão de dados entre o receptor e o transmissor é controlada por um sinal de controle chamado *clock*. Este sinal é usado para sincronizar o transmissor com o receptor, isto é, para informar ao receptor que um dado está sendo transmitido. É utilizado pela unidade de controle para a execução das instruções.

O *clock* é um dispositivo localizado na unidade central de processamento, que gera pulsos elétricos síncronos em um determinado intervalo de tempo (sinal de *clock*). A quantidade de vezes que este pulso se repete em um segundo define a frequência do *clock*. Toda transmissão paralela utiliza um sistema de *clock*. Esses sistemas de *clock*, entretanto, são independentes, isto é, o sistema de *clock* usado na transmissão de dados entre o processador e a memória RAM não é o mesmo usado na transmissão de dados entre o disco rígido e a placa-mãe, por exemplo.

3.1.2 Memórias

A memória tem por função armazenar internamente toda informação que é manipulada pelo computador: os programas e os dados. A memória pode ser classificada quanto à sua velocidade (ou tempo) de acesso, capacidade de armazenamento, custo e volatilidade. Em função dessas características, pode-se estabelecer uma hierarquia de tipos de memórias, conforme Figura 3.1.

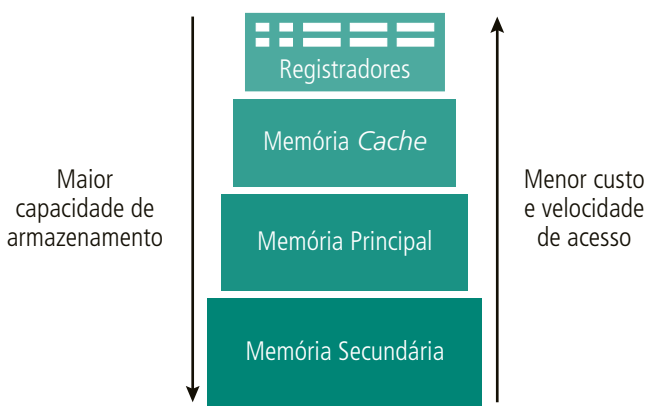

Figura 3.1: Comparativo dos diversos tipos de memória
Fonte: Adaptado de Machado, 2004

a) Registradores

São dispositivos de alta velocidade, localizados fisicamente na unidade central de processamento, para armazenamento temporário de dados. O número de registradores varia em função da arquitetura de cada processador. Existem registradores de uso específico (com propósitos especiais) e de uso geral.

Os registradores de uso específico são:

- Contador de instruções - responsável por armazenar o endereço da próxima instrução que a unidade central de processamento deverá executar. Toda vez que uma instrução já está sendo processada o endereço da próxima instrução a ser processada é armazenado no contador de instruções;

- Apontador de pilha: responsável por armazenar o endereço de memória do topo da pilha. Pilha é uma estrutura de dados onde o sistema mantém informações sobre tarefas que estavam sendo processadas, mas que por algum motivo tiveram que ser interrompidas;

- Registrador de estado: responsável por armazenar informações sobre a execução do programa (status do programa). A cada instrução executada, o registrador de estado é alterado conforme o resultado gerado pela instrução.

b) Memória *cache*

É uma memória volátil de alta velocidade. Quando o processador faz referência a um dado armazenado na memória principal, verifica antes se este dado não está armazenado na memória *cache*. Ao encontrar o dado armazenado na memória *cache*, o processador não acessa a memória principal, diminuindo o tempo de processamento.

Para aumentar o desempenho no funcionamento das memórias *caches* é feita a hierarquização da *cache* em múltiplos níveis. O nível da cache mais alto é chamado de L1 (Level 1), com baixa capacidade de armazenamento e com altíssima velocidade de acesso. O segundo nível, L2 (Level 2), possui maior capacidade de armazenamento, porém com velocidade de acesso inferior a L1, e assim sucessivamente. Quando a CPU necessita de uma informação da memória principal, primeiramente verifica a cache L1, caso não ache, segue para a cache L2, assim por diante. Se não encontrar em nenhum dos níveis, busca o dado na memória principal.

c) Memória principal

É a memória responsável pelo armazenamento dos programas que estão sendo executados pela CPU em um certo instante, bem como dos dados utilizados pelos programas em execução. Para que um programa possa ser executado pela CPU é necessário que ele seja previamente armazenado na memória principal. Existem ainda dois tipos de memória: ROM (*read only memory* – memória somente leitura) e RAM (*random access memory* – memória de acesso randômico).

Quando usamos o termo "memória" para um computador, normalmente estamos nos referindo à sua memória RAM. Se um programa que o usuário pretenda executar não estiver na memória RAM, então ele deve ser transferido de um sistema de memória secundário (como discos rígidos, unidades de CD-ROM e etc.) para a memória RAM.

A RAM é um tipo de circuito eletrônico de memória que permite a leitura e a escrita de dados em seu interior. Só que ela é uma memória volátil (como os registradores e a *cache*), isto é, cortando-se sua alimentação elétrica, apagamos os dados que estavam nela armazenados.

Já a característica principal da ROM é que o seu conteúdo não é perdido quando cortamos a sua fonte de alimentação e por isso é utilizada para manter os programas iniciais do computador. Quando ligamos o micro, o processador não sabe o que fazer; ele precisa executar um programa; este programa necessário para dar o *boot* é gravado em uma memória ROM, localizada na placa-mãe do computador.

Um programa armazenado em ROM recebe o nome de *firmware*. Na memória ROM do micro há basicamente três programas (*firmware*) principais:

- BIOS (*Basic Input/Output System*, Sistema básico de Entrada/Saída);
- POST (*Power On Self Test*, Autoteste ao Ligar);
- *Setup* (programa que permite alterar vários itens da configuração do computador).

A memória principal é composta por unidades de acesso chamadas células, cada uma capaz de armazenar um determinado número de *bits.* Cada célula tem um endereço, conforme Figura 3.2, que é uma referência à posição da célula dentro da memória, como o endereço de uma casa. Quando um programa deseja ler ou escrever um dado em uma célula, deve primeiro especificar qual o endereço de memória desejado. O endereço da célula a ser acessada fica armazenado em um registrador denominado registrador de endereço de memória.

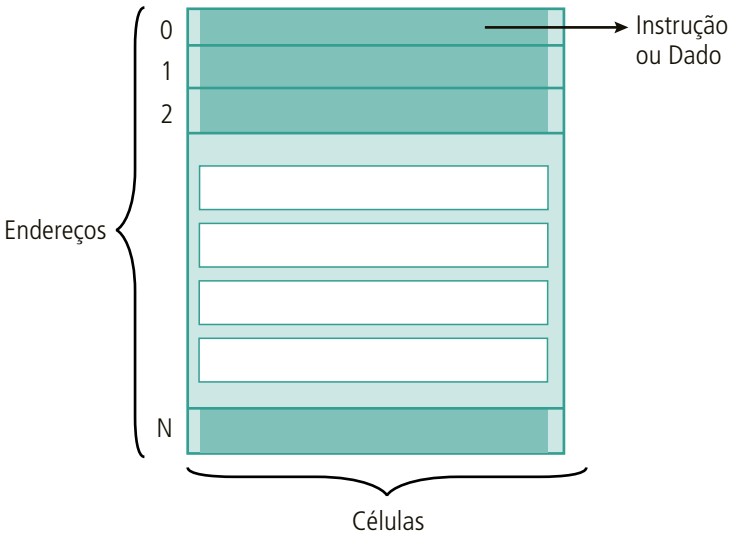

Figura 3.2: Células de memória e seus endereços
Fonte: Adaptado de Machado, 2004

A-Z

Células de memória
Células de memória são unidades de acesso da memória principal.

O número de células endereçadas na memória principal é limitado pelo tamanho do registrador de endereço. No caso de o registrador possuir *n bits*, a memória poderá endereçar 2^n células.

d) Memória secundária

Memória secundária é um termo genérico para designar diversos componentes que permitem gravar e ler dados permanentes. O seu acesso é lento, se comparada com as memórias *cache* ou principal, porém relativamente apresentam custo mais baixo e capacidade de armazenamento superior. Exemplos de memória secundária são as fitas magnéticas, discos rígidos (HDs), CDs, DVDs, etc.

Além das memórias RAM e ROM descritas no texto, temos ainda a PROM (memória programável somente de leitura) que pode ser gravada pelo usuário uma única vez, a EPROM (memória programável e apagável somente de leitura) que pode ser gravada ou regravada por meio de um equipamento que fornece as voltagens adequadas em cada pino (para apagar os dados deve-se utilizar raios ultravioleta no *chip*), e por último temos a EEPROM (memória programável e apagável eletronicamente somente de leitura), que pode ser gravada, apagada ou regravada utilizando um equipamento que fornece as voltagens adequadas em cada pino.

3.1.3 Dispositivos de entrada e saída

Permitem a comunicação entre o computador e o mundo externo. Alguns dispositivos servem para a comunicação homem-máquina, como teclados, monitores de vídeo, impressoras, plotters, entre outros. A implementação de interfaces mais amigáveis permite cada vez mais que pessoas sem conhecimento específico sobre informática possam utilizar o computador. São alguns exemplos desses tipos de dispositivos: *Scanner*, caneta ótica, *mouse*, dispositivos sensíveis à voz humana, e etc.

3.1.4 Barramento

A CPU, a memória principal e os dispositivos de E/S são interligados através de linhas de comunicação denominadas barramentos, barras ou vias. Um

barramento é um conjunto de fios paralelos (linhas de comunicação), por onde trafegam informações, como dados, endereços ou sinais de controle. Um esquema gráfico de um barramento é mostrado na Figura 3.3.

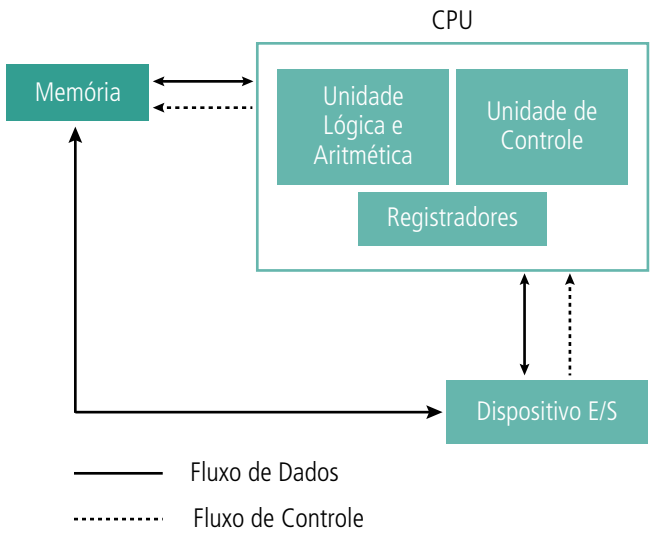

Figura 3.3: Esquema gráfico dos barramentos
Fonte: Adaptado de Machado, 2004

O barramento pode ser classificado como unidirecional (transmissão em um só sentido) ou bidirecional (transmissão em ambos os sentidos). Existem três tipos de barramentos, que se diferenciam uns dos outros de acordo com o que transportam:

- **Barramento de dados:** transmite informações entre a memória principal e a unidade central de processamento;

- **Barramento de endereços:** utilizado pela unidade central de processamento para especificar o endereço da célula de memória que será acessada;

- **Barramento de controle:** é por onde a unidade central de processamento envia os pulsos de controle relativos às operações de leitura e gravação.

3.1.5 *Pipelining*

O conceito de processamento *pipeline* é a divisão de uma tarefa em uma sequência de subtarefas. O processador, por meio de suas várias unidades funcionais pipeline, funciona de forma a permitir que, enquanto uma instrução se encontra na fase de execução, uma outra instrução possa estar na fase de busca.

A técnica de *pipelining* pode ser empregada em sistemas com um ou mais

processadores, em diversos níveis, e tem sido a técnica de paralelismo mais utilizada para maior desempenho dos sistemas de computadores.

Podemos fazer a comparação de um *pipelining* com um motor de carro de quatro tempos e vários cilindros. Enquanto um cilindro está na fase de alimentação, outro está na fase de compressão, outro na de explosão e assim por diante. Isto aumenta bastante a eficiência do motor.

3.2 *Software*

Na Figura 1.4 vimos três camadas/níveis de *software*: aplicativos, utilitários e sistema operacional. Os *software* ou programas executados pelos usuários são atualmente chamados, atualmente, de aplicativos. Antigamente se chamavam simplesmente programas. Quase tudo que o usuário consegue fazer utilizando o computador necessita de um aplicativo. Esses programas podem ser, por exemplo, um editor de textos, uma planilha ou um programa de imposto de renda, de controle de estoques ou de contas a receber.

Antigamente havia uma distinção mais nítida entre os aplicativos e os utilitários, pois havia um personagem a mais no cenário: o operador do computador. Este executava algumas atividades específicas, como copiar de fita para disco e vice-versa ou operar as impressoras. Para executar essas atividades o operador precisava de uma série de programas, como, por exemplo, um programa para localizar um arquivo no computador. A maioria dos programas requeridos pelo operador era chamada de utilitários.

Hoje em dia continua a existir um conjunto de programas, utilizados como interface entre o usuário e o *hardware.* O termo utilitário é, assim, uma referência a *softwares* relacionados com serviços do sistema operacional, como os compiladores, *linkers*, depuradores e outros.

Os *software* aplicativos podem ser identificados como aqueles que estão mais próximos do usuário comum, como os navegadores, editores de texto, jogos, etc. Já os *software* utilitários são aqueles que fazem a intermediação entre os aplicativos e o núcleo do sistema operacional, possuindo funções mais específicas e geralmente mais restritas, como ligadores, depuradores, compiladores, etc.

Dentre os *software* utilitários podemos destacar alguns que dão apoio à programação de computadores: tradutores, compiladores, montadores, in-

terpretadores, ligadores, carregadores, depuradores. Nas próximas seções estaremos abordando esses utilitários com mais detalhes.

3.2.1 Tradutores, compiladores e montadores

Com o surgimento das primeiras linguagens de montagem (*Assembly*) e as linguagens de alto nível, o programador passou a se preocupar menos com aspectos de *hardware* e a escrever em uma linguagem mais próxima da linguagem humana.

Apesar das vantagens proporcionadas pelas linguagens de montagem e de alto nível, que propiciaram um aumento enorme na produtividade dos programadores, os programas não estão prontos para serem executados diretamente pela CPU. Eles deverão passar por uma etapa de conversão, quando a codificação do programa é traduzida para código de máquina. É essa a função do tradutor.

O utilitário "tradutor" tem a função de converter todo o código fonte escrito em linguagem de alto nível para código de máquina.

O tradutor, pelo tipo de linguagem de programação utilizada, pode ser chamado de montador ou compilador:

A denominação montador ocorre quando é gerado o módulo objeto (a linguagem de máquina) a partir de uma linguagem de montagem (*assembler*). A denominação compilador é dada ao utilitário responsável por gerar, a partir de um programa escrito em linguagem de alto nível (*Cobol, C, Delphi, etc.*), um programa em linguagem de máquina (módulo objeto)

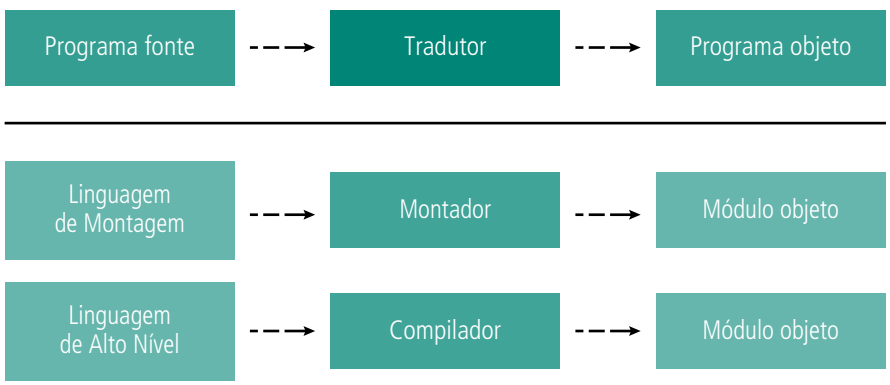

Figura 3.4: Representação das diferenças entre o tradutor, montador e compilador
Fonte: Adaptado de Machado, 2004

3.2.2. Interpretador

Assim é chamado um tradutor que não gera o módulo 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.

Sua desvantagem é o tempo gasto na tradução das instruções de um programa toda vez que este for executado, já que não existe a geração de um código executável.

Alguns exemplos de linguagens interpretadas são o *Basic* e o *Perl.*

3.2.3. *Linker*

O *linker* (ligador) é o utilitário responsável por gerar, a partir de um ou mais módulos objeto, um único programa executável, conforme Figura 3.5. Sua função é resolver todas as referências simbólicas existentes entre os módulos - objeto, reservar memória para a execução do programa e determinar uma região da memória onde o programa será carregado para sua execução.

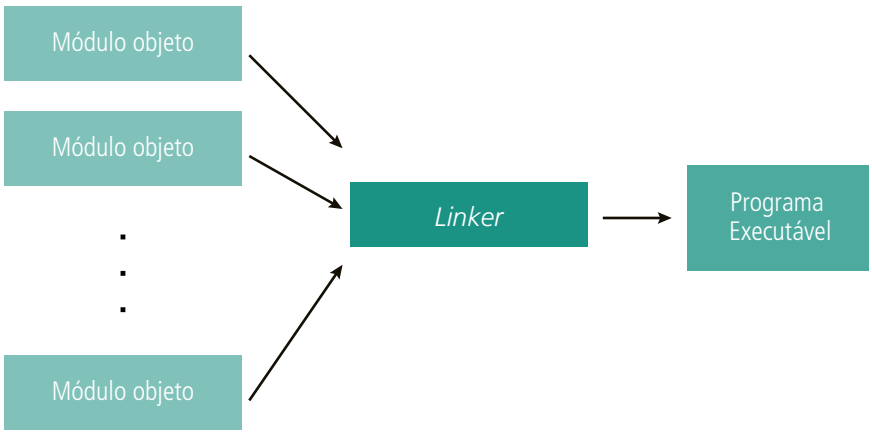

Figura 3.5: Representação do papel do *linker*
Fonte: Adaptado de Machado, 2004

Em ambientes multiprogramáveis esse tipo de alocação fixa feita pelo *linker* é inviável porque neste caso a memória é compartilhada entre diversos programas: é pouco provável que no momento em que o sistema carrega um programa, sua área de memória que foi predeterminada esteja disponível. A solução para isso é permitir que um programa possa ser executado em qualquer região disponível da memória, durante a sua carga (código relocável). Esse tipo de relocação não é realizado pelo *linker,* e sim por outro utilitário, chamado loader, responsável por carregar os programas na memória.

Os IDEs (Ambientes Integrados de Desenvolvimento), como o *Dev C++*, executam essas funções (de compilador e *link*er e até a carga do executável) de uma vez, quando você aperta o F9. Você não percebe quando é executada cada uma delas, embora sejam distintas. Você só consegue ver, na pasta onde você está trabalhando, os programas fonte (com terminação " .c" ou " .cpp") e o executável (terminação " .exe"); o módulo objeto intermediário (criado durante a compilação) você não vê mais.

3.2.4. *Loader*

O *loader* (carregador) é o utilitário responsável por colocar fisicamente na memória principal um programa para sua execução. Pode permitir que um programa seja carregado em regiões diferentes toda vez que for trazido para a memória.

Quando o *loader* carrega um programa para memória principal, ele aloca uma área de código, uma área de dados e uma área de pilha. A área de código armazena o programa executável, a área de dados armazena as variáveis e constantes utilizadas no programa e a área de pilha armazena os endereços de retorno das funções ou procedimentos chamados du-

rante a execução do programa. O funcionamento do carregador depende do código gerado pelo *linker* e, de acordo com este, pode ser classificado como absoluto ou relocável:

- **Loader absoluto** - o *loader* só necessita conhecer o endereço de memória inicial e o tamanho do módulo para realizar o carregamento. Então, o *loader* transfere o programa da memória secundária para a memória principal e inicia sua execução;

- **Loader relocável** - o programa pode ser carregado em qualquer posição de memória e o *loader* é responsável pela relocação no momento do carregamento.

O processo completo de compilação, *link*-edição e carga de um programa é mostrado na Figura 3.6.

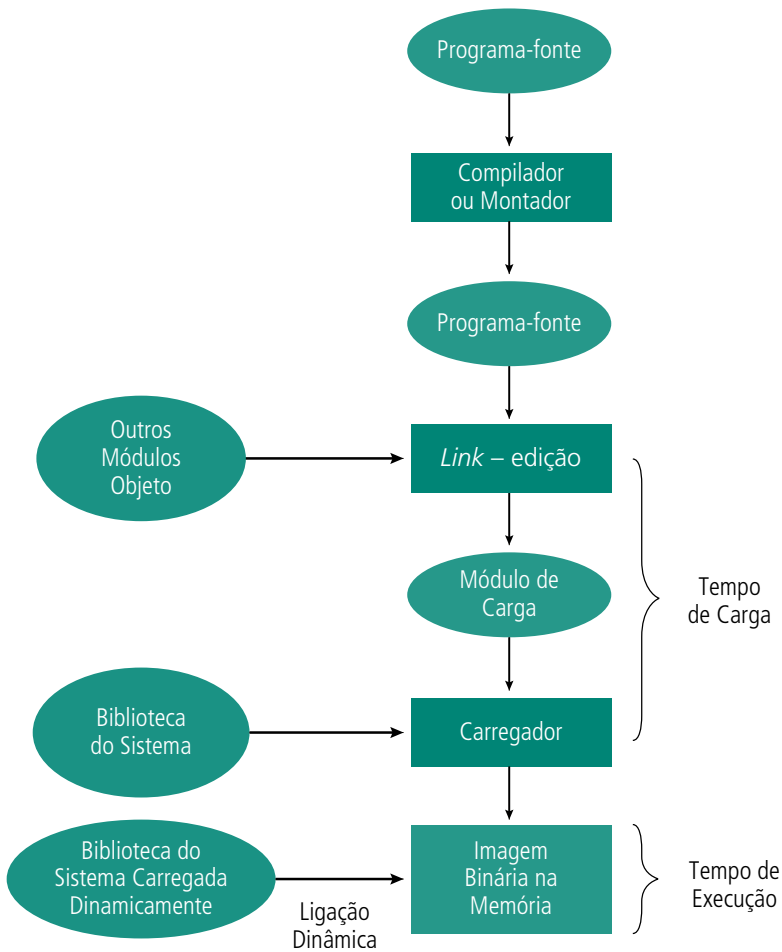

Figura 3.6: As várias etapas do desenvolvimento e processamento de um programa
Fonte: Adaptado de Silberschatz, Galvin e Gagne, 2000

3.2.5. Depurador

O desenvolvimento de programas está sujeito a erros de lógica, independentemente da metodologia ou linguagem utilizadas pelo programador. O depurador é o utilitário que permite ao usuário acompanhar e controlar a execução de um programa a fim de detectar erros na sua estrutura. O depurador ajuda a detectar os erros, mas não os corrige. O depurador geralmente oferece ao usuário os seguintes recursos:

- acompanhar a execução de um programa instrução por instrução;
- possibilitar a alteração e a visualização do conteúdo de variáveis;
- colocar pontos de parada dentro do programa, de forma que, durante a execução, o programa pare nos pontos determinados;
- especificar em forma de envio de mensagem, toda vez que o conteúdo de uma variável for modificado.

Esses recursos de depuração de programas também estão presentes em IDEs, geralmente na opção *Debug*.

Resumo

Nesta aula você pôde estudar e identificar os elementos básicos de uma arquitetura clássica de computadores. Pode compreender as funções e características de cada elemento de *hardware*. Nesta aula, fizemos também um apanhado inicial sobre *softwares* utilitários que, mesmo sem percebermos muitas vezes, nos ajudam e muito em tarefas do dia a dia.

Atividades de aprendizagem

1. O que são memórias voláteis e não voláteis?
2. Quais os benefícios de uma arquitetura de memória *cache* com múltiplos níveis?
3. Diferencie as funções básicas dos dispositivos de E/S.
4. Como a técnica de *pipelining* melhora o desempenho dos sistemas computacionais?
5. Por que o código-objeto gerado pelo tradutor ainda não pode ser executado?
6. Por que a execução de programas interpretados ainda é mais lenta que a de programas compilados?

Aula 4 – Elementos de hardware e software – Parte II

Objetivos

Conhecer conceitos mais específicos sobre programas de sistema.

Analisar o mecanismo de interrupção na concorrência entre processos.

Descrever as operações de entrada e saída.

Conhecer as características dos sistemas em lote e sistemas de tempo compartilhado.

Compreender a funcionalidade de alguns serviços do sistema operacional.

Analisar as características básicas de uma arquitetura de sistema operacional.

4.1 Linguagem de controle

Denominada, também de linguagem de comando, é a forma mais direta de um usuário se comunicar com o sistema operacional (SO). É oferecida por todos os SO para que, através de comandos simples, o usuário possa ter acesso a rotinas específicas do sistema.

Os comandos, quando digitados (ou executados) pelo usuário, são interpretados por um programa denominado interpretador de comandos ou *shell*. A linha de comando é reconhecida, o *shell* verifica sua sintaxe, envia mensagens de erro e faz chamadas de rotinas dos sistemas. O usuário dispõe assim, de uma interface interativa direta com o sistema operacional, para realizar tarefas básicas como acessar um arquivo em disco ou consultar um diretório.

As linguagens de controle evoluíram no sentido de permitir uma interação mais amigável, utilizando interfaces gráficas, colocando os programas em uso em janelas e utilizando ícones para comunicação com o usuário. Quando você cria uma pasta, renomeia ou apaga um arquivo, clica em cima de

uma planilha ou texto para editá-los, você está interagindo com o *shell* e utilizando recursos dessa linguagem de comando.

Embora todos ou quase todos os SO modernos usem uma interface gráfica para interagir com o usuário, o prompt da linha de comandos também continua a existir.

4.2 Programas de sistema ou utilitários

Os utilitários também são chamados comumente programas de sistema. Os SO mais modernos vêm aumentando a coleção desses tipos de programas. Você deve lembrar que há pouco tempo atrás você precisava de um programa específico para gravar arquivos em CDs e DVDs e para assistir um filme. Hoje estas funções estão embutidas no SO.

Os programas de sistema fornecem um ambiente conveniente para a execução de uma série de tarefas de uso do computador ou de periféricos específicos. Alguns deles são simplesmente interfaces de usuário às chamadas ao sistema; outros são consideravelmente mais complexos.

Podem ser divididos em categorias tais como:

a) **Gerência de arquivos** - criam, excluem, copiam, renomeiam, imprimem, listam e geralmente manipulam arquivos e diretórios. Precisam trabalhar com diversos tipos de mídia: CD, DVD, HD, disquete, *pendrive*, etc;

b) **Informações de *status*** - simplesmente pedem ao sistema informações relativas ao status da máquina ou de periféricos: data, hora, quantidade de memória ou espaço em disco disponível, número de usuários conectados, número de documentos aguardando impressão e outras informações semelhantes;

c) **Modificação de arquivo** - vários editores de texto podem estar disponíveis para criar e modificar o conteúdo dos arquivos armazenados em disco, fita ou outro tipo de mídia;

d) **Comunicações** - esses programas oferecem o mecanismo para criar conexões virtuais entre processos, usuários e diferentes sistemas de computação. Permitem aos usuários enviar mensagens às telas uns dos outros, navegar pelas páginas da *web*, efetuar *logon* remotamente ou transferir arquivos de uma máquina para outra;

e) ***Spooling*** - permitem manipular a fila de impressão de documentos em uma ou mais impressoras ligadas ao sistema.

À medida que novos periféricos se popularizam e novas funções são atribuídas ao computador, essa lista de utilitários tende a crescer. Além disto, uma função antes exercida por um utilitário pode ser incorporada definitivamente ao SO, como citamos o exemplo da leitura e gravação de CD e DVD.

4.3. Linguagem de máquina

Todos os programas, para serem processados, precisam estar em linguagem de máquina. Essa é a linguagem que o processador realmente consegue entender. Um programa em linguagem de máquina é totalmente codificado em formato binário, o que torna praticamente impossível o entendimento pelo usuário.

O programa em linguagem de máquina pode ser diretamente processado pela unidade central de processamento (CPU) não requerendo qualquer tipo de tradução ou relocação. Um programa em linguagem de máquina não pode ser executado em outra máquina de modelo diferente, nem em outro SO.

Cada processador possui um conjunto único de instruções de máquina previamente definido pelo fabricante. Estas instruções especificam detalhes como registradores, modos de endereçamento de memória, tipo de dados que caracterizam um processador e suas potencialidades. Por sua vez, cada SO tem um grupo de instruções específico, chamada API – *Application Programlinterface* – que os compiladores usam para traduzir as instruções escritas na linguagem de alto nível para a linguagem entendida pelo *linker*. Isso faz com que um programa compilado em *Linux* não funcione com *Windows*; ou que um programa compilado em um PC não funcione em um *Apple*.

Um exemplo de tentativa de transformar os programas independentes do SO e do modelo do equipamento é a plataforma Java. Pesquise mais sobre o assunto no *site* http://www.java.com/pt_BR

4.4. Mecanismo de interrupção

Os sistemas multiprogramáveis tornaram mais eficiente a utilização dos recursos computacionais, por permitirem a execução simultânea (concorrente) de vários programas, permitindo que, quando um programa estiver realizando uma operação de E/S, outros possam utilizar o processador. Essa possibilidade do processador executar instruções em paralelo com as operações de E/S permitiu que diversas tarefas fossem executadas ao mesmo tempo. Outro aspecto importante que devemos ressaltar é a melhor utilização da memória principal, que antes, em sistemas monotarefa, era subutilizada, agora pode conter vários programas residentes concorrendo pela utilização do processador.

Os sistemas operacionais podem ser vistos como um conjunto de rotinas que executam concorrentemente de forma ordenada. A possibilidade de o processador executar instruções em paralelo com operações de E/S permite que diversas tarefas sejam executadas concorrentemente.

Durante a execução de um programa, alguns eventos inesperados podem ocorrer, interrompendo o seu fluxo normal de execução e ocasionando um desvio forçado. Estes eventos são conhecidos por interrupção ou exceção e podem ser consequência da sinalização de algum dispositivo de *hardware* externo ao processador ou da execução de instruções do próprio programa.

Sempre que ocorre uma interrupção, o sistema operacional é chamado para executar uma rotina de tratamento apropriada. Antes de realizar o desvio, o sistema operacional deve salvar o estado do processo interrompido, para que este possa continuar do ponto onde parou, quando voltar a ser executado. Um exemplo de interrupção ocorre quando um dispositivo avisa ao processador que alguma operação de E/S já está completa. Nesse caso, o processador deve interromper o programa para tratar o término da operação.

Quando isso ocorre, o programa que está em execução é interrompido e o controle desviado para uma rotina responsável por tratar o evento ocorrido, denominada de rotina de tratamento de interrupção. Para que o programa interrompido possa voltar a ser executado posteriormente, se faz necessário que, no momento da interrupção, um conjunto de informações sobre sua execução seja preservado. Essas informações consistem no conteúdo dos registradores, que deverão ser restaurados para que seja dada continuidade à execução do programa, conforme a Figura 4.1.

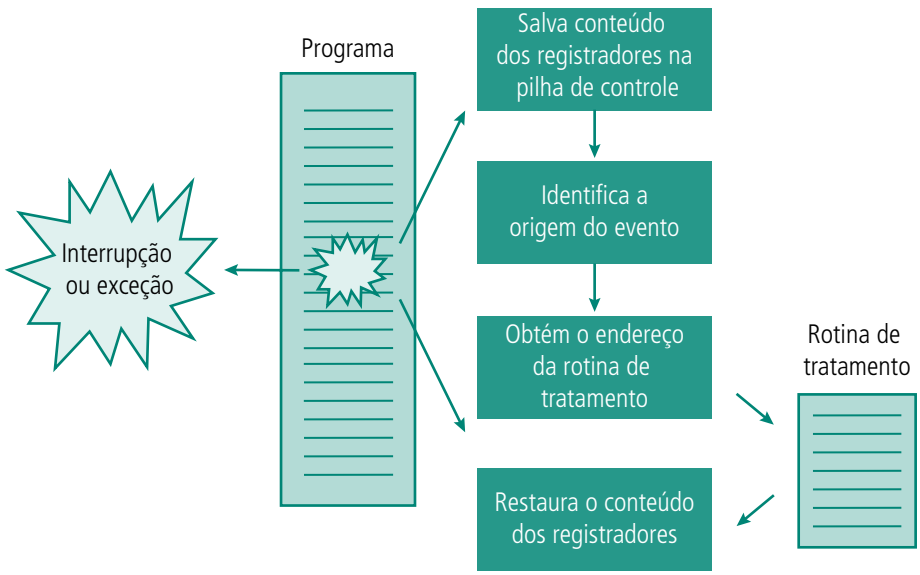

Figura 4.1: Mecanismo de interrupção
Fonte: Adaptado de Machado, 2004

Para cada tipo de interrupção existe uma rotina de tratamento associada,

para a qual o fluxo de execução deverá ser desviado. A identificação do tipo de evento ocorrido é fundamental para determinar o endereço da rotina de tratamento. No momento da interrupção, o processador deverá saber para qual rotina de tratamento deverá encaminhar o fluxo em execução que foi interrompido

Os eventos que causam a interrupção podem ser classificados como síncronos ou assíncronos. Um evento classificado como assíncrono, é independente dos dados de entrada e das instruções do programa, ou seja, pode ocorrer em qualquer ponto do programa. Essas interrupções não estão relacionadas com a instrução do programa corrente, são eventos imprevisíveis, podem ocorrer múltiplas vezes (exemplo: interrupção gerada pelo mouse, teclado, periféricos, etc.). Isto possibilita a ocorrência de múltiplas interrupções simultâneas, o que não seria interessante para nosso programa em execução. Uma maneira de evitar essa situação é a rotina de tratamento inibir as demais interrupções. No caso, na ocorrência de outras interrupções durante a execução da rotina de tratamento, elas serão ignoradas, ou seja, não receberão tratamento. Em outras palavras, elas só serão tratadas quando a rotina de tratamento da interrupção atual terminar. Interrupções com a característica de poder ser desabilitada são chamadas de **interrupções mascaráveis**.

A-Z

interrupções mascaráveis
são aquelas com a característica de poderem ser desabilitadas.

As interrupções que não podem ser desabilitadas são chamadas não-mascaráveis. Alguns exemplos de eventos que geram **interrupções não-mascaráveis** são: pressionar o botão *reset*, falha no *hardware*, etc.

A-Z

Interrupções não-mascaráveis
são aquelas com a característica de não poderem ser desabilitadas.

Já um evento classificado como síncrono, é resultado direto da execução do programa corrente. Tais eventos são previsíveis, e se um mesmo programa for executado várias vezes com a mesma entrada de dados, os eventos síncronos ocorrerão sempre nos mesmos pontos (instruções) do programa.

4.5. Operações de Entrada e Saída (E/S)

Nos primeiros sistemas de computação, os periféricos eram controlados pelo processador por meio de instruções especiais, chamadas instruções de E/S. Essas instruções continham detalhes específicos de cada periférico. Devido a isso, esse modelo criava uma forte dependência entre o processador e os dispositivos de E/S.

Para evitar esse problema, criou-se o controlador ou interface (Figura 4.2), que permitiu ao processador operar de maneira independente dos dispositivos de E/S. Com esse novo elemento, o processador não precisava mais se

comunicar diretamente com os periféricos, nem conhecer detalhes de operação específicos de cada periférico. Com isso, foram muito simplificadas as instruções de E/S do processador.

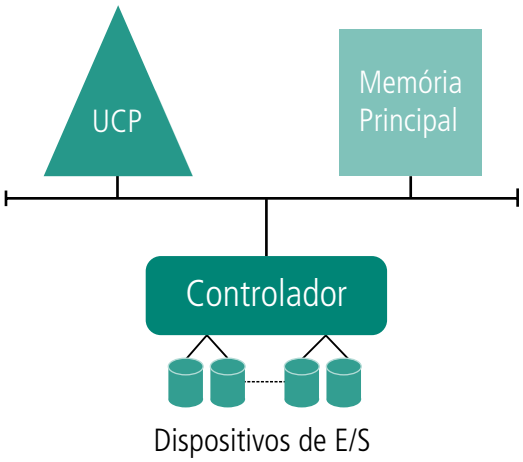

Figura 4.2: Controlador dos dispositivos de E/S
Fonte: Adaptado de Machado, 2004

Por meio do controlador, o processador gerenciava as operações de E/S, sem se preocupar com os detalhes de implementação de cada dispositivo. Com esse novo elemento, o processador não mais se comunicava diretamente com os periféricos, mas sim, pelo controlador.

- **E/S programada** - esse foi um dos primeiros modelos de operação de E/S. Após o processador iniciar a transferência de dados, ficava consultando o estado do periférico sucessivamente, até que a operação de E/S chegasse ao fim, mantendo o processador ocupado até o término da E/S (espera ocupada, do inglês *busy wait)*. Como o processador executa uma instrução muito mais rápida que uma operação de E/S realizada pelo controlador, havia um enorme desperdício de tempo do processador;

- **E/S por *polling*** - em relação ao modelo anterior, a evolução ocorreu ao permitir que algumas instruções pudessem ser executadas entre sucessivas consultas sobre o estado de uma operação de E/S. Com isso introduziu-se certo grau de paralelismo. Isso porque permitiu que outros programas pudessem ser executados, enquanto uma operação de E/S era realizada, mas, em determinados intervalos de tempo o sistema operacional deveria interrompê-los para verificar o estado da operação de E/S. Caso houvesse várias operações de E/S pendentes, o processamento seria interrompido diversas vezes para a verificação do estado das operações;

- **E/S controlada por interrupção** - ao invés de o sistema ficar periodicamente testando o estado das operações pendentes, o próprio controlador interrompe o processador para informar o término da operação. Cabe ao controlador a responsabilidade de controlar as operações de E/S. Quando essa operação termina, o controlador interrompe o processador para que este realize a transferência de dados para a memória principal. Após essa transferência, o processador está livre para executar outros programas. Apesar disso, se houver a transferência de um grande volume de dados, o processador será interrompido diversas vezes, reduzindo o seu desempenho. Para solucionar esse problema, foi implementada a técnica de transferência de dados chamada DMA (*Direct Memory Access*);

- **Acesso Direto à Memória (DMA)** - a técnica DMA permite que o controlador de E/S transmita um bloco de dados entre os dispositivos de E/S e a memória principal. O controlador acessa a memória diretamente, sem a necessidade da intervenção do processador, exceto no início e no final da transferência. O controlador realiza a operação de E/S, bem como a transferência de dados entre a memória e o dispositivo de E/S, e, somente ao final, interrompe o processador para avisar que a operação foi concluída. A área de memória utilizada pelo controlador na técnica de DMA é chamada *buffer* de entrada/saída.

Faça as seis atividades a seguir:

1. Porque o uso do *linker* se tornou inviável em sistemas multiprogramáveis? E qual seria a solução adotada para o problema gerado?

2. Qual a função da linguagem de controle?

3. Por que o mecanismo de interrupção é fundamental para a implementação da multiprogramação?

4. Explique o mecanismo de funcionamento das interrupções.

5. Pesquise sobre o termo *traps* em sistemas operacionais e em que situações elas ocorrem.

6. O que é DMA e qual a vantagem desta técnica?

Um sistema operacional fornece o ambiente no qual os programas são exe-

cutados. Esse ambiente é formado por um conjunto de rotinas que oferecem serviços aos usuários, às aplicações e ao próprio sistema. A esse conjunto de rotinas denominamos *kernel*. A Figura 4.3 nos mostra em que nível se encontra o *kernel* em um sistema computacional. O *kernel* é considerado o núcleo do sistema operacional.

Figura 4.3: Posicionamento do núcleo do sistema operacional em um sistema computacional
Fonte: Adaptado de Machado, 2004

As principais funções do núcleo encontradas nos sistemas operacionais são:

- Tratamento de interrupções e exceções;
- Criação e eliminação de processos e *threads;*
- Sincronização e comunicação entre processos e *threads;*
- Gerência de memória;
- Gerência do sistema de arquivos;
- Gerência dos dispositivos de entrada e saída;
- Suporte a redes locais e distribuídas;
- Contabilização do uso do sistema;
- Auditoria e segurança do sistema.

Podemos destacar como funções principais do *kernel* do sistema operacional as quatro gerências: gerência de processos, gerência de memória, gerência de entrada e saída, e gerência de sistema de arquivos.

Existem vários pontos de vista pelos quais poderíamos analisar um sistema operacional:

- Examinando os serviços oferecidos aos usuários;
- Analisando a interface disponibilizada aos usuários e programadores;
- Desmontando o sistema em seus componentes mais básicos.

Procure enumerar as atividades que você realiza e recursos que você costuma utilizar em computadores no seu dia a dia, para depois comparar com os serviços prestados por um sistema operacional, listados ao final desta aula.

4.6. Sistemas em lote

Os primeiros computadores eram máquinas dependentes de um operador. Os dispositivos de entrada comuns eram leitoras de cartões e unidades de fita, e os de saída eram impressoras, unidades de fita e perfuradoras de cartões. O usuário não interagia diretamente com o computador. Em vez disso, ele preparava uma tarefa (*job*), que consistia no programa, dados e algumas informações de controle sobre a natureza da tarefa (cartões de controle) e o entregava ao operador do computador. Algum tempo depois o resultado do programa podia ser obtido como uma listagem, uma fita, novo pacote de cartões ou uma listagem dos conteúdos dos registradores no caso de erro do programa.

O sistema operacional nesses primeiros computadores era bem simples. Sua principal tarefa era transferir controle automaticamente de uma tarefa para a próxima tarefa. O sistema operacional estava sempre residente na memória e executava uma tarefa de cada vez.

O monitor residente foi considerado o primeiro sistema operacional (rudimentar). Era um programa que ficava permanentemente em memória e sua função era transferir a execução de um *job* para outro.

Para acelerar o processamento, os operadores reuniam as tarefas em lotes com necessidades semelhantes e os executavam no computador como um grupo. Assim os programadores deixavam seus programas com o operador e o operador classificava os programas em lotes com requisitos semelhantes. À medida que o computador ficava disponível, executava outro lote ou *batch*.

A saída de cada tarefa seria enviada de volta ao programador apropriado.

> Neste ambiente de execução, a CPU ficava muitas vezes ociosa porque as velocidades dos dispositivos mecânicos de E/S (leitores de cartão) eram intrinsecamente menores do que as dos dispositivos eletrônicos (CPU). Com o tempo, é claro, melhorias na tecnologia e a introdução de discos resultaram em dispositivos de E/S mais rápidos (Silberschatz, Galvin e Gagne, 2000. p. 5).

Não é muito comum vermos tarefas sendo executadas em *batch*. Entretanto, existem alguns procedimentos que guardam bastante semelhança: os processos de *backup* e recuperação de bancos de dados, bem como o processo de inicialização do *Windows*. Em ambos os casos uma série de tarefas são executadas de forma automática, em sequência e de forma mais ou menos independente da vontade do usuário.

4.7. Escalonamento de tarefas e multiprogramação

A introdução da tecnologia de disco permitiu que o sistema operacional mantivesse todas as tarefas em um disco, em vez de em uma leitora de cartões serial. Surge então o conceito de escalonamento de tarefas e posteriormente de multiprogramação.

A multiprogramação aumenta a utilização da CPU, organizando as tarefas de forma que a CPU esteja sempre ocupada. Assim, o SO mantém várias tarefas na memória simultaneamente, todas competindo pelos recursos do computador, conforme Figura 4.4.

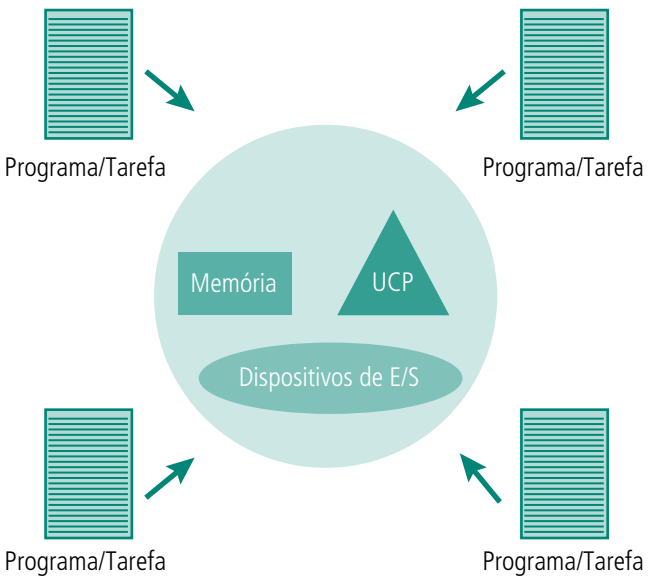

Figura 4.4: Sistema multiprogramável ou multitarefa
Fonte: Adaptado de Machado, 2004

O sistema operacional escolhe e começa a executar a tarefa carregada na memória. Em alguns momentos a tarefa que está em execução deverá espe-

rar a conclusão de alguma outra tarefa, como uma operação de E/S, que é muito mais lenta. Em um sistema operacional não multiprogramado, a CPU ficaria ociosa. Em um sistema de multiprogramação, o sistema operacional simplesmente passa para outra tarefa e a executa. Quando esta segunda tarefa precisar esperar, a CPU passará para outra tarefa e assim por diante. Por fim, a primeira tarefa terminará a sua operação de E/S e terá a CPU de volta. Neste esquema, a CPU nunca fica ociosa.

Todos os *jobs*, ou tarefas, que entram no sistema são mantidos num pool de *jobs* (espécie de fila). Esse pool de *jobs* consiste em todos os processos residentes no disco aguardando alocação da memória principal (espaço na memória). O número de *jobs* que pode ser mantido simultaneamente na memória principal geralmente é muito menor do que o número de *jobs* que pode estar no pool de *jobs*. Se vários *jobs* estiverem prontos para serem carregados na memória e se não houver espaço suficiente para todos, o sistema operacional deverá fazer a escolha. Essa tomada de decisão é chamada de **escalonamento de tarefas** ou de *jobs*.

A-Z

Escalonamento de tarefas consiste em selecionar uma tarefa, de um conjunto de tarefas aptas para execução, que será a próxima a ser executada pela CPU

Os sistemas em *batch* multiprogramados forneciam um ambiente no qual os vários recursos do sistema, como por exemplo a CPU, memória e dispositivos periféricos, eram utilizados de forma eficaz, mas não permitiam a interação do usuário com o sistema de computação. Tempo compartilhado (*time sharing*) ou multitarefa é uma extensão da multiprogramação: a CPU executa vários *jobs* alternando entre eles (multiprogramação), mas as trocas ocorrem com tanta frequência que o usuário pode interagir com seu programa como se tivesse uma máquina exclusiva para ele. Assim se permite a comunicação direta entre o usuário e o sistema. Devido a esse tipo de interação, os sistemas de tempo compartilhado também ficaram conhecidos como sistemas *on-line* ou de tempo real (*real time*). O usuário passa instruções ao sistema operacional ou a um programa diretamente, usando o teclado ou um mouse, e espera por resultados imediatos. O tempo de resposta deve ser curto.

Um sistema operacional de tempo compartilhado permite que muitos usuários compartilhem o computador ao mesmo tempo. Como cada ação ou comando em um sistema de tempo compartilhado tende a ser curto, apenas um pequeno tempo de CPU é necessário para cada usuário. Como o sistema alterna rapidamente de um usuário para outro, cada usuário tem a impressão de que todo o sistema de computação está dedicado a ele, enquanto, na verdade, o computador está sendo compartilhado por muitos usuários.

Nos sistemas *time sharing* os usuários podem interagir com o programa em execução, muitas vezes através de terminais, tendo a ilusão de possuir a máquina dedicada a execução de seu programa. Porém, o ocorre na verdade, é a divisão de tempo de processamento entre os usuários.

4.8. Serviços de Sistemas Operacionais

Um outro enfoque que pode ser dado ao estudo dos sistemas operacionais é encarar o SO como um prestador de serviços: um sistema operacional fornece um ambiente para a execução de programas, ou seja: fornece certos serviços a programas e aos usuários destes programas. Este enfoque é interessante, pois, frequentemente, você verá na literatura abordagens das funções ou estrutura do SO como "serviços". Estes serviços podem ser assim classificados:

a) **Execução de programa** - o sistema deve ser capaz de carregar um programa na memória e executar esse programa. Também deve ser capaz de encerrar a sua execução, quer de forma normal ou anormal (indicando erro);

b) **Operação de E/S** - um programa em execução pode precisar de E/S. Essa operação de E/S pode envolver um arquivo ou dispositivo de E/S. Portanto, o sistema operacional deve fornecer os meios para realizar as operações de E/S;

c) **Manipulação do sistema de arquivos** - os programas precisam ler e gravar arquivos, criar e excluir arquivos por nome; essas atividades devem ser gerenciadas pelo sistema operacional;

d) **Comunicação** - existem muitas circunstâncias nas quais um processo precisa trocar informações com outro processo. Essa comunicação entre processos pode ocorrer com processos em um mesmo computador ou entre processos que estão executando em diferentes sistemas de computação ligados por uma rede. As comunicações podem ser implementadas via memória compartilhada ou pela técnica de troca de mensagens, na qual pacotes de informações são movidos entre processos pelo sistema operacional;

e) **Detecção de erros -** O sistema operacional precisa estar constantemente ciente de possíveis erros. Os erros podem ocorrer no *hardware* da CPU e da memória, em dispositivos de E/S ou no programa de usuário. Para cada tipo de erro o sistema operacional deve tomar a medida adequada para garantir o funcionamento correto e consistente (e não simplesmente travar).

Outras funções dos sistemas operacionais servem não somente para ajudar o usuário, mas para garantir a operação eficiente do sistema em si, tais como:

a) **Alocação de recursos** - recursos (memória e dispositivos de E/S) devem ser alocados para cada um dos processos que estão em execução simultânea;

b) **Contabilização** - é preciso manter um registro dos usuários que utilizam os recursos do computador, em que quantidade e quais recursos. Este registro pode ser usado para contabilização. Estas estatísticas de uso podem ser uma ferramenta valiosa para os pesquisadores que desejam reconfigurar o sistema para melhorar os serviços oferecidos;

c) **Proteção** - quando vários processos independentes forem executados ao mesmo tempo, um processo não poderá interferir na área de outro processo ou do próprio sistema operacional. A proteção visa garantir que todo acesso aos recursos do sistema seja controlado. Estende-se também à proteção dos dispositivos de E/S externos (modens, placa de rede, etc.) de tentativas de acesso inválidas e ao registro de todas as conexões para detecção de invasões.

Resumo

Nesta aula vimos o conceito de interpretador de comandos (*shell*), qual a sua funcionalidade em um sistema operacional. Aprendemos também algumas questões sobre a linguagem de máquina que é utilizada pela CPU para executar os programas. Por fim analisamos alguns conceitos importantes sobre interrupções de *software* e *hardware* e o funcionamento das operações de entrada e saída que envolvem tanto a máquina em si quanto o sistema operacional propriamente dito. Aprendemos sobre a figura do *kernel* em um sistema operacional bem como suas principais funções. Vimos alguns tipos de sistemas operacionais que começaram o histórico de sistemas multiprogramados, a conceituação de *jobs* e tarefas e a importância dos discos magnéticos para a evolução de nossos sistemas atuais. Analisamos a estrutura do sistema como um prestador de serviços.

Atividades de aprendizagem

1. O *kernel* possui 4 funções de gerência que são essenciais para um sistema operacional. Quais são?

2. Por que podemos afirmar que os discos magnéticos foram de fundamental importância para o surgimento da multiprogramação nos sistemas atuais?

3. Quais as principais características dos primeiros sistemas operacionais?

4. "Um bom escalonador de tarefas é aquele que consegue equilibrar bem tarefas orientadas a entrada e saída e tarefas orientadas a processamento". Explique esta frase.

5. Quais são as principais características dos sistemas *time sharing*?

6. No exercício passado no início desta aula foi pedido para você enumerar as atividades e recursos que você costuma utilizar em computadores. Relacione com os serviços e funções dos sistemas operacionais descritos nesta aula.

Aula 5 – Arquitetura do Sistema Operacional

Objetivos

Conhecer os modos de acesso ao processador como forma de proteção do sistema.

Compreender a estrutura das chamadas de sistema utilizadas para a comunicação com o *kernel* do sistema.

Analisar as características de uma arquitetura de sistema operacional dividido em camadas ou não.

Verificar o funcionamento e importância de um interpretador de comandos.

5.1. Modos de acesso

Vimos que os programas de computador ou aplicações em geral, consistem de conjuntos de instruções a serem executadas pelo processador ou CPU. Existem certas instruções que não podem ser disponibilizadas diretamente à disposição das aplicações, pois a sua utilização indevida ocasionaria sérios problemas à integridade do sistema. Estas instruções, como operações de E/S, só podem ser executadas pelo sistema operacional, isso para impedir a ocorrência de problemas de segurança e mesmo a violação do próprio sistema. As instruções que têm o poder de comprometer o sistema são conhecidas como instruções privilegiadas, enquanto as instruções não-privilegiadas são as que não oferecem perigo ao sistema.

Para que uma aplicação possa executar uma instrução privilegiada, o processador implementa o mecanismo de modos de acesso. Existem basicamente dois modos de acesso: modo usuário e modo *kernel*. Quando o processador trabalha no modo usuário, uma aplicação só pode executar instruções não-privilegiadas (instruções que não oferecem riscos), tendo acesso a um número reduzido de instruções; no modo *kernel* a aplicação pode ter acesso ao conjunto total de instruções do processador.

Quando um programa que esteja no modo usuário tenta executar uma instrução privilegiada, é gerado um erro de proteção. O processador sinaliza este erro através de uma exceção, o sistema operacional será chamado e o programa será finalizado. Você deve estar se perguntando como ocorrem as transições entre os modos de acesso, ou seja, do modo usuário para o modo *kernel* e vice-versa. No momento da carga do sistema (*boot*), o sistema operacional inicia em modo *kernel*. Após estar carregado em memória, o sistema operacional permite que os programas de usuários sejam carregados apenas em modo usuário.

Chaveamento dos modos:
Interrupção - modo usuário → modo protegido.
Instrução - modo protegido → modo usuário.

Se um programa estiver executando em modo usuário e ocorrer qualquer tipo de interrupção (interrupção de *hardware*, exceção ou interrupção de *software*), o modo de acesso é alterado para o modo *kernel*. Com isso, a rotina de tratamento é executada em modo *kernel*. Ao final de toda rotina de tratamento, há uma instrução específica que, antes de retornar para o programa do usuário, altera o modo de acesso para o modo usuário (Figura 5.1).

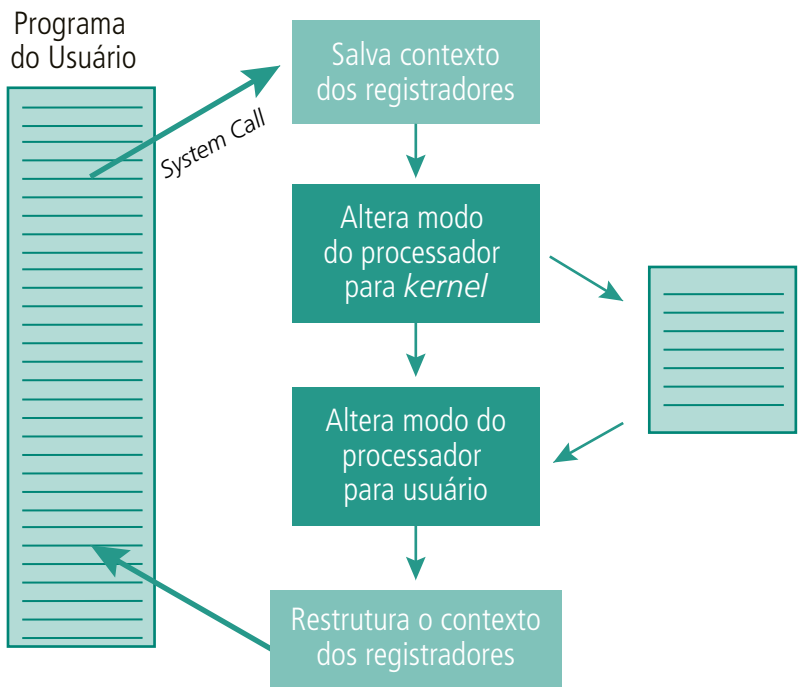

Figura 5.1: Mudança dos modos de acesso durante uma interrupção
Fonte: Adaptado de Machado, 2004

A melhor maneira de controlar o acesso às instruções privilegiadas é permitir

que apenas o sistema operacional tenha acesso a elas. Sempre que uma aplicação necessita de um serviço que incorra em risco para o sistema, a solicitação é feita através de uma *system call.* Esta altera o modo de acesso do processador para um modo mais privilegiado (modo *kernel*). Ao término da rotina do sistema, o modo de acesso é retornado para o modo usuário. Caso um programa tente executar uma instrução privilegiada, sem o processador estar no modo *kernel*, uma exceção é gerada e o programa é encerrado.

Caso uma aplicação tenha acesso a áreas de memória onde está carregado o sistema operacional, um programador mal-intencionado ou um erro de programação poderia gravar nesta área, violando o sistema. Com o mecanismo de modo de acesso, para uma aplicação escrever numa área onde resida o sistema operacional o programa deve estar sendo executado no modo *kernel*.

5.2. *System calls* (Chamadas ao sistema)

As chamadas ao sistema (*system calls*) fornecem a interface entre um processo e o sistema operacional. Essas chamadas estão disponíveis como instruções em linguagem *assembly* e, em geral, são listadas nos manuais usados por programadores em linguagem *assembly*, conforme Figura 5.2.

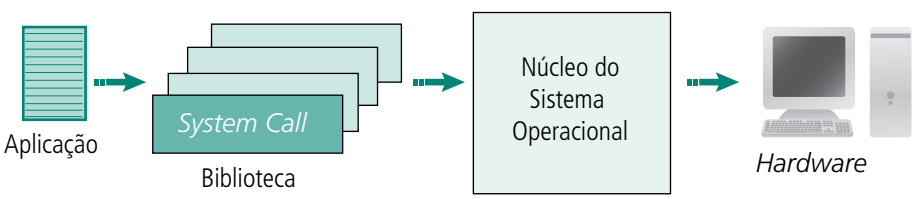

Figura 5.2: Chamada de sistema (*system call*)
Fonte: Adaptado de Machado, 2004

Certos sistemas permitem que as chamadas ao sistema sejam feitas diretamente de um programa de linguagem de nível mais alto e, nesse caso, as chamadas normalmente lembram chamadas de subrotinas ou de funções predefinidas. Várias linguagens – como C, C++ e *Perl* – foram definidas para substituir a linguagem *assembly* na programação de sistemas. Essas linguagens permitem que as chamadas ao sistema sejam feitas diretamente. Por exemplo, as chamadas ao sistema do *Unix* podem ser feitas diretamente a partir de um programa em C ou em C++.

A documentação dos SO costuma disponibilizar estas chamadas com uma série de funções cujo conjunto é chamado de API – Application Program Interface. Esta sigla API pode ser usada também para designar de uma forma genérica como uma grande aplicação (como um gerenciador de banco de dados) aceita receber solicitações de outras aplicações.

Como um exemplo da forma em que as chamadas de sistema são usadas, consideremos escrever um programa simples para ler dados de um arquivo e copiá-los para outro arquivo. Cada passo dado pelo programa praticamente efetua uma chamada ao sistema:

a) A primeira entrada que o programa precisará são os nomes dos dois arquivos: de entrada e de saída. Uma abordagem é fazer o programa pedir ao usuário os nomes dos dois arquivos. Em sistemas baseados em mouse e ícones, um menu de nomes de arquivos geralmente é exibido em uma janela (esta lista de diretórios e arquivos é fornecida por uma chamada ao SO); o usuário pode usar o mouse para selecionar o nome de origem. Uma janela pode ser aberta para que o nome do arquivo de destino seja especificado;

b) Depois que os dois nomes de arquivos tiverem sido obtidos, o programa deve abrir o arquivo de entrada e criar o arquivo de saída. Cada uma dessas operações requer uma chamada ao sistema;

c) Agora que os dois arquivos estão prontos, entramos em um laço que lê dados do arquivo de entrada (uma chamada ao sistema) e os grava no arquivo de saída (outra chamada ao sistema). Lembramos que entre essas operações de leitura e escrita poderão ocorrer alguns erros que gerarão outras chamadas ao sistema. Por exemplo, a operação de escrita pode encontrar erros, dependendo do dispositivo de saída (falta de espaço em disco, fim físico da fita, etc.);

d) Finalmente, depois que o arquivo todo tiver sido copiado, o programa deverá fechar os dois arquivos (outra chamada ao sistema), gravar uma mensagem na console (mais uma chamada ao sistema) e finalmente terminar normalmente (a chamada final ao sistema).

Como podemos observar, os programas fazem uso pesado do sistema operacional. Entretanto, a maioria dos usuários nunca chega a ver esse nível de detalhe.

As chamadas de sistema ocorrem de diferentes maneiras, dependendo do SO e do computador que está sendo usado. Geralmente, mais informações são necessárias além de simplesmente identificar a chamada ao sistema desejada; o tipo e a quantidade exata de informações variam de acordo com a chamada em questão (os parâmetros, que você já viu ao criar suas funções em programação). Por exemplo, para obter entrada é preciso especificar o arquivo ou dispositivo a ser usado como origem e o endereço e o tamanho do *buffer* (espaço) de memória no qual a entrada deve ser lida.

As chamadas de sistema podem ser agrupadas basicamente em cinco categorias principais de acesso ou controle:

- Controle de processo;

- Manipulação de arquivos;
- Manipulação de dispositivos;
- Manutenção de informações;
- Comunicações.

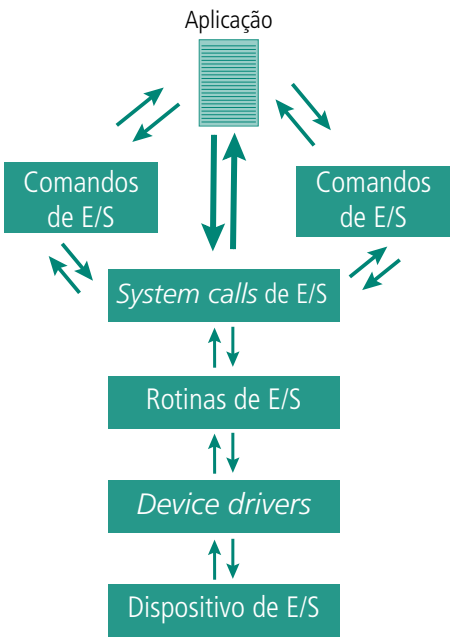

Figura 5.3: Operações de entrada/saída através das *system calls*
Fonte: Adaptado de Machado, 2004

Um dos objetivos principais das *system calls* de E/S é simplificar a interface entre as aplicações e os dispositivos. Com isso, elimina-se a necessidade de duplicação de rotinas idênticas nos diversos aplicativos, além de esconder do programador características específicas associadas à programação de cada dispositivo. São as chamadas de sistema que irão intermediar a solicitação de uso de algum dispositivo feito pelas aplicações. O programador ao elaborar uma aplicação não precisa fazer nenhuma referência à configuração do dispositivo a ser utilizado. A Figura 5.3 ilustra a comunicação entre a aplicação e os dispositivos de E/S de maneira simplificada.

5.3. Arquiteturas do núcleo (*kernel*)

O projeto de um sistema operacional é bastante complexo e deve atender a diversos requisitos, algumas vezes conflitantes, como confiabilidade, portabilidade, fácil manutenção, flexibilidade e desempenho.

A estrutura do núcleo do sistema operacional, ou seja, a maneira como o código do sistema é organizado e o inter-relacionamento entre seus diversos componentes, pode variar conforme a concepção do projeto. A seguir serão abordadas as principais arquiteturas dos sistemas operacionais.

5.3.1. Arquitetura monolítica

Neste tipo de organização, arquitetura monolítica, a forma mais comum de ser encontrada é aquela que estrutura o sistema como um conjunto de rotinas que podem interagir livremente umas com as outras, conforme Figura 5.4. A estrutura monolítica pode ser comparada com uma aplicação formada por vários procedimentos que são compilados separadamente e depois *linkados*, formando um grande e único programa executável.

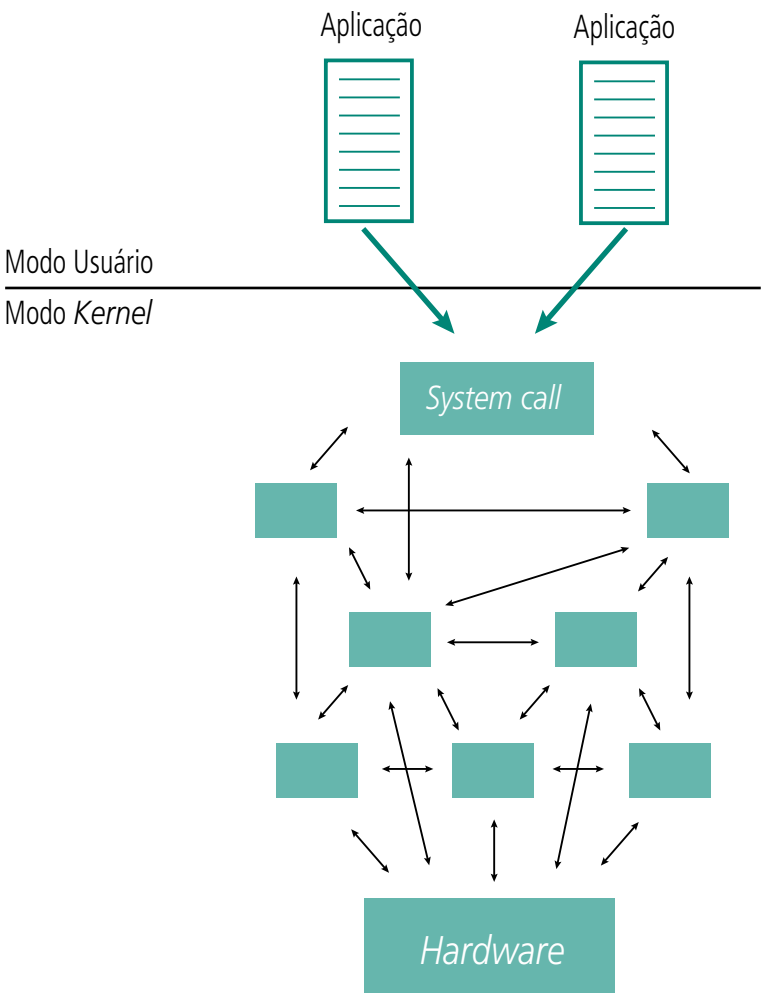

Figura 5.4: Arquitetura monolítica
Fonte: Adaptado de Machado, 2004

Um sistema tão grande e complexo quanto um sistema operacional moderno deve ser cuidadosamente construído para que funcione bem e possa ser facilmente modificado. Uma abordagem comum é dividir a tarefa em pequenos componentes em vez de ter um sistema monolítico.

5.3.2. Arquitetura de camadas

Um sistema em camadas divide o sistema operacional em camadas sobrepostas. Cada módulo oferece um conjunto de funções que podem ser utilizadas por outros módulos. Você viu chamarmos o computador de máquina de níveis no primeiro capítulo; por sua vez o próprio SO pode ser encarado desta forma.

A principal vantagem da abordagem em camadas é a modularidade. As camadas são selecionadas de forma que cada uma utilize as funções (operações) e serviços apenas das camadas de nível mais baixo. Essa abordagem simplifica a depuração e verificação do sistema, além de criar uma hierarquia de níveis de modos de acesso, protegendo as camadas mais internas. A Figura 5.5 apresenta esta formação em camadas para o DOS; entretanto, o nível de segurança é baixo, pois é permitida a comunicação com camadas mais baixas sem passar pelas intermediárias.

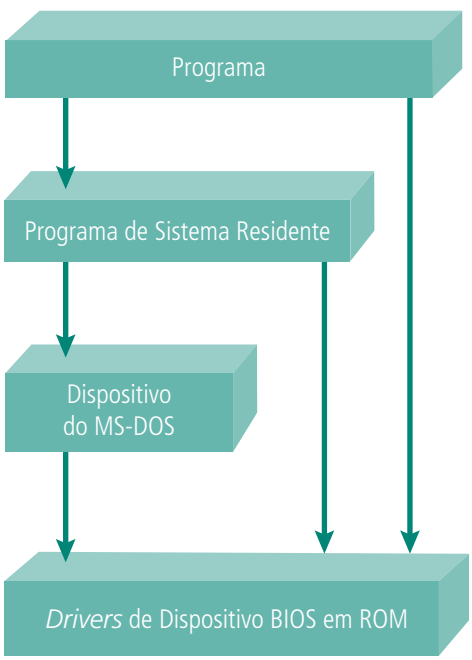

Figura 5.5: Estrutura em camadas do MS-DOS
Fonte: Adaptado de Silberschatz, Galvin e Gagne, 2000

A primeira camada pode ser depurada sem preocupação com o resto do sistema, porque, por definição, ela só utiliza o *hardware* básico (que é

considerado correto) para implementar suas funções. Depois que a primeira camada é depurada, pode-se presumir seu funcionamento correto enquanto a segunda camada é depurada e assim por diante. Se for encontrado um erro durante a depuração de determinada camada, o erro deve estar nessa camada, porque as camadas inferiores já foram depuradas. Assim, o projeto e a implementação do sistema são simplificados quando o sistema é dividido em camadas.

A principal dificuldade da abordagem em camadas está na definição adequada das várias camadas. Como uma camada semente poderá usar as camadas que estão em um nível inferior, é preciso haver um planejamento cuidadoso.

Um último problema com implementações em camadas é que elas tendem a ser menos eficientes. Por exemplo, quando um programa de usuário executa uma operação de E/S (entrada e saída), ele executa uma chamada de sistema (*system call*) que é desviada para a camada de E/S, que chama a camada de gerência de memória, que, por sua vez, chama a camada de escalonamento de CPU, que é então passada para o *hardware*. Em cada camada, os parâmetros podem ser modificados, os dados precisam ser transferidos e assim por diante. Cada camada acrescenta novo custo à chamada ao sistema; o resultado final é uma chamada ao sistema que demora mais do que em um sistema sem camadas.

As limitações da eficiência causaram reações contra a estrutura em camadas nos últimos anos. Menos camadas com mais funcionalidades estão sendo projetadas, fornecendo a maior parte das vantagens do código modularizado e evitando difíceis problemas da definição e interação em camadas.

5.3.3. Arquitetura microkernel

À medida que o Unix se expandiu, o *kernel* tornou-se grande e difícil de gerenciar. Foi desenvolvido, então, um sistema operacional usando a abordagem de *microkernels*. O resultado é um *kernel* menor. Uma tendência dos sistemas operacionais modernos é tornar o núcleo do sistema operacional menor e o mais simples possível. Em geral, os *microkernels* fornecem gerência mínima de memória e processos, além de um recurso de comunicação.

A principal função do *microkernel* é fornecer um recurso de comunicação entre programa cliente e os vários serviços que também estão em execução no espaço de usuário. Para implementar esta ideia, o sistema é dividido em

processos, sendo cada um responsável por oferecer um conjunto de serviços, como serviços de arquivo, serviços de criação de processos, serviços de memória, serviços de escalonamento, etc.

Se o programa cliente desejar acessar um arquivo, ele deverá interagir com o servidor de arquivos. Mas o programa cliente e o serviço nunca vão interagir diretamente; em vez disso, eles se comunicam indiretamente trocando mensagens com o *microkernel*.

Sempre que uma aplicação deseja algum serviço, ela solicita ao processo responsável. Neste caso, a aplicação que solicita um serviço é chamada de cliente, enquanto o processo que responde à solicitação é chamado de servidor. Um cliente, que pode ser uma aplicação de um usuário ou um outro componente do sistema operacional, solicita um serviço enviando uma mensagem. É função do núcleo do sistema realizar a comunicação, ou seja, a troca de mensagens entre o cliente e o servidor.

Os benefícios da abordagem do *microkernel* incluem a facilidade de expandir o sistema operacional. Todos os novos serviços são adicionados ao espaço de usuário e, consequentemente, não exigem modificação do *kernel*. Quando o *kernel* precisa ser modificado, as alterações tendem a ser menores, porque o *microkernel* é um *kernel* menor. O *microkernel* também fornece mais segurança e confiabilidade, pois a maior parte dos serviços está sendo executada como processos de usuário, em vez de *kernel*. Se um serviço falhar, o resto do sistema operacional permanece inalterado.

O *kernel* coordena a troca de mensagens entre as aplicações (clientes e servidoras). O *Windows* NT é projetado para executar várias aplicações, incluindo Win32 (aplicações nativas do *Windows*), OS/2 e POSIX. A estrutura cliente-servidor do *Windows* NT está representada na Figura 5.6.

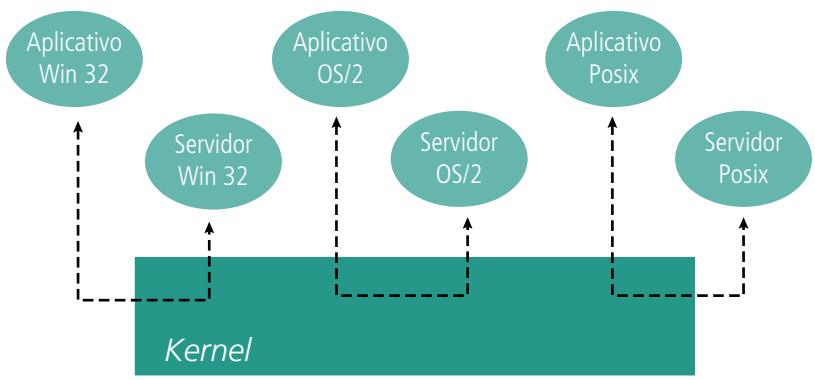

Figura 5.6: Estrutura do Windows NT 4.0 (núcleo e servidores de aplicações)
Fonte: Adaptado de Machado, 2004

A utilização deste modelo permite que os servidores executem em modo usuário, ou seja, não tenham acesso direto a certos componentes do sistema. Apenas o núcleo do sistema, responsável pela comunicação entre clientes e servidores, executa no modo *kernel*. Como consequência, se um erro ocorrer em um servidor, este servidor pode parar, mas o sistema não ficará inteiramente comprometido.

Como os servidores se comunicam através de trocas de mensagens, não importa se os clientes e servidores estão sendo processados em um sistema com um único processador, com múltiplos processadores ou ainda em um ambiente de sistema distribuído.

Apesar de todas as vantagens deste modelo, sua implementação na prática é muito difícil devido a certas funções dos sistemas operacionais exigirem acesso direto ao *hardware*, como operações de entrada e saída. Na realidade, o que é implementado mais usualmente é uma combinação do modelo de camadas com o modelo cliente-servidor.

Um sistema computacional é formado por níveis, onde a camada de nível mais baixo é o hardware. Acima desta camada encontramos o sistema operacional que oferece suporte para as aplicações. O modelo de máquina virtual, ou virtual machine (VM), cria um nível intermediário entre o hardware e o sistema operacional, denominado gerência de máquinas virtuais. Este 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.

5.4. Interpretador de comandos

Um dos utilitários mais importantes para um sistema operacional é o interpretador de comandos (ou *shell*), que é a interface entre o usuário e o sistema operacional. Alguns sistemas operacionais incluem o interpretador de comandos no *kernel* (núcleo do sistema operacional). Outros podem tratar o interpretador de comandos como um programa especial que fica executando quando um *job* é iniciado ou quando um usuário entra no sistema (em sistemas de tempo compartilhado).

Muitos comandos do usuário são passados ao sistema operacional por instruções de controle digitadas em uma linha da tela. Esse programa é chamado de interpretador da linha de comandos. Sua função é simples: obter o comando seguinte e executá-lo. Antes das interfaces gráficas esta era a única forma de comunicação entre a pessoa do usuário e o SO.

Os sistemas operacionais geralmente se diferenciam bastante na área do *shell*, com um interpretador de comandos amigável ao usuário tornando o sistema mais agradável a alguns e menos a outros. Estas diferenças se acentuaram com as interfaces gráficas.

O interpretador de comandos (*shell*) pode ser utilizado na seguinte forma:

a) O mouse é movido para posicionar o ponteiro sobre imagens na tela, ou ícones, que representam programas, arquivos ou funções;

b) Dependendo da localização do ponteiro do mouse, clicar em um botão do mouse pode chamar um programa, selecionar um arquivo ou diretório ou abrir um menu que contém comandos;

c) O clicar do mouse é a "chamada" a algum programa, arquivo ou pasta e irá utilizar o interpretador da linha de comando (*shell*). O clique do mouse já é, portanto, um comando que deverá ser interpretado pelo *shell*.

As instruções de comando lidam com a criação e gerência de processos, tratamento de E/S, gerência de armazenamento secundário, gerência de memória principal, acesso ao sistema de arquivo e proteção, e redes; enfim, tudo o que a pessoa do usuário precisa para operar seus programas e atender suas necessidades.

Resumo

Nesta aula aprendemos sobre funcionamento da comunicação entre os programas de usuários e o núcleo do sistema, pelas chamadas de sistema. Também vimos a forma como o *hardware* participa da proteção dos diversos tipos de processos e, consequentemente, dos diversos tipos de instruções que irão executar na CPU. Por fim, analisamos algumas formas de se projetar o *kernel* de um sistema operacional e suas implicações no desempenho final do sistema.

Atividades de aprendizagem

1. O que são instruções privilegiadas e não-privilegiadas? Qual a relação dessas instruções com os modos de acesso?

2. Como o kernel pode ser protegido pelo mecanismo de modos de acesso?

3. O que é um system call e qual a sua importância para a segurança do sistema?

4. Como as system calls são utilizadas por um programa?

5. Compare as arquiteturas monolíticas e de camadas. Quais as vantagens e desvantagens de cada arquitetura?

6. Pesquise comandos disponíveis em linguagens de controle de sistemas operacionais.

7. Pesquise software disponíveis para utilização de máquinas virtuais e quais as suas principais características.

Aula 6 – Introdução à gerência de processos, memória e arquivos

Objetivos

Conhecer as funções de gerência principais de um sistema operacional.

Analisar as funções do elemento processo dentro de um sistema operacional.

Verificar a administração da utilização de recursos pelo sistema operacional.

6.1. Gerência de processos

Um programa não faz nada a não ser que suas instruções sejam executadas por uma CPU. Um processo pode ser considerado um programa em execução, mas sua definição será ampliada à medida que explorarmos melhor o conceito. Um programa de usuário executado em tempo compartilhado é um processo. Um processador de texto executado por um usuário individual em um PC é um processo. Uma tarefa de sistema, como enviar saída para uma impressora, também é um processo.

Consideraremos que um processo é um *job* ou um programa de tempo compartilhado. Um processo precisa de determinados recursos – tempo de CPU, memória, arquivos e dispositivos de E/S (entrada e saída) – para realizar sua tarefa. Estes recursos são dados ao processo quando ele é criado ou alocados a ele durante sua execução. Além dos vários recursos físicos e lógicos que um processo obtém quando é criado, vários dados de inicialização podem ser fornecidos.

Um processo é um programa em execução. Um programa por si só não é um processo; um programa é uma entidade passiva, como o conteúdo de um arquivo armazenado em disco, enquanto um processo é uma entidade ativa, com um contador do programa especificando a próxima instrução a ser executada.

A execução de um processo deve ser sequencial. A CPU executa uma instru-

ção do processo após a outra até o processo terminar. Além disso, a qualquer momento, no máximo uma instrução é executada em nome do processo. Assim, embora dois processos possam ser associados com o mesmo programa, eles são considerados duas sequências de execução separadas. É comum ter um programa que utilize muitos processos para sua execução.

A-Z

Multiplexando
Multiplexação é a divisão de algum serviço oferecido entre vários processos. Você verá este conceito usado também para um canal de comunicação.

Um processo é a unidade de trabalho em um sistema. Em um sistema teremos uma coleção de processos que são desde processos do próprio sistema operacional (aqueles que executam códigos do SO) até os processos de usuário (aqueles que executam código do usuário). Todos esses processos podem executar concorrentemente, **multiplexando** a CPU entre eles.

O sistema operacional é responsável pelas seguintes atividades em relação à gerência de processos:

A-Z

Deadlocks
Deadlock caracteriza uma situação em que ocorre um impasse, onde dois ou mais processos ficam impedidos de continuar suas execuções, ou seja, ficam bloqueados. Um processo aguarda a liberação de um recurso que está sendo utilizado por um outro processo que, por sua vez, aguarda a liberação de outro recurso alocado ou dependente do primeiro processo.

a) Criar e excluir processos de usuário e de sistema;

b) Suspender e retomar processos;

c) Fornecer mecanismos para a sincronização de processos;

d) Fornecer mecanismos para a comunicação de processos;

e) Fornecer mecanismos para o tratamento de ***deadlocks***.

6.2. Gerência de memória principal

A gerência do espaço da memória principal é uma das preocupações importantes dos sistemas de computação modernos. A memória principal é um repositório de dados rapidamente acessíveis compartilhados pela CPU e dispositivos de E/S (entrada e saída).

A-Z

DMA
Direct Memory Access (Acesso Direto à Memória) é uma forma de controle de entradas e saídas sem a utilização constante do microprocessador, como se houvesse uma comunicação direta entre o dispositivo DMA de E/S e a memória. Isto ocorre, por exemplo, com a placa de vídeo e o HD.

O processador central lê as instruções da memória principal durante o ciclo de busca de instruções e lê e grava dados da memória principal durante o ciclo de busca de dados. As operações de E/S (entrada e saída) implementadas via **DMA** também fazem a leitura e escrita de dados na memória principal. A memória principal geralmente é o único dispositivo de armazenamento que a CPU pode endereçar e acessar diretamente. Para que a CPU processe dados do disco, esses dados devem primeiro ser transferidos para a memória principal por chamadas de E/S geradas pela CPU. Do mesmo modo, as instruções dos programas devem estar na memória principal para que a CPU as execute.

Para melhorar a utilização da CPU e a velocidade da resposta do computador aos seus usuários, é preciso manter vários programas na memória principal. Existem muitos esquemas diferentes de gerência de memória. A seleção de um esquema de gerência de memória para um sistema específico depende de muitos fatores, especialmente do projeto de *hardware* do sistema.

O sistema operacional é responsável pelas seguintes atividades em relação à gerência de memória:

a) Manter registro das partes da memória que estão sendo usadas no momento e por qual processo;

b) Decidir quais processos deverão ser carregados na memória quando houver espaço disponível;

c) Alocar e desalocar espaço na memória, conforme necessário.

Os sistemas operacionais de tempo compartilhado ou de tempo real são ainda mais complexos porque os usuários podem ir se conectando e solicitando a execução de programas até um momento em que pode não haver mais memória real disponível. É possível ainda manter um tempo de resposta razoável utilizando um recurso chamado memória virtual. Quando não há mais memória física disponível, algum processo ocioso naquele momento é passado rapidamente para um disco que agora irá servir como extensão da memória principal. Quando chegar a vez de este processo ser atendido, um outro (ou mais de um) processo ocioso é descarregado no disco, cedendo lugar para este voltar para a memória principal. Esta movimentação de ida e volta da memória para o disco é chamada de paginação ou *swap* e está ilustrado na Figura 6.1.

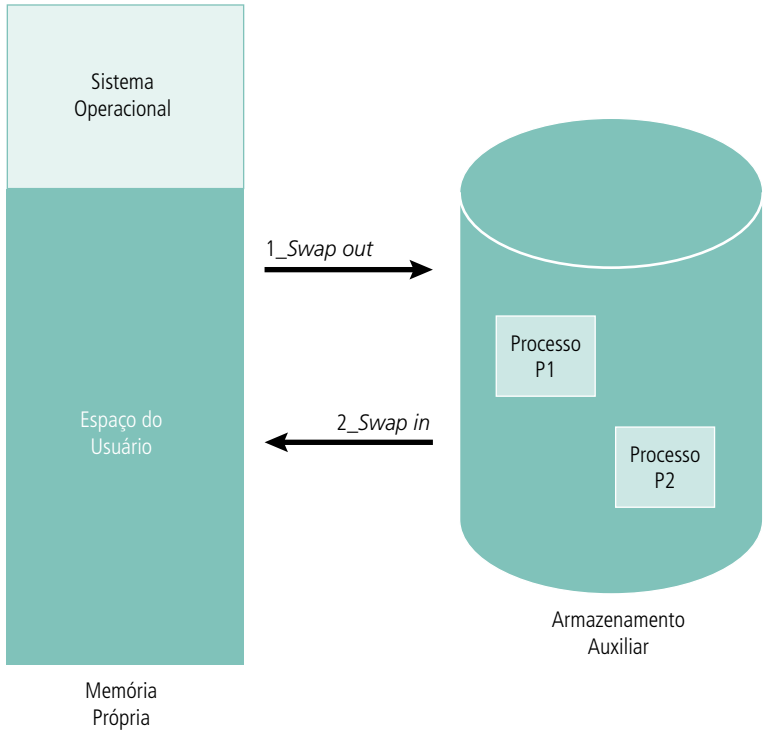

Figura 6.1: Troca de dois processos usando um disco como armazenamento auxiliar (secundário)
Fonte: Adaptado de Silberschatz, Galvin e Gagne, 2000

Esse recurso é bastante interessante e aumenta a capacidade de atendimento aos usuários, pois o tempo de acesso a disco também é muito menor do que o "tempo de resposta" dos usuários. Além desta utilidade, este conceito de memória virtual facilita também a vida dos programadores que, basicamente, passam a não se preocupar com a quantidade de memória física disponível; isso passa a ser mais um problema do SO.

6.3. Gerência de arquivos

A gerência de arquivos é um dos componentes mais visíveis de um sistema operacional. Os computadores podem armazenar informações em vários tipos diferentes de meios físicos: fita magnética, disco magnético e etc. Cada um desses meios possui suas próprias características e organização física. Cada meio é controlado por um dispositivo, como uma unidade de disco, que também têm suas características exclusivas. Essas propriedades incluem: velocidade de acesso, capacidade, taxa de transferência de dados e método de acesso (sequencial ou aleatório).

O sistema operacional mapeia os arquivos nos meios físicos e acessa esses

arquivos através dos dispositivos de armazenamento.

O conceito de arquivo é bastante geral. Um arquivo é uma coleção de informações relacionadas definidas por seu criador. Geralmente os arquivos representam dados nos mais diversos formatos e, às vezes, programas (fonte e objeto). Os arquivos de dados podem ser numéricos ou alfanuméricos, representando dados escritos ou figuras, músicas e animações. Além disso, podem ter forma livre (por exemplo, arquivos de texto) ou podem ter uma formatação rígida (por exemplo, campos fixos como em planilhas ou bancos de dados). Um arquivo consiste em uma sequência de *bits*, *bytes*, linhas ou registros cujos significados são definidos por seus criadores. Uma consideração importante no projeto de um sistema de arquivos, e de todo o sistema operacional, é se o sistema deverá reconhecer e oferecer suporte a todos os tipos de arquivos. Quando um sistema operacional reconhece o tipo de arquivo, ele poderá operar com o arquivo de forma razoável. Uma técnica comum para implementar os tipos de arquivo é incluir o tipo como parte do nome do arquivo. O nome é dividido em duas partes – um nome e uma extensão, geralmente separada por um caractere de ponto. Veja no Quadro 6.1, os tipos de arquivos mais comuns. Dessa forma, o usuário e o sistema operacional podem saber imediatamente a partir do nome qual é o tipo de arquivo em questão.

Quadro 6.1: Tipos de arquivos comuns

Tipo de Arquivo	Extensão Comum	Função
Executável	exe, com, bin, ou nada	Programa de linguagem de máquina pronto para executar
Objeto	obj, o	Linguagem de máquina, compilado, sem linkedição
Código-fonte	c, cc, pas, java, asm, a	Código fonte em várias linguagens
Batch	bat, sh	Comandos para o interpretador de comandos
Texto	txt, doc	Dados textuais, documentos
Processador de textos	wpd, tex, doc, etc	Vários formatos de processador de textos
Biblioteca	lib, a, dll	Bibliotecas de rotinas para programadores
Impressão ou visualização	ps, dvi, gif	Arquivos ASCII ou binário em um formato para impressão ou visualização
Arquivo compactado	arc, zip, tar	Arquivos correlatos agrupados em um arquivo único, às vezes compactado, para fins de arquivamento ou armazenamento

Fonte: Adaptado de Silberschatz, Galvin e Gagne, 2000

O sistema operacional se encarrega de gerenciar as mídias de armazenamento em massa de arquivos, como discos, e os dispositivos que os controlam.

Os arquivos são normalmente organizados em diretórios para facilitar seu uso. E quando vários usuários têm acesso aos arquivos, pode ser desejável controlar quem poderá acessar os arquivos e de que forma poderá fazê-lo.

O sistema operacional é responsável pelas seguintes atividades em relação à gerência de arquivos:

a) Criar e excluir arquivos;

b) Criar e excluir diretórios;

c) Fornecer suporte para manipular arquivos e diretórios;

d) Mapear arquivos no armazenamento secundário;

e) Prover ferramentas de *backup*.

A maioria dos sistemas de computação usa discos (HD) como o principal meio de armazenamento para programas e dados. A maioria dos programas incluindo compiladores, montadores, rotinas de classificação, editores e formatadores, são armazenados em um disco até serem carregados na memória e utilizam o disco como origem e destino de seu processamento.

6.4. Gerência de dispositivos

O sistema operacional deve tornar as operações de entrada e saída (E/S) o mais simples possível para o usuário e suas aplicações. Denominamos de rotinas de entrada e saída ao conjunto de rotinas que o sistema operacional possui para possibilitar a comunicação de qualquer dispositivo que possa ser conectado ao computador. Essas rotinas fazem parte de um subsistema de E/S que permitem ao usuário realizar as operações de E/S sem se preocupar com detalhes, informações a respeito do dispositivo que está acessando.

Em gerência do sistema de E/S (entrada e saída) cabe ao sistema operacional ocultar dos usuários (programas e pessoas) as peculiaridades dos diferentes dispositivos de *hardware*. A gerência de dispositivos de entrada/saída é uma das principais e mais complexas funções de um sistema operacional. Sua implementação é estruturada por meio de camadas, em um modelo semelhante ao apresentado para o sistema operacional como um todo. As camadas de mais baixo nível escondem características dos dispositivos das camadas superiores, oferecendo uma interface simples e confiável para suas aplicações.

As operações de E/S devem ser realizadas através de chamadas ao sistema (*system calls*) que por sua vez chamam as rotinas de E/S do núcleo do sistema operacional. Desse modo, é possível que manipulemos arquivos em dispositivos de E/S, sem ter que alterar o código para cada dispositivo. Por exemplo: leitura de um arquivo armazenado em *pen drive*; não precisamos alterar o código do dispositivo a cada leitura ou a cada tipo de *pen drive* ou a cada máquina que acessamos.

As camadas são divididas em dois grupos, conforme Figura 6.2. O primeiro visualiza os diversos tipos de dispositivos do sistema de um modo único (a), ou seja, essas camadas trabalham de forma independente da configuração do dispositivo. Oferecem serviços de gerenciamento controlado pelo sistema operacional, enquanto o segundo é específico para cada dispositivo, necessitando a instalação e configuração do *hardware*, identificando e controlando cada dispositivo (b). Observem que grande parte das camadas trabalham de forma independente do dispositivo físico instalado.

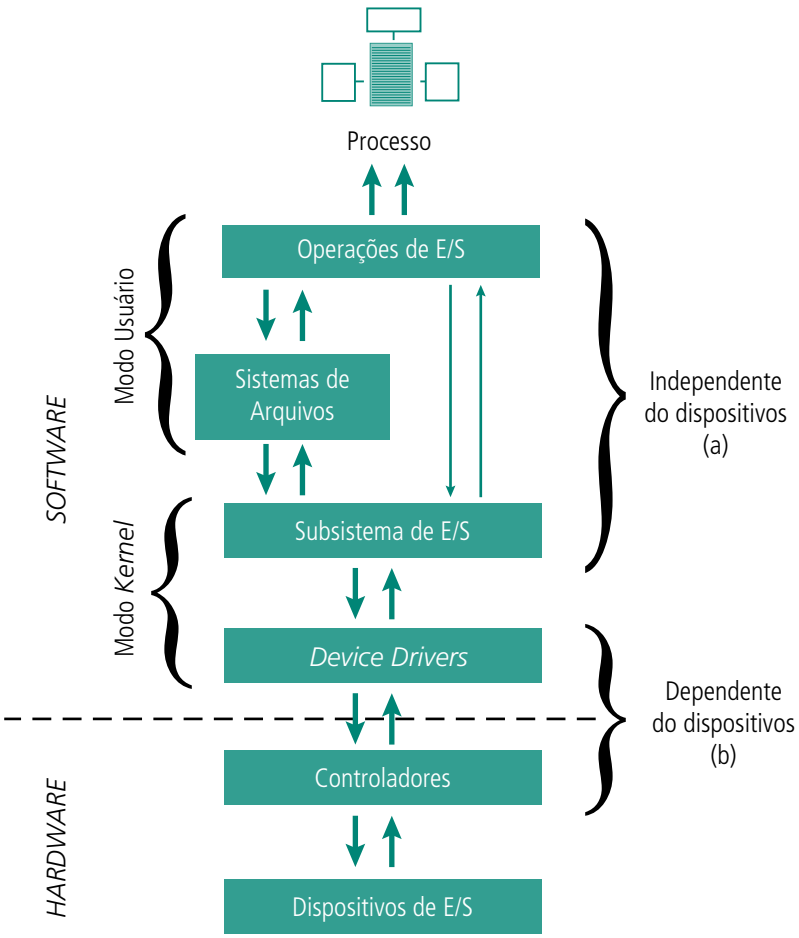

Figura 6.2: Gerência de dispositivos
Fonte: Adaptado de Machado, 2000

Resumo

Vimos nesta aula as principais funções de gerência do sistema operacional. Aprendemos alguns conceitos importantes sobre processos, memória primária e secundária, e dispositivos. Analisamos o funcionamento e a estrutura básica dos subsistemas principais de um sistema operacional. Podemos perceber o quão complexo é implementar um sistema operacional enquanto um *software* dividido em camadas que interagem entre si.

Atividades de aprendizagem

1. Quais recursos são necessários para a criação de um processo?

2. Qual a diferença básica entre um processo e um programa?

3. Por que, muitas vezes, "quebrar" um programa em mais de um processo pode ser vantajoso?

4. Quais as principais atividades da gerência de processos?

5. Pesquise sobre plataformas *multicore* e como elas podem colaborar com o desempenho de um computador em termos de processamento.

6. Pesquise sobre a transferência de dados via DMA.

7. O que vem a ser a técnica de *Swapping*?

8. Qual a diferença básica entre arquivos, diretórios e partições?

9. Associe a modularidade de um subsistema de Entrada e Saída com os *Drivers* de Dispositivos.

Referências

DEITEL, H.M. **Sistemas Operacionais**. 3ª.ed. Tradução de Arlete Simille Marques. São Paulo: Pearson Prentice Hall, 2005.

MACHADO, Francis Berenger. **Arquitetura de Sistemas Operacionais**. Rio de Janeiro: Editora LTC, 2004.

SILBERSCHATZ, A. & GAGNE, G. & GALVIN, P. B. **Fundamentos de Sistemas Operacionais**. Tradução de Adriana Cashin Rieche. Rio de Janeiro, 2004.

TANENBAUM, A.S. **Sistemas Operacionais Modernos**. 2ª.ed. Tradução de Ronaldo A. L Gonçalves. São Paulo, 2009.

Currículo do professor-autor

Mestre em Informática e Cientista da Computação pela UFES. Professor do Instituto Federal do Espírito Santo desde 2009, tendo ministrado disciplinas de sistemas operacionais e redes para os cursos Técnico em Informática e Tecnólogo em Redes de Computadores, presencial e à distância. Experiência de 7 anos em docência e 10 anos na área de informática em geral (programação de sistemas, coordenação de equipes e gerência de projetos).

e-Tec Brasil
Escola Técnica Aberta do Brasil

ISBN: