

SISTEMAS OPERACIONAIS

AUTOR

RICARDO BALIEIRO



SISTEMAS OPERACIONAIS

AUTOR DO ORIGINAL
RICARDO BALIEIRO

1^a EDIÇÃO
SESES
RIO DE JANEIRO 2015



Conselho editorial FERNANDO FUKUDA, SIMONE MARKENSON, JEFERSON FERREIRA FAGUNDES

Autor do original RICARDO BALIEIRO

Projeto editorial ROBERTO PAES

Coordenação de produção RODRIGO AZEVEDO DE OLIVEIRA

Projeto gráfico PAULO VITOR BASTOS

Diagramação FABRICO

Revisão linguística ADERBAL TORRES BEZERRA

Imagen de capa NOME DO AUTOR — SHUTTERSTOCK

Todos os direitos reservados. Nenhuma parte desta obra pode ser reproduzida ou transmitida por quaisquer meios (eletrônico ou mecânico, incluindo fotocópia e gravação) ou arquivada em qualquer sistema ou banco de dados sem permissão escrita da Editora. Copyright SESES, 2015.

Dados Internacionais de Catalogação na Publicação (cip)

B186s Balieiro, Ricardo

Sistemas operacionais / Ricardo Balieiro.

Rio de Janeiro : SESES, 2015.

144 p. : il.

ISBN 978-85-5548-085-0

1. Sistemas multiprogramáveis. 2. Gerenciamento do processador.
3. Gerenciamento de memória. 4. Gerenciamento de dispositivos de entrada e saída.I. SESES. II. Estácio.

CDD 005.43

Diretoria de Ensino — Fábrica de Conhecimento
Rua do Bispo, 83, bloco F, Campus João Uchôa
Rio Comprido — Rio de Janeiro — RJ — CEP 20261-063

Sumário

Prefácio	7
1. Introdução à Sistemas Operacionais	10
Conceitos Fundamentais de Sistemas Operacionais	11
Evolução Histórica dos Sistemas Operacionais	15
Classificação de Sistemas Operacionais	20
Interrupções	27
Conceitos de concorrência	28
Estruturas dos Sistemas Operacionais.	29
2. Processos	36
Conceito de Processo	36
Estados de um processo	45
Threads	47
Comunicação entre Processos	49
Sincronização entre Processos.	52
3. Gerênciа de Processador	64
Fundamentos	64
Critérios de Escalonamento	67
Escalonamento primeiro a entrar primeiro a sair (FIFO – First - IN - First - OUT)	68

Escalonamento por job mais curto primeiro (SJF – Shortest - Job - First)	71
Escalonamento circular (Round Robin)	72
Escalonamento por Prioridade	77
Escalonamento por Múltiplas filas com realimentação	84
Cálculo estimado de tempo de resposta.	86
4. Gerência de Memória	92
Funções	92
Estruturas de memória	94
Espaço de Endereçamento Físico e Lógico	96
Estratégias de alocação	97
Memória Virtual.	111
5. Gerência de Entrada e Saída	120
Introdução	121
Componentes de hardware de ENTRADA E SAÍDA	122
Componentes de Software de ENTRADA E SAÍDA	126
Sistema de Arquivos	131
Conceitos de Arquivos e Diretórios	132
Métodos alocação	138
Gerência de espaços livres	142
Proteção de acesso	143

Prefácio

Prezados(as) alunos(as)

Não podemos imaginar hoje o mundo sem o computador. Da mesma forma, não podemos imaginar o computador sem o sistema operacional. Ligamos o computador e o sistema operacional entra em cena, permitindo que acessemos nossos programas e arquivos para os mais variados fins, para o trabalho, a diversão, se relacionar através das redes sociais, internet, etc. Os sistemas operacionais nos oferecem uma maneira fácil de efetuarmos nossas tarefas através de suas interfaces gráficas e atraentes. Mas nem sempre foi assim. Houve um tempo em que os computadores eram configurados através do próprio hardware.

É importante que tenhamos uma visão detalhada dos mecanismos envolvidos em um sistema operacional moderno para que possamos utilizá-lo melhor e de forma mais eficiente. Conhecer sua evolução como também todos os problemas enfrentados e soluções adotadas, permite que possamos avançar mais na criação de novos sistemas sem ser barrados pelos problemas já conhecidos. Aprender sobre sistemas operacionais ajuda-nos a entendermos como são gerenciados os nossos arquivos, diretórios, processador, memória, periféricos etc., e assim, conseguirmos um maior desempenho do sistema. Estudar como os sistemas operacionais controlam e buscam as informações tão rapidamente, como controla acesso, sincronização entre diversos processos em execução, permite que tenhamos sólidos conceitos, que podemos utilizar em diversas áreas de desenvolvimento e gerenciamento de sistemas.

Diante de todos os aspectos acima descritos, acreditamos que com o estudo atencioso do material aqui presente, você com certeza será um profissional em destaque no mercado de trabalho!

Bons estudos e sucesso!

1

Introdução à Sistemas Operacionais

1 Introdução à Sistemas Operacionais

Para podermos aprofundar em todos os aspectos envolvidos em um sistema operacional, necessitamos criar uma base sólida de conhecimentos básicos que serão primordiais para a sequência dos próximos capítulos. Diante disto, estudaremos a evolução e os diversos tipos de sistemas operacionais, como também suas principais funções e estrutura.



OBJETIVOS

- Conhecer os conceitos fundamentais contidos nos sistemas operacionais.
 - Estudar os aspectos que impulsionaram a evolução dos sistemas operacionais.
 - Discutir sobre os diversos tipos de sistemas operacionais atuais.
-



REFLEXÃO

Você se lembra dos sistemas operacionais que utilizou nos últimos anos? Consegue imaginar um computador sem um sistema operacional? E o seu celular ou tablet, será que possui um sistema operacional?!

1.1 Conceitos Fundamentais de Sistemas Operacionais

A tecnologia tem avançado muito rapidamente em todas as áreas do conhecimento e novos equipamentos são lançados a todo o momento para os mais variados fins. Exemplo disso, além dos computadores, são os *tablets*, *smartphone*, *smartwatch*, vídeo *games*, TVs e a nossa porta os *google-glass*.

Independente do tipo de equipamento, formato ou tamanho, todos têm um ponto em comum, necessitam de um sistema operacional para funcionarem.



Figura 1 – Sistemas operacionais em diversos equipamentos.

Disponível em : <www.extremetech.com>



Figura 2 – Smartwatch e Google-Glass.

1.1.1 Definição

Segundo Silberschatz et al. (2004), um sistema operacional é um programa que efetua o gerenciamento dos componentes físicos do computador (*hardware*), como também uma base para os programas aplicativos. Além disso, atua como intermediário entre o usuário e o *hardware* do computador.

Existem muitos tipos de sistemas operacionais, cujo aspecto varia de acordo com o tipo de funções e *hardware* ao qual será utilizado. Se observarmos os *tablets*, *smartphone*, *smartwatch* teremos um sistema operacional projetado para facilitar a interface do usuário com os programas a serem executados. Nos *desktops* (computadores pessoais) esta otimização são voltadas para aplicações comerciais, jogos, etc. Em ambientes corporativos, onde a utilização de computadores de grande porte são mais requisitados, os sistemas operacionais têm um projeto mais voltado para a otimização de *hardware*. Na indústria, cujos equipamentos necessitam de alta precisão e confiabilidade, utilizam-se sistemas operacionais em tempo real que permitem a resposta a um evento ser feito em um espaço de tempo determinado.

1.1.2 Objetivo

A função do sistema operacional é permitir uma interface homem máquina mais amigável com o usuário, isto porque encapsula as complexas rotinas de acesso a recursos de *hardware*, tais como, interface de programação e gerenciamento de recursos.

Tarefas simples como salvar um pequeno texto, sem o sistema operacional seria algo que demandaria um grande esforço e extensos conhecimentos da arquitetura interna do computador. Esta pequena tarefa necessitaria as seguintes etapas simplificadas:

1. Localizar os dados do arquivo na memória.
1. Obter o nome do arquivo e local de gravação.
2. Validar os dados do item 2.
3. Posicionar o cabeçote de leitura e gravação no cilindro correto.
4. Posicionar o cabeçote de leitura e gravação na trilha correta.
5. Posicionar o cabeçote de leitura e gravação no setor correto.
6. Salvar o arquivo.

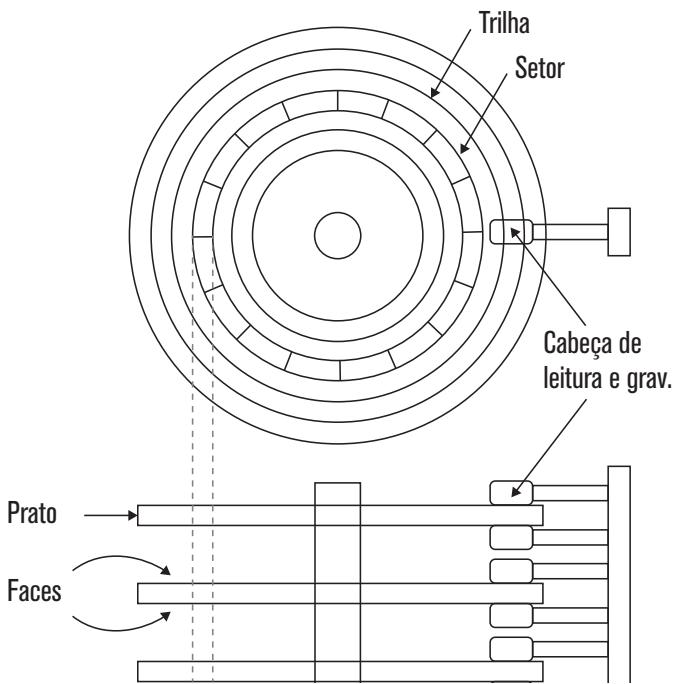


Figura 3 – Disco Rígido.

De acordo com Tanenbaum e Woodhyll (1999) e Silberschatz et al. (2004), os sistemas operacionais podem ser conceituados de duas formas: *topdown* e *bottom-up*.

No modo *topdown* (“de-cima-a-baixo”) que é o modo de visão do usuário, o sistema operacional é um *software* que permite a interação entre o *hardware* e os programas aplicativos. Assim para o usuário, o sistema operacional fornece:

- Acesso ao sistema.
- Possibilidade de criar e gerenciar arquivos e diretórios.
- Ambiente para execução de programas.
- Acesso aos dispositivos de entrada e saída.
- Acesso ao conteúdo de arquivos.
- Detecção de erros.

Já no modo *bottom-up* (“de-baixo-a-cima”), é considerado um gerenciador de recursos. Isto porque controla a utilização dos recursos de *hardware* pelas

aplicações como também quais e quando as aplicações podem ser executadas. Podemos citar como recursos:

- Tempo de CPU.
- Espaço em memória.
- Espaço em disco.
- Acesso aos dispositivos de comunicação.
- Bibliotecas de *software*.

Para que isso fique mais claro, podemos analisar o sistema operacional dividindo-o em quatro componentes: usuários, programas aplicativos, sistema operacional e o *hardware*.

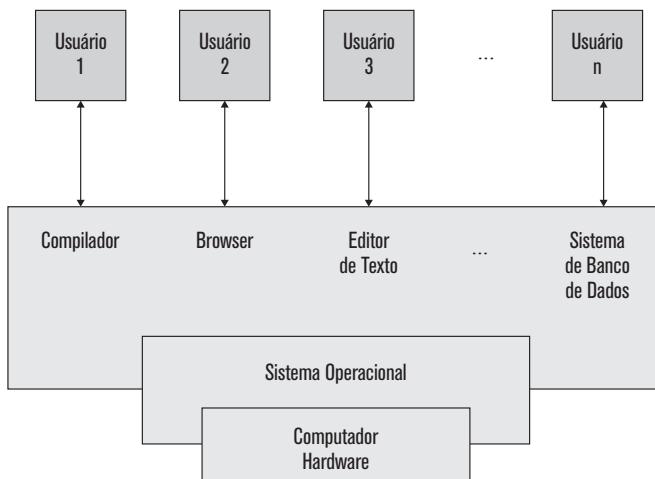


Figura 4 – Sistema computacional.

Fonte: Oliveira et al. (2010).

Nós usuários utilizamos os programas aplicativos (editor de texto, planilhas eletrônicas, navegador web, etc) para diversas tarefas, tanto para trabalho como para lazer. Os aplicativos por sua vez dependem dos recursos de *hardware* para executar suas funções. O *hardware* é a parte mais básica dos componentes, sendo composto pela unidade central de processamento (CPU - *Central Processing Unit*), memórias (RAM e ROM), dispositivos periféricos (teclado, monitor, mouse, impressora, etc.) e unidades de armazenamento (disco rígido).



ATENÇÃO

O sistema operacional é considerado por muitas pessoas como sendo um programa que fica executando o tempo todo no computador, enquanto o restante é considerado como programas aplicativos (TANENBAUM; WOODHYLL, 1999).

Neste ponto entra o sistema operacional fornecendo um ambiente de integração que possibilite a execução dos programas e o controle e coordenação da utilização dos recursos de *hardware* pelos aplicativos. Silberschatz et al. (2004) considera o sistema operacional como uma alocador de recursos. É de responsabilidade do sistema operacional resolver conflitos na utilização destes recursos. Por exemplo, imagine três computadores em rede e os usuários destes computadores resolvem mandar ao mesmo tempo um texto para uma impressora compartilhada na rede. Qual texto o sistema operacional deve imprimir primeiro? Imprimir parte do texto do usuário 1, em seguida, parte do usuário 2 e assim por diante? Como o sistema operacional resolve este conflito de alocação de recursos? A solução adotada neste caso é, primeiramente o sistema operacional armazenar os arquivos enviados em uma fila de impressão no disco rígido. Em seguida, imprime o primeiro arquivo da fila e em seguida o próximo até não haver mais nenhum para impressão.

Este é um pequeno exemplo do que acontece a todo o momento em um sistema operacional, que é procurar gerenciar a utilização dos recursos da forma mais eficiente possível, procurar minimizar as falhas, efetuar controle de acesso para que os usuários possam acessar apenas os recursos que lhes foram autorizados, entre outros. Em resumo, os objetivos fundamentais dos sistemas operacionais são executar os aplicativos dos usuários e facilitar a resolução de seus problemas.

1.2 Evolução Histórica dos Sistemas Operacionais

Para estudarmos a evolução histórica dos sistemas operacionais devemos estudar a evolução dos computadores, isto porque os dois estão diretamente ligados. Segundo Machado e Maia (2007) a máquina de cálculos de equações polinomiais conhecida como Máquina Analítica (*Analytical Engine*) inventada pelo matemático inglês Charles Babbage em 1822 é o que mais se assemelha a um computador atual. Isto porque possuía os conceitos de unidade central de processamento, memória, unidade de controle e dispositivos de entrada/saída.

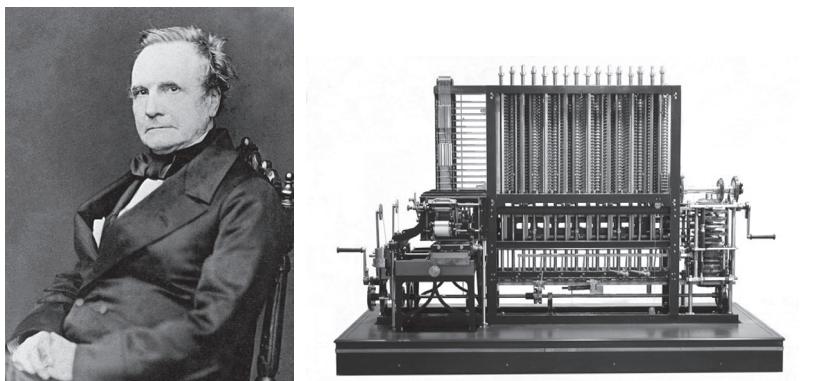


Figura 5 – Charles Babbage - Máquina Analítica.



ATENÇÃO

A Máquina Analítica de Charles Babbage não possuía sistema operacional, apenas as sequências de instruções executadas pela Máquina Analítica que era de responsabilidade daquela considerada a primeira programadora da história, Augusta Ada Byron discípula de Babbage (SILBERSCHATZ et al., 2004).

1.2.1 Década de 1940: válvulas e painéis com plugs

Durante a segunda guerra mundial houve um esforço muito grande no desenvolvimento de máquinas que pudessem agilizar os procedimentos manuais efetuados na área militar, principalmente para cálculos balísticos. Assim, em 1943 começou a ser desenvolvido o ENIAC (*Electronic Numerical Integrator Analyzer and Computer* – Computador Integrador Numérico Eletrônico), primeiro computador eletrônico de grande porte idealizado pelos cientistas norte-americanos John Eckert e John Mauchly, da Electronic Control Company.

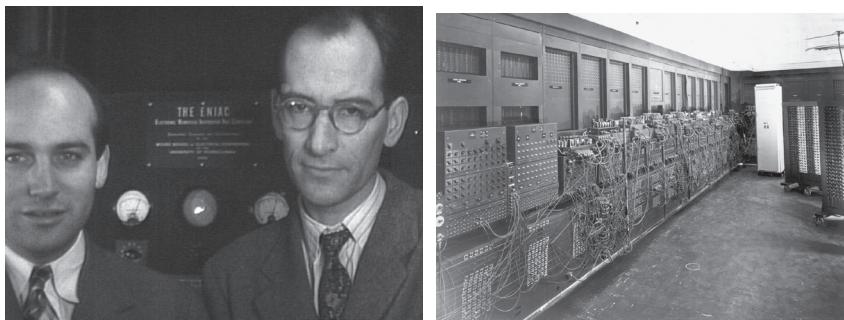


Figura 6 – John Eckert, John Mauchly e o ENIAC

Disponível em: <<https://sites.google.com/site/worldcyber> / - Dr. J. Presper Eckert, Dr. John Mauchly ENIAC of 1946>.

A arquitetura dos modernos computadores que temos nos dias de hoje, conhecida como *Arquitetura von Neumann*, foi idealizado por um dos consultores do projeto ENIAC, o professor John von Neumann. Segundo a arquitetura von Neumann, uma máquina digital (computador) teria os seguintes componentes:

1. Memória: capaz de armazenar em um mesmo espaço dados e instruções dos programas;
2. Unidade de processamento (CPU – *Central Processing Unit*): responsável por executar as instruções dos programas armazenados na memória;
3. Unidade de controle (CU – *Control Unit*): responsável pela interpretação das instruções de programa, como também, controlar a sequência de tempo das atividades necessárias para sua execução;
4. Unidade aritmética e lógica (ALU – *Arithmetical and Logical Unit*): responsável pela execução das operações aritméticas (somas, subtrações, etc.) e lógicas (comparações, AND, OR, etc.) contidas nas instruções dos programas;
5. Registradores: pequenas áreas de memória localizada na CPU para armazenamento temporário de dados dos programas que estão sendo executados, resultados de instruções, estado interno da CPU, etc.
6. Dispositivos de entrada e saída: responsável por traduzir os dados inseridos pelo usuário no computador (ex.: teclado, cartões perfurados, fitas ou discos magnéticas etc.) para a memória como também traduzir da memória para um formato externo (ex.: fitas ou discos magnéticos, telas de vídeo, etc.)

Neste período não havia ainda os conceitos de sistema operacionais, desta forma, era responsabilidade do usuário operar, programar e efetuar a manutenção do computador durante o período que o equipamento ficava a sua disposição. A programação, composta basicamente por cálculos numéricos, era feita diretamente nos painéis do computador.

1.2.2 Década de 1950: transistores e sistemas batch

Na década de 1950 surgiram os transistores que permitiram uma grande diminuição do tamanho dos computadores, que anteriormente eram feitos a válvula, o que proporcionou um aumento do poder de processamento dos equipamentos. Grandes empresas e corporações começaram a adquirir computadores, conhecidos como *Mainframes*. Os Mainframes permitiram que houvesse uma separação entre os operadores, programadores e técnicos de manutenção.

A programação, feita através de cartões perfurados, eram entregues ao operador do computador para que fossem processados. Os programas, também denominados *Jobs*, eram lidos por uma leitora e gravados em uma fita de entrada. O computador então lia a fita e executava um programa de cada vez. O resultado do processamento era então gravado numa fita de saída. Esta técnica, onde são processados um conjunto de programas, ficou conhecido como *processamento batch*.

Em 1953 os usuários do computador IBM 701, do Centro de Pesquisas da General Motors, desenvolveram o primeiro sistema operacional, chamado de *Monitor*. O Monitor, chamado assim pela sua simplicidade, tinha como objetivo automatizar as tarefas manuais executadas na época.

Neste período surgiram as primeiras linguagens de programação de alto nível, tais como FORTRAM, ALGOL E COBOL. Houve então um grande avanço no desenvolvimento e manutenção dos programas que não mais tinham uma relação direta com o *hardware* dos computadores. Consequentemente, os sistemas operacionais evoluíram para atender as demandas das linguagens de programação e assim facilitar o trabalho de codificar, executar e depurar os programas.

1.2.3 Década de 1960 - 1980: circuitos integrados e multiprogramação

Com o surgimento dos circuitos integrados os computadores tiveram uma redução de custo de aquisição o que proporcionou sua viabilização nas empresas.

Várias inovações foram implementadas nos sistemas operacionais, tais como multiprogramação, multiprocessamento, *time-sharing* e memória virtual.

A década de 1970 foi marcada com a miniaturização dos componentes (*chips*) baseadas nas tecnologias de Integração em Larga Escala (*Lage Scale Integration* – LSI) e a Integração em Muito Larga Escala (*Very Lage Scale Integration* – VLSI), o surgimento das primeiras redes de computadores, além do desenvolvimento de novas linguagens de programação de alto nível.

Na década de 1980 os fabricantes de computadores passam a produzir microcomputadores utilizando microprocessadores. A IBM então cria a filosofia de computadores pessoais o que impulsionou a evolução dos sistemas operacionais. Os microcomputadores da época possuíam baixa capacidade de armazenamento e as versões iniciais dos sistemas operacionais eram *monousuário/monotarefa*. Os sistemas operacionais evoluíram para *monousuário/multitarefa* com a incorporação de discos rígidos e outros periféricos nos microcomputadores.

Em meados da década de 1980 crescem as redes de computadores pessoais utilizando sistemas operacionais para rede e sistemas operacionais distribuídos. Os sistemas operacionais para rede permitem que os usuários se conectem a máquinas remotas e utilizem recursos compartilhados. O usuário tem plena consciência da existência de vários computadores conectados. Já no sistema operacional distribuído os usuários não têm consciência onde estão armazenados seus arquivos ou onde estão sendo executados seus programas. Apesar do sistema operacional distribuído ser composto de múltiplos computadores conectados, as operações são executadas de tal forma que o usuário tem impressão de estar trabalhando um único computador.

1.2.4 Década de 1990 - 2000: Windows e Linux

A rede mundial de computadores, a Internet, surge na década de 1990 e com a decorrência de sua rápida evolução, força os sistemas operacionais a oferecerem suporte ao protocolo TCP/IP utilizado na Internet. Nesta mesma década os sistemas operacionais como o Windows da Microsoft e o Unix, passam a adotar as interfaces gráficas.

Surge o Linux em 1991 através do desenvolvimento do finlandês Linus Torvalds e de trabalhos colaborativos de diversos programadores. A forma colaborativa e os avanços da Internet possibilitaram que outros *softwares* abertos, como o já citado sistema operacional Linux, o banco de dados MySQL, o servidores *web*

Apache entre outros, pudessem ser desenvolvidos e distribuídos sem custos aos seus usuários. Para Machado e Maia (2007) a década de 2000 aponta para uma mudança radical no desenvolvimento de computadores frente as exigências cada vez maiores de equipamentos mais eficientes e com maior poder de processamento. Os sistemas operacionais voltados para as novas arquiteturas de processadores *64 bits*, são dotados de interfaces usuário-máquina que exploram cada vez mais imagens, sons e linguagens naturais para proporcionar ao usuário uma interatividade com o computador mais intuitiva, natural e simples.

CONEXÃO

Para entender melhor as diferenças entre Windows e Linux, veja o link: <http://www.guiadopc.com.br/artigos/3394/as-10-principais-diferencias-entre-o-windows-e-o-linux.html>. O autor faz uma comparação entre as 10 principais diferenças entre o Windows e o Linux. Leia e confira!

1.3 Classificação de Sistemas Operacionais

Com a evolução dos computadores, houve a necessidade da evolução dos sistemas operacionais para suportar os novos recursos de *hardware* e das aplicações por ele suportado. Neste sentido, os sistemas operacionais podem ser classificados conforme o seu processamento, tarefas, usuários e interface.

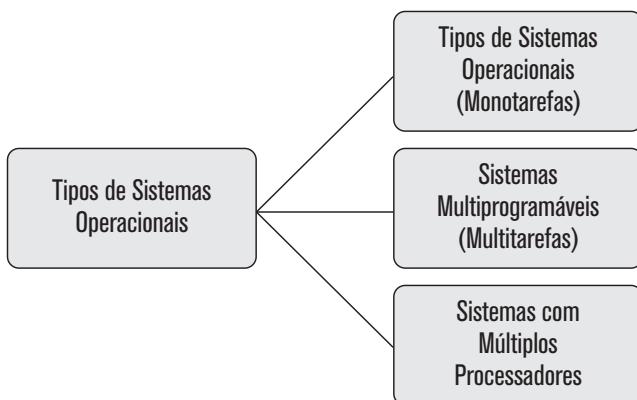


Figura 7 – Tipos de sistemas operacionais.

1.3.1 Sistema Monoprogramáveis/Monotarefas

Os sistemas operacionais *monoprogramáveis* foram os primeiros sistemas a serem utilizados. Este tipo de sistema operacional tem a característica de permitir a execução de um único programa por vez.



Figura 8 – Sistemas Monoprogramáveis/Monotarefa.

Assim o processador, memória e os periféricos do computador ficam dedicados exclusivamente para um único programa em execução. Os sistemas operacionais monoprogramáveis também são conhecidos como sistemas *monotarefas*. Além de executar um programa por vez, os recursos do sistema são alocados para uma tarefa por vez. Nesse sentido quando o programa está esperando a entrada de um dado pelo usuário, o processador fica parado, sem qualquer tipo de processamento.

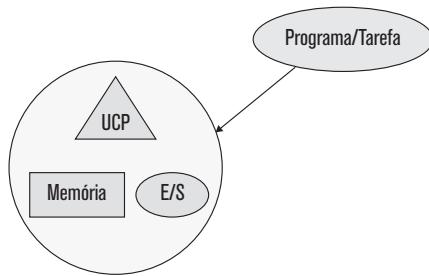


Figura 9 – Sistemas Monoprogramáveis/Monotarefa.

Fonte: Machado e Maia (2007).

Os sistemas operacionais monoprogramáveis/monotarefa surgiram para serem utilizados nos primeiros computadores, os Mainframes e posteriormente aos computadores pessoais utilizados por um usuário por vez. O MS_DOS, sistema operacional muito utilizado nas décadas de 1980 e 1990 é um exemplo de sistemas operacionais monoprogramáveis/monotarefa. Por não compartilhar recursos, como processador, memória e dispositivos de entrada/saída, este tipo de sistema operacional é de simples implementação.

1.3.2 Sistema Multiprogramáveis/Multitarefa

Os sistemas operacionais *multiprogramáveis/multitarefa* têm como características permitir o compartilhamento dos recursos do computador com vários usuários e aplicações.

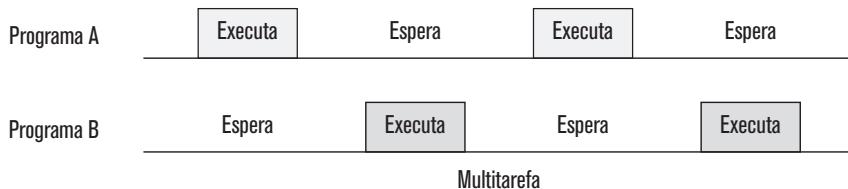


Figura 10 – Sistemas multiprogramáveis/multitarefa.

Para que isso aconteça, os programas são carregados em memória e a utilização do processador é efetuado por apenas um programa, ficando os demais enfileirados, aguardando a sua vez. Neste ambiente cada programa (processo) recebe um tempo para a utilização do processador. No final do tempo, um novo programa passa a utilizar o processador. Para o usuário final a impressão é de que vários programas estão sendo executados simultaneamente. Este tipo de alternância de processos é denominado de *concorrência*. Este acesso concorrente aos recursos disponível é gerenciado pelo sistema operacional de forma ordenada e protegida.

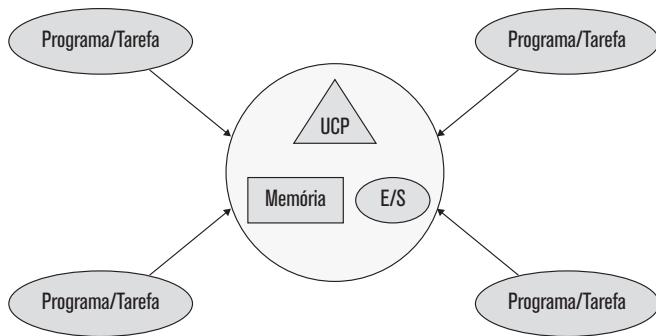


Figura 11 – Sistemas multiprogramáveis/multitarefa.

Fonte: Machado e Maia (2007).

A implementação dos sistemas operacionais multiprogramáveis são mais complexas, mas apresentam uma maior eficiência que os monoprogramáveis e uma sensível redução de custo em função da possibilidade de compartilhar os recursos disponíveis de *hardware* entre diferentes aplicações.



CONEXÃO

Leia um pouco mais sobre Sistemas multiprogramáveis/multitarefa em: <<http://prezi.com/atwpl2cicalv/sistemas-multiprogramaveismultitarefa/>>.

1.3.3 Classificação de Processamento

De acordo como as aplicações são gerenciadas, os sistemas multiprogramáveis podem ser classificados como: Batch, sistemas de tempo compartilhado ou sistema de tempo real.

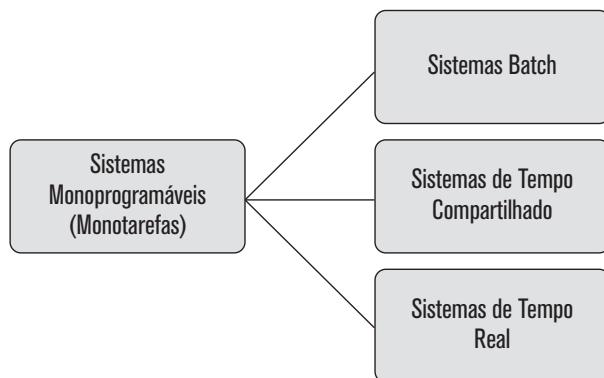


Figura 12 – Classificação de sistemas multiprogramáveis.

1.3.3.1 Sistemas Operacionais Batch (lote)

Os primeiros sistemas multiprogramáveis foram os Sistemas Batch. Neste sistema, a programação era feita com cartões perfurados. Os cartões eram lidos e armazenados em disco ou fita, que em seguida eram utilizados para carregar os dados para processamento. Uma vez executados, o resultado do processamento era armazenado em disco ou fita de saída. Apesar de apresentarem um tempo de resposta muito longos eram considerados altamente eficientes.

1.3.3.2 Sistemas operacionais de tempo compartilhado (*time-sharing*)

Os sistemas operacionais de tempo compartilhado (*time-sharing*) são sistemas onde o tempo do processador é dividido em pequenas partes (*time-slice*) permitindo a cada programa utilizar uma destas partes para a sua execução.

O controle do sistema operacional é feito em um computador central. Os usuários interagem com o computador central através de comandos digitados em terminais compostos por teclado, monitor e *mouse*. Grande parte das atuais aplicações comerciais é processada em sistemas de tempo compartilhado.

1.3.3.3 Sistemas de tempo real

Nos sistemas de tempo real (*real-time*) o tempo do processador é distribuição de acordo com a prioridade de cada aplicação. Devido à natureza dos sistemas de tempo real, cujas aplicações são de controle de processo, a resposta na execução de uma tarefa deve estar dentro de limites rígidos de tempo. O conceito de tempo compartilhado não é aplicado neste tipo de sistema. Assim o sistema operacional deve garantir a disponibilidade de todos os recursos necessários para execução de um programa até que este termine ou que surja um de maior prioridade. O sistema operacional não define as prioridades de execução de um programa, esta definição é feita pela própria aplicação. Para Machado e Maia (2007) o sistema de tempo real é empregado em aplicações de controle e monitoramento de processos onde o tempo de processamento é fator crítico, como em casos de usinas termoelétricas ou nucleares, refinarias de petróleo, tráfego aéreo, etc.

1.3.4 Sistemas com múltiplos processadores

Os sistemas com múltiplos processadores têm como característica possuir mais de um processador interligado e trabalhando em conjunto. Os processadores podem estar num mesmo computador ou espalhados fisicamente em uma rede de computadores. Dessa forma diversos programas podem ser executados simultaneamente, ou um programa pode ser dividido em partes e executados em processadores diferentes numa mesma máquina ou em várias. Devido a sua capacidade de ampliar consideravelmente o poder de processamento a medida que novos processadores são adicionados, este tipo de sistema é muito empregado em aplicações de processamento de imagens, simulações, prospecção de petróleo, processamento científico entre outros.

No desenvolvimento de sistemas operacionais com múltiplos processadores é de vital importância que se conheça a forma de comunicação entre os processadores, o grau de compartilhamento da memória principal e dos dispositivos de entrada e saída. Isto porque, estes fatores definem a classificação dos sistemas em: Fortemente acoplados e Fracamente acoplados.

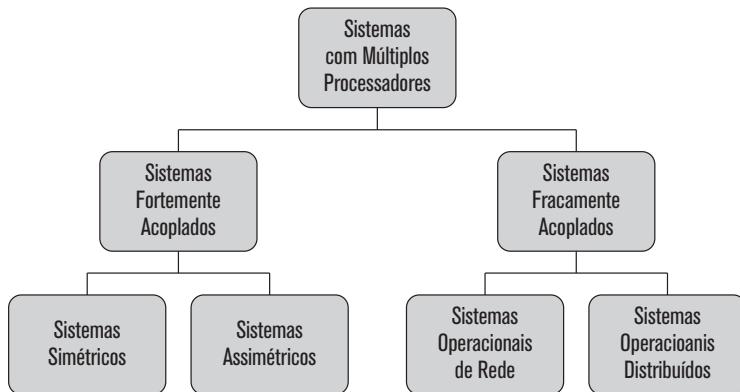


Figura 13 – Tipos de sistemas com múltiplos processadores.

1.3.4.1 Sistemas fortemente acoplados

Os sistemas fortemente acoplados, conhecidos também como multiprocessadores, têm como características possuir vários processadores compartilhando uma única memória física e dispositivos de entrada e saída e apenas um sistema operacional efetuando o gerenciamento. Segundo Silberschatz et al. (2004) os sistemas fortemente acoplados podem ser divididos em Multiprocessamento Simétrico (SMP – Symmetric Multiprocessing) e Multiprocessamento Assimétrico (NUMA – Non-Uniform Memory Access). Os sistemas multiprocessamento simétrico os processadores compartilham o mesmo tempo de memória. No multiprocessamento assimétrico, o tempo de acesso à memória pode variar dependendo da localização física dos processadores em relação à memória.

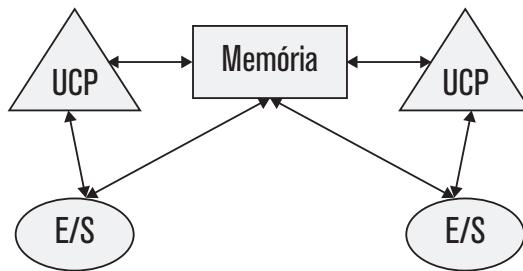


Figura 14 – Sistema fortemente acoplados.

Fonte: Machado e Maia (2007).

1.3.4.2 Sistemas fracamente acoplados

Os sistemas fracamente acoplados, conhecidos também como *multicomputadores*, conectam vários sistemas computacionais (computadores) através de linhas de comunicação. Os sistemas computacionais funcionam de forma independente, assim cada qual tem sua própria UCP, memória, dispositivos de entrada e saída e sistema operacional.

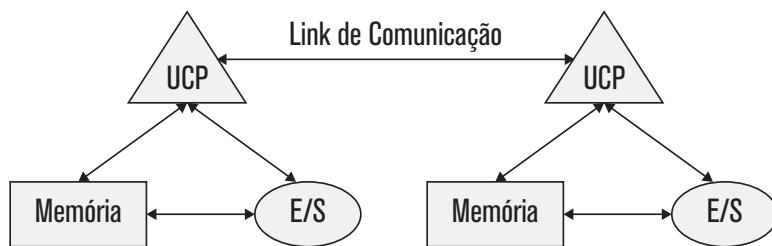


Figura 15 – Sistemas fracamente acoplados.

Fonte: Machado e Maia (2007).

Levando em consideração o grau de integração dos computadores da rede, os sistemas fracamente acoplados podem ser classificados como sistemas operacionais de rede e sistemas distribuídos. Nos Sistemas Operacionais de Rede (SOR) os recursos de um computador, tais como diretórios, impressora, serviços, etc. podem ser compartilhados com outro computador da rede. Neste tipo de sistema o usuário tem pleno conhecimento dos computadores da rede, quais recursos e serviços que estão disponíveis em cada um. No sistema distribuído os recursos e serviços individuais disponíveis em cada computador e tratado com um único conjunto. Assim o usuário final tem a sensação de estar trabalhando em um

único sistema centralizado e não em uma rede de computadores. Entre diversas vantagens oferecidas pelos Sistemas distribuídos destaca-se o balanceamento de carga, muitas vezes denominado de *clusters*. O balanceamento de carga permite que quando um programa é aceito para execução, o sistema escolhe o computador com menor carga de processamento para executá-lo.

1.4 Interrupções

Quando um programa entra em execução, suas instruções são executadas em sequência pelo processador. Pode ocorrer situações que force o processador lançar algum evento que cause o desvio da sequência original de execução do programa. Estes eventos são conhecidos como *interrupções* ou *exceções*.

Um evento é gerado pelo processador quando este recebe um sinal de algum dispositivo de *hardware* informando uma mudança de estado em algum dispositivo, como por exemplo, a conclusão de uma entrada\saída em disco ou da execução de instruções do próprio programa.

A Figura 16 mostra um diagrama do mecanismo de interrupção. O programa é composto de inúmeras instruções que devem ser executadas em uma determinada ordem pelo processador. Após executar cada instrução, o processador verifica se houve alguma ocorrência de interrupção. Caso tenha ocorrido, o processador suspende a execução da próxima instrução e desvia para uma rotina pré-definida para tratar o evento, chamada de *rotina de tratamento de interrupção*. Antes de desviar para a rotina, os dados dos registradores são salvos permitindo que após o tratamento da interrupção, o processador possa retornar ao código que estava sendo executado quando recebeu a interrupção.

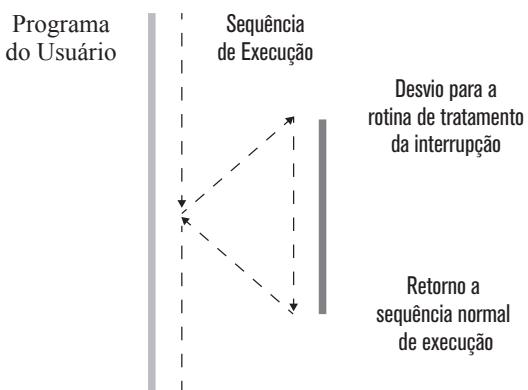


Figura 16 – Mecanismos de interrupção e exceção.

Exemplo de um tratamento de interrupção:

- está sendo executado um determinado programa pelo processador;
- um *pen-drive* é conectado a porta USB;
- a placa de dispositivo de entrada e saída envia uma interrupção para o processador;
- o processador para a execução do programa e desvia para um tratamento de interrupção;
- a rotina de tratamento é executada, atualizando as estruturas do sistema operacional e exibindo uma mensagem ao usuário informando da conexão do *pen-drive*;
- finalizando a rotina de tratamento da interrupção o processador retorna à execução do programa interrompido anteriormente.

A utilização de interrupção permitiu o desenvolvimento de concorrência nos computadores.

1.5 Conceitos de concorrência

Os primeiros sistemas operacionais, os monoprogramáveis, apresentavam uma arquitetura que limitava muito seu desempenho. Isto acontecia devido à utilização pouco eficiente dos recursos computacionais como processadores, memória, dispositivos de entrada e saída etc. Pouco eficiente no sentido de serem recursos de alto custo e ficarem ociosos em grande parte do tempo. O processador, por exemplo, ficava ocioso enquanto era efetuada a entrada de dados, por exemplo, a digitação do usuário (Figura 17). A memória, que permitia carregar apenas um programa por vez, não era ocupada totalmente, permanecendo grandes áreas livres sem utilização.



Figura 17 – Sistemas Monoprogramável

Com o surgimento dos sistemas operacionais multiprogramáveis, foi possível carregar vários programas na memória, concorrendo pela utilização do processa-

dor. Assim quando um programa solicita uma operação de entrada/saída, que normalmente são muito lentas comparadas com a velocidade de processamento de uma instrução, outro programa assume o uso do processador (Figura 18).

O controle do acesso concorrente a diversos recursos é implementado por mecanismos de proteção do sistema operacional para garantir a integridade dos programas e do próprio sistema operacional. Assim, podemos visualizar o sistema operacional como sendo um conjunto de rotinas que são executadas de forma concorrente e ordenada.

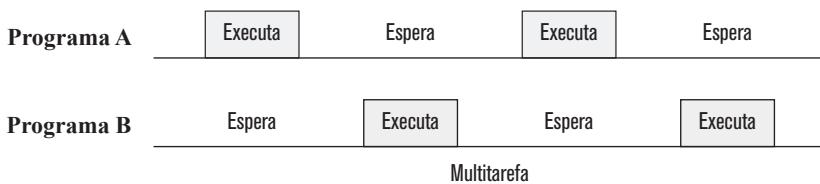


Figura 18 – Sistemas Multiprogramável.

1.6 Estruturas dos Sistemas Operacionais.

Os sistemas operacionais são diferentes dos demais programas que são executados sequencialmente, tendo início, meio e fim. A execução das rotinas está baseada em eventos relacionados às tarefas internas do sistema operacional e do *hardware*. Dessa forma o sistema operacional através de um conjunto de rotinas, oferece serviços aos usuários do sistema, aos programas que estão sendo executados e a outras rotinas do próprio sistema. O conjunto destas rotinas é chamado *Kernel* (cérebro) ou núcleo do sistema.

As principais funções do *Kernel* são:

- tratamento de interrupções e exceções;
- criação e eliminação de processos e *threads*;
- sincronização e comunicação entre processos e *threads*;
- escalonamento e controle dos processos e *threads*;
- gerência de memória;
- gerência de sistema de arquivos;
- gerência de dispositivos de E/S;
- suporte a redes locais e distribuídas;
- contabilização do uso do sistema;
- auditoria e segurança do sistema.

1.6.1 Modos de Acesso

Existe uma preocupação muito grande dos desenvolvedores de sistemas operacionais quanto ao acesso direto ao Kernel por usuário e seus aplicativos. Uma instrução indevida executada diretamente pode corromper o kernel comprometendo todo o sistema. Instruções deste tipo são denominadas instruções privilegiadas. A execução de uma destas instruções está vinculada aos chamados de modos de acesso. Os modos de acesso são:

- Modo usuário: não permite que o usuário ou aplicações executem instruções privilegiadas.
- Modo Kernel: permite acesso total as instruções privilegiadas, tanto pelo usuário quanto pelas aplicações.

1.6.2 System Calls

Para que o usuário ou alguma aplicação possa usufruir de algum serviço fornecido pelo *Kernel* deve acessá-lo através de um mecanismo conhecido como *System Calls*. O *System Calls* é um intermediário entre as aplicações do usuário e o sistema operacional. Usuários ou aplicações acessam os serviços do Kernel chamando uma das suas rotinas através de uma *System Call*. O serviço é processado, com bases nos parâmetros definidos na *System Call*, e retorna à aplicação os resultados obtidos.



ATIVIDADE

1. De acordo com o material, qual das opções abaixo pode ser classificada como sistema operacional?
 - a) Android e MS-Excel.
 - b) Windows e Internet Explorer.
 - c) Linux e Windows.
 - d) Android e MS-Word.
 - e) Linux e Broffice.
2. Quais as principais funções do kernel?

3. Para que uma aplicação possa utilizar os serviços disponíveis pelo Kernel deve efetuar a comunicação com o Kernel por meio de:
 - a) Device Drivers
 - b) Batch
 - c) Scripting
 - d) System Calls
 - e) Shell
-



REFLEXÃO

Neste capítulo aprendemos sobre as principais características e funções do sistema operacional, o que nos auxilia a utilizá-lo de forma mais eficiente. Em seguida vimos a evolução dos sistemas monoprogramáveis para os multiprogramáveis o que nos permitiu estudar as diversas formas como o computador tem sido controlado ao longo dos anos. Vimos também os diversos tipos de sistemas operacionais, como os sistemas de tempo compartilhado, tempo real e com múltiplos processadores. Conceitos muito importantes quando há necessidade de escolher um sistema operacional para uma aplicação específica. Fechamos estudando conceitos como interrupções e concorrências, que são bases para muitos programas e não somente para o sistema operacional.



LEITURA

Para você avançar mais o seu nível de aprendizagem envolvendo os conceitos de sistemas operacionais e demais assuntos deste capítulo, consulte as sugestões de *links* abaixo:

MAZIERO, C. A. Sistemas Operacionais: Conceitos e Mecanismos. Disponível em: <<http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/so:so-cap01.pdf>>. Acesso em: set. 2014.



REFERÊNCIAS BIBLIOGRÁFICAS

CARL. Emopulse Smile smartwatch available on pre-order. Disponível em: <<http://www.kitguru.net/channel/generaltech/carl/emopulse-smile-smartwatch-available-on-pre-order/>>. Acesso em: set. 2014.

COMPUTER HISTORY ARCHIVES PROJECT. Disponível em: <<https://sites.google.com/site/worldcyber/>>. Acesso em: set. 2014.

DEITEL, H. M.; DEITEL, P. J.; CHOFFNES, D. R. Sistemas Operacionais. 3^a ed. São Paulo, Editora Prentice-Hall, 2005.

FERREIRA, A. L. Milhares de Imagens - Mude o fundo de tela do seu PC, Tablet ou Smartphone. Disponível em: <http://visualdicas.blogspot.com.br/2013/09/milhares-de-imagens-mude-o-fundo-de.html>. Acesso em: set. 2014.

HENNESSY, J. L.; PATTERSON, D. A. Arquitetura de Computadores: uma abordagem quantitativa. 3^a ed. Rio de Janeiro: Campus, 2003.

MACHADO, F. B.; MAIA, L. P. Arquitetura de sistemas operacionais. 4^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2007.

MATSUMOTO. Entenda o que é formatação de disco rígido. Disponível em: <http://www.socialbits.com.br/2012/11/entenda-o-que-e-formatacao-de-disco-rigido/>. Acesso em: set. 2014.

MAZIERO, C. A. Sistemas Operacionais: Conceitos e Mecanismos. Disponível em: <<http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/so:so-livro.pdf>>. Acesso em: set. 2014.

MEGATECH BRASIL. Disponível em: <<http://megatechbrasil.com/tecnologia/google-glass-ja-a-venda-nos-eua/>>. Acesso em: set. 2014.

OLIVEIRA, R. S.; CARISSIMI, A. S.; TOSCANI, S. S. Sistemas Operacionais. 4^a ed. Porto Alegre : Editora Bookman, 2010.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G Fundamentos de Sistemas Operacionais. 6^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2004.

SWADE, D. The Babbage Engine. Disponível em: <<http://www.computerhistory.org/babbage/>>. Acesso em: set. 2014.

TANENBAUM, A. S.; WOODHYLL, A. S. Sistemas operacionais projeto e implementação. 2^a ed Porto Alegre: Bookman, 1999.



NO PRÓXIMO CAPÍTULO

No capítulo seguinte, estudaremos sobre os conceitos relacionados aos processos. Você aprofundará seus conhecimentos de como é feito o compartilhamento de um espaço de endereçamento (memória) por vários fluxos de execução. Serão apresentados os principais conceitos relacionados aos processos.

2

Processos

2 Processos

Como um sistema operacional consegue executar vários programas ao mesmo tempo?

A resposta para esta pergunta está nos processos. Veremos que os programas podem ser divididos em vários processos e que estes processos têm a necessidade de compartilhar diversos recursos, tais como processador, memória, dispositivos de entrada e saída, etc. Vamos neste capítulo estudar o que são os processos e como o sistema operacional faz o seu gerenciamento.



OBJETIVOS

- Compreender os principais conceitos relacionados aos processos.
- Aprofundar nossos conhecimentos de como é feito o compartilhamento de um espaço de endereçamento (memória) por vários fluxos de execução.
- Discutir sobre os problemas envolvidos no compartilhamento de recursos entre os processos e as soluções encontradas para resolvê-los.



REFLEXÃO

Você se lembra dos sistemas operacionais multiprogramáveis? Caso não se lembre, faça uma breve consulta e anote os principais conceitos.

2.1 Conceito de Processo

Como vimos anteriormente os sistemas operacionais multiprogramáveis permitem carregar mais de um programa em memória e executá-los concorrentemente. Segundo Silberschatz et al. (2004) um processo é um programa em execução. Assim há uma diferenciação entre programas e processos. O programa é algo estático enquanto os processos são dinâmicos. Os programas são formados por sequências de comandos e instruções e não tem a possibilidade de alterar o seu estado. Os processos por sua vez, são quem executam os comandos dos programas, e à medida que os comandos são executados o processo pode sofrer alterações. Quando abrimos um navegador da Internet (ex.: Internet Ex-

pler) um processo é associado a ele. Se abrirmos um segundo Internet Explorer, um novo processo será criado, totalmente independente do primeiro. Ou seja, podemos ter vários processos relacionados a um único programa e gerenciados individualmente pelo sistema operacional.

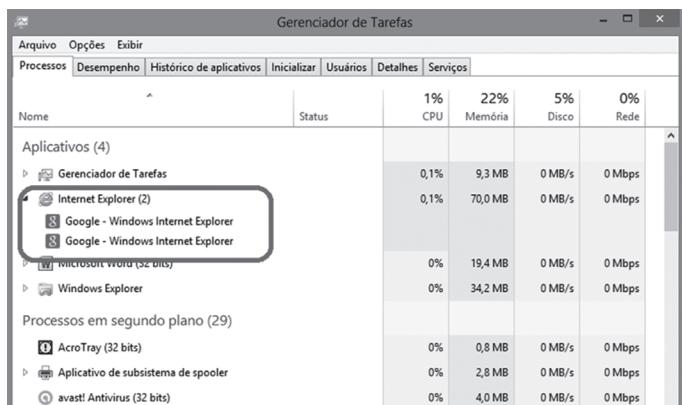


Figura 19 – Gerenciador de tarefas Windows 8.0.

O Gerenciador de Tarefas do Windows 8.0 permite que possamos visualizar os processos que estão em execução no computador. Notem que temos o grupo de *Aplicativos* e o grupo *Processos em segundo plano*. Os processos ligados a execução de um programa são listados no grupo “Aplicativo”. Existem muitos processos que não têm ligação direta com algum programa e sim com o sistema operacional. Estes processos, tais como gerenciador de memória, gerenciador de rede, antivírus etc., são executados em segundo plano.

CONEXÃO

Leia um pouco mais sobre processos em: <<http://www.guiky.com.br/2009/12/o-que-sao-os-processos-do-sistema-operacional.html>>. Entenda um pouco mais com este outro artigo: <<http://www.tecmundo.com.br/memoria/3197-o-que-sao-processos-de-um-sistema-operacional-e-por-que-e-importante-saber.htm>>.



ATENÇÃO

O termo Processo foi utilizado pela primeira vez na década de 60 pelos projetistas do sistema MULTICS. Desde esta época, houve um aumento em sua definição, tais como tarefa, programa em execução, o “espírito animado” de um procedimento, etc. (DEITEL et al., 2005).

2.1.1 Pseudoparalelismo

Quando estamos fazendo um determinado trabalho no computador, como por exemplo, digitando um trabalho da faculdade, podemos executar um programa para escutar nossas músicas enquanto efetuamos a digitação, abrir o *browser* da Internet para efetuar alguma pesquisa e rodar um programa para tratamento de imagens.

A impressão que temos é de que todos estes programas estão sendo executados em paralelo, ou seja, todos ao mesmo tempo, mas não é bem assim que as coisas acontecem.

Neste cenário, para que todos os programas sejam executados, o sistema operacional associa um processo a cada programa em execução, assim, teremos quatro processos sendo executados. Cada processo tem diversas características próprias, tais como, seus estados atuais, recursos necessários à execução de suas tarefas, valores de registradores, variáveis etc.

Como há quatro processos em execução, os quatro concorrem pela utilização da CPU, forçando o seu compartilhamento. O compartilhamento da CPU é feito através de uma rápida alternância entre um processo e outro, executando cada um num rápido intervalo de tempo, podendo chegar a dezenas ou centenas de milissegundos. Esta alternância é conhecida como *pseudoparalelismo*. É importante destacar que o paralelismo real acontece em *hardware* dos sistemas multiprocessadores, ou seja, possuem duas ou mais CPU que compartilham a mesma memória física.

Na Figura 20 temos o detalhamento deste processo de alternância com quatro processos em memória. A Figura 20(a) ilustra os quatro programas na memória associados a quatro processos. Cada processo é executado independente uns dos outros, cada um executando o seu próprio fluxo de controle (Figura 20(b)). A Figura 20(c) exibe o progresso de execução de cada um dos processos. Podemos notar que apenas um processo foi executado em um determinado instante. No tempo 1 a CPU executa as instruções do processo “A”. No tempo

2 o sistema operacional decide inicializar o processo “B”, mas antes, todos os dados dos registradores do processador são salvos no processo “A”. Este ciclo acontece para todos os processos enquanto estiverem em execução. Este procedimento, efetuado pelo sistema operacional, de troca de processo por outro é chamado de *mudança de contexto*.

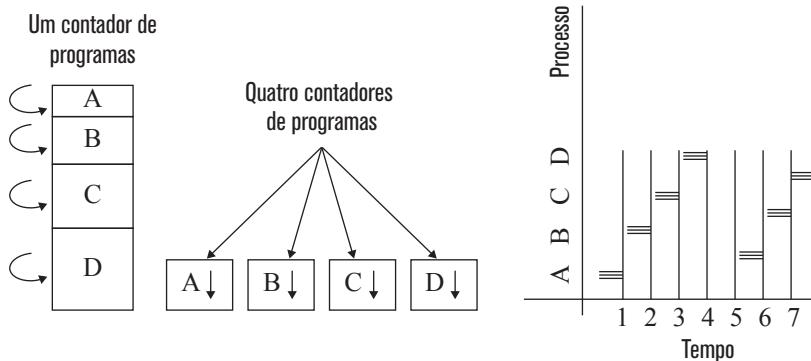


Figura 20 – (a) Multiprogramação de quatro programas. (b) Modelo conceitual de quatro processos sequenciais independentes. (c) Só um programa está ativo em qualquer dados instante.
Fonte: Tanenbaum e Woodhyll (1999).

São grandes as vantagens de utilização de processos pelo sistema operacional, tais como:

- Simplicidade: um processo pode ser decomposto em vários outros processos permitindo que se realizem várias operações independentes em um sistema.
- Velocidade: se um processo esta em espera, por exemplo, aguardando alguma solicitação a um dispositivo de entrada e saída, este processo é substituído por outro.
- Segurança: cada processo pode estar associado a um determinado direito.

Em geral os processos podem ser classificados em três classes:

- Interativos (*Foreground*): são processos que necessitam de algum tipo de interação com o usuário. Normalmente esta interação é relativa a solicitações de entrada e saída, feitas através de interface gráfica, que exigem do sistema operacional um tempo de resposta rápido. Exemplo: editor de texto, planilhas eletrônicas, jogos, etc.

- *Batch (Background)*: são processos que realizam o processamento de dados de entrada produzindo um conjunto de dados de saída sem que haja a intervenção do usuário. Exemplo: *backups*, compiladores, programas de cálculo numérico, etc.
- *Daemons*: são processos carregados pelo sistema operacional durante sua inicialização e permanece em execução até que o sistema seja finalizado. Processos *Daemons* ficam em espera em segundo plano até que seja requerido algum serviço. Exemplo: gerenciamento de log do sistema, serviços de *e-mail*, etc.

2.1.2 Blocos de controle de processos (PCBs)

O sistema operacional ao criar um processo, cria um bloco chamado *Blocos de controle de processos* (PCBs), também conhecido como *descritor de processo*. O bloco de controle de processo contém diversas informações que auxiliam o seu gerenciamento pelo sistema operacional. Entre estas informações estão:

- Número de Identificador de processo (*PID – Process Identification Number*): número pelo qual o processo é identificado pelo sistema operacional.
- Estado do processo: estado atual do processo (Pronto, Execução, Em Espera, etc.).
- Contador de programa: indica qual instrução do processo deverá ser executada.
- Prioridade de escalonamento: indica qual a prioridade do processo na escala de prioridade de execução.
- Credenciais: indica quais as permissões o processo possui.

2.1.3 Contexto

Na seção anterior vimos que um processo tem diversas características, entre elas, o seu estado. O estado representa a situação atual do processo, ou seja, os valores de suas variáveis, as instruções que esta executando, os recursos em uso. Conforme a execução do processo avança, o seu estado se altera. O estado de um processo em um determinado instante é conhecido como *contexto*.

Um processo é formado por três partes: *contexto de hardware*, *contexto de software* e *espaço de endereçamento*. Através destas três partes um processo guarde todas as informações do programa em execução.

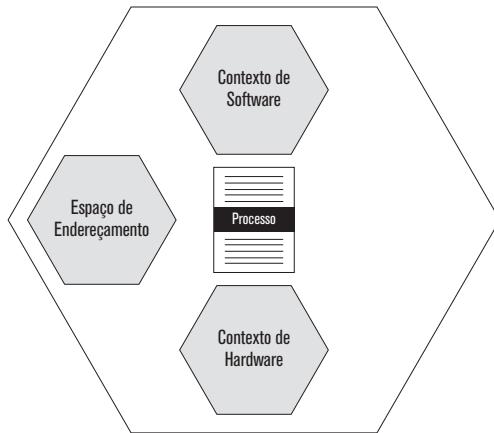


Figura 21 – Estrutura do processo.

2.1.4 Contexto de Hardware

Em sistemas de tempo compartilhado (multiprogramados), no qual há um revezamento na utilização do processador pelos processos, o contexto de *hardware* é de suma importância. Através do contexto de *hardware*, o sistema operacional é capaz de efetuar a troca de um processo por outro no processador, esta operação é conhecida como *troca de contexto*.

A troca de contexto envolve os registradores do processador. Um processador possui vários registradores entre os de usos gerais e os específicos, tais como o Program Counter (PC), o Stack Pointer(SP) e o Registrador de Status (PSW). Os registradores são utilizados para armazenar informações enquanto um processo está em execução.

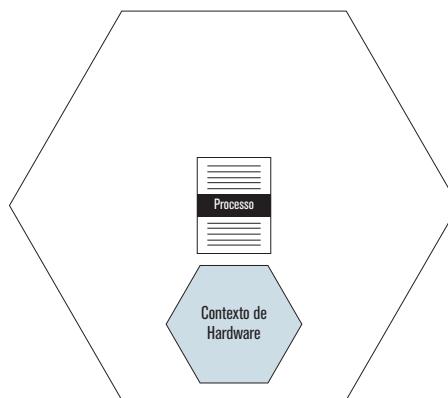


Figura 22 – Contexto de hardware.

A troca de contexto segue a seguinte sequencia. Ao executar um processo “A”, o seu contexto de *hardware* é armazenado nos registradores do processador. Quando o sistema operacional decide trocar de processo, executar o processo “B”, primeiramente salva os dados dos registradores no contexto de *hardware* do processo “A” e em seguida disponibiliza a utilização do processador para o processo “B”. O contexto de *hardware* do processo “B” é armazenado nos registradores do processador. A necessidade de armazenar os registradores no contexto de *hardware* quando um processo perde o uso do processador é que quando este processo voltar a ser executado possa continuar exatamente de onde foi interrompido.

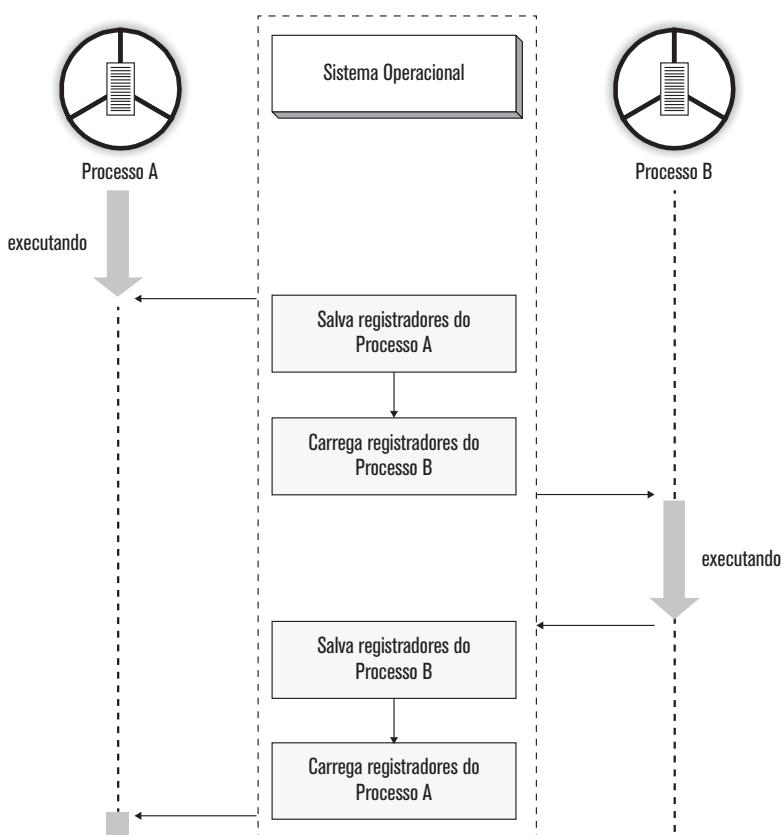


Figura 23 – Mudança de contexto.

Fonte: Machado e Maia (2007).

2.1.5 Contexto de software

Quando um processo é criado, o sistema operacional especifica os limites e características de recursos que o processo pode alocar. Estas informações são armazenadas no contexto de *software* e são relativas à: quantidade máxima de arquivos que o processo pode abrir simultaneamente, qual a prioridade de execução, tamanho do *buffer* utilizado em operações de entrada e saída, etc. As especificações de limites de recursos permitidos a um processo alocar são definidos pelo administrador do sistema e armazenado num arquivo do sistema operacional denominado *arquivo de usuários*. O sistema operacional consulta o arquivo de usuários para prover as informações do contexto de *software*.

As informações do processo são divididas em três grupos no contexto de *software*: identificação, quotas e privilégios.

- Identificação: quando um processo é criado, por um usuário ou por outro processo, recebe do sistema operacional um número de identificação única (PID – *Process Identification Number*) e a identificação única do usuário (UID – *User Identification*) referente ao usuário ou processo que o criou. O sistema operacional e outros processos utilizam o PID para fazer referências aos processos em execução.
- Quotas: o limite de recursos que um processo pode alocar e definidos em quotas. Um processo em execução pode ter necessidade de mais recursos do que lhe foi estipulado. Esta situação de insuficiência de quota pode causar a lentidão na execução do processo, sua interrupção ou nem mesmo ser executado.
- Privilégio: cada processo possui privilégios que lhe permite executar ações a ele mesmo, como alterar seus limites de alocação de memória, sua prioridade de execução etc. Além dele mesmo, o processo pode ter privilégios de efetuar ações a outros processos e ao sistema operacional.

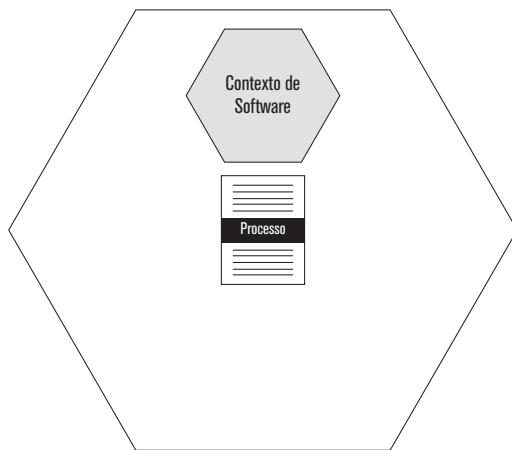


Figura 24 - Contexto de software.

2.1.6 Espaço de endereçamento

Os processos são responsáveis pela execução das instruções de um programa. Para tanto, as instruções e os dados do programa são carregados em uma área de memória pertencentes ao processo para serem executados. Esta área de memória é chamada de *espaço de endereçamento*.

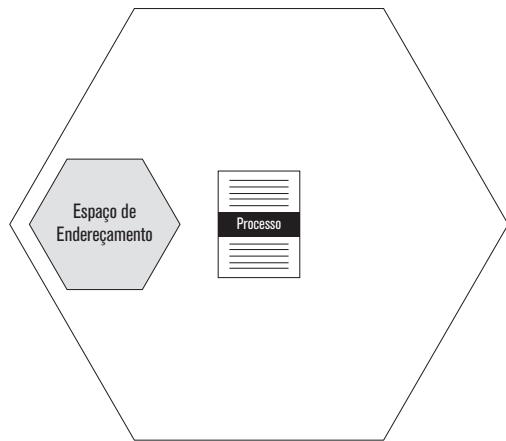


Figura 25 – Espaço de endereçamento.

2.2 Estados de um processo

Os processos executados pelo processador normalmente possuem necessidades distintas de duração, importância e comportamento. O sistema operacional tem a responsabilidade de gerenciar e estabelecer uma ordem de execução destes processos. Como visto anteriormente, no sistema multiprogramável há um padrão de compartilhamento do processador entre os diversos processos em execução. Assim há uma troca constante de processos em *execução no processador o que faz com que o processo passe por estados* distintos durante o seu ciclo de vida. Conforme avança a execução de um processo, o seu estado pode ser alterado tanto por eventos gerados pelo próprio processo como pelo sistema operacional. Os estados possíveis para um processo são:

- Pronto: este estado indica que o processo aguarda para ser executado. Como normalmente há vários processos em estado de Pronto, primeiramente o sistema operacional efetua um enfileiramento destes processos (Figura 26). Em seguida utiliza um mecanismo denominado *escalonamento* com o objetivo de determinar a ordem e os critérios de uso do processador pelos processos.
- Em Execução: este estado indica que o processo está sendo executado pelo processador. A troca de um processo por outro na utilização do processador deve respeitar os critérios de escalonamento.
- Em Espera: este estado indica que o processo está aguardando por um evento externo ou ser atendida alguma chamada ao sistema operacional. Os processos neste estado são organizados em fila e assim que o evento esperado ou o retorno da chamada ao sistema operacional acontece, o processo é transferido para o estado de pronto.



Figura 26 – Fila de processos aguardando a execução pelo processador.

Fonte: Oliveira et al. (2010).

2.2.1 Mudança de estado de um processo

De acordo com Silberschatz et al. (2004) há quatro mudanças de estado que podem ocorrer durante o seu ciclo de vida de um processo:

- Novo → Pronto: esta mudança de estado ocorre quando o processo é admitido pelo sistema operacional e termina de ser carregado na memória, aguardando apenas a oportunidade de ser escolhido pelo escalonador para ser executado.
- Pronto → Em Execução: esta mudança ocorre quando o escalonador escorre um processo na fila de pronto para ser executado.
- Em Execução → Pronto: esta mudança ocorre quando o tempo de uso do processador pelo processo chega ao fim. O processo retorna para a fila de processos prontos até ser novamente habilitado pelo escalonador para usar o processador.
- Em Execução → Em Espera: esta mudança ocorre quando o processo aguarda um evento ou necessita utilizar algum recurso que não esteja disponível no momento, por exemplo. Isto faz com que o processo libere o uso do processador e fique em estado de espera enquanto aguarda o evento ou a disponibilidade do recurso.
- Em Espera → Pronto: esta mudança ocorre quando o processo recebe o evento esperado ou o recurso solicitado fica disponível para uso. Consequentemente o processo volta para a fila de pronto a fim de aguardar ser processado.
- Em Execução → Terminado: esta mudança ocorre quando o processo finaliza sua execução ou quando acontece algum erro e o processo precisa ser abortado.

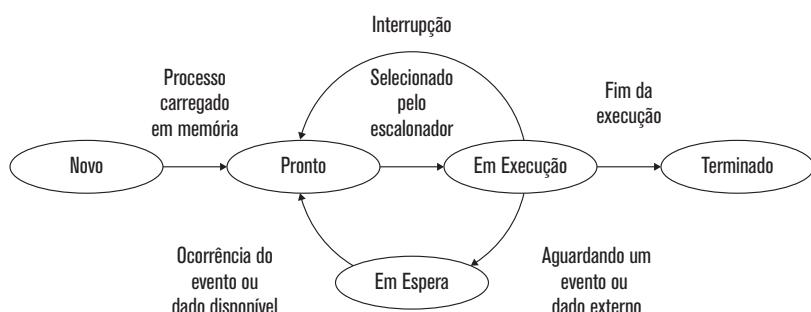


Figura 27 – Diagrama de estados de um processo.

Fonte: Silberschatz et al. (2004).

2.2.2 Classificação de processos

Os processos podem ser classificados de acordo com vários critérios entre eles, em função de modo como utiliza o processador e os dispositivos de entrada e saída:

- Processo *CPU-Bound*: são processos que utilizam de modo muito intenso o processador durante o seu ciclo de vida. A maior parte do tempo os estados destes processos são *Pronto* ou *Em Execução*.
- Processo *I/O-Bound*: este tipo de processo está mais ligado aos dispositivos de entrada/saída do que propriamente ao processador. O estado deste processo fica *Em Espera* durante grande parte do seu ciclo de vida devido aos longos tempos aguardando respostas das solicitações feitas aos dispositivos de entrada e saída.

Durante o ciclo de vida de um processo são realizadas entradas e saídas de dados no processo. Estas entradas e saídas são feitas através de canais de comunicação disponibilizada pelo processo. O processo tem no mínimo dois canais de comunicação associados a ele. Os canais permitem acesso de um processo a outros processos, comunicação com o usuário, arquivos, etc.

- Processo em *foreground*: são processos que mantém com o usuário uma ligação direta enquanto esta sendo processado. Esta ligação do processo com o usuário é feita através dos canais de comunicação associados ao teclado, *mouse*, monitor, etc.
- Processos em *background*: são processos que não tem ligação direta com o usuário enquanto esta sendo processado. Este tipo de processo é muito utilizado em processamentos do tipo *batch*.

2.3 Threads

Anteriormente estudamos que os processos possuem uma série de características tais como vistos em contexto de *software*, de *hardware*, espaço de endereçamento, etc. Além das características, há associado a um processo um fluxo de execução que é conhecido como *thread*. Os primeiros sistemas operacionais eram *monothread*, ou seja, cada processo tinha o seu espaço de endereçamento individual e era associado a um único fluxo de execução (*thread*).

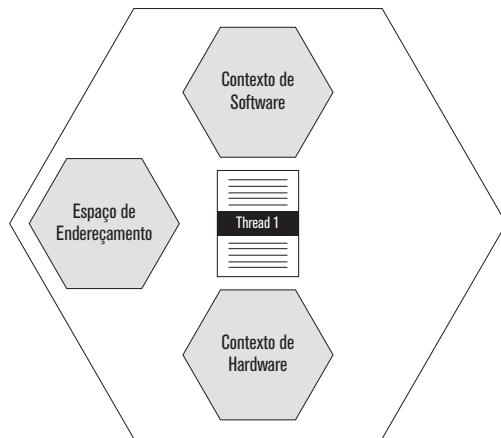


Figura 28 – Processo monothread.

Em sistemas operacionais *multithreading*, um único processo pode estar associado a vários fluxos de execução (*threads*). Como as *threads* existem no interior do processo, todas compartilham o mesmo espaço de endereçamento (código e dados).

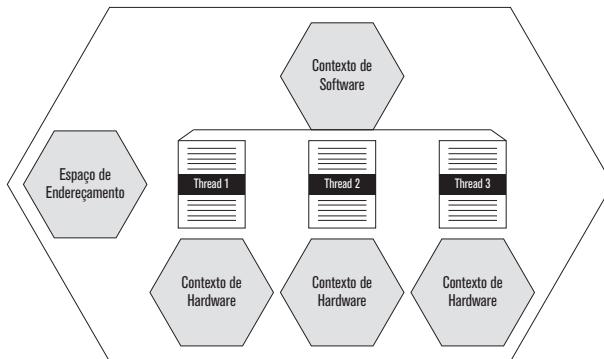


Figura 29 – Processo multithread.

Compartilham o mesmo espaço de endereçamento faz com que o gerenciamento de processos das aplicações concorrentes fiquem mais eficientes devido a drástica diminuição de tempo gasto com a criação, eliminação e troca de contextos destes processos. A troca de um *thread* por outro dentro de um mesmo processo é muito mais rápida do que a troca entre dois processos. Os *threads* compartilham uso do processador como se fossem processos separados. Assim os estados dos *threads* seguem os mesmos princípios dos processos. Logo se uma *thread* esta

aguardando que um determinado recurso fique disponível, seu estado passa de *Em Execução* para *Em Espera*, liberando o processador para outra *thread*.

Um exemplo de utilização de *threads* pode ser dado através da execução de um vídeo com som em inglês e legendas em português. O programa de visualização do vídeo pode criar um processo com três linhas de execução (três *threads*). Sendo um *thread* para tratar a exibição da imagem (vídeo) um segundo *thread* para tratar o áudio e o terceiro para tratar a legenda. O usuário final visualiza a imagem, o som e legenda, tudo ao mesmo tempo, mas o processamento de cada *thread* é executado individualmente no processador.



CONEXÃO

Entenda como são criadas as threads em linguagem de programação.

<http://www.macoratti.net/14/07/c_gptwf.htm>.

2.4 Comunicação entre Processos

Nos dias atuais temos acesso a computadores com multiprocessadores, os quais, a maioria dos programas levam em consideração em seu desenvolvimento, para obter melhor desempenho.

Muitos problemas solucionados por um programa não são executados sequencialmente, são divididos em várias tarefas interdependentes que relacionam entre si a fim de atingir um objetivo comum. Os códigos dos programas são estruturados de tal forma que permitem serem executados concorrentemente através de múltiplos processos ou *threads*. Programas com estas características são chamados de *aplicações concorrentes*.

Aplicações concorrentes proporcionam um melhor desempenho mesmo não havendo um paralelismo real na execução dos processos ou *threads*. Onde há o paralelismo real, em casos de sistemas com múltiplos processadores, as aplicações concorrentes melhora ainda mais estas vantagens.

Programas desenvolvidos para serem executados de modo sequencial, que executam apenas um único fluxo por vez, não se beneficiam das vantagens oferecidas por um sistema com múltiplos processadores. Programas como editores de texto, navegadores *web* e jogos, caso fossem um programa sequencial, não teríamos a alta interatividade que temos nestas aplicações. Os editores de texto, por exemplo, permitem a digitação enquanto efetuam a revisão ortográfica. Os

navegadores *web* interagem com usuário através das páginas de Internet ao mesmo tempo em que tarefas de comunicação com a rede estão sendo executadas.

Os bancos de dados, essencial nas aplicações atuais, principalmente aplicações empresariais como Sistema integrado de gestão empresarial (SIGE ou ERP – *Enterprise Resource Planning*) são outro exemplo de aplicações concorrentes. Um sistema ERP integra todos os dados e processo de uma organização em um único sistema e estes dados são armazenados em um banco de dados. Caso o banco de dados utilizado por um ERP fosse sequencial, atenderia um único usuário por vez. Logo se um usuário estivesse dando entrada de mercadoria no estoque, todo o restante da empresa estaria parado, aguardando o término desta tarefa, tornando este tipo de sistema inviável para uma empresa.

Para que os processos concorrentes possam cooperar entre si, é necessário que haja uma comunicação entre eles para troca e compartilhamento de informações, além de um sincronismo, efetuado pelos sistemas operacional para que as atividades sejam executadas sem erros. Diversos mecanismos podem ser utilizados para efetuar a comunicação entre os processos, como variáveis compartilhadas na memória principal, quanto à memória é compartilhada entre os processos. Quando a memória não é compartilhada, os processos podem trocar mensagens para compartilhar as informações entre si.

Para ilustrar o uso de memória compartilhada e a eficiência dos programas concorrentes em relação aos programas sequenciais, Oliveira et al. (2010) propõem uma pequena tarefa de impressão de arquivo, que deve ser executada por um programa. O programa deve ler um arquivo do disco rígido, efetuar a formatação adequada para imprimir e por fim, enviar para a impressora.

A figura 30 exibe as três operações feitas utilizando um programa sequencial. Neste caso é utilizado apenas um processo.



Figura 30 - Acessando arquivo e impressora através de um programa sequencial.

Fonte: Oliveira et al. (2010).

A Figura 31 exibe a linha do tempo necessária para que o processo possa efetuar a sequência de passos para a realização da tarefa:

4. A primeira coisa efetuada pelo processo é enviar um comando de leitura do arquivo para o disco rígido.
5. Em seguida o processo fica em modo espera até que a leitura seja finalizada.
6. O arquivo é formatado pelo processo.
7. Envia os dados formatados direto para o *buffer* da impressora. Como normalmente o *buffer* é relativamente pequeno são necessárias várias repetições das quatro etapas até que o arquivo seja totalmente impresso.

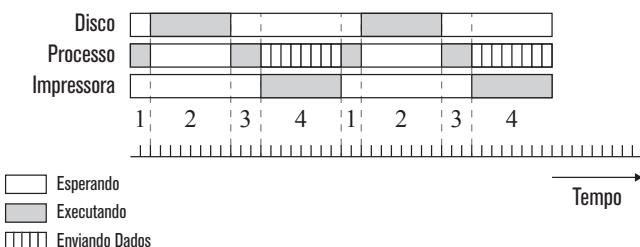


Figura 31 – Linha do tempo do programa sequencial.

Fonte: Oliveira et al. (2010)

A Figura 32 exibe a mesma tarefa sendo executada por um programa concorrente. Logo foi utilizado dois processos na execução da tarefa: processo leitor e processo impressor.

A função do processo leitor é ler o arquivo do disco rígido, efetuar a devida formatação e colocar na variável compartilhada. O processo impressor retira os dados da variável e envia para o *buffer* da impressora.

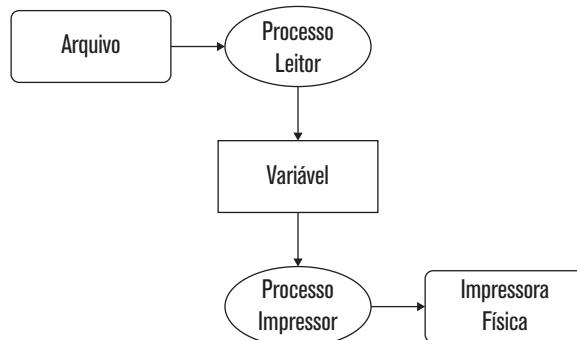


Figura 32 – Acessando arquivo e impressora através de um programa concorrente.

Fonte: Oliveira et al. (2010).

Através do gráfico, podemos notar a utilização simultânea do disco e da impressora. Dessa forma, o tempo de impressão será bem menor o que torna o programa concorrente mais eficiente que o programa sequencial.

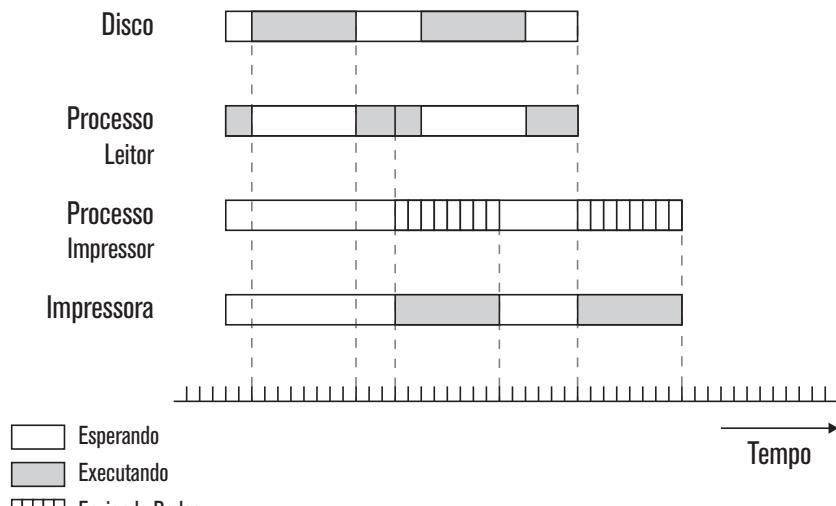


Figura 33 – Linha do tempo do programa sequencial.

Fonte: Oliveira et al. (2010).

2.5 Sincronização entre Processos.

Na seção anterior, vimos que as aplicações concorrentes em sistemas operacionais multiprogramáveis permitiram um grande avanço e eficiente em relação aos programas sequenciais. A eficiência decorre do trabalho cooperativo entre os processos e do acesso compartilhado de recursos, como memória, dispositivos de entrada/saída e arquivos. Em muitos casos o acesso a compartilhado pode gerar situações de inconsistência de informações ou problemas intermitentes de difícil reprodução e solução do mesmo. Estes problemas impulsionaram a criação de mecanismos nos sistemas operacionais a fim de ordenar a execução dos processos, chamados de *sincronização*.

2.5.1 Problemas das condições de corrida e região crítica

O problema de compartilhamento de recursos entre processos concorrentes fica evidente com o exemplo a seguir, onde a ordem de execução de dois processos interfere no resultado final, conhecido como condições de corrida (*race conditions*).

Utilizando o diagrama da Figura 34, têm-se dois processos que postam arquivos para impressão e o *spooler* (tabela), que recebe arquivos a serem impressos e ao finalizar a impressão, o arquivo é retirado do *spooler*. Em um determinado momento, o Processo “A” e “B” decidem imprimir um arquivo ao mesmo tempo. Os procedimentos que se seguem são:

1. Os dois processos consultam a variável de *Entrada* do *spooler* para obter o próximo *slot* livre e para postarem os arquivos para impressão figura 34 (a).
2. Os dois obtém o mesmo *slot* livre para impressão, o *slot* 10 (figura 34 (b)).
3. O processo “A” posta o arquivo “Arq_A.doc” para a impressão e incrementa a variável de Entrada para 11 (figura 34 (c)).
4. No mesmo instante o processo “B”, posta o arquivo “Arq_B.doc”, sobrepondo o “Arq_A.doc”, e em seguida incrementa a variável de Entrada para 10 (figura 34 (d)).
5. CONCLUSÃO: o processo “A” nunca receberá a impressão do arquivo “Arq_A.doc”.

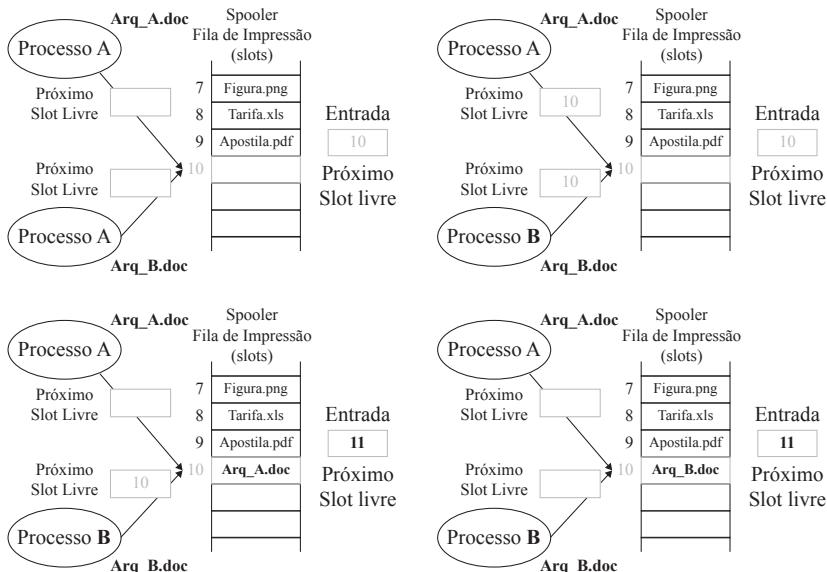


Figura 34 – Condições de corrida.

A solução para o problema de condições de corrida (*race conditions*) se inicia pela identificação da parte do programa que gera disputa dos recursos compartilhados e da parte que não gera disputa. A parte que gera disputa é conhecida como região *crítica* ou *seção crítica* e a que não gera disputa é conhecida como *códigos reentrante* ou *código público*.

Uma vez identificado à região crítica, o próximo passo para a solução do problema é atender os seguintes aspectos:

- Exclusão mútua: garantir que apenas um processo por vez terá acesso à região crítica.
- Progresso: caso um processo deseja utilizar uma região crítica que não está em uso, outros processos não podem bloqueá-lo de utilizar a região crítica.
- Espera limitada: caso um processo deseja utilizar uma região crítica, o mesmo não deve esperar indefinidamente para a utilização da região crítica.

A literatura oferece várias soluções para o problema da região crítica e que satisfazem os três aspectos descritos anteriormente. A seguir são abordadas algumas destas soluções.



ATENÇÃO

Uma das tarefas dos programadores é testar o programa que esta sendo criado. Esta tarefa não é nada fácil quando o programa em teste contém condições de corrida. Esta condição faz com que a maioria dos testes tenham bons resultados, mas, de tempos em tempos, algo estranho e inexplicável acontece (TANENBAUM; WOODHYLL, 1999).

2.5.2 Sincronismo por software

Esta solução leva em consideração a utilização apenas de *software*. A entrada da região crítica é feita através de uma variável de sincronismo *busy* (Figura 35). A variável *busy* inicialmente contém o valor 0 (zero) o que indica que qualquer processo pode entrar na região crítica. Um processo ao entrar na região crítica, altera *busy* para 1 impedido o acesso de outro processo a região. Suponha que temos dois processos, “A” e “B”, sendo executados concorrentemente e que desejam entrar na região crítica. O processo “A” lê a variável *busy* com valor 0 (linha 5), entra na região

crítica e antes de mudar o valor de *busy* para 1 (linha 6), ocorreu uma interrupção que passou o controle para o processo “B”. O processo “B” lê *busy* igual a 0, entra na região crítica e altera *busy* para 1. Ocorre novamente a interrupção voltando a executar o processo “A”. Neste momento, para o processo “A” o valor de *busy* ainda é 0, o que ocasiona também a sua entrada na região crítica.

```
1 int busy = 0 ;           // a seção está inicialmente livre
2
3 void enter (int task)
4 {
5     while (busy) ;      // espera enquanto a seção estiver ocupada
6     busy = 1 ;           // marca a seção como ocupada
7 }
8
9 void leave (int task)
10{
11    busy = 0 ;           // libera a seção (marca como livre)
12}
```

Figura 35 – Sincronismo por software.

Fonte: Maziero (2014).

Abordagem de *software* apresenta uma propriedade denominada *busy-waiting* (espera ativa). *Busy-waiting* ocorre quando um processo tem que aguardar em um laço a sinalização de um evento. Segundo Oliveira et al. (2010) a propriedade *Busy-waiting* aliado ao fato que a implementação de soluções de *software* serem altamente complexas, faz com que não sejam muito empregadas na prática.

2.5.3 Sincronismo por Hardware

Em sistemas que utilizam um único processador a solução adotada é desabilitar as interrupções. O processo ao entrar na região crítica, desabilita todas as interrupções e habilita ao sair. Dessa forma o processador não efetua a troca de processo enquanto a interrupção estiver desabilitada. Este tipo de solução apresenta algumas restrições:

- Segurança: não é uma solução segura pelo fato do processo, por algum motivo, não habilitar novamente as interrupções e assim não conseguir ser finalizado.
- Eficiência: a perda de eficiência esta relacionada aos periféricos que dependem das interrupções para serem atendidos imediatamente durante a utilização de regiões críticas. Os periféricos somente serão atendidos depois que o processo sair da região crítica.

- Limitação: desabilitar a interrupção depende de acesso privilegiado e processos do usuário pode não ter esta permissão. Em máquinas onde há vários processadores executando simultaneamente, desabilitar as interrupções não funciona. Isto porque ao desabilitar a interrupção em um processador, os demais continuaram funcionando. Desabilitando todas interrupções em todos os processadores, ainda assim os processadores estariam executando simultaneamente os processos.

2.5.4 Semáforos

Em 1965 o matemático holandês E.W.Dijkstra sugeriu a utilização de um mecanismo conhecido como *semáforo* para o sincronismo entre processos. A proteção da região crítica tornou-se simples com a utilização do semáforo. Logo, os projetos de sistemas operacionais e aplicações concorrentes passaram a adotar o mecanismo se semáforo como principal forma de implementar exclusão mútua e a sincronização condicional entre processos. Os semáforos são classificados em: binário ou contadores. Semáforos binários só podem assumir valores 0 e 1, por isso também são chamados de *mutexes (mutual exclusion semaphores)*. Já os semáforos contadores permitem qualquer valor desde que sejam inteiros positivos.

O semáforo é composto por uma variável do tipo inteiro que é controlada por duas instruções: *Down* e *Up*. *Down* e *Up* são instruções que garantem uma única ação atômica e indivisível, ou seja, não podem ser interrompidas quando estiverem em execução.

O recurso compartilhado fica associado a um semáforo que controla o acesso dos processos concorrentes ao recurso. O recurso estará disponível para uso caso a variável semáforo esteja maior que 0, caso contrário, o recurso está sendo utilizado por algum processo.

A instrução *Down* é executada quando um processo deseja entrar em sua região crítica. Estando o valor da variável semáforo maior que 0, este valor é decrementado de 1 e a região crítica é liberada para uso do processo. Se em seguida, outro processo executar a instrução *Down*, verá que o valor da variável é 0, não podendo assim acessar sua região crítica, ficando então na fila de estado de espera. O processo que está na região crítica, ao sair, executa a instrução *Up* que incrementa o valor da variável semáforo, liberando o acesso do recurso. Caso haja processos na fila de espera, associados ao recurso, um desses processos terá seu estado alterado para pronto pelo sistema.

2.5.5 Monitores

O mecanismo de semáforo é muito importante para a implementação de exclusão mútua, mas por não ser estruturado, pode ser difícil sua utilização em um programa. Este fato exige do desenvolvedor um cuidado redobrado na programação, teste e depuração para não causar problemas intermitentes, difícil de reproduzir e solucionar. Em 1974 C.A.R Hoare propôs um mecanismo de sincronização de alto nível chamado *Monitor*. Os monitores oferecem as funcionalidades dos semáforos, porém tem a característica de ser estruturado, logo seu código é mais legível e consequentemente mais imune a *bugs*.

Formado por um módulo, que encapsula as variáveis e procedimentos, o monitor diminui a complexidade da programação além de implementar a exclusão mútua entre os procedimentos de forma automática. A exclusão mútua entre procedimentos significa que, em um determinado instante, apenas um processo de cada vez tem a permissão de executar um dos procedimentos do monitor. Caso outro processo chame qualquer um dos procedimentos, este ficará bloqueado, aguardando em uma fila de entrada até que seja sua vez.

2.5.6 Troca de mensagens

Quando a memória não é compartilhada, os processos trocam mensagens para compartilhar informações entre si. Esta situação pode ocorrer quando os processos estão executando em diferentes computadores. A troca de mensagens é feita utilizando um canal de comunicação entre os processos. O canal de comunicação pode ser um *buffer* ou um *link* de rede entre os computadores. Uma vez o canal de comunicação estabelecido entre os processos, às mensagens podem ser trocadas utilizando duas funções básicas: *Send* e *Receive*.

O processo transmissor (origem) utiliza a função *Send*(Receptor, Mensagem) para enviar mensagens (dados) para um processo receptor (destino). O processo receptor recebe as mensagens enviadas através da função *Receive* (transmissor, mensagem). A mensagem só pode ser lida após ter sido enviada. Logo é necessário que haja um sincronismo entre os processos envolvidos na troca das mensagens.

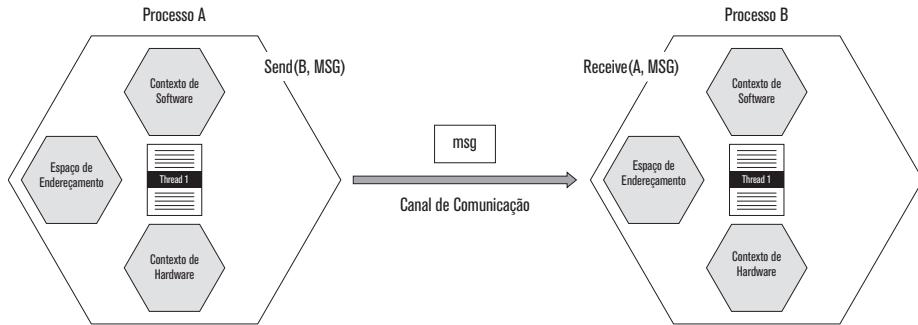


Figura 36 – Transmissão de mensagem.

Para que a comunicação entre processos aconteça, os processos necessitam saber como se referir um ao outro. A forma como os processos se referenciam caracteriza o tipo de comunicação entre eles podendo ser:

- Comunicação direta: este tipo de comunicação exige que o processo que deseja enviar ou receber uma mensagem, deve explicitar o nome do processo receptor ou transmissor da mensagem. O tipo de comunicação direta tem a restrição de permitir a troca de mensagens somente entre dois processos.

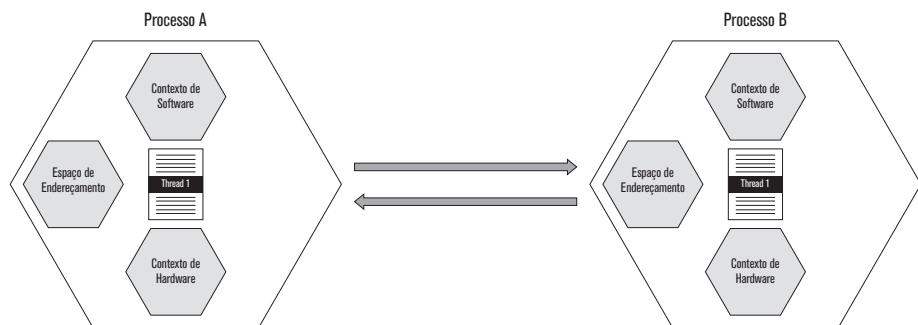


Figura 37 – Comunicação direta.

- Comunicação indireta: neste tipo de comunicação, são utilizadas caixas postais para o recebimento e envio de mensagens. O processo transmissor deposita mensagens na caixa postal que são retiradas pelo processo receptor. Neste tipo de comunicação pode haver várias caixas postais para a comunicação dos processos. Cada caixa recebe uma identificação

única, assim um processo pode enviar ou receber mensagens de outro processo utilizando caixas postais diferentes, desde que os processos compartilhem o uso da mesma caixa.

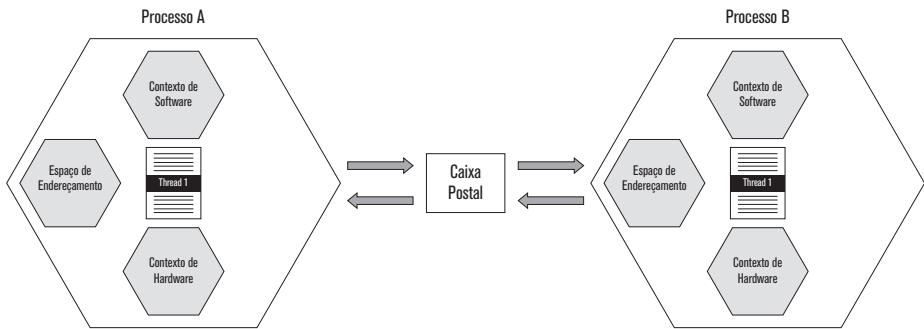


Figura 38 – Comunicação indireta.



ATIVIDADE

1. O processo é composto por três partes, entre elas o contexto de hardware. Das opções abaixo, qual opção que faz parte do contexto de hardware?
 - f) Endereço de memória
 - g) Registrador PC
 - h) PID
 - i) Privilégio
 - j) Tempo de processador
2. O processo é composto por três partes, entre elas o contexto de software. Das opções abaixo, qual opção que faz parte do contexto de software?
 - a) Registrador SP
 - b) Registrador PC
 - c) PID
 - d) Endereço de memória
 - e) Registrador de status
3. Por que os processos I/O-Bound ficam no estado “Em Espera” durante grande parte do seu ciclo de vida?



REFLEXÃO

Neste capítulo estudamos como os programas são executados pelo processador através dos processos. Posteriormente vimos como os processos são estruturados em contexto de *hardware*, *software* e espaço de endereçamento, permitindo um gerenciamento mais eficiente do sistema operacional. Em seguida, discutimos as *threads*, comunicação e sincronização entre processos, conceitos que nos auxiliam entender melhor a multiprogramação. Sugerimos que você faça todos os exercícios propostos e pesquise outras fontes para aprofundar seus conhecimentos. Em caso de dúvidas, retorne aos tópicos e faça a releitura com bastante atenção.



LEITURA

Para você avançar mais o seu nível de aprendizagem envolvendo os conceitos de sistemas operacionais e demais assuntos deste capítulo, consulte as sugestões de *links* abaixo:

MACORATTI, J. C. VB.NET – Trabalhando com Threads. Disponível em: <http://www.macoratti.net/vbn_thd1.htm>. Acesso em: set. 2014.

MACORATTI, J. C. Trabalhando com MultiThreads no VB.NET. Disponível em: <http://www.macoratti.net/vbn_thrd.htm>. Acesso em: set. 2014.



REFERÊNCIAS BIBLIOGRÁFICAS

DEITEL H. M.; DEITEL P. J.; CHOFFNES D. R. Sistemas Operacionais. 3^a ed. São Paulo, Editora Prentice-Hall, 2005.

HENNESSY, J. L.; PATTERSON, D. A. Arquitetura de Computadores: uma abordagem quantitativa. 3^a ed. Rio de Janeiro: Campus, 2003.

MACHADO, F. B.; MAIA, L. P. Arquitetura de sistemas operacionais. 4^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2007.

MAZIERO, C. A. Sistemas Operacionais: Conceitos e Mecanismos. Disponível em: <<http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/sosolivro.pdf>>. Acesso em: set. 2014.

OLIVEIRA, R. S.; CARISSIMI, A. S.; TOSCANI, S. S. Sistemas Operacionais. 4^a ed. Porto Alegre : Editora Bookman, 2010.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G Fundamentos de Sistemas Operacionais. 6^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2004.

TANENBAUM, A. S.; WOODHYLL, A. S. Sistemas operacionais projeto e implementação. 2^a ed Porto Alegre: Bookman, 1999.



NO PRÓXIMO CAPÍTULO

No capítulo seguinte, estudaremos sobre os principais conceitos relacionados ao escalonamento de processos. Você observará as diferenças entre os diversos tipos de escalonadores. Serão apresentados como é feito a avaliação de um escalonador.

3

Gerência de Processador

3 Gerência de Processador

Vimos que os processos têm a necessidade de compartilhar diversos recursos do sistema, principalmente o processador. Como o sistema operacional gerencia quando dois ou mais processos disputam o uso do processador? Quais os métodos utilizados para acesso ao processador, suas vantagens e desvantagens? Estas são perguntas que foram sendo solucionadas à medida que os sistemas operacionais e o próprio hardware foram evoluindo. A importância dada à forma como os processos são escalonados, para a utilização do processador, será um dos principais temas de nossa discussão.



OBJETIVOS

- Estudarmos os principais conceitos relacionados ao escalonamento de processos.
- Observar as diferenças entre os diversos tipos de escalonadores.
- Entender como é feita a avaliação de um escalonador.



REFLEXÃO

Você se lembra dos estados de um processo? Como os processos são classificados em função de modo como utilizam o processador e os dispositivos de entrada e saída? Seria interessante você procurar relembrar estes conceitos no capítulo 2, pois irá ajudá-lo a compreender melhor a função do gerenciamento do processador.

3.1 Fundamentos

Estudamos em capítulos anteriores que em sistemas multiprogramáveis podemos ter vários processos em estado de Pronto indicando que estão aguardando para serem executados. Neste momento, o sistema operacional, através do *escalonador de processo (scheduler)*, deve decidir qual processo será executado primeiramente pelo processador. A tomada de decisão, feita pelo escalonador, leva em consideração critérios de uso do processador pelos processos com base nas *políticas de escalonamento*. A política de escalonamento é a estrutura base da gerência do processador.

São funções básicas da política de escalonamento:

- Manter o processador ocupado por maior tempo possível.
- Efetuar o balanceamento de uso do processador entre processos.
- Aplicações críticas devem ser executadas prioritariamente.
- Maximizar o *throughput* (número de processos finalizados em um dado intervalo de tempo) do sistema.
- Processos interativos devem ter tempos de resposta razoáveis.

Uma vez definido qual processo deve ser executado, o sistema operacional passa a execução para o módulo conhecido como *Dispatcher* (despachante). O módulo *Dispatcher* é responsável por efetuar a troca de contexto e dar o controle do processador para o processo selecionado pelo escalonador. Esta operação de troca de contexto gasta um determinado tempo que é conhecido como *latência do dispatcher*, ou seja, o tempo para interromper a execução de um processo para executar outro.

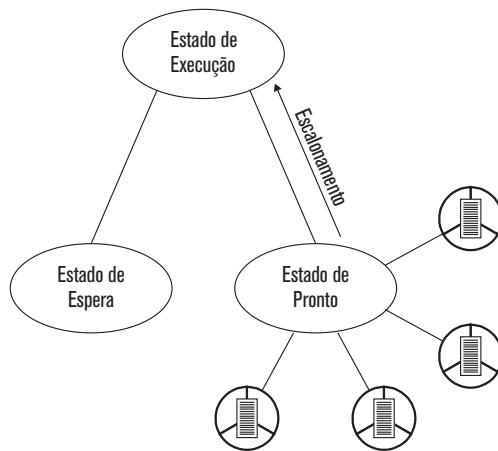


Figura 39 – Escalonamento.

Fonte: Machado e Maia (2007)

3.1.1 Escalonamento Não-Preemptivo e Preemptivo

Inicialmente, seria razoável pensar que a atividade de *políticas de escalonamento* para a troca de um processo em execução, seria trocá-lo pelo primeiro processo que está na fila de pronto. Mas a tomada de decisão do escalonador leva

em conta outros aspectos que tão somente o primeiro da fila de pronto para efetuar a troca. Um dos aspectos considerados pelo escalonador para atender a troca de um processo em execução por outro, é mudança de estado do processo.

4. O processo é carregado na memória e fica com estado de Pronto.
5. O processo muda de estado Em Execução para Pronto.
6. O processo muda de estado Em Execução para Em Espera.
7. O processo muda de estado Em Espera para Pronto.
8. O processo muda de estado de Em Execução para Terminado.



Figura 40 – Diagrama de estados de um processo.

Fonte: Silberschatz et al. (2004).

Tão logo o processo tenha mudado o seu estado, o escalonador pode interromper um processo em execução para dar lugar a outro. Esta tarefa de troca de processo pelo sistema operacional é conhecida como *preempção*. Levando em consideração a *preempção*, os escalonadores podem ser classificados como:

- Escalonador Não preemptivo: este tipo de escalonador foi um dos primeiros a serem utilizados em sistemas multiprogramáveis. Permite que o processo seja executado do início ao fim sem ser interrompido até ser finalizado. Escalonadores não-preemptivos mantêm o processo em execução até que seu estado mude para “Em Espera” ou “Terminado”, item 3 e item 5.
- Escalonador Preemptivo: este tipo de escalonador tem a capacidade de trocar um processo, que poderia continuar executando, por outro. Estão nesta situação os processos com mudanças de estados citados acima de 1 a 5. Escalonadores preemptivos permitiram que sistemas de tempo real pudessesem dar prioridade na execução de processos críticos.



ATENÇÃO

Os sistemas operacionais foram criados para executar os programas dos usuários. Nesta tarefa são consumidos, indiretamente, valiosos e escassos recursos do sistema. Este consumo é denominado sobrecarga pelo fato dos recursos não estarem sendo utilizados diretamente pelas aplicações (DEITEL et al., 2005).

3.2 Critérios de Escalonamento

As características principais de uma política de escalonamento variam de acordo com o tipo de sistema operacional. Em sistemas de tempo real a principal característica da política de escalonamento é garantir a prioridade de execução a processos considerados críticos em relação aos demais. Esta política não se aplica a sistemas de tempo compartilhado, pelo fato deste tipo de sistema priorizar, na política de escalonamento, que não haja *starvation*.

Starvation é a situação onde um processo nunca consegue ter acesso a um recurso compartilhado. A política nesta situação é garantir que todos os processos tenham o mesmo tempo de uso do processador.

Para Machado e Maia (2007), independente do tipo de sistema operacional, a política de escalonamento tem diversas funções básicas, tais como:

- Utilização do processador: o ideal é manter o processador ocupado o maior tempo possível. Uma faixa considerada ideal para ocupação do processador é entre 30% a 90%. Abaixo de 30% indica que o processador está sendo pouco utilizado. Acima da faixa superior, 90%, deve se ter cuidado pelo fato de estar próxima do limite do processador. As políticas de escalonamento procuram maximizar a utilização do processador.
- *Throughput*: indica o número de processos executados por unidade de tempo. Logo *Throughput* representa a produção do sistema e deve ser maximizado. Para longos processos, o valor de *Throughput* será baixo. Para curtos o valor de *Throughput* será alto. As políticas de escalonamento procuram maximizar o *Throughput*.
- Tempo de Processador: tempo em que um processo fica em execução no processador. Este tempo é influenciado apenas pelo código da aplicação e entrada de dados e não pelas políticas de escalonamento. As políticas de escalonamento não têm efeito sobre este tempo.

- **Tempo de Espera:** tempo total de espera, para ser executado, de um processo na fila de Pronto. As políticas de escalonamento procuram reduzir ao máximo este tempo.
- **Tempo de Turnaround:** indica o tempo total de vida de um processo, desde sua criação até o momento que é encerrado. No cálculo deste tempo estão os tempos gastos com alocação de memória, espera na fila de pronto, interrupções de entrada e saída. As políticas de escalonamento procuram reduzir ao máximo este tempo.
- **Tempo de resposta:** indica o tempo gasto para produzir uma resposta a uma solicitação de uma aplicação ou sistema. Este tempo está relacionado com a velocidade dos dispositivos de entrada e saídas. As políticas de escalonamento procuram reduzir ao máximo este tempo.

CONEXÃO

Aprofunde seus conhecimentos sobre escalonamento:

<<http://www.oficinadanet.com.br/post/12781-sistemas-operacionais-o-que-e-escalonamento-de-processos>>.

3.3 Escalonamento primeiro a entrar primeiro a sair (FIFO – First - IN - First - OUT)

Os primeiros sistemas monoprogramáveis utilizavam o algoritmo de escalonamento não-preemptivo FIFO (*First-In-First-Out*) também conhecido como escalonamento FCFS (*First-Come-First-Served*). Já na arquitetura de sistemas multiprogramáveis o escalonamento FIFO se apresentou pouco eficiente. Assim sendo, houve várias mudanças na versão original para que o mesmo pudesse ser usado, de forma parcial, em sistemas de tempo compartilhado.

O escalonamento FIFO se baseia em uma fila simples do processador. Após ser criado e receber o estado de Pronto, o processo entra na fila de pronto. O escalonador sempre coloca o primeiro processo da fila para ser executado. A liberação do processador pelo processo em execução somente acontece caso o processo mude seu estado para “Em Espera”, “Terminado” ou quando ocorrer um erro na execução.

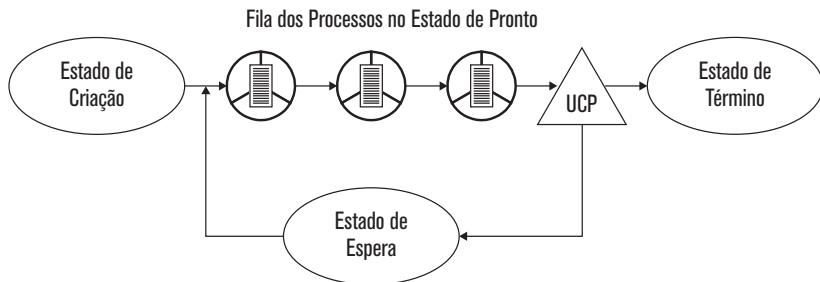


Figura 41 – Escalonamento FIFO.

Fonte: Machado e Maia (2007).

A vantagem do escalonamento FIFO é que é de fácil entendimento e implementação, além do que nunca acontecerá o fenômeno *starvation*, logo, todos os processos da fila de pronto serão executados. A desvantagem é com relação ao desempenho, o escalonamento FIFO tem grande sensibilidade à ordem de chegada dos processos para serem executados. Se os primeiros processos que estiverem na fila forem processos com tempo de execução muito grande, consequentemente teremos o Tempo Médio de Espera e o *Turnaround* muito elevado. Este cenário poderia acontecer se inicialmente chegasse à fila processos do tipo *CPU-Bound*, que fazem muito uso do processador (Ex.: programas gráficos de alta resolução).

Caso os primeiros processos forem de curta duração, o Tempo Médio de Espera e o *Turnaround* serão pequenos. Podemos citar como processos de curta duração, os processos *I/O-Bound* que fazem muito uso de dispositivos de entrada e saída e pouco do processador (Ex.: cópia de arquivos).

Dessa forma não há possibilidade de prever quando um processo entrará em execução. Por exemplo, se simularmos a ordem de chegada de três processos para serem executados, com tempos distintos, iniciando pelo processo com maior tempo, conforme a tabela e a figura a seguir, o cálculo do tempo médio de espera ficaria $(0+6+10)/3 = 5,33$.

ORDEM DE CHEGADA	PROCESSO	TEMPO DE ESPERA	TEMPO DE EXECUÇÃO
1º	P3	0	6
2º	P2	6	4
3º	P1	10	2

Tabela1 – Tempo de espera.

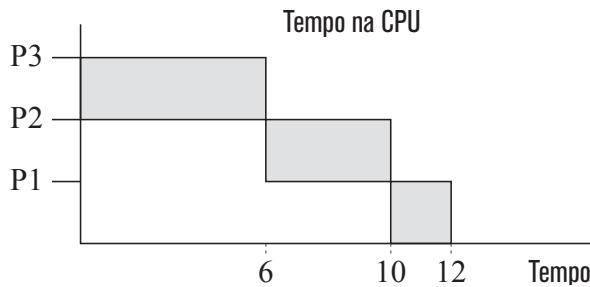


Figura 42 – Escalonamento FIFO.

Simulando a ordem de chegada dos processos iniciando com os de menor tempo, teremos o tempo médio de espera $(0+2+6)/3 = 2,66$.

ORDEM DE CHEGADA	PROCESSO	TEMPO DE ESPERA	TEMPO DE EXECUÇÃO
1º	P1	0	2
2º	P2	2	4
3º	P3	6	6

Tabela 2 – Tempo de espera.

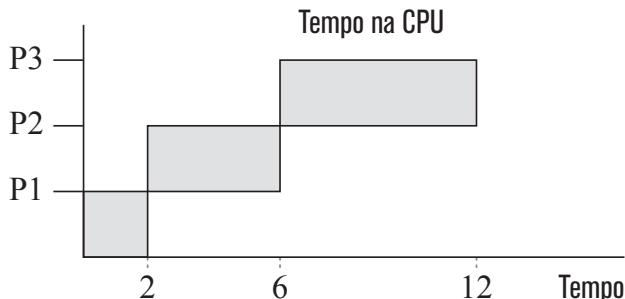


Figura 43 – Escalonamento FIFO.

3.4 Escalonamento por job mais curto primeiro (SJF – Shortest - Job - First)

Como o escalonador FIFO, o escalonador SJF (*Shortest-Job-First*) também conhecido como SPN (*Shortest-Process-Next*) foi inicialmente utilizado nos primeiros sistemas monoprogramáveis, de escalonamento não-preemptivo.

O princípio de funcionamento do escalonamento SJF também se baseia em uma fila de Pronto onde ficam os processos que aguardam por execução. O escalonador, com base em conhecimento prévio do uso do processador e estimativa do tempo de execução dos processos da fila, escolhe o processo com menor tempo de execução para ser o primeiro a ser executado. Se houver processos com tempos de execução iguais, será utilizado o critério de ordem de chegada, o primeiro a chegar será o primeiro a ser executado. Uma vez em execução, o processo não libera o processador até finalizar seu processamento.

Na tabela abaixo, o primeiro processo a chegar é o de maior tempo e o último de menor tempo. Verificando os tempos, o escalonamento SJF inicia a ordem de execução começando pelo último até o primeiro. Consequentemente teremos o tempo médio de espera $(0+2+6)/3 = 2,66$.

ORDEM DE CHEGADA	PROCESSO	TEMPO DE ESPERA	TEMPO DE EXECUÇÃO	TEMPO DE ESPERA
1º	P3	6	3º	6
2º	P2	4	2º	2
3º	P1	2	1º	0

Tabela 3 – Tempo de espera

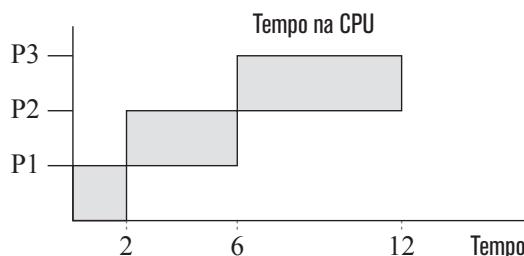


Figura 44 – Escalonamento SJF.

Para atender a necessidade de preempção, foi criada uma versão do escalonador SJF conhecida como SRTF (*Shortest-Remaining-Time-First*). Esta versão permite que o escalonador compare, o tempo restante necessário para que um processo em execução termine seu processamento, com os tempos estimados dos processos que estão na fila de pronto. Caso o tempo restante para terminar o processamento for maior que o tempo estimado de algum processo na fila, o processo em execução é substituído por um de menor tempo.

Este algoritmo de escalonamento apresenta um problema de ter que prever quanto tempo de processador um processo vai necessitar para finalizar seu processamento, e isso não é conhecido. Em geral, o sistema operacional faz uma estimativa com base no tempo anterior que o processo utilizou o processador.

A vantagem do escalonador SJF é um menor tempo médio de espera para os processos a serem executados. Esta redução de tempo médio de espera faz com que o escalonador SJF seja mais eficiente que o escalonador FIFO. Podemos citar como desvantagem a possibilidade de ocorrer *starvation* para processos do tipo *CPU-Bound*, que ocupam muito tempo de uso do processador, caso processos de tempos curtos, como os *I/O-Bound* forem continuamente adicionados para a execução.

3.5 Escalonamento circular (Round Robin)

Com o desenvolvimento dos sistemas multiprogramáveis surgiu um escalonador para atender os requisitos de sistemas compartilhado denominado *Round Robin* (escalonamento circular). No escalonamento circular, o sistema operacional determina um período de tempo que um processo pode ser executado pelo processador, este período de tempo é conhecido como fatia de tempo (*time-slice*) ou *quantum* e normalmente pode variar entre 10-100 milissegundos. Esta política de escalonamento evita que um processo não monopolize o processador. Quando o processo está utilizando o processador é dito que é um *surto do processador*. O surto pode ser com relação aos dispositivos de entrada e saída quando o processo está usando estes dispositivos. O surto de um processador, referente a um processo, pode ser menor que um *quantum*, se o tempo necessário para finalizar um processo for menor que o *quantum*.

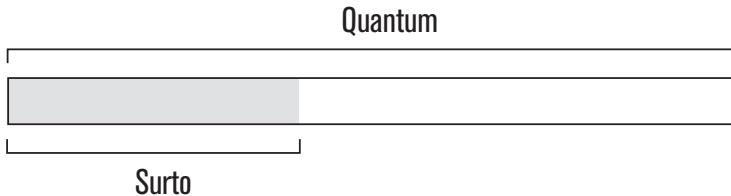


Figura 45 – Surto de um processador.

O escalonador *Round Robin* forma uma fila circular de processos com estado de pronto. Todo processo carregado na memória com o estado de pronto, sempre entra no final da fila. O escalonador, por sua vez sempre escolhe o primeiro da fila para ser executado. O processo selecionado recebe um *quantum* e passa a utilizar o processador. Ao término do *quantum*, o processo em execução é interrompido, pelo sistema operacional, e volta para o fim da fila de pronto. Situação como esta, tende a acontecer com maior frequência em processos do tipo *CPU-Bound*.

Quando este mesmo processo tiver acesso novamente ao processador, retomará a execução de onde parou no ciclo anterior. Esta forma de troca de processo levando em consideração um período de tempo é conhecida como *preempção por tempo*. A troca de um processo em execução por outro pode acontecer antes do final de *quantum*, quando o processo já tenha terminado todo o seu processamento ou voluntariamente passe para o estado de “Em Espera”.

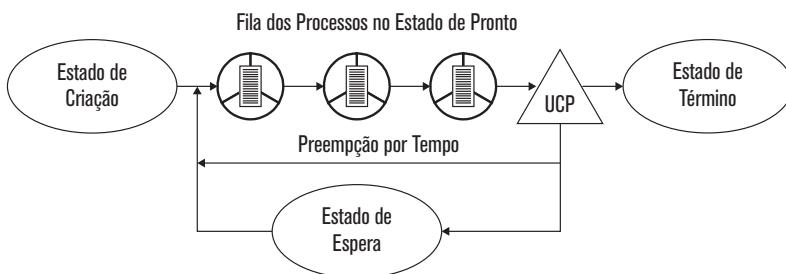


Figura 46 – Escalonamento circular.

Fonte: Machado e Maia (2007).

O exemplo a seguir tem se três processos com diferentes tempos necessários para sua execução. O tempo de *quantum* equivale a duas unidades de tempo. O gráfico de escalonamento circular ilustra a execução dos processos de acordo com o *quantum* atribuído a cada um. Note que o processo B terminou antes do tempo de *quantum*, o que causou a troca de processo.

PROCESSO	HORA DE CHEGADA	DURAÇÃO
A	0	24
B	2	3
C	1	4

Tabela 4 - Processos

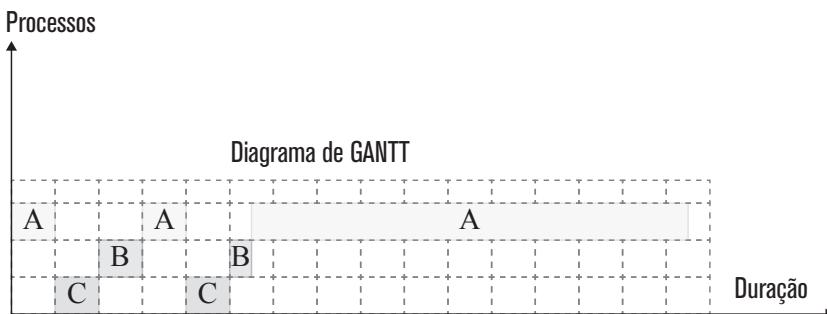


Figura 47 – Escalonamento Circular.

Um ponto crítico no desempenho do escalonamento circular é a definição do *quantum*. Além do *quantum*, o sistema operacional necessita de certa quantidade de tempo para executar a parte administrativa relacionada à troca de processos – salvar e carregar contexto de *hardware*, *software* etc. Dependendo do tempo definido para o *quantum*, pode haver maior ou menor número de trocas de contexto aumentando o tempo administrativo e consequentemente perda de eficiência.

Por exemplo, caso tenha um processo que o tempo total para ser executado é de 10 unidades de tempo e cada unidade de tempo for 20ms, então o processo necessitará de 200ms para ser executado. Para o tempo administrativo serão necessários 5ms para efetuar todo o procedimento para troca de contexto. Por fim podemos simular o *quantum* com os seguintes valores:

- f) Se o *quantum* for definido em 12 unidades de tempo, 240ms, o processo terá tempo suficiente para executar todo o seu processamento e não

haverá troca de contexto, consequentemente não haverá nenhum custo adicional de tempo administrativo.

- g) Se o *quantum* for definido em 6 unidades de tempo, 120ms, será necessário uma troca de contexto para que o processo possa finalizar o processamento. Logo haverá um custo adicional de 5ms gastos com o tempo administrativo.
- h) Se o *quantum* for definido em 1 unidade de tempo, 20ms, serão necessários 9 trocas de contexto para que o processo possa finalizar o processamento. Logo haverá um custo adicional de 45ms gastos com o tempo administrativo.

QUANTUM	TEMPO EM MS	Nº TROCA	TEMPO ADMINISTRATIVO
a	12	240	0
b	6	120	5
c	1	20	45

Tabela 5 - Tempo Quantum.



Figura 48 – Quantidade de quantum para executar um processo.

Conclui-se que, com o tempo definido para o *quantum* for muito grande, este tempo poderá ser maior que qualquer tempo dos processos na fila de pronto o que causaria o escalonador circular funcionar como se fosse um escalonador FIFO. Se o *quantum* for muito pequeno, haverá muitas trocas de contexto, e assim, haverá grande perda de eficiência do sistema.

A troca de contexto não acontece somente quando o *quantum* é muito pequeno, mas também em processos do tipo *I/O-Bound*. Processos do tipo *I/O-Bound* entram com maior frequência em estado de “Em Espera” pelo fato de efetuarem constantes acessos aos dispositivos de entrada e saída. Diferenças de características dos tipos de processos *CPU-Bound* e *I/O-Bound* tendem a ocasionar um desbalanceamento de uso do processador.

Com o objetivo de balancear o processador, surgiu uma versão aprimorada do escalonador circular chamado *escalonamento circular virtual*. O *escalonamento circular virtual* criou uma segunda fila auxiliar de pronto que recebe os processos que saem do estado de “Em Espera”. O escalonador sempre verifica esta fila quando há necessidade de troca de contexto. Caso haja processo na fila auxiliar, este terá prioridade em relação a um processo da fila de pronto. Se a fila auxiliar estiver vazia, o escalonador utilizará processos da fila de pronto. Este mecanismo de troca dinâmica da ordem em que os processos são executados é conhecido como *mecanismo adaptativo*.

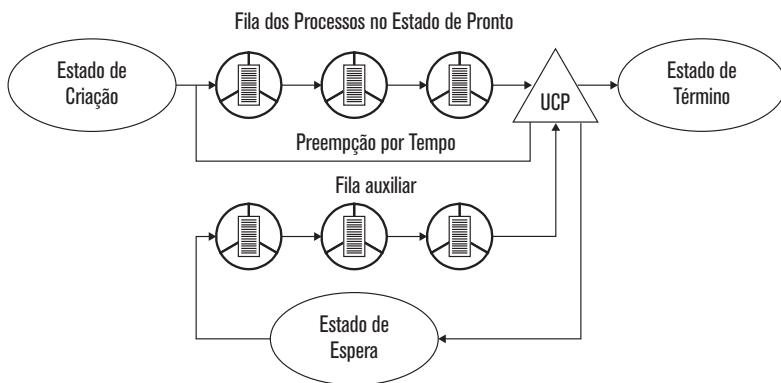


Figura 49 – Escalonamento circular Virtual.

Fonte: Machado e Maia (2007).

Os tempos atribuídos para os processos da fila auxiliar são diferentes da fila de tempo. O cálculo de tempo que o processo da fila auxiliar recebe para a execução (Figura 50 (b)), leva em conta o seu último tempo de ocupação do processador (Figura 50 (a)).

Este ajuste dinâmico do escalonamento dos processos para

Tempo recebido após fila auxiliar = *Quantum* – Último tempo em execução

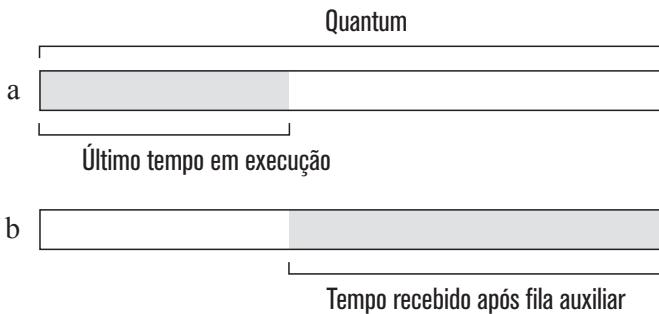


Figura 50 – Tempo recebido após fila auxiliar.

3.6 Escalonamento por Prioridade

As prioridades de escalonamentos de processos podem ser definidas com bases em diferentes critérios. Estudamos três deles, a ordem de chegada (ex.: escalonador FCFS), menor tempo de execução (ex.: escalonador SJF), troca dinâmica (ex.: escalonador *Round Robin*). Maziero (2014) define outros critérios que podem ser usados na definição de prioridades, podendo ser o comportamento do processo, se este é em lote, interativo ou processo de tempo-real, como também o proprietário do processo, podendo ser administrador, gerente ou estagiário.

Em geral, as prioridades podem ser definidas com base em critérios internos ou externos. No caso de critérios internos, as prioridades dos processos são dimensionadas com base em algum valor de grandeza, tempo de médio de utilização do processador, tempo médio de utilização de dispositivos de entrada e saída, ordem de chegada, requisitos de memória etc. Já as prioridades externas são definidas por critérios que não têm interferência do sistema operacional, como por exemplo, tipo de processo, departamento responsável, custo, etc.

O escalonamento por prioridade associa um valor, chamado *prioridade de execução*, a um processo que entra na fila de pronto, o processo com a maior prioridade será o primeiro a ser escalonado para execução. Caso haja dois ou mais processos com o mesmo nível de permissão, será escalonado o primeiro entre eles (escalonamento FIFO). Isto faz com que o escalonamento por prioridade seja o ideal para sistemas de tempo real, aplicações de controle de processo e quando a necessidade de definir a prioridade de determinados processos em sistemas de tempo compartilhado.

O valor da prioridade é um número que pode variar de sistema para sistema. Segundo Silberschatz et al. (2004), em alguns sistemas a faixa de prioridades varia entre 0 a 7 e em outros de 0 a 4095. O valor 0 em alguns casos define a prioridade mais baixa e em outros, a mais alta. No exemplo a seguir o valor 0 define a prioridade mais baixa.

PROCESSO	TEMPO DE PROCESSADOR (u.t)	PRIORIDADE
A	10	2
B	4	1
C	3	3

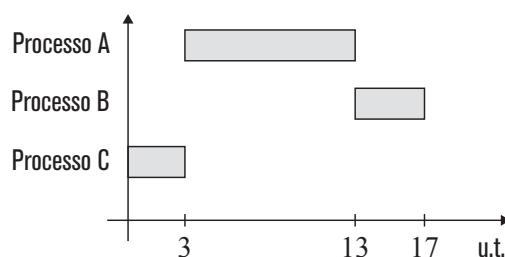


Figura 51 – Exemplo de Escalonamento por prioridades.

Fonte: Machado e Maia (2007).

Diferente do escalonador circular, que divide o tempo do processador para uso dos processos, no escalonamento por prioridade não existe este conceito de fatia de tempo, o que pode deixar um processo com maior prioridade utilizando o processador por tempo indeterminado (*starvation*) e desta forma processos de menor prioridade podem nunca serem executados.

Logo, para que um processo de maior prioridade não utilize o processador por tempo indeterminado (*starvation*) é utilizada uma estratégia conhecida como *aging* (*envelhecimento*). A estratégia *aging* pode ser implementada de duas formas segundo alguns autores. A implementação, segundo Tanenbaum e Woodhyll (1999), consiste em *diminuir gradativamente* a prio-

ridade do processo em execução a cada interrupção de relógio. Desta forma, a prioridade do processo em execução, em um determinado momento, ficará abaixo da prioridade de um processo na fila de pronto o que causará a perda do uso do processador. Esta estratégia é denominada *preempção por prioridade*. Já para Silberschatz et al. (2004) a estratégia *aging* consiste em aumentando gradativo da prioridade dos processos que estão aguardando durante muito tempo na fila de pronto. Quando um processo da fila de pronto atingir um valor de prioridade maior do que o processo em execução, o processo em execução será substituído pelo da fila de pronto.

! ATENÇÃO

Silberschatz et al. (2004) relata um fato de starvation que ocorreu em 1973 quando o IBM 7094 do M1T foi desligado. Ao desligar o IBM 7094, foi encontrado um processo de baixa prioridade que aguardava ser executado desde 1967.

O processo pode perder o uso do processador caso haja a alteração de seu estado de “Em Execução” para “Em espera” ou quando entrar um processo de maior prioridade na fila de pronto. O escalonamento por prioridade pode ter diversas filas de estado de pronto com prioridades distintas. Os processos na fila com maior prioridade tem preferência no uso do processador.

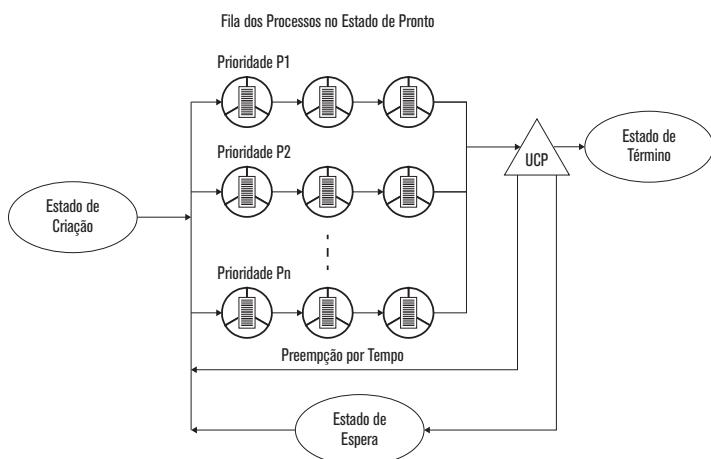


Figura 52 – Exemplo de fila de prioridades.

Fonte: Machado e Maia (2007).

O escalonador de prioridade pode ser implementado de forma mais simples no modelo não-preemptivo. No modelo não-preemptivo, se chegar um processo na fila de pronto com prioridade maior do que o processo em execução, não ocasionará a troca de processo. Neste caso, o processo que chegou à fila de pronto com maior prioridade, será colocado no início da fila. A prioridade de execução pode ser classificada como:

- Estática: o valor de prioridade é associado ao processo no momento de sua criação e não é alterado durante o seu ciclo de vida.
- Dinâmica: o valor de prioridade associado ao processo pode variar durante seu ciclo de vida levando em consideração critérios estabelecidos pelo sistema operacional.



CONEXÃO

Recomendamos a leitura deste artigo para melhor compreensão sobre escalonamento de processos: <http://www.soffner.com.br/Semana5_SOII.htm>.

3.6.1 Escalonador circular com prioridades

O processador, no escalonamento por prioridade, fica desbalanceado em relação dos processos *CPU-Bound* e *I/O-Bound*. Processos do tipo *I/O bound* tende a utilizar mais recursos de entrada e saída, o que faz com que o mesmo passe a maior parte do tempo em estado de “Em Espera” aguardando que uma solicitação de entrada e saída termine. Um melhor balanceamento do processador foi alcançado com a implementação do *escalonador circular com prioridades*. O escalonador circular com prioridades adicionou, à já existente prioridade de processo, o conceito de fatia de tempo. Desta maneira um processo pode perder o uso do processador quando houver um processo com maior prioridade ou o seu *quantum* finalizar ou quando entrar em estado de “Em Espera”.

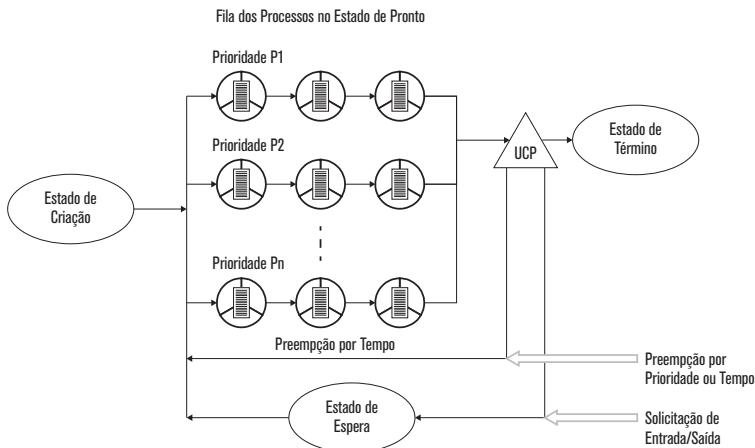


Figura 53 – Escalonamento circular com preempção por tempo ou prioridade.

Fonte: Machado e Maia (2007).

A classificação do escalonador circular com prioridades pode ser:

- Estática: o valor de prioridade é associado ao processo no momento de sua criação e não é alterado durante o seu ciclo de vida.
- Dinâmica: o valor de prioridade associado ao processo pode variar durante seu ciclo de vida levando em consideração critérios estabelecidos pelo sistema operacional.

O tipo escalonador circular com prioridades dinâmicas melhora o平衡amento entre os processos *CPU-Bound* e *I/O-Bound* mencionado anteriormente. Como os processos *I/O-Bound* tende a ficar em estado de “Em Espera” muito tempo, para compensar este tempo, a sua prioridade é alterada para um valor maior, assim quando saí do estado “Em Espera”, entra em uma fila de pronto com prioridades mais altas. Consequentemente, o processo será atendido mais rapidamente pelo escalonador.

3.6.2 Escalonamento por múltiplas filas

Um dos mais antigos escalonadores de prioridade é o escalonador de múltiplas filas. O escalonador de múltiplas filas foi utilizado no Sistema de compartilhamento por tempo CTSS (*Compatible Time-Sharing System* – Sistema compatível de divisão por tempo), desenvolvido pelo Centro Computacional do MIT em 1961 e utilizado no IBM 7090. A implementação do Escalonamento por múltiplas filas baseia-se em diversas filas de pronto, cada qual com um nível de prioridade.

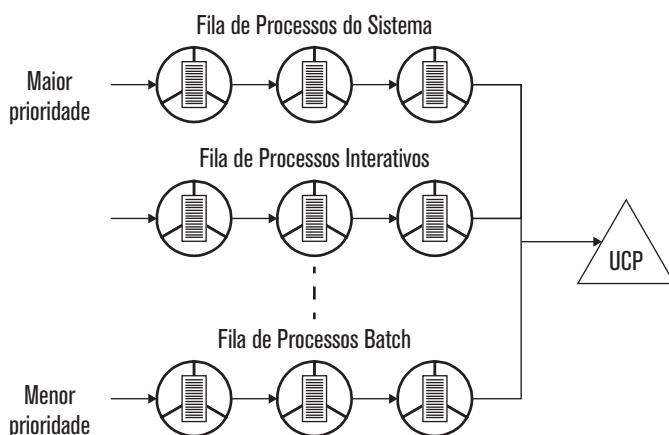


Figura 54 – Múltiplas filas de escalonamento.

Fonte: Machado e Maia (2007).

Neste sistema, o processo não possui prioridade, a prioridade é característica da fila, assim, o sistema usa como critério uma ou mais propriedades de cada processo para associá-lo a uma desta fila.

Como vimos anteriormente, os processos são divididos em três categorias, interativos (*foreground*), batch (*Background*) e *daemons*, sendo que cada tipo tem uma necessidade distinta de uso do processador de acordo com o tempo de resposta exigida para cada um deles. Esta necessidade, de tempo de resposta, é um exemplo da necessidade de um escalonamento diferenciado para cada tipo de processo. Para Silberschatz et al. (2004) a escolha de qual fila um determinado processo deve entrar, leva em consideração alguma propriedade do processo, podendo ser o tipo de processo, sua prioridade, tamanho de memória, etc. Esta associação acontece no momento da criação do processo e permanece até o fim

do seu processamento. Isto pode ser uma desvantagem, pelo fato do sistema operacional não levar em conta que alguns processos podem alterar suas propriedades no decorrer de sua execução e assim poderem ser agendados em filas de maior ou menor prioridade.

Para atender as necessidades acima descritas para os processos, cada fila tem seu próprio algoritmo de escalonamento. Por exemplo, processos do tipo *foreground* podem ser agendados em uma fila cujo escalonador seja do tipo escalonamento circular, enquanto processos do tipo *background* em fila com algoritmo do tipo FIFO. No escalonamento por múltiplas filas, um processo de uma fila de menor prioridade somente será escalonado se não houver processos em filas de maior prioridade. Por exemplo, se tivéssemos as filas conforme a Figura 55, e algum estudante executasse algum programa, o processo deste programa entraria na fila “Processos de estudantes”. Como as filas acima da de “Processos de Estudantes” tem prioridade absoluta, o processo somente seria escalonado se as demais filas estivessem completamente vazias. Uma vez em execução, o processo do estudante pode sofrer preempção caso um processo entre em uma das filas de maior prioridade.

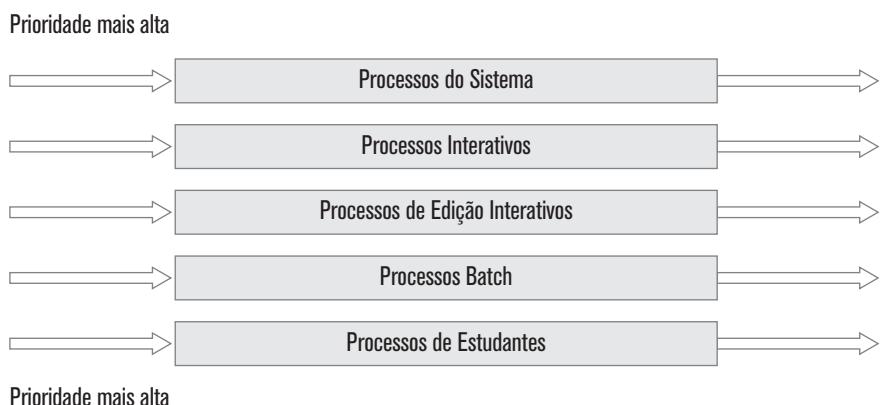


Figura 55 – Escalonamento com múltiplas filas de prioridade.

Para que os processos, como o do estudante, não fique indefinidamente sem execução, caso as filas acima estejam sempre ocupadas, existe a possibilidade de dividir o tempo de processamento entre as diversas filas. Esta divisão pode ser feita, por exemplo, atribuindo 80% do tempo do processador para as filas de processos do tipo *foreground*, com escalonamento circular, e 20% para

os processos do tipo *background*, com escalonamento do tipo FIFO. As filas por sua vez, dividem o tempo recebido entre os seus processos.

3.7 Escalonamento por Múltiplas filas com realimentação

A vantagem do escalonamento por múltiplas filas é permitir um escalonamento de baixo custo, mas por outro lado, tem a desvantagem de ser inflexível em relação a mudança de processo entre as filas.

Vimos que o sistema operacional analisa as propriedades de um processo no momento de sua criação e associa-o permanentemente a uma das filas de prioridade. Uma vez atribuído a uma fila, o processo não tem como se mover para outra. De maneira geral, isso é coerente se levarmos em conta a natureza dos processos (*foreground* e *background*). Se um processo é do tipo *foreground*, será associado a uma fila com prioridades de *foreground*, se for *background* será associado a uma fila com prioridades de *background*.

O problema dessa abordagem é que processos do tipo *CPU-Bound*, em geral tem maior prioridade do que processos do tipo *I/O-Bound*. Se houver muitos processos *CPU-Bound* entrando numa fila de maior prioridade, um processo do tipo *I/O-Bound* pode ter que aguardar um tempo excessivo para ser escalonado. A solução deste problema veio com o escalonamento *Escalonamento por Múltiplas filas com Realimentação*. Este escalonador permite que um processo possa trocar de fila durante seu processamento. São implementadas diversas filas com níveis de prioridades distintas. O comportamento do processo é analisado dinamicamente pelo sistema que então ajusta sua prioridade de execução e mecanismo de escalonamento. Esta análise de comportamento é feita, pelo sistema, ao longo do ciclo de vida do processo, permitindo que o mesmo possa ser direcionado para uma determinada fila de acordo com sua mudança de comportamento. As filas apresentam dois tipos de escalonamento. A fila de menor prioridade utiliza o escalonamento circular. As demais filas, de maior prioridade, apresentam um escalonamento FIFO adaptado com fatia de tempo. Um processo que estiver em uma fila de menor prioridade somente será escalonado quando as filas de maior prioridade estiverem vazias. A fatia de tempo, atribuída a uma fila, é dimensionada de acordo com a prioridade da fila. Uma fila com prioridade maior terá menos tempo de processamento e fila com prioridade menor, mais tempo de processamento. Este mecanismo, do escalonador por Múltiplas filas com realimentação, que efetua ajustes conforme as mudanças no comportamento do processo é

chamado de *mecanismo adaptativo*. Em geral, os mecanismos adaptativos geram sobrecarga do sistema, mas no caso dos resultados obtidos no escalonador por Múltiplas filas com realimentação, com estes mecanismos, justificam este aumento de sobrecarga (DEITEL et al., 2005).

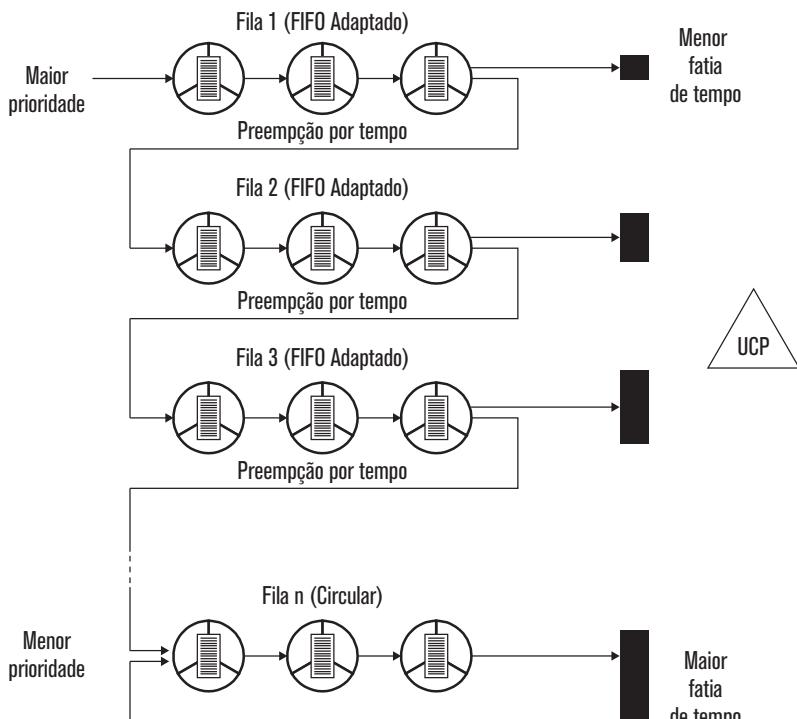


Figura 56 – Escalonamento por múltiplas filas com realimentação.

Fonte: Machado e Maia (2007).

O processo ao ser criado, não é associado à fila de pronto e sim direcionado para o final da fila de maior prioridade. Como cada fila tem uma determinada fatia de tempo, é atribuído ao processo uma parte deste tempo. Ao chegar ao início da fila, o processo é escalonado e obtém acesso ao processador. Se durante a sua execução, ocorrer a chegada de um processo em uma fila de maior prioridade, o processo sofrerá a preempção para dar lugar ao processo de prioridade mais alta.

A preempção por prioridade ou por solicitação a um recurso do sistema faz com que o processo seja associado para o final da mesma fila que estava anteriormente. Caso a preempção do processo for por tempo, ou seja, esgotou o *quantum* atribuído a ele, o processo é associado ao final de uma fila de menor prioridade. Esta estratégia faz com que processos do tipo *CPU-Bound* entrem na fila de maior

prioridade e ganhe rapidamente o uso do processador. Como sua característica é de utilizar o maior tempo possível o processador, irá gastar seu *quantum* de tempo e será associado a uma fila de menor prioridade. Já os processos *I/O-Bound* conseguem ter um melhor tempo de resposta, isto porque este tipo de processo tem uma grande tendência de sofrer preempção por solicitação a um recurso do sistema, e com isso, voltar para o final da mesma fila de prioridade de onde saiu, permanecendo um maior tempo em filas com prioridades mais altas.

3.8 Cálculo estimado de tempo de resposta.

Estudamos nas seções anteriores diversos tipos de escalonadores e suas características, tais como modo de funcionamento, parâmetros, etc. Estas características podem definir a escolha do tipo de escalonador para um determinado sistema. Outros critérios podem ser utilizados levando em conta a máxima utilização do processador, tempo de resposta ou *Throughput* (número de processos executados em um determinado intervalo de tempo).

Uma vez os critérios definidos, podemos utilizar um determinado método para avaliar os escalonadores. Silberschatz et al. (2004) cita quatro métodos de avaliação: modelagem determinista, modelos de filas, simulações e implementação. Para o nosso estudo do cálculo estimado do tempo de resposta, utilizaremos a modelagem determinista. A modelagem determinista é um tipo de avaliação analítica. Segundo Silberschatz et al. (2004) a avaliação analítica permite obter um número que define o desempenho do escalonador a partir de um volume de trabalho predeterminado. A tabela 6 determina o tempo estimado de processador e a ordem de chegada dos processos. Será utilizado as seguintes siglas: T_r = Tempo de Resposta e T_e = Tempo Estimado de CPU.

PROCESSO	TEMPO ESTIMADO DE CPU
P1	7
P2	3
P3	5
P4	4

Tabela 6 - Tempo estimado de processador.

Utilizando o escalonador FIFO, temos:

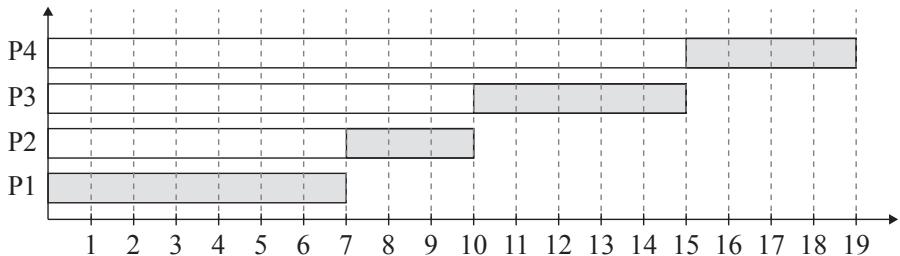


Figura 57 – Escalonador FIFO.

Tempo de Resposta

Processo P1 → $T_r = 7$	→	$T_r = 7\text{ms}$
Processo P2 → $T_r = 7 + 3$	→	$T_r = 10\text{ms}$
Processo P3 → $T_r = 7 + 3 + 5$	→	$T_r = 15\text{ms}$
Processo P4 → $T_r = 7 + 3 + 5 + 4$	→	$T_r = 19\text{ms}$

Utilizando o escalonador SJF, teríamos:

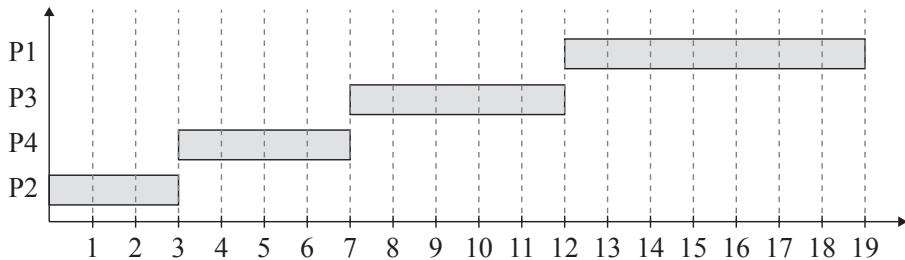


Figura 58 – Escalonador SJF.

Tempo de Resposta

Processo P1 →	$T_r = 3 + 4 + 5 + 7 \rightarrow$	$T_r = 19\text{ms}$
Processo P2 →	$T_r = 3$	$\rightarrow T_r = 3\text{ms}$
Processo P3 →	$T_r = 3 + 4 + 5$	$\rightarrow T_r = 12\text{ms}$
Processo P4 →	$T_r = 3 + 4$	$\rightarrow T_r = 7\text{ms}$

Utilizando o escalonador RR, temos:

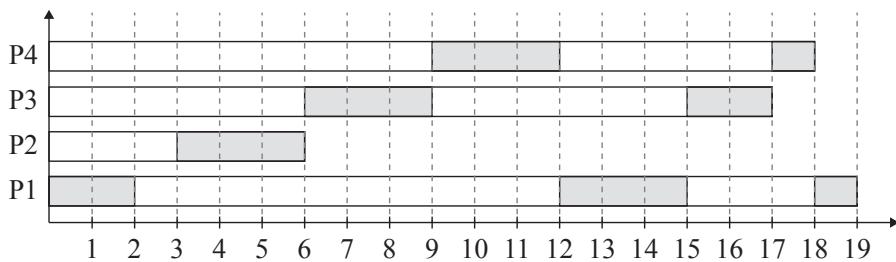


Figura 59 – Escalonador RR.

Tempo de Resposta

Para o cálculo utilizaremos Quantum = 30m.

$$\text{Processo P1} \rightarrow T_r = 3 + 3 + 3 + 3 + 3 + 2 + 1 + 1 \rightarrow T_r = 19\text{ms}$$

$$\text{Processo P2} \rightarrow T_r = 3 + 3 \rightarrow T_r = 6\text{ms}$$

$$\text{Processo P3} \rightarrow T_r = 3 + 3 + 3 + 3 + 3 + 2 \rightarrow T_r = 17\text{ms}$$

$$\text{Processo P4} \rightarrow T_r = 3 + 3 + 3 + 3 + 3 + 2 + 1 \rightarrow T_r = 18\text{ms}$$

ATIVIDADE

1. De acordo com o material, o que é política de escalonamento de um processo?
2. Qual a diferença entre o escalonamento preemptivo e não-preemptivo?
3. Descreva as principais diferenças entre os escalonamentos do tipo Circular e FIFO?

REFLEXÃO

Neste capítulo estudamos as diversas políticas de escalonamento e suas funções básicas. Abordamos as características de diversos escalonamentos, tais como o FIFO, SJF, Round Robin, por prioridades, circular com prioridade, assuntos mais densos e de relativa complexidade. Estudando estas características podemos definir a escolha do tipo de escalonador para um determinado sistema. Por fim vimos alguns métodos para avaliar os escalonadores. Todos estes conhecimentos, certamente serão imprescindíveis para sua vida profissional.



LEITURA

Para você avançar mais o seu nível de aprendizagem envolvendo os conceitos de sistemas operacionais e demais assuntos deste capítulo, consulte as sugestões de *links* abaixo:

Capítulo 8 do livro

DEITEL H. M.; DEITEL P. J.; CHOIFFNES D. R. Sistemas Operacionais. 3^a ed. São Paulo, Editora Prentice-Hall, 2005.



REFERÊNCIAS BIBLIOGRÁFICAS

DEITEL H. M.; DEITEL P. J.; CHOIFFNES D. R. Sistemas Operacionais. 3^a ed. São Paulo, Editora Prentice-Hall, 2005.

HENNESSY, J. L.; PATTERSON, D. A. Arquitetura de Computadores: uma abordagem quantitativa. 3^a ed. Rio de Janeiro: Campus, 2003.

MACHADO, F. B.; MAIA, L. P. Arquitetura de sistemas operacionais. 4^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2007.

MAZIERO, C. A. Sistemas Operacionais: Conceitos e Mecanismos. Disponível em: <<http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/so:so-livro.pdf>>. Acesso em: set. 2014.

OLIVEIRA, R. S.; CARISSIMI, A. S.; TOSCANI, S. S. Sistemas Operacionais. 4^a ed. Porto Alegre : Editora Bookman, 2010.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G Fundamentos de Sistemas Operacionais. 6^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2004.

TANENBAUM, A. S.; WOODHYLL, A. S. Sistemas operacionais projeto e implementação. 2^a ed Porto Alegre: Bookman, 1999.



NO PRÓXIMO CAPÍTULO

No capítulo seguinte, estudaremos as estruturas de memória. Você entenderá as diferenças entre os espaços de endereçamento físico e lógico. Serão apresentados os diversos modos de alocação da memória.

4

Gerência de Memória

4 Gerência de Memória

Estudamos que, na multiprogramação, vários processos são mantidos na memória a fim de que o sistema operacional tenha um grande desempenho. Os programas, cada vez mais, têm a necessidade de utilizar uma quantidade maior de memória, sem contar a exigência de memórias com acesso cada vez mais rápido. Este cenário faz com que o sistema operacional procure efetuar um gerenciamento da memória o mais eficiente possível, começando por alocar o máximo possível de processos na memória, controlar os processos dos usuários e ter o cuidado de não ocupar muita memória. Estes são apenas alguns pontos para começarmos a refletir sobre o gerenciamento de memória. Os conceitos que iremos estudar sobre o assunto com certeza serão de grande importância para a sua formação.



OBJETIVOS

- Expor as estruturas de memória.
- Entender as diferenças entre os espaços de endereçamento físico e lógico.
- Compreender os diversos modos de alocação da memória.



REFLEXÃO

Você se lembra de como os processos são escalonados, certo? Estes conceitos nós estudamos no capítulo 3. A partir deste momento, veremos quais as estratégias que o sistema operacional utiliza para a alocação dos processos na memória.

4.1 Funções

Estudamos anteriormente que um processo é um programa em execução e que os sistemas operacionais multiprogramáveis permitem carregar mais de um programa em memória e executá-los. Isto faz com que a memória seja responsável pelo *armazenamento principal* do sistema operacional, onde são carregados os programas, e os dados que o que os programas acessam, durante a sua execução.

Dependendo do tamanho do programa, ele pode estar armazenado parcial ou total na memória para ser executado. Assim sendo, há uma grande necessi-

dade de gerenciar a alocação da memória a fim de que a mesma seja utilizada da forma mais eficiente possível, para isso o gerenciador de memória deve tentar manter a memória com o máximo de programas (processos) em execução.

Com vários programas em memória, o gerenciador de memória deve garantir que um programa não acesse áreas destinadas a outro programa, como também das áreas destinadas ao sistema operacional. Em situações onde os processos têm a necessidade de trocar dados, o gerenciador de memória oferece mecanismos protegidos de compartilhamento de dados.

Outro ponto a ser considerado é de que, mesmo com toda a evolução tecnológica, proporcionando às memórias um grande aumento de sua capacidade de armazenamento e velocidade de acesso, os programas também evoluíram na mesma proporção, exigindo que o sistema operacional gerencie a memória disponível para que um determinado programa não venha a ocupar toda a memória disponível.

Em geral, os programas devem estar armazenados permanentemente no computador, em um meio não-volátil, ou seja, a memória principal não é adequada para esta tarefa pelo fato de ser do tipo volátil, isto é, se a energia for desligada, todos os dados se perdem.

Este fato faz com que os discos rígidos, que são não voláteis, de baixo custo e de grande capacidade, seja o meio utilizado pelo sistema operacional, como sistema de *armazenamento secundário*, armazenando os programas e os seus dados.

É muito importante frisar que o processador somente executa instruções que estão carregadas na memória principal, assim, o gerenciador de memória tem a função de transferir os programas, que serão executados, do armazenamento secundário (discos rígidos) para o armazenamento principal (memória). Esta transferência do programa da memória secundária para principal deve ser gerenciada para que não cause perda de desempenho do sistema. Esta perda de desempenho pode ocorrer se houver uma grande quantidade de acesso à memória secundária. O tempo de acesso à memória secundária é muito mais lento se comparado com a memória principal. Dessa forma o sistema operacional deve otimizar a quantidade de operações de entrada e saída da memória secundária. Pode acontecer, da memória principal estar totalmente ocupada o que faz com que o sistema operacional utilize um mecanismo conhecido como *swapping*. O mecanismo de *swapping* transfere os processos da memória principal, temporariamente para a memória secundária, permitindo que novos processos sejam carregados na memória principal e executados.

Outras funções do gerenciado de memória é controlar os espaços que serão utilizados pelos programas e os que não serão, alocar espaços para os dados, conforme os programa solicitem e desalocar quando não mais são necessários. Nesta unidade estudaremos os componentes de *hardware* que compõem a memória e os mecanismos de controles utilizados pelo sistema operacional para o gerenciamento da memória

CONEXÃO

Aprofunde seus conhecimentos sobre gerência de memória.

<<http://regulus.pcs.usp.br/~jean/so/AULA%2013%20-%20Ger%C3%Aancia%20de%20Mem%F3ria.pdf>>.

4.2 Estruturas de memória

O armazenamento principal (memória) e o armazenamento secundário (disco rígido) faz parte de uma *hierarquia de memória*. Outros componentes do computador, utilizados para armazenar dados, fazem parte desta hierarquia de memória, entre eles estão os dispositivos de armazenamento externos como as fitas magnéticas, *pendrives*, *CD-Roms*, *DVD-Roms*, *Blu-ray*, etc. e os internos como os registradores e o cache interno do processador, o cache externo da placa-mãe, etc. (MAZIERO, 2014). Devido às características de cada componente, o sistema operacional deve fornecer diversas funcionalidades para permitir que os programas controlem todos os aspectos destes componentes. Aspectos no sentido de componentes que podem ser acessados somente para leitura ou leitura e escrita. Outros são de acesso sequencial ou aleatório, síncrono ou assíncrono, etc. Outro fator que pode influenciar o sistema como um todo é a velocidade de acesso a estes componentes. Patterson e Hennessy (2005) detalha, na tabela abaixo, o tempo de acesso de cada componente e sua taxa de transferência. O tempo de acesso indica quanto tempo a memória gasta para transferir uma informação para o barramento de dados após uma determinada ação. Uma vez iniciada a transferência, uma determinada quantidade de *bytes* por segundo podem ser lidos ou escritos em um meio de armazenamento. A esta quantidade de *bytes* por segundo lidos ou escrito por segundo e dado o nome de taxa de transferência.

MEIO	TEMPO DE ACESSO	TAXA DE TRANSFERÊNCIA
Cache L2	1 ns	1 GB/s (1 ns/byte)
Memória RAM	60 ns	1 GB/s (1 ns/byte)
memória flash (NAND)	2 ms	10 MB/s (9100 ns/byte)
Disco rígido IDE	10 ms (tempo necessário para o deslocamento da cabeça de leitura e rotação do disco até o setor desejado)	80 MB/s (12 ns/byte)
DVD-ROM	de 100 ms a vários minutos (caso a gaveta do leitor esteja aberta ou o disco não esteja no leitor)	10 MB/s (100 ns/byte)

Tabela 7 - Tempos de acesso e taxas de transferência.

Fonte: Patterson e Hennessy (2005)

Levando em consideração as principais características dos componentes de armazenamento, como custo, capacidade de armazenamento, tempo de acesso e taxa de transferência, é possível definir uma pirâmide de hierarquia de memória.

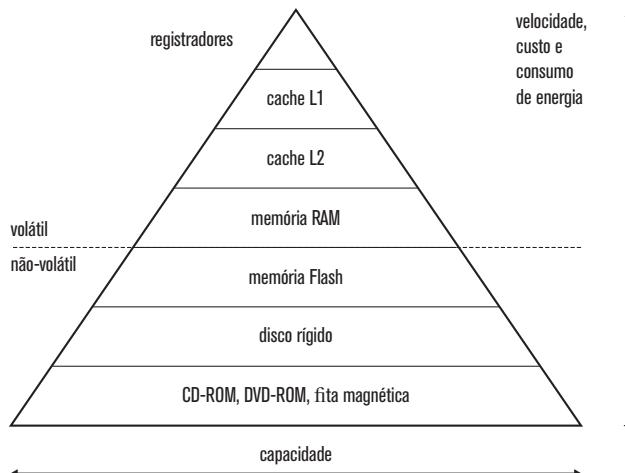


Figura 60 – Pirâmide clássica de hierarquia de memória.

Na base pirâmide estão os componentes que armazenam informações acessadas com menor frequência pelo processador. Isto permite utilizar componentes de custo mais baixo e alta capacidade de armazenamento. Por outro lado, são consideradas como memórias mais lentas e *não-voláteis*. No topo da pirâmide temos os componentes que armazenam informações acessadas com maior frequência pelo processador. São memórias com custo elevado e baixa capacidade de armazenamento. No entanto, são memórias extremamente rápidas, apesar de serem voláteis.

4.3 Espaço de Endereçamento Físico e Lógico

Os processos sendo executados no processador possuem instruções que fazem referências a endereços de memória. Estes endereços de memória gerados pelo programa são chamados *endereços lógicos* ou *endereços virtuais*. Os endereços lógicos estão relacionados ao programa e não necessariamente são iguais aos endereços na memória real do computador. Como o processador apenas pode acessar posições de memória principal, ou seja, endereços físicos, o endereço lógico é traduzido para endereço físico. Esta tradução é conhecida como *mapeamento*. Para que não haja perda de desempenho, o mapeamento é feito por um dispositivo de *hardware* conhecido como Unidade de Gerenciamento de Memória (*MMU - Memory Management Unit*) figura 61.

A figura 61 exibe um exemplo de mapeamento de endereço lógico para físico segundo Silberschatz et al. (2004). Neste exemplo, a Unidade de Gerenciamento de Memória possui um Registrador de Relocação que está apontando para o endereço físico de valor igual a 14000. O valor do registrador de relocação sempre é somado aos endereços de memória que estão sendo acessados pelo processo do usuário em execução no processador. Assim, se o processo está fazendo referência ao endereço lógico de valor 0 ao passar pelo MMU, será mapeado para o endereço físico de valor 14000. Caso o processo faça uma referência ao endereço lógico de valor 346 será mapeado para o endereço físico de valor 14360. Mesmo que o usuário crie um ponteiro no programa para acessar o endereço físico de memória 346, não acessará esta posição física, pois os programas de usuários tratam de endereços lógicos e nunca acessam diretamente os endereços físicos reais. Existem outros métodos diferentes de mapeamento que serão abordados nas próximas seções.

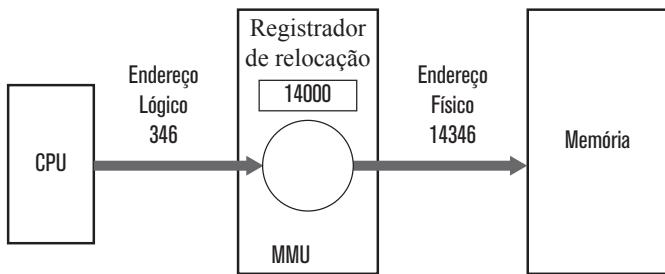
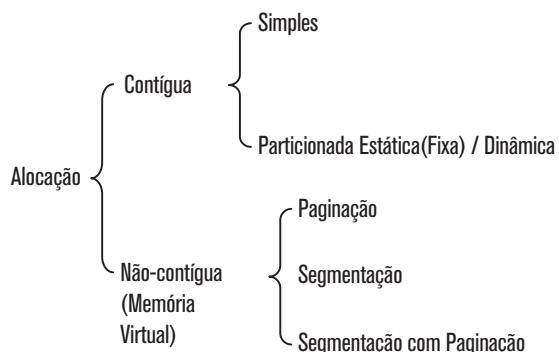


Figura 61 – Tradução de endereço lógico para físico.

Dados e instruções de programas podem ser associados a endereços de memória em 3 momentos: em tempo de compilação, em tempo de carga e em tempo de execução. Em tempo de compilação se a posição da memória for conhecida. Neste caso os endereços podem ser associados em valores absolutos. Em tempo de carga os endereços serão gerados em código relocável. E o último e mais utilizados na maioria dos sistemas operacionais, é o em tempo de execução. Neste caso a associação somente é feita no momento que o processo for executado.

4.4 Estratégias de alocação

Nas seções seguintes, estudaremos as formas de alocação de memória. A alocação de memória pode ser organizada dos seguintes formas:



A alocação contígua será estudadas nos próximos tópicos e a alocação não contígua no tópico de Memória Virtual.

4.4.1 Alocação contígua simples

Os primeiros sistemas monoprogramáveis utilizava a forma de alocação contígua de memória. Como os sistemas monoprogramáveis permitiam que apenas um programa fosse carregado na memória por vez, a alocação contígua era a forma mais simples de gerenciamento de memória consistindo de dividir a memória entre o sistema operacional e o programa carregado. Tanenbaum e Woodhyll (1999) apresenta três formas de organizar a memória para acomodar o sistema operacional e um programa do usuário. A figura 62(a) e (b) utiliza uma abordagem somente com memória RAM(*Random Access Memory* – Memória de Acesso Aleatório). Notem que o sistema operacional pode ser alocado tanto na parte baixa como na parte alta da memória.

! ATENÇÃO

Em geral, o sistema operacional é alocado na parte baixa da memória devido a posição do vetor de interrupções que normalmente ocupa a memória baixa. O vetor de interrupções é uma tabela que contém os endereços das rotinas de tratamento de interrupção (MAZIERO, 2014).

Na configuração da figura 62 (a) o sistema operacional é alocado na parte baixa da memória ficando o restante da memória disponível para o programa a ser executado. Já na configuração (b) ocorre o inverso, o sistema operacional é alocado na parte superior e o programa na inferior. A figura 62 (c) mostra uma configuração utilizando uma memória RAM e uma ROM (*Read-Only Memory* – Memória Apenas de Leitura). Nesta abordagem os *drivers* de dispositivo ficam alocados na parte superior da memória ROM e o sistema operacional e o programa do usuário na memória RAM.

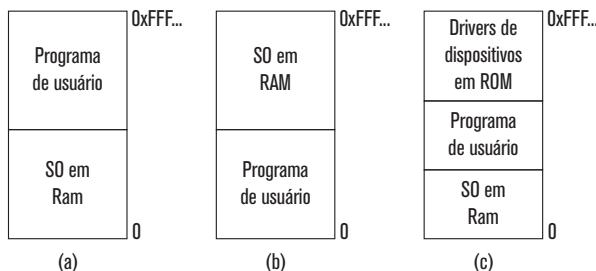


Figura 62 – Formas de organização da memória.

Os antigos computadores de grande porte utilizavam a estrutura da figura 62 (a). A abordagem da figura 62 (b) tem sido utilizada em muitos sistemas embarcados e estrutura da figura 62 (c) foi utilizada por pequenos sistemas MS-DOS que denominava a parte do sistema na ROM de BIOS (*Basic Input Output* – Sistema Básico de Entrada e Saída). Nesta forma de alocação, o usuário pode acessar qualquer endereço de memória, ele tem o controle total sobre a memória principal. Esta característica permite que um determinado programa possa, de forma intencional ou não, acessar a área destinada ao sistema operacional o que poderia causar um colapso do sistema. Para resolver este problema foi adotado um registrador (figura 63) que tem a função de delimitar as áreas do sistema operacional e do programa. Caso um programa faça referência a um endereço de memória, o sistema checa o delimitador para verificar se o endereço não está invadindo uma área ocupada pelo sistema operacional. Se estiver invadindo, então o sistema operacional termina o programa e envia uma mensagem de erro para o usuário sobre a violação de acesso da memória principal.



Figura 63 – Proteção de memória.

Fonte: Machado e Maia (2007)

4.4.2 Sobreposições (overlay)

Com o passar do tempo, a memória física passou a não ser suficiente para conter todo o programa do usuário que eram maiores que a memória principal. A solução encontrada foi a utilização de uma técnica conhecida como *overlay* (sobreposições). Nesta técnica o programa é desenvolvido de forma que possa ser dividido em partes lógicas independentes de forma a permitir que uma parte possa ser substituída por outra quando não estiver em uso. Por exemplo,

a figura 64 ilustra um programa dividido em quatro partes, de “a” a “d”. A parte principal do programa (a) deve ficar na memória principal durante todo tempo em que o programa deve ser executado. Assim é carregado na memória principal logo acima do sistema operacional. O restante da memória, área de *overlay*, será a área comum compartilhada pelas demais partes do programa. Quando a parte principal (a) efetuar uma chamada a uma das partes, por exemplo, parte (b), este módulo é carregado na área de *overlay*. Caso o parte principal chame a parte (c), a parte (b) será sobreposta à parte (c) na área de *overlay*. Se a parte principal chamar um módulo que já esteja carregado, a carga será desconsiderada.

A área de *overlay* é definida pelo programador utilizando comandos da linguagem de programação utilizada para desenvolver o programa. Entretanto, uma sobreposição que não for muito bem planejada pode se tornar complexa e de difícil manutenção. Outro fator importante a ser considerado é com relação ao desempenho. Para garantir o desempenho do programa, o programador deve ter o cuidado de evitar trocas constantes entre as partes que ocupam a área de *overlay* (memória principal) e a memória secundária.

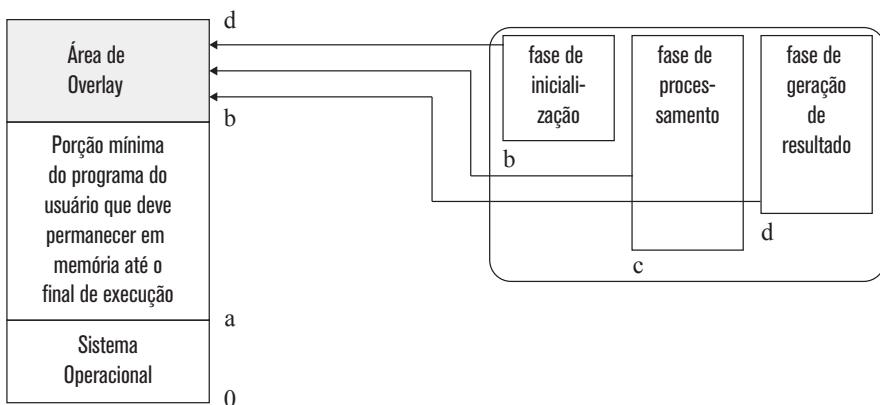


Figura 64 – Técnica de overlay.

4.4.3 Alocação contígua particionada fixa

Com a evolução dos sistemas monoprogramáveis para os multiprogramáveis, houve uma grande necessidade de aprimoramento no modo de alocação da memória. Isto pelo fato de que, um dos fatores de eficiência dos sistemas multiprogramáveis decorrem de ter vários programas alocados na memória ao mesmo tempo para serem executados. Assim, surgiu a *alocação particionada fixa*. Esta abordagem consiste em dividir a memória em tamanhos fixos, denominadas *partições*.

A partição da memória pode ser de tamanhos iguais ou diferentes. O tamanho de cada partição era definido no momento da inicialização do sistema operacional levando em consideração os programas que seriam executados. Caso houvesse a necessidade de um particionamento diferente do que tinha sido configurado, era necessário que o sistema operacional fosse reinicializado com a nova configuração.

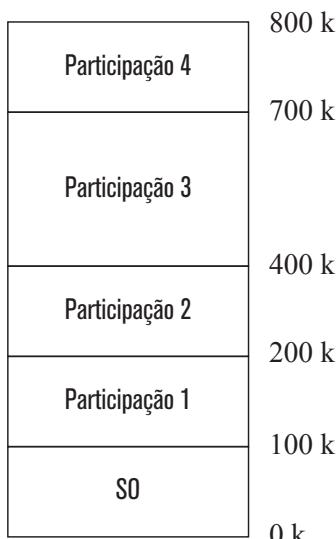


Figura 65 – Partição da memória.

Este esquema tornava o gerenciamento de memória relativamente simples pelo fato dos programas, ao serem compilados, recebiam um endereço físico da memória principal a partir de onde deveriam ser carregados. Isto fazia com que o programa somente executasse em uma partição específica. Dessa maneira, um programa tinha que aguardar a desocupação de sua partição para ser executado, mesmo tendo outras partições livres. Por exemplo, o programa “A” está sendo alocado na Partição 4 cujo tamanho é de 100k. O programa “D” está alocado na Partição 1, também com tamanho de 100k. A Partição 3 de tamanho 300k está livre, mas a restrição imposta pelos endereços absolutos impedem que os programas “B”, “E” ou “F” sejam alocados nesta partição causando um grande desperdício de memória. Esta forma de gerenciamento de memória ficou conhecida como *alocação particionada estática absoluta*.

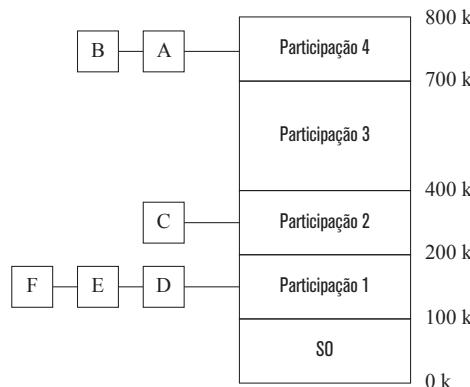


Figura 66 – Alocação particionada estática absoluta.

O problema de desperdício de memória da alocação particionada estática absoluta foi superado com a implementação de compiladores e carregadores de realocação. Assim, todos os endereços de memória, utilizados por um programa, ao ser compilado, passam a ser relativos ao início do programa e não mais endereços físicos da memória. Esta abordagem permite que ao carregar um programa, o carregador transforma os endereços relativos em endereços absolutos a partir de participação onde foi alocado o programa. Esta forma de gerenciamento de memória ficou conhecida como *alocação particionada estática relocável*.

Desta forma, o programa pode ser alocado em qualquer participação que comporte o seu tamanho. Contudo, houve um grande aumento na complexidade dos compiladores e carregadores.

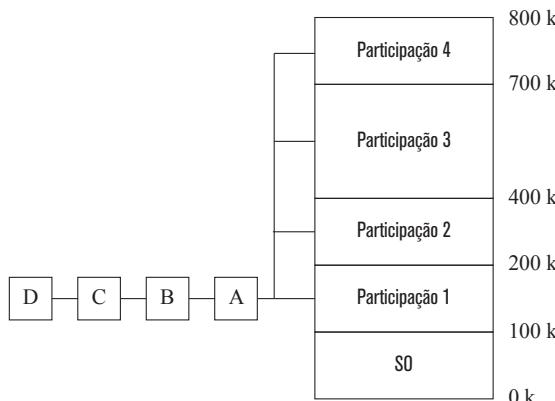


Figura 67 – Alocação particionada estática relocável.

Para decidir em qual partição um programa deve ser carregado, o gerenciador de memória utiliza uma tabela conhecida como *tabela de alocação de partições*. A tabela de alocação de partições permite que o gerenciador verifique qual partição está livre e qual o seu tamanho. Caso a partição livre seja grande o suficiente para conter o programa, ele será carregado nesta partição e a tabela será atualizada para informar que a partição está ocupada.

Memória Principal		
Sistema Operacional		
1	Programa C	
2	Área Livre	
3	Programa B	

PARTIÇÃO	TAMANHO	LIVRE
1	2 Kb	Não
2	5 Kb	Sim
3	8 Kb	Não

Figura 68 – Tabela de alocação de partições.

Fonte: Machado e Maia (2007).

Uma vez o programa carregado em uma partição, o próximo passo é gerenciar o acesso de memória feita pelo programa para que não invada a área destinada ao sistema operacional ou outros programas alocados em outras partições. A solução encontrada foi a utilização de dois registradores de fronteiras: *registrador de base* e *de limite*. Cada partição é delimitada pelo seu próprio registrador de base e de limite. Estes registradores contêm o endereço inicial e final da partição. Quando o programa faz uma requisição de um endereço de memória, o gerenciador verifica se o endereço é maior ou igual ao valor do limite inferior e se é menor que o valor do limite superior. Se estiver dentro da faixa permitida, a requisição é atendida, caso contrário o sistema termina o programa com uma mensagem de erro para o usuário sobre a violação de acesso da memória principal.

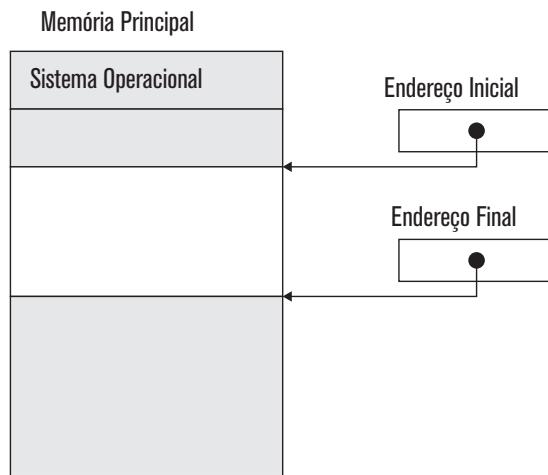


Figura 69 – Registradores de fronteiras.

Fonte: Machado e Maia (2007).

4.4.4 Alocação contígua particionada dinâmica

Um grande problema enfrentado nas abordagens anteriores, onde a partição é fixa é a fragmentação da memória conhecida com *fragmentação interna*. A fragmentação interna acontece quando um programa não ocupa totalmente a partição em que foi alocado.

Além disso, uma partição pode ser muito pequena que não possa acomodar um programa que esteja esperando.

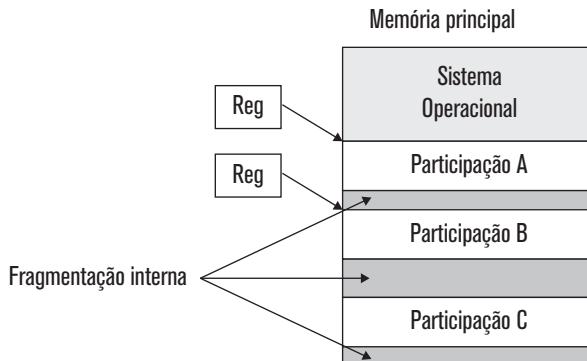


Figura 70 – Fragmentação interna.

A solução encontrada foi permitir que um programa ocupasse o espaço que for necessário, de acordo com sua necessidade, desde que haja espaço disponível na memória. Assim, foi substituída a partição de tamanho fixo pela *alocação particionada dinâmica*. Inicialmente a memória é organizada de forma a não ter nenhuma partição. Conforme há a necessidade de carregar um programa, o sistema operacional cria a partição dinamicamente, de acordo com as necessidades do programa. Consequentemente, eliminou-se a fragmentação interna, mas por outro lado, aumentou a complexidade da tabela de alocação de partições. A tabela de alocação de partições passou a não ter mais um número fixo de entradas, agora o número de entradas varia de acordo com o número de partições.

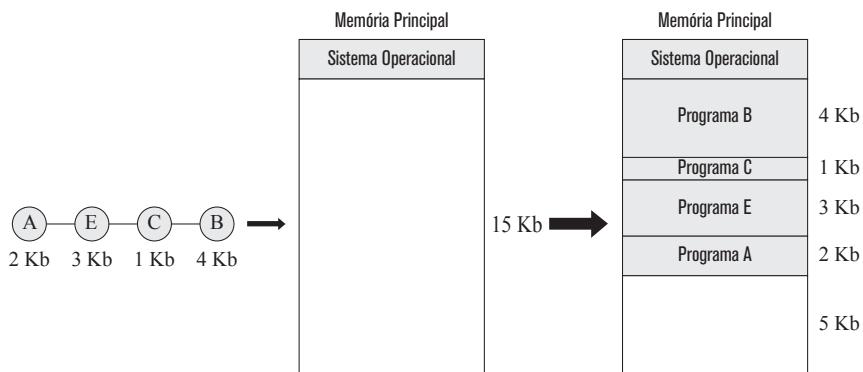


Figura 71 – Alocação particionada dinâmica.

Fonte: Machado e Maia (2007).

Esta solução gerou um outro problema. Conforme os programas terminam sua execução e desalocam a partição, vão deixando cada vez mais, pequenos espaços vazios na memória. Estes espaços vão se espalhando pela memória e muitas vezes são tão pequenos, que não é possível utilizá-los para alocar um novo programa. Este problema é conhecido como *fragmentação externa*.

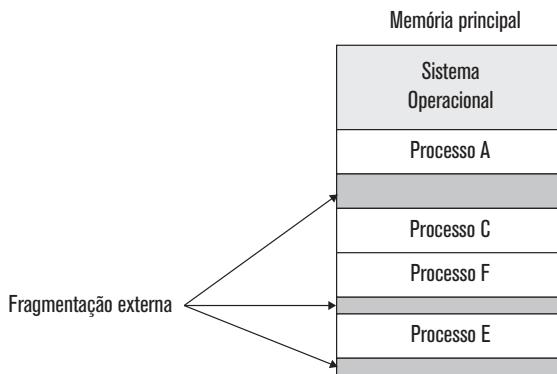


Figura 72 – Fragmentação externa.

É possível resolver o problema de fragmentação externa através de duas soluções. A primeira solução consiste em reunir os espaços livres adjacentes quando um programa libera a partição. Na figura abaixo, quando o Programa C finaliza sua execução, sua partição de 1kb é combinada com as duas partções adjacentes formando um espaço maior de 8kb.

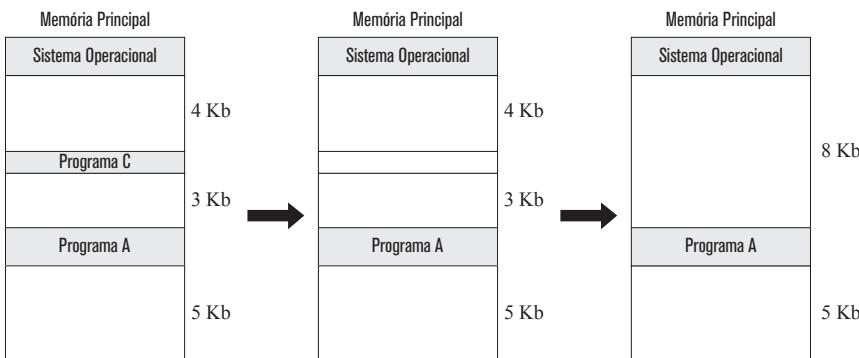


Figura 73 – Solução para Fragmentação externa.

Fonte: Machado e Maia (2007).

A segunda solução exige que todas as partções na memória sejam realocadas de forma que sejam eliminados todos os espaços vazios entre elas. Esta solução é conhecida como *realocação dinâmica* ou *alocação particionada dinamicamente com realocação*. Esta abordagem praticamente elimina a fragmentação externa, mas há um excessivo consumo de recurso do sistema, como processamento e área de disco rígido, além de ser muito complexa.

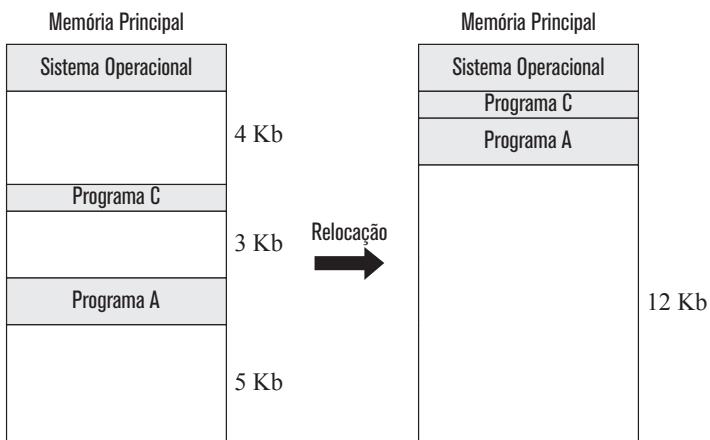


Figura 74 – Solução para Fragmentação externa.

Fonte: Machado e Maia (2007).



CONEXÃO

Entenda um pouco mais sobre gerenciamento de memória

<http://www.ricardobarcelar.com.br/aulas/soa/mod2-ger_memoria.pdf>.

4.4.5 Estratégia de posicionamento de memória

Inicialmente, quando um programa necessita ser alocado, o sistema pode escolher qualquer partição livre que contenha o espaço suficiente para recebê-lo. A escolha é feita pelo gerenciador de memória utilizando diversas estratégias e levando em consideração diversos fatores com o objetivo de determinar em qual a melhor partição livre, um programa esperando por memória, pode ser alocado. Consequentemente a escolha de uma terminada estratégia pode influenciar no desempenho do sistema. Independente de qual estratégia seja adotada, a fragmentação externa tende a diminuir ou praticamente ser eliminada. Para implementar qualquer uma das estratégias, o sistema operacional mantém uma tabela com os dados das áreas livres e qual a sua quantidade de espaço disponível, permitindo desta forma a agilidade na identificação das áreas livres.

Memória Principal		
Sistema Operacional		
Área Livre 1	4 Kb	
Programa C		
Área Livre 2	5 Kb	
Programa A		
Área Livre 3	3 Kb	

PARTIÇÃO	TAMANHO	LIVRE
1	2 Kb	Não
2	5 Kb	Sim
3	8 Kb	Não

Figura 75 – Tabela de memórias livres.

Fonte: Machado e Maia (2007).

Entre as estratégias mais utilizadas estão:

- Estratégia o primeiro que couber (*First-Fit*): percorre a tabela de memórias livres procurando pela *primeira* área capaz de comportar o programa a ser alocado. Esta estratégia minimiza o trabalho de busca e é extremamente rápida principalmente se houver áreas livres muito longas. Nesta estratégia, a tabela de memórias livres não possui nenhuma ordenação.

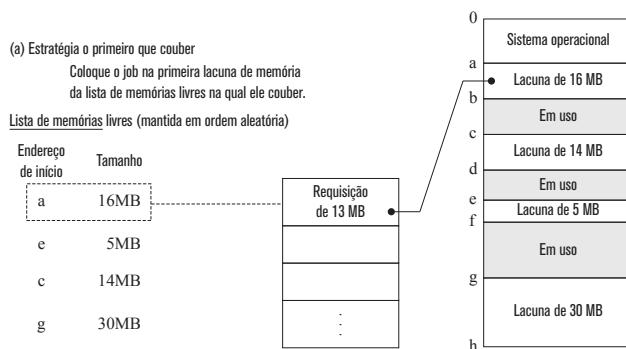


Figura 76 – Estratégia o primeiro que couber (*First-Fit*).

Fonte: Deitel et al. (2005).

- Estratégia o que pior couber (*Worst-Fit*): percorre a tabela de memórias livres procurando pela *maior* área possível, encontrando, aloca o programa.

O restante que sobra, também será grande podendo ser utilizada para alocar um programa grande. Por ter que pesquisar em todas as áreas livres, para encontrar a de maior área livre, sobrecarrega o sistema e tende a deixar pequenas áreas livres sem utilização. Nesta estratégia, a tabela de memórias livres é ordenada em ordem descendente de tamanho de área livre.

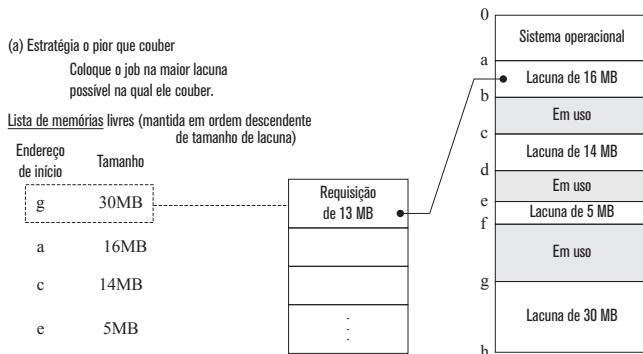


Figura 77 – Estratégia o que pior couber (Worst-Fit).

Fonte: Deitel et al. (2005).

- Estratégia o que melhor couber (*Best-Fit*): percorre a tabela de memórias livres procurando pela *menor* área possível, encontrando, aloca o programa. O restante que sobra, também será pequeno, e desta forma terá pouco espaço sem utilização. O problema é que este pouco espaço sem utilização, com o tempo, pode causar o problema de fragmentação. Nesta estratégia, a tabela de memórias livres é ordenada em ordem ascendente de tamanho de área livre.

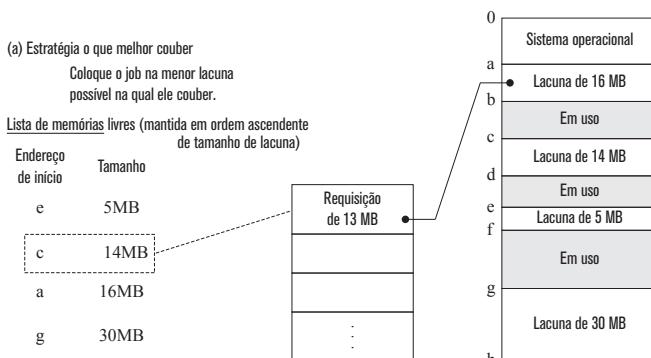


Figura 78 – Estratégia o que melhor couber (Best-Fit).

Fonte: Deitel et al. (2005).

Nas pesquisas efetuadas por Johnstone e Wilson (1999) foram demonstradas que as estratégias mais eficientes são a Estratégia o que melhor couber (*Best-Fit*) e Estratégia o primeiro que couber (*First-Fit*), sendo que a *First-Fit* se mostrou bem mais rápida que a anterior.

4.4.6 Troca de memória (swapping)

Nas abordagens anteriores, o programa era alocado na memória e somente saía quando era finalizado. Isto faz com que se a memória principal estiver toda ocupada, um novo programa não poderá ser alocado e consequentemente, não será executado. Além disso, pode ocorrer de ter partições livres, mas não com o tamanho suficiente para alocar um determinado programa pronto para ser executado. A solução para estes cenários é o mecanismo denominado *swapping* (troca). O mecanismo de *swapping* transfere um processo da memória principal, temporariamente para a memória secundária, permitindo que um novo processo sejam carregado na memória principal e executado. Este procedimento é conhecido *swap-out*. Em um determinado momento o processo, que foi transferido para a memória secundária, é carregado de volta para a memória principal. Este procedimento é conhecido *swap-in*.

Para que o mecanismo de *swapping* seja utilizado, o gerenciador de memória, antes de qualquer coisa, reserva um espaço em disco rígido (memória secundária) para que possa receber os processos da memória. O mecanismo de *swapping* permitiu que o sistema operacional executasse um maior número de processos que normalmente caberia na memória. Um ponto a ser observado é que o *swapping* tende a aumentar o tempo de execução dos programas devido ao fato da troca de um processo para o disco rígido ser demorado. Para diminuir as quantidades de troca, os processos escolhidos para *swapping* são processos no estado de Em Espera ou aqueles que estejam em estado de Pronto, mas com poucas chances de ser escalonado. Assim o gerenciador de memória tende a ter um desempenho melhor evitando trocas desnecessárias.

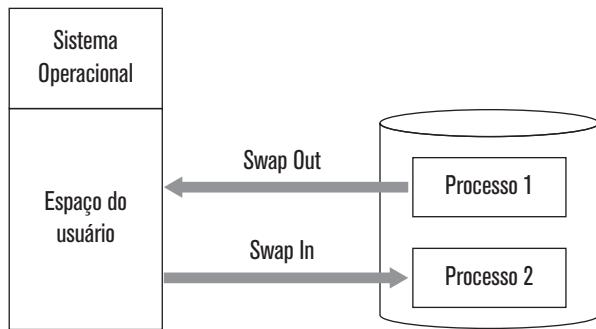


Figura 79 – Swapping.

4.5 Memória Virtual.

Estudamos nas seções anteriores que os programas maiores que o tamanho da memória principal são divididos em módulos e armazenados na memória secundária. Quando este programa entra em execução, os módulos são trocados (*swapping*) entre a memória principal e secundária e vice-versa pelo sistema operacional. A divisão em módulos era feita pelo próprio programador, tarefa que dava muito trabalho principalmente se tratando de programas muito grandes. Em 1961, Fotheringham criou um método que combina a memória principal e secundária de forma a dar a ilusão que a memória é muito maior do que a memória principal. Este método ficou conhecido como *memória virtual*.

Desta forma o programador não tem mais a necessidade de dividir os programas em módulos por causa da limitação de memória. O disco rígido (memória secundária) passa a ser utilizada como extensão da memória RAM (memória principal), e onde são armazenados parte dos programas e dados que estão carregados e executando. Assim, o sistema operacional transfere parte dos programas e dados, da memória secundária para a principal, somente no momento que realmente serão executados. Esta técnica cria-se o espaço de endereçamento virtual (lógico) e o espaço de endereçamento real (físico) (ver seção: Espaço de Endereçamento Físico e Lógico).

Devido ao grande crescimento de capacidades das memórias RAM nos últimos anos, que combinadas podem passar dos 16Gbs, não há necessidade de utilizar em tempo integral a virtualização da memória como era necessário até alguns anos atrás. Em alguns casos, ainda é necessário a utilização da memória virtual, mas é importante salientar, como foi feito anteriormente, que esta técnica pode causar perda de desempenho do computador tendo em vista que o acesso ao disco rígido é mais lento que o acesso à memória principal.

4.5.1 Paginação

O gerenciador de memória utiliza uma técnica onde o espaço de endereçamento real e o espaço de endereçamento virtual são divididos em blocos do mesmo tamanho. Esta técnica é conhecida como paginação. Os blocos obtidos da divisão são conhecidos como *páginas*, sendo que os blocos do endereçamento real são denominados *páginas reais ou frames* e os blocos do endereçamento virtual são denominados *páginas virtuais*. A figura 80 exibe o esquema de paginação formado pela memória virtual, memória principal e a tabela de páginas. Cada processo possui a sua própria *tabela de páginas*. A tabela de páginas é montada durante o carregamento do processo e é responsável pelo mapeamento do endereço virtual para o endereço real. Toda página virtual do processo possui uma entrada na tabela de processos (Entrada na Tabela de Páginas – ETP).

A figura exibe os endereços descritos na forma binária, sendo que os *bits* de mais alta ordem (três primeiros números da esquerda para direita) correspondem ao número da página e os *bits* de mais baixa ordem (2 últimos números) correspondem ao deslocamento dentro da página.

Tomemos como exemplo, mapear referente ao endereço virtual (lógico) Y2 para o endereço real (físico). O endereço virtual de Y2 corresponde a 00101 sendo que o número da página é 001 e o deslocamento 01. Através da tabela de páginas encontramos que a entrada da página virtual 001 foi carregada na página real 101. Basta agora unir os *bits* de deslocamento ao número da página real, encontrado então o endereço real 10101.

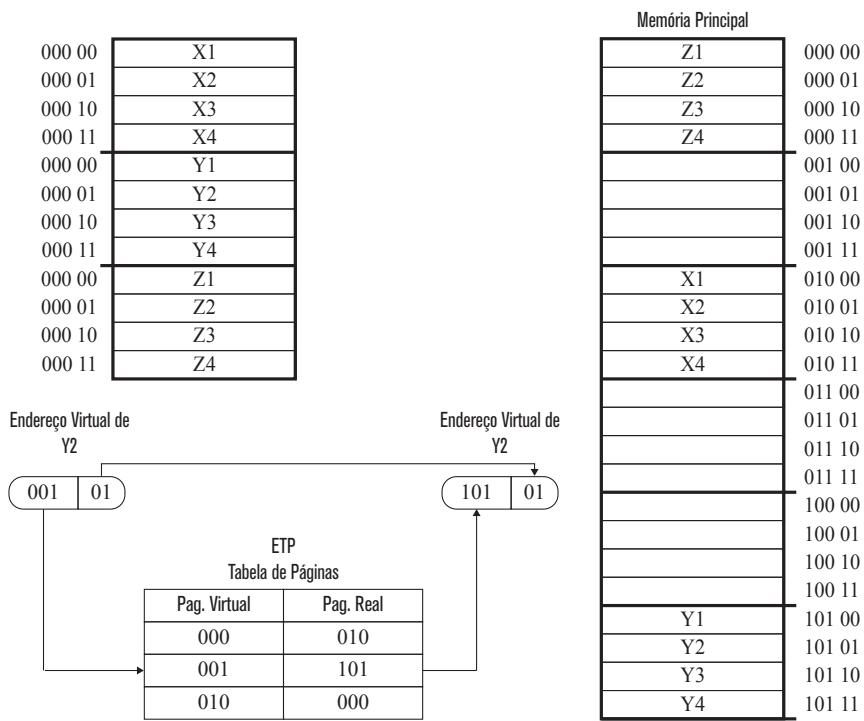


Figura 80 – Esquema de paginação.

4.5.2 Segmentação

Enquanto a paginação divide a memória virtual e blocos do mesmo tamanho, a segmentação é uma técnica que permite dividir tanto o espaço de endereço virtual e real em blocos de tamanhos diferentes.

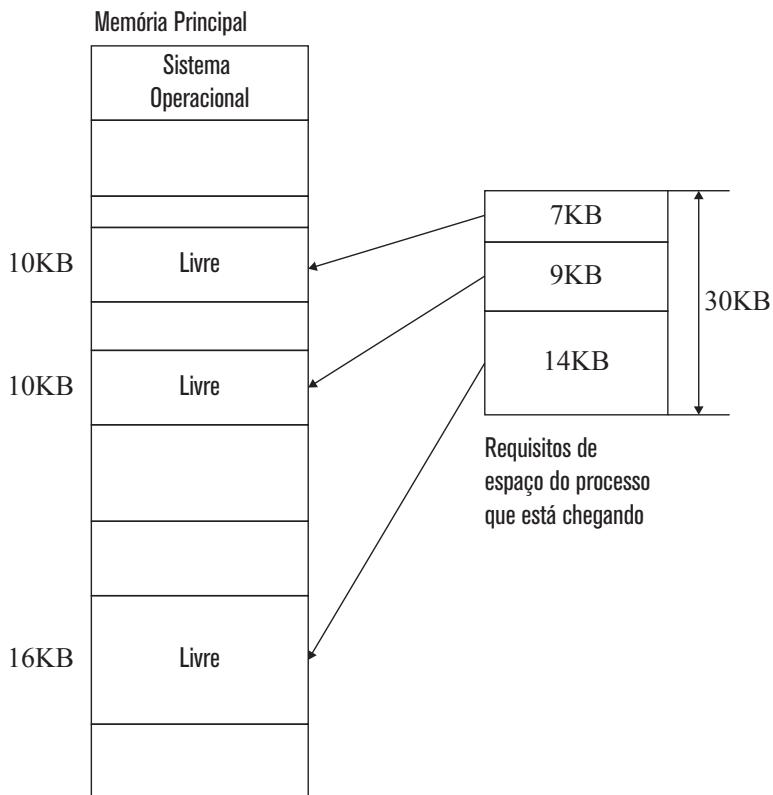


Figura 81 – Sistema de segmentação.

Fonte: Deitel et al. (2005).

A divisão do tamanho do bloco está relacionado a divisão lógica de um programa. Uma divisão lógica de blocos para um programa poderia ser, por exemplo, instruções, dados e pilhas. Estes blocos são chamados de *segmentos*. Os segmentos não necessariamente precisam ser do mesmo tamanho ou ocupar posições adjacentes na memória principal. Os endereços virtuais são mapeados para o endereço real através de uma tabela de segmentos. As informações mantidas na tabela de segmento são semelhantes à tabela de paginação.

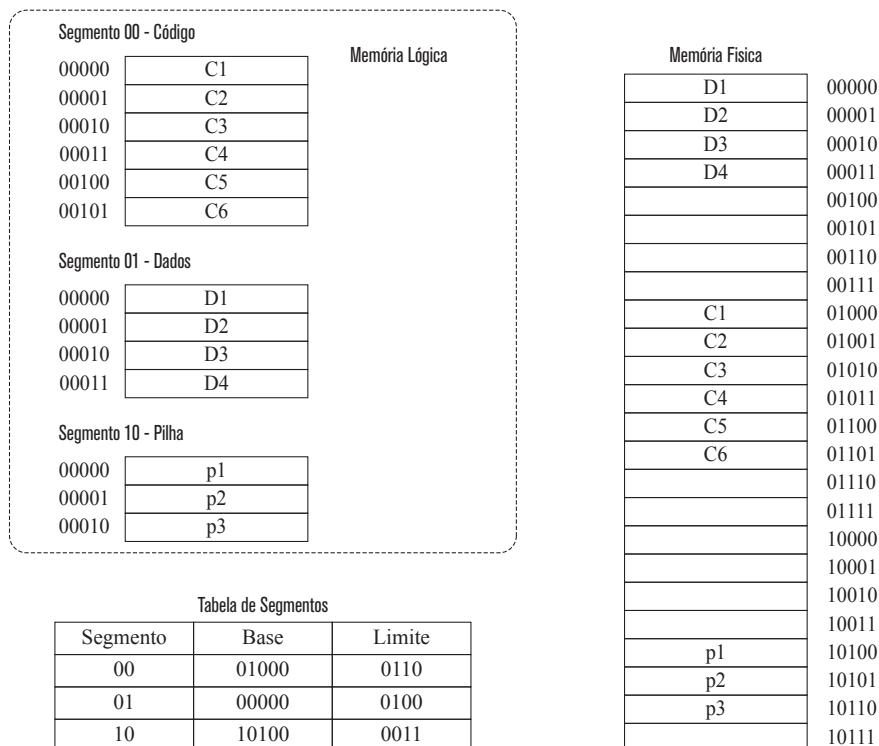


Figura 82 – Esquema de segmentação.

4.5.3 Segmentação com paginação

Neste tipo de gerenciamento de memória são combinadas as técnicas de segmentação e paginação. Os segmentos são arranjados ao longo de páginas de memória virtual, permitindo desta forma as vantagens das duas técnicas. O mapeamento do endereço virtual para um endereço real utiliza uma tabela de segmentos e uma tabela de páginas. O endereço virtual é formado por um *número de segmento, número de página e deslocamento*. Cada segmento está associado a uma tabela de páginas. A Figura 83 mostra como é feito o mapeamento de um determinado endereço virtual. O número de segmento indica qual tabela de segmentos o endereço está associado. Na tabela de segmentos indica qual tabela de páginas o segmento está associado. Combinando o número da página com o deslocamento temos o endereço real.

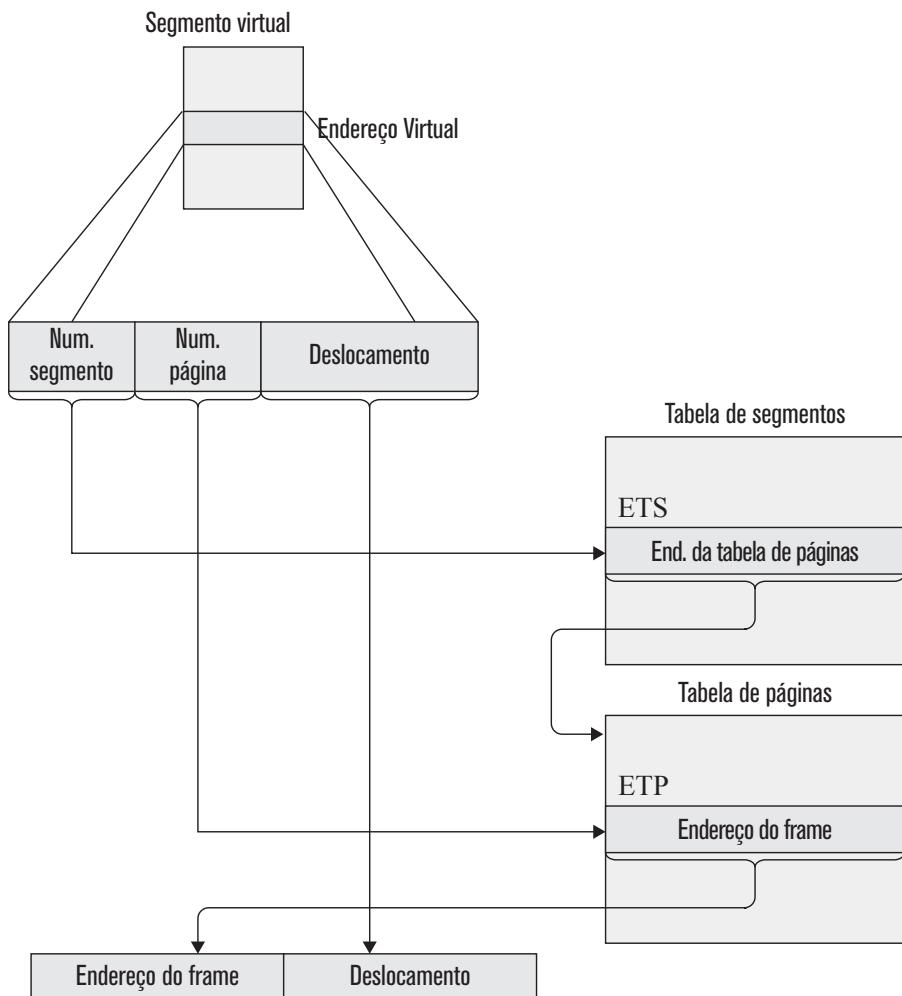


Figura 83 – Segmentação com paginação.

Fonte: Machado e Maia (2007)



ATIVIDADE

1. Quais são as formas de alocação de memória pelo sistema operacional?
2. Como funciona a técnica conhecida como Sobreposições (*overlay*)?



REFLEXÃO

Neste capítulo estudamos o gerenciamento da memória, para tanto iniciamos vendo as estruturas de memória, espaçamentos de endereçamento físico e lógico. Para melhor entender como o sistema operacional efetua este gerenciamento, abordamos as diversas formas de alocação da memória, como a alocação contígua e não-contígua. É importante que, baseado nos conceitos e arquiteturas apresentadas, você seja capaz de identificar as vantagens e desvantagens de cada modelo de alocação de memória. Reflita sobre isso e identifique as diferenças entre cada uma delas.



LEITURA

Para você avançar mais o seu nível de aprendizagem envolvendo os conceitos de sistemas operacionais e demais assuntos deste capítulo, consulte as sugestões de *links* abaixo:
Capítulo 4 do livro

TANENBAUM, A. S.; WOODHYLL, A. S. *Sistemas operacionais projeto e implementação*. 2^a ed Porto Alegre: Bookman, 1999.



REFERÊNCIAS BIBLIOGRÁFICAS

DEITEL H. M.; DEITEL P. J.; CHOHNES D. R. Sistemas Operacionais. 3^a ed. São Paulo, Editora Prentice-Hall, 2005.

HENNESSY, J. L.; PATTERSON, D. A. Arquitetura de Computadores: uma abordagem quantitativa. 3^a ed. Rio de Janeiro: Campus, 2003.

JOHNSTONE, M. S.; WILSON, P. R. The memory fragmentation problem: solved? ACM SIGPLAN Notices, 34(3): 26–36, 1998.

MACHADO, F. B.; MAIA, L. P. Arquitetura de sistemas operacionais. 4^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2007.

MAZIERO, C. A. Sistemas Operacionais: Conceitos e Mecanismos. Disponível em: <<http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/so:so-livro.pdf>>. Acesso em: set. 2014.

OLIVEIRA, R. S.; CARISSIMI, A. S.; TOSCANI, S. S. Sistemas Operacionais. 4^a ed. Porto Alegre : Editora Bookman, 2010.

PATTERSON, D. A.; HENNESSY, J. L. Organização e Projeto de Computadores. 3^a Ed. Rio de Janeiro: Campus, 2005.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G Fundamentos de Sistemas Operacionais. 6^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2004.

TANENBAUM, A. S.; WOODHYLL, A. S. Sistemas operacionais projeto e implementação. 2^a ed Porto Alegre: Bookman, 1999.



NO PRÓXIMO CAPÍTULO

No capítulo seguinte, estudaremos os principais componentes de hardware e software de entrada e saída. Você entenderá como é feita a comunicação dos dispositivos com o computador. Será apresentado como o sistema operacional efetua o gerenciamento dos arquivos e diretórios.

5

Gerência de Entrada e Saída

5 Gerência de Entrada e Saída

Atualmente podemos conectar, além dos já existentes, uma infinidade de dispositivos em nosso computador, tais como, máquinas fotográficas, filmadoras, tablets, celulares etc. Efetuar um gerenciamento de todos estes dispositivos é considerado a função mais complexa do sistema operacional e uma das principais. Outro ponto é com relação aos sistemas de arquivos. Quantas informações de alta relevância guardamos em nossos arquivos? Quantos dias de trabalho podem estar armazenados em um simples arquivo? Quais as consequências de perdemos um destes arquivos? Para responder a este questionamento, estudaremos como o sistema operacional efetua o gerenciamento dos arquivos de forma a garantir segurança e integridade dos seus dados.



OBJETIVOS

- Estudar os principais componentes de hardware e software de entrada e saída.
 - Entender como é feita a comunicação dos dispositivos com o computador.
 - Compreender como o sistema operacional efetua o gerenciamento dos arquivos e diretórios.
-



REFLEXÃO

Você se lembra de alocação de memória, contígua e paginação? Caso não se lembre, faça uma breve consulta e anote os principais conceitos.

5.1 Introdução

Uma das principais funções do sistema operacional é fornecer mecanismos que possam servir de comunicação entre o usuário e os diversos componentes de um computador. Estes componentes são conhecidos como *dispositivos de entrada e saída*, tais como *mouse*, teclado, monitor, telas sensíveis ao toque, impressora, *pen drive*, etc. Não somente interação homem\máquina, mas também entre outros computadores (*modems*, placas de redes), outros dispositivos como celulares, câmeras digitais, *tablets*, etc., ou mesmo dispositivos que permitem o armazenamento de informações (disco rígido, fita magnética, CD-ROM, etc.).

Assim, o sistema operacional mantém uma estrutura que permite controlar a diversidade de dispositivos de entrada e saída, através de operações, tais como envio de comando para os dispositivos, captura das interrupções geradas, tratamento de erros, interface entre os dispositivos e o restante do sistema os programas dos usuários, entre outros.

Os dispositivos de entrada e saída podem ser classificados em periféricos de entrada, periféricos de saída ou periféricos de entrada e saída, de acordo com o fluxo de dados entre o computador e o dispositivo. Segundo Oliveira et al. (2010) um periférico é qualquer dispositivo conectado a um computador de forma a permitir a sua interação com o mundo externo. Como exemplo de periféricos, podemos citar:

- Periféricos de entrada: teclado, *mouse*, *webcam*, *scanner*, etc.
- Periféricos de saída: monitor, impressora, caixas de som, etc.
- Periféricos de entrada e saída: monitor *touchscreen*, *modem*, *pendrive*, *joystick*, impressoras com *scanners*, etc.

Cada um destes periféricos possuem suas próprias características e necessidades de mecanismos de acesso específicos. Consequentemente, geram um grande desafio no sentido de desenvolvimento e manutenção de sistemas operacionais.

Gerenciar todas estas diversidades de periféricos de entrada e saída é uma das funções principais e mais complexas efetuada pelo sistema operacional. Além disso, deve fornecer um conjunto de rotinas e serviços para que os dispositivos possam ser acessados pelos usuários e os seus programas. Para tanto, o sistema operacional utiliza uma arquitetura de camadas para melhor gerenciar os dispositivos de entrada e saída.

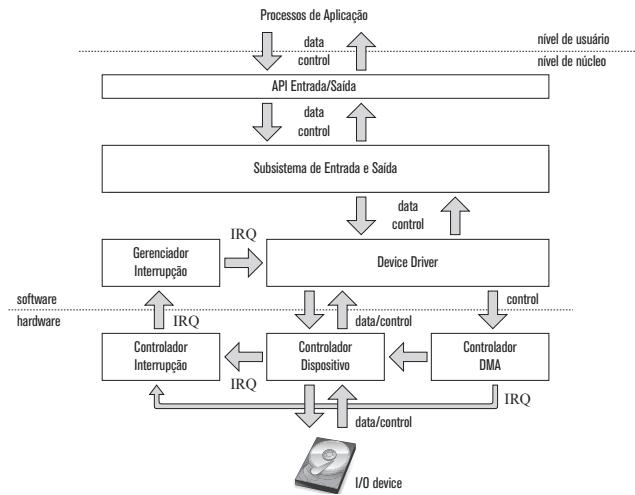


Figura 84 – Arquitetura de entrada e saída.

Fonte: Adaptado de Maziero (2014).

5.2 Componentes de hardware de ENTRADA E SAÍDA

O sistema operacional tem como função gerenciar a comunicação entre os programas e os diversos dispositivos de entrada e saída. O sistema operacional não acessa diretamente os dispositivos. O acesso e gerenciamento aos dispositivos feito pelo sistema operacional passam pela controladora. Estudaremos os aspectos envolvidos nesta comunicação, bem como as características destes dispositivos.

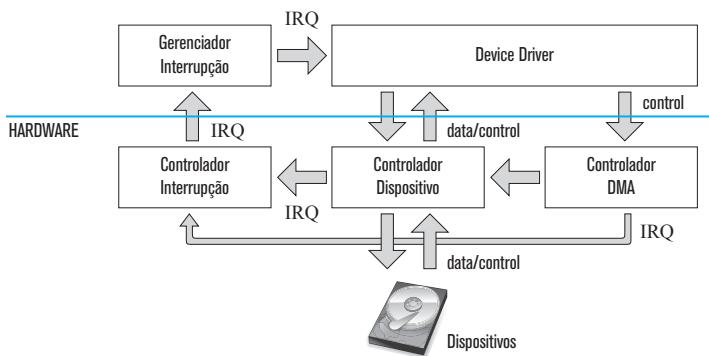


Figura 85 – Componentes de hardware de E/S.

Fonte: Adaptado de Maziero (2014).

5.2.1 Dispositivos de Entrada e Saída

Atualmente há diversos dispositivos periféricos que permitem ao homem se comunicar com o computador, como também entre computadores. Segundo Maziero (2014) os periféricos podem variar em suas características, velocidade e forma de transferência dos dados e método de acesso. A velocidade típica de transferência de dados de alguns dispositivos é ilustrada na tabela a seguir.

DISPOSITIVO	VELOCIDADE
Teclado	10 B/s
Mouse ótico	100 B/s
Interface infravermelho (IrDA-SIR)	14 KB/s
Interface paralela padrão	125 KB/s
Interface de áudio digital S/PDIF	384 KB/s
Interface de rede <i>Fast Ethernet</i>	11.6 MB/s
Chave ou disco USB 2.0	50 MB/s
Interface de rede <i>Gigabit Ethernet</i>	116 MB/s
Disco rígido SATA 2	300 MB/s
Interface gráfica <i>high-end</i>	4.2 GB/s

Tabela 8 - Velocidades de dispositivo de E/S.

Fonte: Mazieiro

A classificação dos dispositivos pode variar de autor para autor. A primeira classificação e utilizada anteriormente neste capítulo, é a feita por Carpinelli (2001) onde os dispositivos de E/S são classificados em:

- **Dispositivos de entrada:** são os dispositivos utilizados apenas para a entrada dados no sistema. Exemplo: teclado, *mouse*, *webcam*, *scanner*, etc.
- **Dispositivos de saída:** são os dispositivos utilizados apenas para saída de dados do sistema. Exemplo: monitor, impressora, caixas de som, etc.
- **Dispositivos de entrada e saída:** são os dispositivos utilizados tanto para a entrada como a saída de dados no sistema. Exemplo: monitor *touchscreen*, *modem*, disco rígido, etc.

Para Stallings (2002) os dispositivos podem ser classificados de acordo com sua finalidade e como interage com o usuário e outros dispositivos. Assim temos:

9. Voltados para a comunicação com o usuário: são dispositivos adequados para a comunicação com o usuário. Exemplo: teclado, *mouse*, monitor, impressora, caixas de som, etc.
10. Voltados para a comunicação com a máquina: são dispositivos adequados para a comunicação com dispositivos eletrônicos. Exemplo: disco rígido, fita magnética, *CD-ROMs*, controladores, etc.
11. Voltados para a comunicação com dispositivos remotos: são dispositivos adequados para a comunicação remota. Exemplo: placa de rede, *modem*, etc.

A classificação dada por Tanenbaum e Woodhyll (1999) leva em consideração a forma em que os dados podem ser acessados e/ou obtidos a partir destes dispositivos. Podendo ser:

- Dispositivos de blocos: são dispositivos que transferem ou armazenam dados em blocos de tamanho fixo, cada um desses blocos possui um endereço, sendo que cada bloco pode ser acessado independentemente. Exemplo: disco rígido, CD-ROM, etc.
- Dispositivos de caractere: são dispositivos que enviam ou recebem um fluxo de caracteres. Os caracteres não possuem endereçamento e não podem ser acessados de forma individual. Exemplo: teclado, *mouse*, impressoras, placas de rede, etc.

5.2.2 Controlador de entrada e saída

Para que os periféricos sejam conectados ao computador, o sistema disponibiliza componentes de *hardware* conhecidos como interfaces. As interfaces, por sua vez, são conectadas aos barramentos de dados para que, independente do tipo de periférico, todos se comunicam com o computador. As interfaces implementam um outro *hardware* denominado *controlador*.

O controlador é responsável por gerenciar diretamente os dispositivos de entrada e saída e também permitir a comunicação entre estes dispositivos.

vos e o sistema operacional. Normalmente o controlador em um computador é uma placa de circuito impresso inserida em um determinado *slot* da placa mãe. De modo geral, o controlador pode ser entendido como um processador, que possui alguns registradores, dedicados a gerenciar as complexas funções necessárias para o acesso aos periféricos de entrada e saída. Cada dispositivo necessita de um controlador diferente, isto porque o controlador tem que conhecer como o dispositivo funciona para poder efetuar o seu controle.

O controle é feito através de diversas funções como escrever dados, ler dados, executar comandos, ler *status*, etc. Para tanto, a comunicação entre a UCP e o controlador é feita através de um conjunto de registradores: registrador de dado, registrador de *status* e registrador de comando.

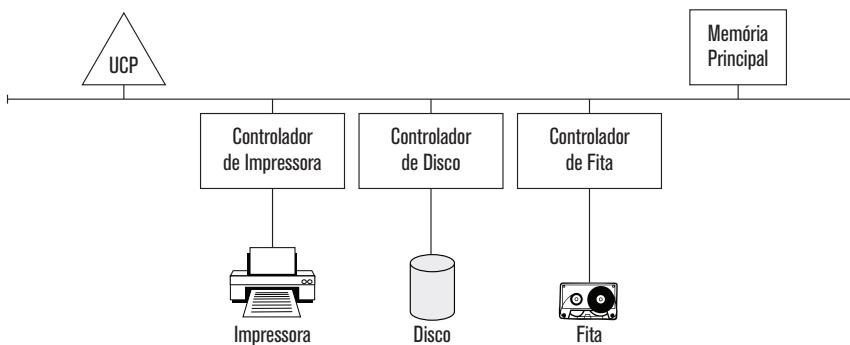


Figura 86 – Controladores.

Fonte: Machado e Maia (2007).

Além dos registradores, o controlador possui um *buffer* que é utilizado para armazenar os dados lidos nos dispositivos até que seja formado um bloco. Quando o bloco é formado, o controlador faz a checagem de erros, e se não houver nenhum erro, o bloco pode ser transferido para a um *buffer* de entrada e saída na memória principal. Esta transferência do bloco do *buffer* interno do controlador para a memória principal pode ser feita pela UCP ou por um controlador DMA (Acesso Direto à Memória - *Direct Memory Access*).

A utilização do DMA permite, entre outras coisas, que seja liberada a UCP durante o operação de busca de dados nos dispositivos de blocos. Sem o DMA a UCP teria que executar loops para a transferência do bloco para a memória principal. A UPC apenas inicia a operação de Entrada e saída informando o endereço inicial na memória e a quantidade de bytes a ser transferido. Desta forma, várias controladoras possuem suporte a DMA, principalmente as controladoras de dispositivos de bloco (SILBERSCHATZ et al., 2004).



CONEXÃO

Aprofunde seus conhecimentos sobre Gerência de Entrada e Saída:

<<http://www.devmedia.com.br/como-funcionam-os-dispositivos-de-entrada-e-saida/28275>>.

5.3 Componentes de Software de ENTRADA E SAÍDA

O objetivo principal dos componentes de *software*, em relação aos dispositivos de entrada e saída, é padronizar ao máximo o acesso a estes dispositivos. Isto permite que novos dispositivos, como por exemplo, novos discos rígidos sejam acrescentados ao computador sem que seja necessário mudar o sistema operacional. A padronização permite ainda que uma aplicação possa abrir um determinado arquivo no disco rígido sem se preocupar com a complexidade envolvida no acesso e leitura do disco.

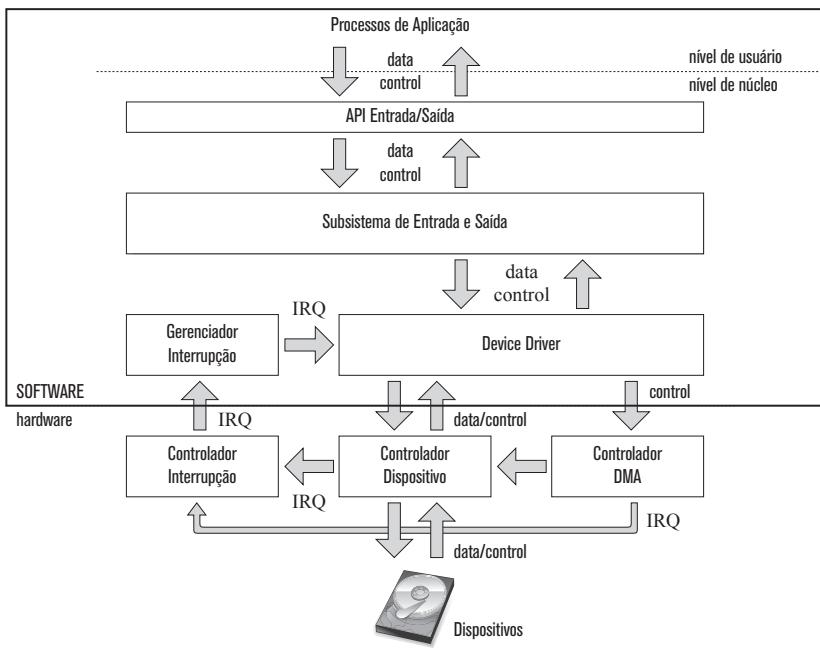


Figura 87 – Componentes de Software de E/S.

Fonte: Adaptado de Maziero (2014).

5.3.1 Drivers

A comunicação entre a camada de subsistemas de entrada e saída e os controladores é feita pelos *devices drivers* ou simplesmente *drivers*. Os *drivers* são camadas de *software* responsável pela implementação de rotinas específicas que permitem o acesso, inicialização e o gerenciamento de um determinado dispositivo de entrada e saída. As rotinas permitem que, ao receber uma instrução de acesso ou controle de dispositivo, esta instrução seja traduzida de uma forma que o dispositivo possa entendê-la.

A camada dos *drivers* é considerada uma camada *dependente dos dispositivos*. Cada *driver* é implementado levando em conta um determinado dispositivo de entrada ou um grupo de dispositivo com características semelhantes. Isto porque é função dos *drivers* programar os registradores internos dos controladores a fim de fazer o acesso e a gerência destes dispositivos.

Outro fator a ser considerado com relação aos *drivers* é o alto grau de dependência do sistema operacional e o restante do *kernel* do sistema em que será instalado. A implementação dos *drivers*, leva em consideração o sistema operacional, para que as corretas instruções de acesso ao dispositivo estejam presentes no *driver*.

O alto grau de dependência entre os *driver* e o restante do *kernel* do sistema fazia com que em sistemas mais antigos, ao ser instalado um novo driver, o *kernel* tinha que ser recompilado e em seguida, reinicializado o sistema. Com a modernização dos sistemas operacionais, não há mais a necessidade de reiniciar o sistema após uma instalação, pois os *drivers* são carregados dinamicamente.

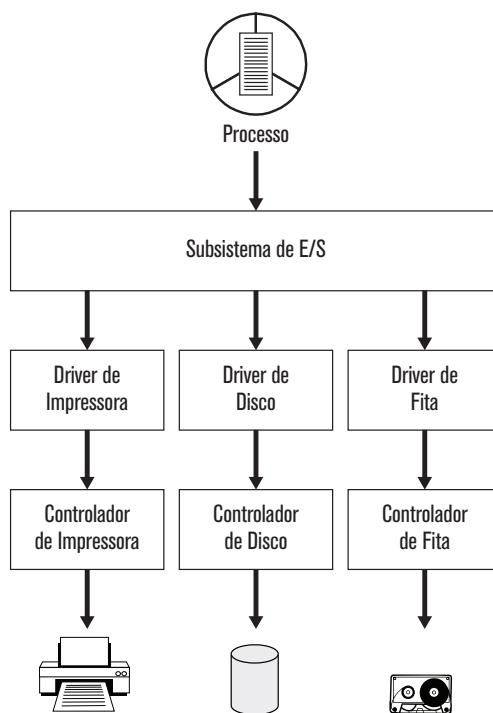


Figura 88 – Device drivers.

Fonte: Machado e Maia (2007).

A figura a seguir ilustra o gerenciador de dispositivo do Windows 7. Através do gerenciador de dispositivo, podemos visualizar, atualizar ou configurar os *drivers* dos dispositivos que estão instalados no computador. Quando um de-

terminado *hardware* está com mau funcionamento, o sistema operacional sinalizado no gerenciador de dispositivo. Verificando esta sinalização, é possível efetuar as devidas correções.

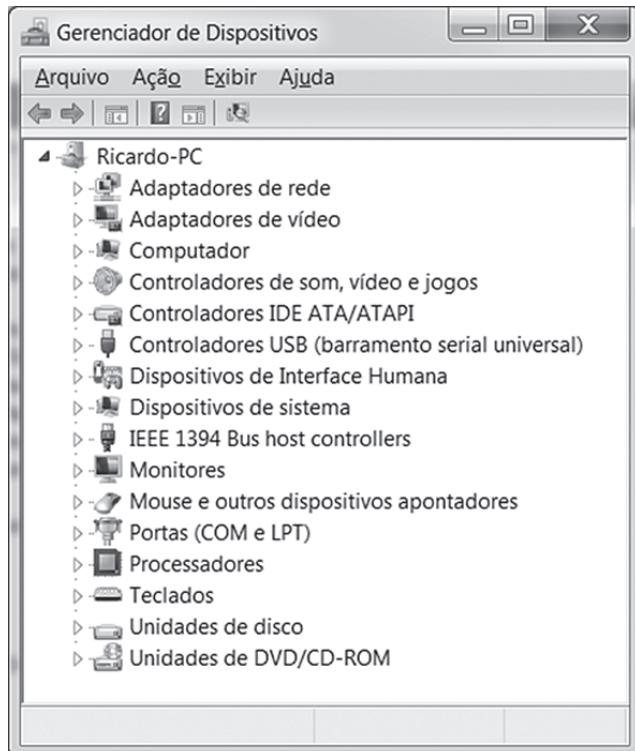


Figura 89 – Gerenciador de Dispositivos do Windows 7.

5.3.2 Subsistemas de entrada e saída

Atualmente há uma grande quantidade de periféricos conectados ao computador, para diversas funções, com velocidades diferentes e formatos variados. Para isolar a complexidade de operações específicas para controle da grande variedade de dispositivo de *hardware* e usuários e os seus programas, o sistema operacional implementa uma camada de *software*, denominada *Subsistemas de Entrada/Saída*.

A função dos Subsistemas de Entrada/Saída é fornecer diversas rotinas necessárias para controle da grande quantidade de dispositivos, isolando o usuário e os programas da complexidade de gerenciamento dos dispositivos de entrada e

saída, como também permitir uma maior flexibilidade do sistema. Isto é feito através da abstração de uma camada para outra, ou seja, uma camada de baixo nível oferece rotinas simples de comunicação com sua camada superior. Dessa forma, a camada superior não necessita conhecer toda a complexidade da camada inferior para efetuar a comunicação. As camadas mais próximas dos dispositivos, normalmente são as que têm a tarefa de tratar os erros destes dispositivos.

Esta estratégia permite uma maior padronização e diminuição do número de rotinas de acesso aos periféricos de entrada e saída. Assim o subsistema de E/S oferece um conjunto de rotinas, chamadas de rotinas de entrada/saída, pelo qual o usuário consegue acessar os dispositivos, independentes de ter que conhecer toda sua complexidade. Dessa forma, o subsistema de E/S é considerado *independente de dispositivo*.

Consequentemente, ao instalar um novo dispositivo no computador, não há necessidade de alterar as rotinas dos programas do usuário.

5.3.3 Aplicação do usuário

Os dispositivos de entrada e saída são acessados através dos aplicativos e linguagens de programação utilizadas pelos usuários. Por exemplo, o caso de um aplicativo de planilha eletrônica, as operações para abrir e salvar um arquivo no disco rígido (dispositivo de armazenamento) feitas pelo aplicativo ficam transparentes para o usuário. As linguagens de programação, utilizadas para desenvolvimento dos aplicativos, possuem um conjunto de comandos de alto nível que são traduzidos pelo compilador da linguagem em funções que são disponíveis em bibliotecas de entrada e saída. Estas bibliotecas contêm funções de chamadas ao sistema operacional e são fornecidas pelos fabricantes do compilador. Uma característica importante destas bibliotecas e que são fornecidas como um módulo objeto (relocável), ou seja, a biblioteca é ligada com o programa desenvolvido pelo programador para compor o aplicativo final.

As operações de entrada e saída podem ser classificadas de acordo com o seu sincronismo, podendo ser *síncrona* ou *assíncrona*. A maioria dos comandos de alto nível funciona na forma síncrona. Neste modelo de sincronismo, o processo chama uma operação e fica em estado de espera aguardando o final da chamada. Na chamada assíncrona, o processo chama a operação e não necessita aguardar o final da chamada em estado de espera, o processo continua pronto para ser executado.

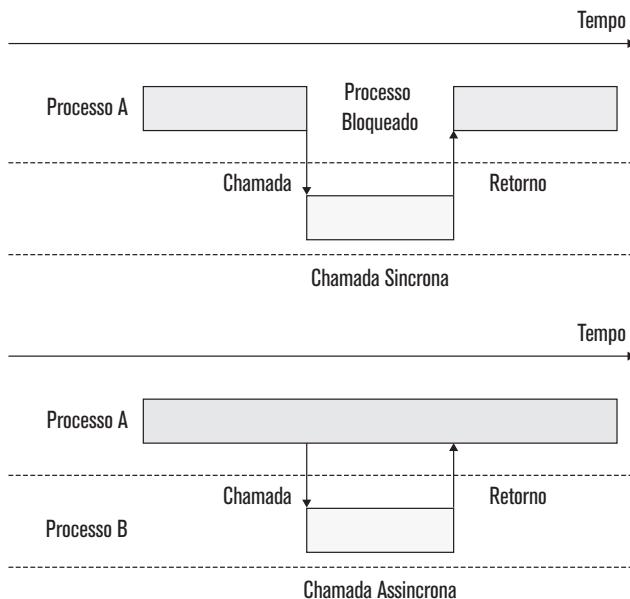


Figura 90 – Chamada síncrona e assíncrona.

5.4 Sistema de Arquivos

Estudamos em capítulos anteriores o armazenamento primário e secundário do sistema operacional. O armazenamento secundário e o local onde os arquivos (programas e dados) são armazenados de forma confiável e íntegro, mesmo que o computador permaneça longos períodos de tempo desligado. O armazenamento secundário pode ser feito por diversos tipos de dispositivos de entrada e saída como, por exemplo, disco rígido, discos ópticos, fitas magnéticas, etc. Os arquivos são gerenciados e organizados pelo sistema operacional através do sistema de arquivos.

Segundo Deitel et al. (2005) o sistema de arquivos tem a função de:

- Gerenciamento de arquivos: fornecer estrutura para que os arquivos possam ser armazenados, referenciados, compartilhados e que possam estar em segurança.
- Integridade do arquivo: garantir que os dados armazenados em um arquivo não sejam corrompidos por qualquer motivo que seja.
- Métodos de acesso: fornecer mecanismo de acesso aos dados armazenados.

Estas características permitem que o usuário possa, através dos sistemas de arquivo, estruturar os arquivos da melhor forma para que possam ser utilizados em cada aplicação. O sistema de arquivos deve prover mecanismos em que os arquivos possam ser compartilhados entre usuários de uma forma segura e controlada. O sistema de arquivo deve garantir a independência do dispositivo, ou seja, o usuário pode efetuar operações com os arquivos através de nomes simbólicos. Nomes simbólicos são nomes lógicos de fácil interpretação do usuário, por exemplo, *MeuDiretório:MeuArquivo.txt*. Caso fosse necessário utilizar nomes de dispositivos físicos para operações com arquivos, teria que especificar o lugar do dispositivo onde o arquivo deveria ser acessado, por exemplo, disco 2, blocos 782-791. Além disso, os usuários devem poder criar, modificar, eliminar arquivos e que o sistema de arquivo ofereça uma interface amigável e consistente a realização de todas as tarefas citadas anteriormente. Deitel et al. (2005) destaca ainda a importância do sistema de arquivo fornecer mecanismo que facilitem a criação de cópias de segurança e recuperação que permitam ao usuário a recuperar quaisquer dados danificados ou perdidos.



CONEXÃO

Leia mais sobre Sistema de Arquivos em:
[<http://pt.kioskea.net/contents/612-o-sistema-de-arquivos>](http://pt.kioskea.net/contents/612-o-sistema-de-arquivos).

5.5 Conceitos de Arquivos e Diretórios

Silberschatz et al. (2004) define os arquivos como sendo um conjunto de dados relacionados que é gravado no armazenamento secundário através de um nome. A gravação é feita no armazenamento secundário principalmente por ser um dispositivo não volátil como um disco rígido, discos magnéticos e discos ópticos.

O conjunto de dados contidos em um arquivo pode ser uma sequência de *bit*, *bytes*, linhas ou registros. Os dados ainda podem estar na forma numérica, alfabética, alfanumérica ou binária. Desta forma é possível armazenar um arquivo figura, texto, programa executável, programa fonte, música, vídeo, etc. Assim, os arquivos podem representar programas ou dados.

Na visão do usuário e dos programas os arquivos são uma maneira de gravar informações no disco rígido e de recuperá-las de volta em algum momento.

Tanto a gravação quanto a leitura deve ser feita de forma que o usuário não tenha que conhecer os detalhes de como o disco rígido trabalha realmente. Isto é feito através de uma visão lógica uniforme fornecida pelo sistema operacional para o armazenamento de informações.

Quando um arquivo é criado por um programa, ele recebe um nome, composto por uma cadeia de caracteres. Através do nome o arquivo pode ser acessado pelo programa que o criou ou por outros programas. Tanenbaum e Woodhyll (1999) afirma que dependendo do sistema operacional há regras diferentes para os nomes dos arquivos, mas todos os sistemas operacionais aceitam como nomes válidos de arquivos, sequências de caracteres com oito letras. Sistemas mais modernos permitem até 255 caracteres para nomes de arquivos. Outro ponto a ser observado é com relação a nomes escritos com letras maiúsculas e nomes escritos com letras minúsculas. No sistema Windows, por exemplo, o arquivo com nome *SISTEMA_OPERACIONAL.DOCX* é a mesma coisa que *sistema_operacional.docx* e *Sistema_Operacional.DocX*. Já para o UNIX, os nomes de arquivos *SISTEMA_OPERACIONAL.DOCX*, *sistema_operacional.docx* e *Sistema_Operacional.DocX* são arquivos distintos. Em geral, os sistemas operacionais permitem nomes de arquivos com duas ou mais partes separadas por um ponto. A parte após o ponto é chamada de *extensão*. A extensão tem como finalidade informar algo sobre o arquivo ou uma convenção definida para ele. Por exemplo, um arquivo com o nome *noticia.txt*, a extensão *.txt*, tem como convecção indicar que o arquivo provavelmente contém um texto. A tabela a seguir exibe algumas extensões mais comuns para arquivos.

EXTENSÃO	SIGNIFICADO
file.back	Arquivo de cópia de segurança
file.c	Programa fonte em C
file.gif	Imagen no formato de intercâmbio gráfico da Compuserve (graphical interchange format)
file.html	Arquivo de auxílio

EXTENSÃO	SIGNIFICADO
file.jpg	Documento do Word Wide Web em linguagem de marcação de hipertexto (hypertext markup language – HTML)
file.mp3	Imagen codificada com padrão JPEG
file.mpg	Música codificada no formato de áudio MPEG – camada 3
file.o	Filme codificado com padrão MPEG
file.pdf	Arquivo-objeto (saída do compilador, ainda não-ligado)
file.ps	Arquivo no formato portátil de documentos (portable document format – PDF)
file.tex	Arquivo no formato PostScript
file.tex	Entrada para o programa de formatação TEX
file.txt	Arquivo de textos
file.zip	Arquivo comprimido

Tabela 9 - Extensões típicas de arquivos.

Os arquivos possuem um conjunto de informações de controle conhecidas como atributos. Dependendo do sistema operacional, os atributos de um arquivo podem variar. Em geral os atributos são:

- Tamanho: o total de dados que estão armazenados no arquivo.
- Localização: o local onde o arquivo está armazenado.
- Acessibilidade: restrição quanto ao acesso dos dados do arquivo.
- Tipo: indica o formato dos dados do arquivo permitindo saber como o arquivo é utilizado. Arquivo utilizado como imagem, vídeo, áudio, etc.
- Datas: os arquivos mantêm as datas de criação, último acesso e a sua última modificação.

A figura abaixo exibe as telas, no Windows 7, com os principais atributos de um arquivo. Estas telas podem ser acessadas clicando com o botão direito do mouse sobre um arquivo e escolhendo a opção “Propriedade”.

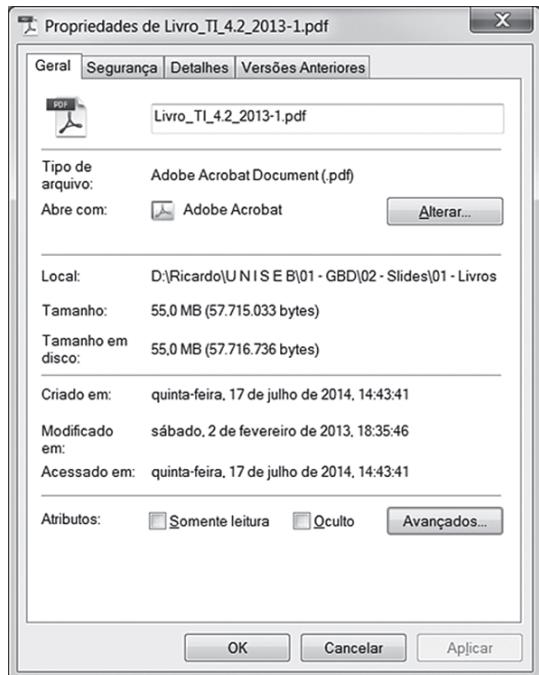


Figura 91 – Propriedade de um arquivo no Windows 7.

Os dados armazenados e recuperados em um arquivo, feitas por uma aplicação ou pelo sistema operacional, são feitas através de um conjunto de operações. As operações básicas oferecidas pelo sistema operacional são:

- Criar: permite a criação de um novo arquivo. A criação de um arquivo envolve definir nome, atributos, localização, permissões, etc.
- Abrir: permite a abertura do arquivo. Antes de abrir, o sistema operacional efetua várias operações, tais como verificar se o arquivo existe na localização informada, se as permissões informadas permitem o acesso ao arquivo, etc.
- Ler: permite a leitura do arquivo.
- Escrever: permite adicionar dados ao arquivo.
- Copiar: permite que um arquivo seja copiado.

- Renomear: permite que um arquivo seja renomeado.
- Listar: permite exibir ou imprimir o conteúdo de um arquivo.
- Fechar: impedir que novas referências sejam feitas a um arquivo até que o mesmo seja reaberto.
- Destruir: elimina o arquivo do armazenamento secundário.

Deitel et al. (2005) descreve as operações que podem ser feitas para a manipulação do conjunto de dados contidos em um arquivo. Estas operações são:

- Ler: a memória de um processo recebe uma cópia dos dados contidos em um arquivo.
- Escrever: os dados contidos na memória de um processo são copiados para um arquivo.
- Atualizar: permite alterar os dados contidos em um arquivo.
- Inserir: permite inserir dados em um arquivo.
- Apagar: permite remover dados de um arquivo.

5.5.1 Estrutura de arquivos

A forma como os dados estão organizados internamente dentro de um arquivo é conhecida como *estrutura de arquivos*. As estruturas de arquivos podem ser feitas de diversas formas. Algumas das estruturas mais comuns encontradas para arquivos são apresentadas na figura 92.

A primeira forma de organização (figura 92 (a)) é uma sequência de *bytes* não estruturados. Neste tipo de estrutura, a aplicação que define toda a organização interna do arquivo. A forma de estruturação de um arquivo é apresentada na figura 92 (b). Nesta estrutura, um arquivo contém um conjunto de registros de comprimento fixo. Os registros possuem uma estrutura interna para organizar os dados. Na terceira forma (figura 92 (c)) os registros não tem a necessidade de serem do mesmo comprimento.

O registro possui um . O campo chave permite uma organização em forma de árvore de registro, como também, permitir a rápida localização de uma chave particular dentro da árvore.

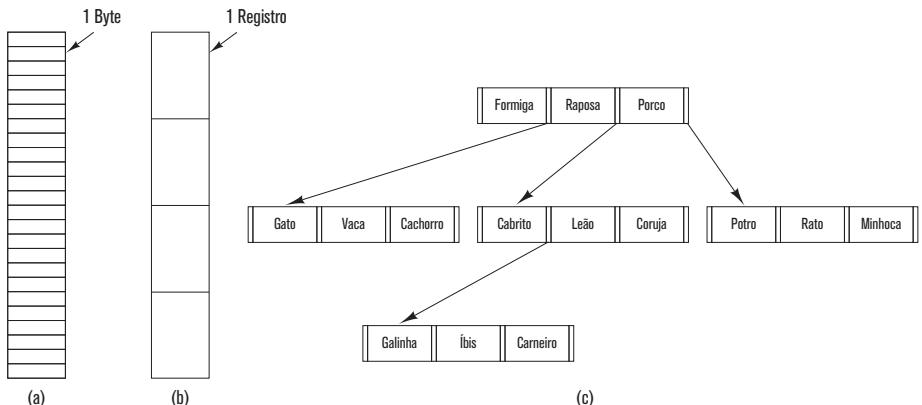


Figura 92 – Tipos de estruturas de arquivos. (a) Sequencia de bytes. (b) Sequencia de registros. (c) Árvore.

Fonte: Tanenbaum e Woodhyll (1999)

5.5.2 Sistema de Diretório

Um sistema pode conter uma elevada quantidade de arquivos armazenados em seu disco rígido. A organização lógica destes arquivos é feita pelo sistema de arquivos através de *estruturas de diretórios*.

A forma como os diretórios são organizados pode variar de acordo com o sistema operacional. As formas mais comuns de organização são mostradas na figura 99. A primeira forma de estrutura é que foi utilizada nos primeiros computadores é mostrada na figura 99 (a). Esta estrutura, considerada a forma mais simples de organização, é uma estrutura de *diretório de nível único (ou plano)*. No diretório de nível único, todos os arquivos de todos os usuários são mantidos em um único diretório. O problema desta estrutura é que os usuários criem arquivos com o mesmo nome.

A figura 93 (b) exibe uma estrutura de *diretório em dois níveis*. Com a estrutura em dois níveis é possível que cada usuário possua o seu próprio diretório. Desta forma, os usuários podem criar seus arquivos com qualquer nome sem ter a necessidade de verificar se outro usuário criou um arquivo com o mesmo nome. Para permitir que os usuários possam agrupar seus arquivos de forma

lógica surgiu a abordagem mostrada na figura 93 (c). Esta estrutura é conhecida como *diretório estruturados em árvore*. Nesta estrutura os usuários podem criar quantos diretórios forem necessários permitindo que possam agrupar seus arquivos de forma lógica.

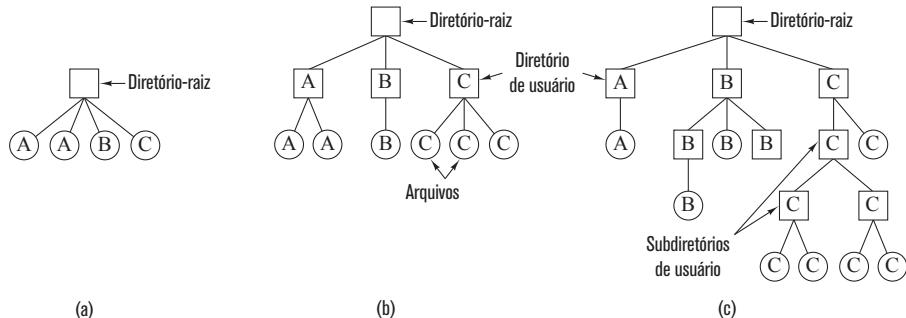


Figura 93 – Três projetos de sistemas de arquivos. (a)Diretório de nível único (ou plana). (b) Diretório em dois níveis. (c) Diretório em árvore.

Fonte: Tanenbaum e Woodhyll (1999).

Segundo Deitel et al. (2005), um diretório é um arquivo onde são armazenados os nomes e as localizações referentes a outros arquivos.

5.6 Métodos alocação

A alocação de espaço em armazenamento secundário enfrenta problemas semelhantes aos de alocação da memória principal. Toda vez que se aloca e libera espaços no disco rígido, este fica cada vez mais fragmentado. A fragmentação do disco faz com que os arquivos fiquem espalhados por blocos, muitas vezes distantes uns dos outros, que podem causar perda de desempenho do sistema.

Existem três formas mais utilizadas para alocar espaço em disco: contíguo, encadeado e indexado.

5.6.1 Alocação Contígua

A abordagem de Alocação Contígua é a mais simples de alocação de espaço no disco rígido. Na alocação contígua um arquivo ocupa um conjunto de blocos sequenciais. Para acessar um arquivo, o sistema operacional necessita conhecer o endereço físico do bloco e o tamanho do arquivo. O problema neste tipo de alocação é que se não houver a quantidade de espaço contíguo suficiente para armazenar um arquivo, este arquivo não pode ser criado. Outro problema está relacionado ao tamanho do arquivo que tende a mudar de tamanho conforme é alterado pelo usuário. Isto pode acarretar um aumento no tamanho do arquivo e talvez não haja espaço contíguo disponível para suportá-lo.

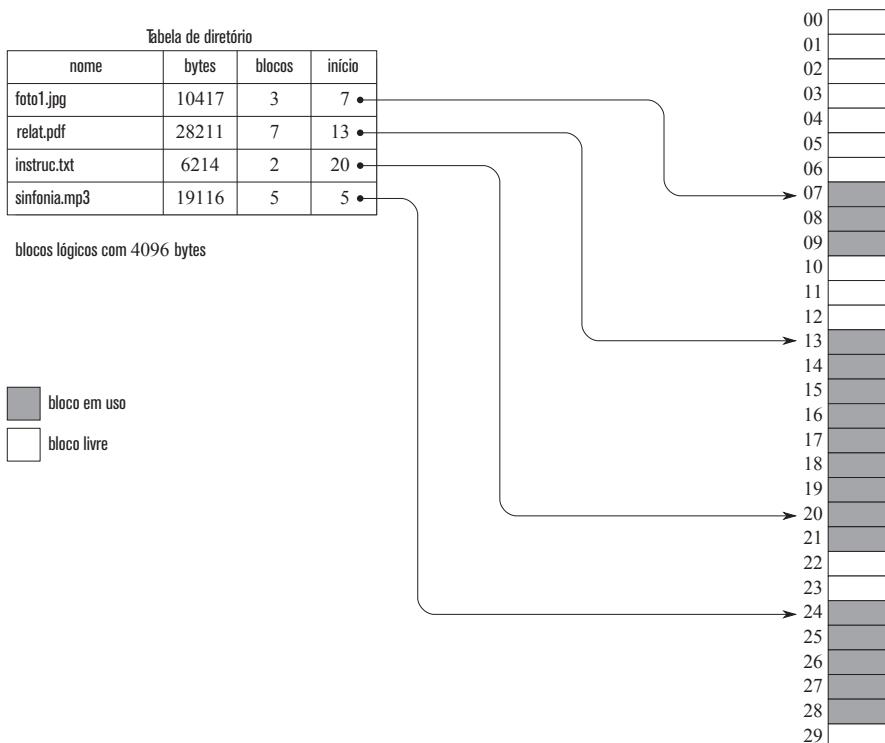


Figura 94 – Estratégia de alocação contígua.

Fonte: Maziero (2014).

5.6.2 Alocação Encadeada

A alocação encadeada permite que os blocos sejam alocados de forma não contígua. Os blocos de um arquivo podem estar armazenados em locais diferentes no disco rígido. O diretório contém um ponteiro para o primeiro bloco do arquivo, este bloco, por sua vez, aponta para o próximo, e assim sucessivamente o método de alocação encadeada resolve boa parte dos problemas encontrados na alocação contígua. O problema neste tipo de alocação, está no fato dos blocos dos arquivos estarem espalhados pelo disco rígido, há um aumento no tempo de acesso destes arquivos.

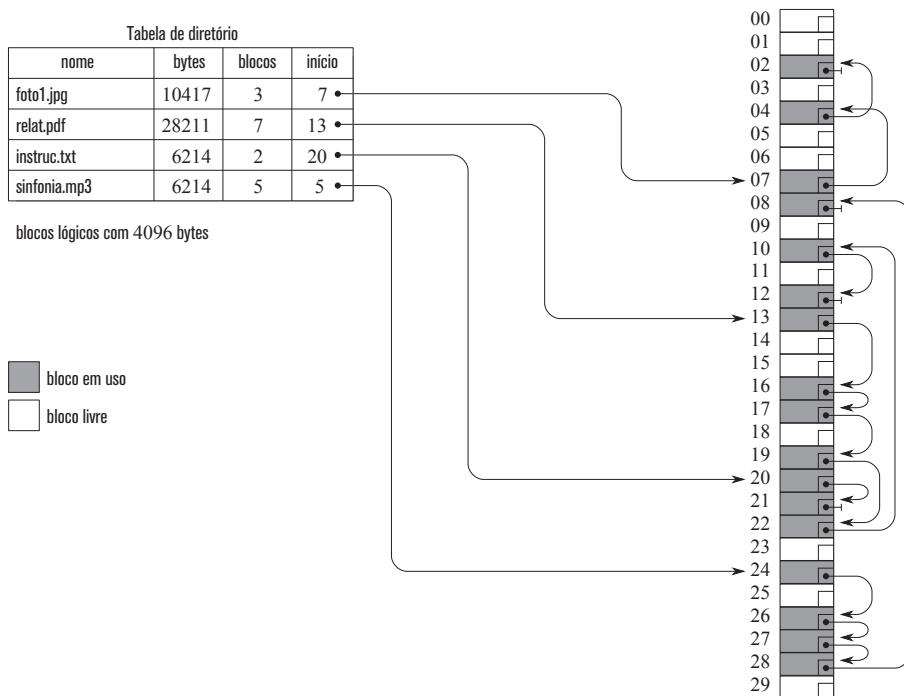


Figura 95 – Alocação Encadeada.

Fonte: Maziero (2014).

5.6.3 Alocação indexada

Um problema da alocação encadeada é que para acessar um determinado bloco é necessário iniciar a busca no primeiro bloco e ir navegando na lista até encontrar o bloco desejado. Não é possível acessar o bloco diretamente. Este problema foi solucionado com a alocação indexada. Neste método todos os ponteiros de um arquivo são colocados em um mesmo bloco chamado *bloco de índices*, conforme figura abaixo. Desta forma qualquer bloco do arquivo pode ser acessado diretamente.

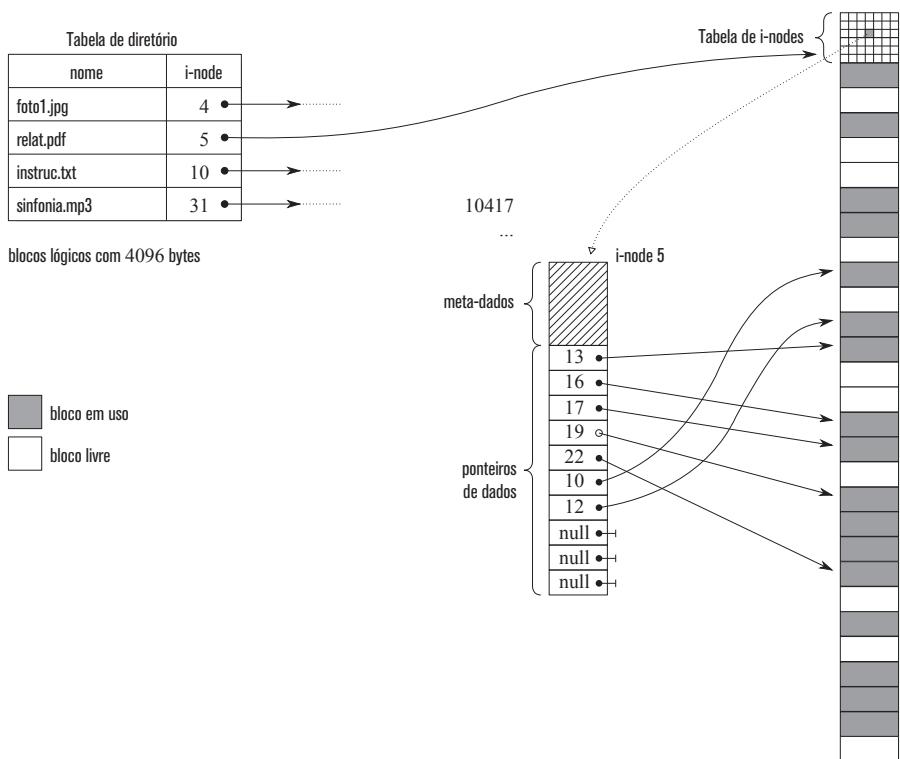


Figura 96 – Alocação indexada.

Fonte: Maziero (2014).

5.7 Gerência de espaços livres

Devido a quantidade limitada de espaço em disco rígido, o sistema operacional tem que manter informações dos espaços livres no disco para que possam ser utilizados. Estes espaços livres incluem os já existentes além daqueles liberados quando um arquivo é excluído do sistema. As estratégias mais comuns adotadas pelo sistema operacional para manter estas informações são:

- Mapa de *bits*: neste método cada bloco no disco é representado por um *bit*. Se o bloco estiver em uso, o seu *bit* no mapa de *bits* tem o valor igual a 1. Se o bloco está livre, o valor referente a seu *bit* tem o valor igual a 0 (figura 97 (a)).
- Lista encadeada: este método mantém uma lista encadeada com todos os blocos livres (figura 97 (b)).
- Tabela de blocos livres: esta abordagem leva em consideração a utilizada de blocos contíguos. A tabela mantém o endereço do primeiro bloco de cada segmento e a quantidade da sequência de blocos (figura 97 (c)).

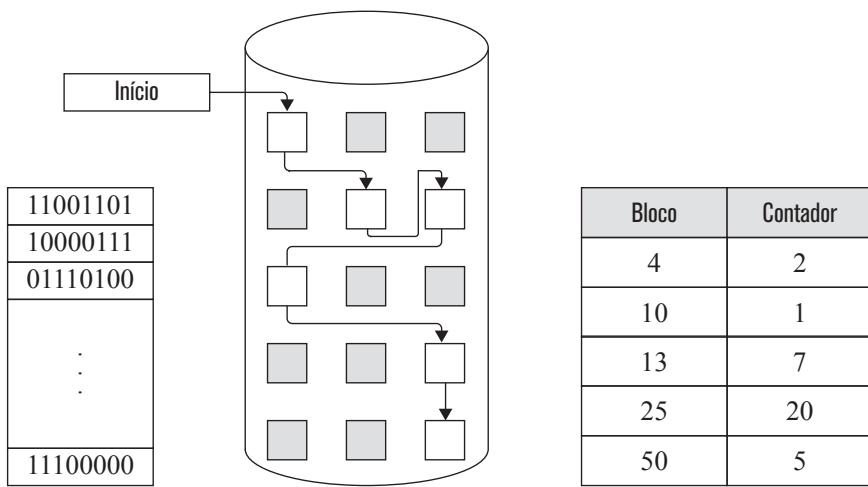


Figura 97 – Gerência de espaços livres.

Fonte: Machado e Maia (2007).

5.8 Proteção de acesso

Devido o acesso aos arquivos por diversos usuários e levando em consideração que muitos destes arquivos podem conter dados confidenciais como senhas, números de cartão de crédito, etc., há uma grande necessidade do sistema operacional fornecer mecanismos de controle de acesso aos arquivos. Entre os tipos de operações que podem ser controladas são:

- Leitura: controla qualquer operação para visualizar, editar ou copiar o arquivo.
- Escrever: controla qualquer operação para incluir ou alterar o arquivo.
- Executar: controla qualquer operação de carregamento e execução do arquivo.
- Excluir: controla qualquer operação para excluir o arquivo.

O controle de acesso a um arquivo pode ser feito através de senha de acesso. Somente o usuário que possuir a senha de acesso pode efetuar as operações descritas acima em relação ao arquivo. Outra forma de controle é baseada em grupos de usuários. Nesta abordagem o acesso a um arquivo é dado a um grupo de usuário. Os usuários então são adicionados a este grupo quando necessitarem fazer acesso ao arquivo.



ATIVIDADE

1. De acordo com o material, qual é a forma que o sistema operacional acessa os dispositivos de entrada e saída?
2. Os dispositivos de entrada e saída podem ser classificados levando em consideração a análise de diversos autores. Quais são as formas de classificação de um dispositivo quando são consideradas as formas como os dados são acessados a partir deste dispositivo.



REFLEXÃO

Neste capítulo estudamos os principais componentes de *hardware* e *software* de entrada e saída. Vimos como o sistema operacional tem a difícil tarefa de gerenciar a complexidade dos diversos tipos de dispositivos de entrada e saída que podem variar em função, velocidade, etc.

Fomos além, e compreendemos como são feitas as comunicações entre estes dispositivos e o sistema operacional, através das controladoras, *drives*, etc. Por fim estudamos os principais aspectos envolvidos no gerenciamento de arquivos e diretórios. A compreensão das dificuldades do sistema de gerenciamento de dispositivos de entrada e saída como também do sistema de gerenciamento de arquivos é de grande relevância para um melhor entendimento das soluções providas para um sistema operacional.

Não pare os seus estudos, mantenha-se sempre lendo e pesquisando sobre os diversos temas relacionados aos sistemas operacionais. Esperamos que todos estes conhecimentos adquiridos sejam um diferencial em sua vida profissional. Desejamos a você um grande sucesso!



LEITURA

Para você avançar mais o seu nível de aprendizagem envolvendo os conceitos de sistemas operacionais e demais assuntos deste capítulo, consulte as sugestões de *links* abaixo:

Capítulos 11 e 12 do livro

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G **Fundamentos de Sistemas Operacionais**. 6^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2004.



REFERÊNCIAS BIBLIOGRÁFICAS

CARPINELLI, J.D. *Computer Systems Organization and Architecture*. Boston, MA: Addison-Wsley, 2001.

DEITEL H. M.; DEITEL P. J.; CHOHNES D. R. **Sistemas Operacionais**. 3^a ed. São Paulo, Editora Prentice-Hall, 2005.

HENNESSY, J. L.; PATTERSON, D. A. **Arquitetura de Computadores**: uma abordagem quantitativa. 3^a ed. Rio de Janeiro: Campus, 2003.

JOHNSTONE, M. S.; WILSON, P. R. **The memory fragmentation problem**: solved? ACM SIGPLAN Notices, 34(3): 26–36, 1998.

MACHADO, F. B.; MAIA, L. P. **Arquitetura de sistemas operacionais**. 4^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2007.

MAZIERO, C. A. **Sistemas Operacionais:** Conceitos e Mecanismos. Disponível em: <<http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/so:so-livro.pdf>>. Acesso em: set. 2014.

OLIVEIRA, R. S., CARISSIMI, A. S., TOSCANI, S. S. Sistemas Operacionais. 4^a ed. Porto Alegre : Editora Bookman, 2010.

PATTERSON, D. A.; HENNESSY, J. L. **Organização e Projeto de Computadores.** 3^a Ed. Rio de Janeiro: Campus, 2005.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G **Fundamentos de Sistemas Operacionais.** 6^a ed. Rio de Janeiro: LTC - Livros Técnicos Editora S.A., 2004.

STALLINGS, W. **Arquitetura e Organização de Computadores,** 5^a Edição, Prentice Hall, São Paulo, 2002.

TANENBAUM, A. S.; WOODHYLL, A. S. **Sistemas operacionais projeto e implementação.** 2^a ed Porto Alegre: Bookman, 1999.



EXERCÍCIO RESOLVIDO

Capítulo 1

1. De acordo com o material, qual das opções abaixo pode ser classificada como sistema operacional?
 - a) Android e MS-Excel.
 - b) Windows e Internet Explorer.
 - c) Linux e Windows.
 - d) Android e MS-Word.
 - e) Linux e Broffice.

Dos programas acima citados, os que são considerados sistemas operacionais são: Android, Windows e Linux. Assim, a opção correta é a C.

2. Quais as principais funções do kernel?

As principais funções do kernel são a sincronização e comunicação entre processos, escalonamento e controle dos processos, gerência de memória, gerência de sistema de arquivos, gerência de dispositivos de E/S.

3. Para que uma aplicação possa utilizar os serviços disponíveis pelo Kernel deve efetuar a comunicação com o Kernel por meio de:
 - a) Device Drivers
 - b) Batch
 - c) Scripting
 - d) System Calls
 - e) Shell

A resposta certa é a D. O System Calls é um intermediário entre as aplicações do usuário e o Kernel.

Capítulo 2

1. O processo é composto por três partes, entre elas o contexto de hardware. Das opções abaixo, qual opção que faz parte do contexto de hardware?

- a) Endereço de memória
- b) Registrador PC
- c) PID
- d) Privilégio
- e) Tempo de processador

A resposta correta é a B. Das opções acima, apenas o Registrador PC faz parte do contexto de hardware.

2. O processo é composto por três partes, entre elas o contexto de software. Das opções abaixo, qual opção que faz parte do contexto de software?

- a) Registrador SP
- b) Registrador PC
- c) PID
- d) Endereço de memória
- e) Registrador de status

A resposta correta é a C. Das opções acima, apenas o PID faz parte do contexto de software.

3. Por que os processos I/O-Bound ficam no estado “Em Espera” durante grande parte do

seu ciclo de vida

Porque os processos I/O-Bound estão mais ligados aos dispositivos de entrada e saída, cujo acesso são mais lentos que outros componentes do sistema

Capítulo 3

1. De acordo com o material, o que é política de escalonamento de um processo?
Através da política de escalonamento, o sistema operacional gerencia o processador definindo qual processo deve ser executado, tornando possível a multiprogramação.
2. Qual a diferença entre o escalonamento preemptivo e não-preemptivo?
O escalonamento não-preemptivo permite que o processo seja executado do início ao fim sem ser interrompido até ser finalizado. Já no escalonamento preemptivo, o escalonador tem a capacidade de trocar um processo, que poderia continuar executando, por outro.
3. Descreva as principais diferenças entre os escalonamentos do tipo Circular e FIFO?
O escalonamento circular o sistema operacional determina um período de tempo que um processo pode ser executado pelo processador. No final do tempo, o processo é trocado por outro. Já no escalonamento FIFO o processo utiliza o processador até terminar para então ser trocado por outro. O escalonador FIFO é não-preemptivo enquanto o circular é preemptivo.

Capítulo 4

1. Quais são as formas de alocação de memória pelo sistema operacional?
O sistema operacional pode alocar a memória de forma Contígua ou Não-Contígua. A forma Contígua pode ser Simples ou Particionada Estática (Fixa) / Dinâmica. Já a Não-Contígua pode ser por Paginação, Segmentação ou Segmentação com Paginação.
2. Como funciona a técnica conhecida como Sobreposições (overlay)?
Nesta técnica o programa é desenvolvido de forma que possa ser dividido em partes lógicas independentes de forma a permitir que uma parte possa ser substituída por outra quando não estiver em uso.

Capítulo 5

1. De acordo com o material, qual é a forma que o sistema operacional acessa os dispositivos de entrada e saída?

O sistema operacional não acessa diretamente os dispositivos de entrada e saída. O acesso aos diapositivos é feito através de uma controladora.

2. Os dispositivos de entrada e saída podem ser classificados levando em consideração a análise de diversos autores. Quais são as formas de classificação de um dispositivo quando são consideradas as formas como os dados são acessados a partir deste dispositivo.

De acordo com Tanenbaum e Woodhyll (1999), que leva em consideração a forma em que os dados podem ser acessados e/ou obtidos a partir dispositivos de E/S, os dispositivos podem ser classificados em: dispositivos de blocos ou dispositivos de caractere.
