

Índice

Bibliografia	0
Introdução	0
Hierarquia de Memória	99
Histórico dos sistemas operacionais	372
Classificações de Sistemas Operacionais	407
Tratamento de E/S em Sistemas Multiprogramáveis	540
Processos e Estrutura de Sistemas Operacionais	642
Contexto de <i>Software</i>	673
Gerência do Processador	821
Exercícios	904
Sincronização na Comunicação entre processos	973
Gerência da Memória	1108
Outra visão de GERÊNCIA DE MEMÓRIA	1268

[Índice](#)

Bibliografia

- *Arquitetura de Sistemas Operacionais* (Francis B. Machado)
- *Sistemas Operacionais Modernos* (Andrew S. Tanenbaum)

[Índice](#)

Introdução

Definicao de Sistema: Conjunto de partes, funcionalmente independentes, que trabalham de maneira harmônica objetivando um fim comum.

- 1- Conjunto de partes – Não pressupõe uma seqüência;
- 2- Funcionalmente independentes – Cada parte do sistema executa uma função bem definida, diferente de qualquer outra parte;
- 3- trabalham de maneira harmônica – Executam sua função da melhor maneira possível em cooperação com as outras partes;
- 4- fim comum - O objetivo do programa.

➤ *O hardware e o sistema operacional*

Sistema Operacional: É o programa (Software (SW)) de computador responsável por duas tarefas básicas:

- Gerenciamento dos recursos de sistema
- Interface com o usuário

Interface – É a forma do usuário interagir com o sistema

Gerência de recursos: (Elementos básicos do Sistema de Computação)

- CPU: (Central Processing Unit) – Processador, Parte do sistema onde são executadas as instruções.
- MEMÓRIA: Real ou Primária – Armazenamento de dados.
- PERIFÉRICOS: “O que sobrou” – Dispositivos de entrada e/ou saída de dados.

Para entendimento do conceito de sistema operacional, objeto desta disciplina, vamos tomar como exemplo uma configuração de *hardware* (conjunto de componentes eletrônicos interligados, composto de processador, memória principal, e dispositivos de entrada e saída: discos magnéticos, monitor, impressora, etc.) de um microcomputador PC. Basta abrir o caderno de informática de um jornal e escolher uma configuração, dentre as tantas existentes com os mais diferentes preços. Ex:

Componente	Especificação (capacidade, marca, modelo)
Placa-mãe (<i>mainboard</i> , <i>motherboard</i>)	ASUS P2L97
Processador (UCP)	INTEL PENTIUM III
Clock	800MHz
Memória Principal (RAM)	64Mb
Memória Secundária (FD, HD, CD-ROM)	Zip Drive 3 ½ “, 10 Gb e 52X
Memória Cache	256Kb <i>on-board</i>
Placa e monitor de vídeo	DIAMOND AGP com 8Mb, SVGA colorido
Kit Multimídia	CREATIVE qq com acessórios
Placa Fax/Modem	US-Robotics de 56K
Gabinete, teclado, mouse, estabilizador	Mini-torre, ABNT2 com 104 teclas, padrão MICROSOFT, qq de 1KVA
Outros: impressora, scanner, câmera digital, joystick	Jato de tinta HP, de mesa com boa resolução, etc.

Se numa próxima etapa, de posse de todos esses componentes, um bom técnico conectá-los de maneira correta, será que esse microcomputador funciona? Alguns ajustes no SETUP (programa de configuração/verificação gravado em um componente da placa-mãe) se fazem necessários, mas certamente o computador ligaria e mostraria toda sua configuração de *hardware*.

A pergunta agora seria um pouco mais específica: do jeito que foi feita a montagem e configuração dos componentes, qual seria a utilidade desse microcomputador? A resposta é simples, NENHUMA. Para que o usuário possa utilizar seus programas em benefício de suas atividades profissionais (editores de texto, planilhas eletrônicas, banco de dados, etc.), ou mesmo lazer (jogos), é necessário que seja instalado um SOFTWARE INDISPENSÁVEL que tem como função mais abrangente GERENCIAR todos os recursos (*hardware e software*) disponíveis no sistema de computação (no caso o microcomputador), chamado SISTEMA OPERACIONAL. O sistema operacional é software : um conjunto de rotinas que são executadas pelo processador para **facilitar** o acesso aos componentes de *hardware* (processador, memória, dispositivos de E/S), e **gerenciar** o uso do sistema de computação (*hardware e software*).

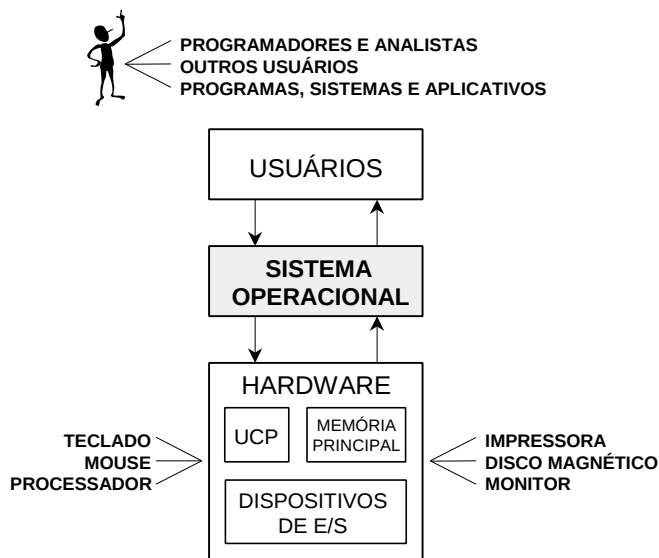


Figura 1 - O que é um sistema operacional?

Então, podemos definir o SISTEMA OPERACIONAL como :

conjunto de rotinas que controlam a execução das aplicações de usuário cujas funções principais são :

- fazer a interface entre o homem e o hardware da máquina; fazer a comunicação entre o usuário, o computador e seus periféricos;
- gerenciar o compartilhamento dos recursos de hardware e software entre várias tarefas e/ou programas, de forma a fazer com que o acesso a esses recursos seja feito de forma **segura e eficiente**.

Algumas atividades que envolvem o sistema operacional:

- leitura de um disquete (acionar a cabeça de leitura e gravação, posicionar trilha e setor, dados do disco para a memória)
- quando um usuário solicita a execução de um programa, o sistema operacional deve alocar espaço na memória para carregar e acessar o programa.

Quais são as etapas anteriores à carga do sistema operacional em um microcomputador?

Quando o microcomputador é ligado, realiza o **POST (Power On Self Test)** que consiste em um conjunto de testes para determinar se o *hardware* está funcionando corretamente (verificação da RAM, vídeo e outros dispositivos; localização da unidade de disco de inicialização; etc.). Estes testes são realizados a partir do BIOS (*Basic Input Output System*, podendo variar conforme cada fabricante).

Se o BIOS for compatível com a tecnologia *Plug and Play*, algumas rotinas adicionais serão realizadas para reconhecimento dos dispositivos. Após a inicialização dos dispositivos, o computador localiza e lê o setor de inicialização, contendo o arquivo carregador, que será carregado para a memória principal e passará a controlar a carga do sistema operacional (*boot*).

➤ **A divisão do hardware de um computador**

Como já havia sido comentado, os componentes do *hardware* de um computador são divididos em três grupos/subsistemas básicos:

1. **Unidade Central de Processamento (UCP ou processador)** – é o “cérebro” do computador, responsável pela execução de todos os programas armazenados na memória principal. A UCP é composta de várias partes:

- **ULA (Unidade Lógica e Aritmética)** : realiza as operações aritméticas (soma, subtração, divisão e multiplicação) e lógicas (AND, OR , XOR), necessárias à execução das instruções;
- **UC (Unidade de Controle)** : responsável pelo controle de todo o hardware. (ver, p.ex., leitura e escrita na memória principal);
- **Registradores** : áreas de memória para armazenamento de dados e instruções temporários (*registradores de dados*) e informações de controle (*registradores de controle*). Registradores de controle importantes :

Program Counter (PC) ou Contador de Instruções (CI) : contém o endereço da próxima instrução a ser executada.

Instruction Register (IR) ou Registrador de Instruções (RI) : armazena a instrução que está sendo executada;

Stack Pointer (SP) ou Apontador de Pilha (AP) : aponta para o endereço do topo da pilha.

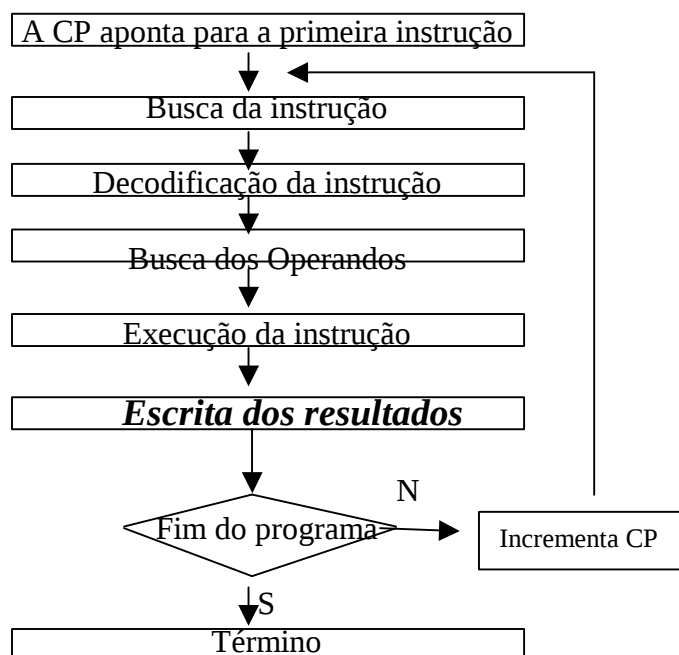
A pilha é uma estrutura de dados na qual o ultimo dado a chegar é o primeiro a sair. A pilha do processador contém vários endereços de memória, com o objetivo de indicar à CPU o “caminho da execução”. As instruções são armazenadas na memória principal em endereços consecutivos. Se o fluxo dessas instruções for contínuo, o PC irá incrementar de 1 a cada execução de uma instrução. Se houver um

desvio para uma rotina, o PC passará a conter o primeiro endereço de memória da rotina. Ao término da rotina o fluxo deverá retornar para o mesmo “lugar” (endereço) onde estava quando houve a chamada da rotina. É por isso que este endereço tem que ficar guardado na pilha.

Faça um esquema, contendo o conteúdo do PC e a pilha para o caso de um programa em execução chamar a Rotina Calcula quando está executando a instrução armazenada no endereço 02001110. As instruções da Rotina Calcula estão armazenadas na memória dos endereços 0011002A até o endereço 00111000. A instrução armazenada no endereço 00110040 faz uma chamada à Rotina Ordem. A Rotina Ordem está armazenada nos endereços 00111000 até 0011147B. (acompanhar em sala com o professor)

- **Clock** : dispositivo interno ao processador que gera pulsos elétricos síncronos em um determinado intervalo de tempo (sinal de *clock*). A quantidade de vezes que este pulso se repete em um segundo define a *frequência do clock*, medida em Hertz. O sinal de *clock* é utilizado pela unidade de controle para a execução das instruções.

Passos para executar um Programa



Vejamos alguns conceitos relacionados aos processadores de um computador:

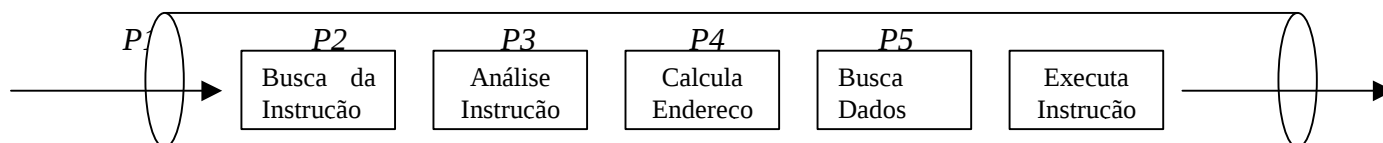
1.a) *Pipelining* – a execução de uma instrução pelo processador é feita em vários passos :

- **busca** da instrução na memória principal
- **decodificação** da instrução
- **execução** efetiva da instrução
- **escrita** do resultado da execução da instrução (em um registrador)

Depois disso o processador faz a busca da próxima instrução. O conjunto desses passos é chamado **ciclo de instrução**. Normalmente a busca, a decodificação e a escrita são feitas, cada uma, em um ciclo de clock. Entretanto, para a execução podem ser necessários mais de um ciclo, o que vai depender da *arquitetura do processador* e da própria instrução.

Os processadores com *pipelining* tem uma unidade para cada passo do ciclo de instruções, sendo que tem mais de uma unidade para o passo de execução.

Pipelining: Esse processamento nos passa a idéia de uma produção em série, onde cada tarefa é dividida em uma seqüência de sub-tarefas executadas em diferentes estágios.



Vamos acompanhar a execução de um programa com as instruções número 1,2 , 3, 4, ..., 25. Suponha que não há nenhum desvio de fluxo, ou seja, que estas instruções serão executadas efetivamente nesta ordem. Preencha os campos abaixo com os números das instruções . (Acompanhamento do professor)

CICLO nº	Busca	Decodifica	Executa 1	Executa 2	Escreve
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
28					
29					

Em quantos ciclos estas instruções seriam executadas sem o pipelining ? Quantos ciclos foram necessários à execução com o pipelining? Comente estes resultados. O que aconteceria se no passo execução2 da instrução numero 10 fosse concluído que a próxima instrução a ser executada é a instrução numero 2 (desvio de fluxo) ? Faça nova comparação entre a execução com o pipelining e sem o mesmo. Comente.

1.b) Arquitetura CISC X Arquitetura RISC

Um processador com arquitetura *CISC* (*Complex Instruction Set Computer*) caracteriza-se por possuir instruções complexas que são interpretadas pelo nível de microporgramação, onde são convertidas em microinstruções. O número de registradores é pequeno e qualquer instrução pode referenciar a memória principal. Exemplos de processadores dessa arquitetura: VAX (DEC), 80x86 e o Pentium (INTEL), 68xxx (Motorola), além da série 360 e todos os mainframes (IBM).

Já um processador de arquitetura RISC (*Reduced Instruction Set Computer*) se caracteriza por possuir poucas instruções de máquina, em geral bastante simples, que são executadas diretamente pelo *hardware*, sem a necessidade da camada de interpretação. Na sua maioria, estas instruções não acessam a memória principal, trabalhando principalmente com registradores que, neste tipo de processador, se apresentam em grande número. Estas características, além de ajudarem as instruções serem executadas em alta velocidade,

facilitam a implementação de *pipeline*. Exemplos de processadores dessa arquitetura: Sparc (SUN), RS-6000 (IBM, PA-RISC (HP), Alpha AXP (DEC), R_x0000 (MIPS).

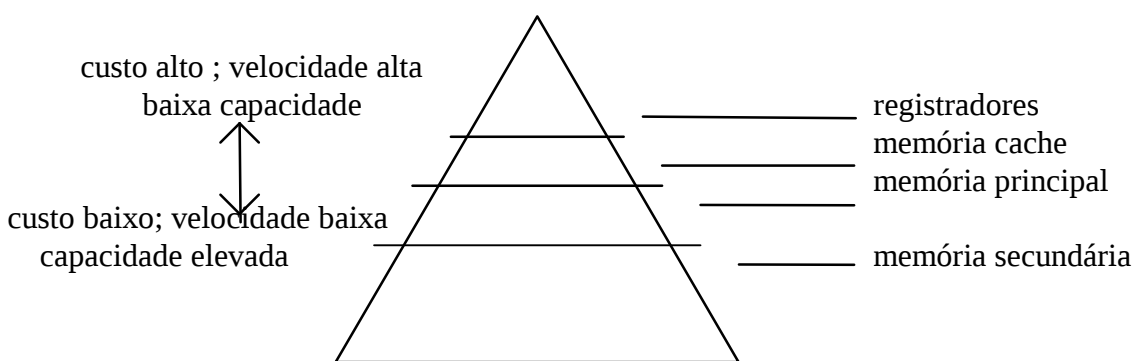
[Índice](#)

Hierarquia de Memória

Em geral em um mesmo computador coexistem diferentes tipos de memória com diferentes tipos de aplicações. Por exemplo é fundamental que as transferências de informações internas ao processador central (CPU) sejam realizadas no menor espaço de tempo possível (para não subutilizá-la). Neste caso a velocidade de transferência é crítica, enquanto a quantidade de informação a ser manipulada é mínima. Já em outros casos a capacidade de armazenamento pode ser o fator crucial.

Essas diferentes características desejáveis, aliadas ao custo (ver figura abaixo) torna inviável a implementação de um computador com um único tipo de memória. Na realidade, existem muitas memórias em um computador, que se interligam de forma estruturada, constituindo um sistema, que denominamos **subsistema de memória**.

Esse subsistema é projetado de modo que seus componentes sejam organizados hierarquicamente, conforme a figura abaixo:



A forma piramidal com base larga simbolizando uma elevada capacidade de armazenamento, tempo de acesso (leitura/escrita) e o custo do tipo de memória em questão.

Os principais parâmetros para análise das características de cada tipo de memória são: **Tempo de acesso** (leitura/escrita), **Capacidade de armazenamento**, **Volatilidade**, e **Tecnologia de fabricação**.

Tempo de acesso: Indica quanto tempo a memória gasta para colocar uma informação disponível no barramento de dados após uma determinada posição ter sido endereçada para leitura.

Capacidade de armazenamento: Refere-se a quantidade máxima de informação que pode ser guardada em uma memória, a unidade de medida mais comum é o “byte”, embora possam também ser utilizadas outras unidades dependendo do tipo de memória, por exemplo: setores (no caso de discos), “bits”(no caso de registradores), etc. Dependendo do tamanho da memória, isto é, da sua capacidade de armazenamento, indica-se o valor numérico total de elementos de forma simplificada, através da inclusão das abreviações K(quilo), M(mega),G(giga),T(tera), e etc.

Volatilidade: As memórias podem ser do tipo volátil ou não volátil. Uma memória do tipo não volátil é aquela que retém a informação nela contida mesmo quando sua fonte de alimentação é desligada, enquanto que a memória do tipo volátil não é capaz de suportar sequer “quedas” de tensão.

Como exemplo de memórias do tipo volátil podemos citar: registradores e as memórias de semicondutores do tipo **RAM** (Random Access Memory), como exemplo de memórias do tipo não volátil podemos ter: memórias magnéticas e óticas como discos e fitas e, as memórias semicondutoras do tipo **ROM** (Read Only Memory) e **EPROM** (Erasable Programable Read Only memory) e **EEPROM** (Electrically Erasable PROM).

Tecnologia de fabricação: Com a evolução dos computadores diversas tecnologias vêm sendo empregadas na construção de memórias, algumas destas tecnologias já estão obsoletas, enquanto outras ainda

não se apresentam disponíveis no mercado comercial, como é o caso da memória de bolha (buble memory), como tecnologias mais conhecidas utilizadas podemos citar:

Memórias semicondutoras, que são dispositivos fabricados com circuitos eletrônicos e baseados em semicondutores, são rápidas e relativamente caras, e as **Memórias Magnéticas**, que são dispositivos eletromecânicos, nestes dispositivos as informações são armazenadas sob forma de campos magnéticos, eles possuem características magnéticas semelhantes as fitas cassete de uso doméstico.

Registradores

Em um sistema de computação, o destino final do conteúdo de qualquer tipo de memória é o processador(CPU). Isto é, o objetivo final de cada uma das memórias é armazenar informações destinadas a serem, em algum momento utilizadas pela CPU. Ela é responsável pela execução das instruções, pela manipulação dos dados e pela produção dos resultados das operações.

As ações operativas da CPU são realizadas na ALU(Unidade lógica e Aritmética) e na FPU(Unidade de ponto flutuante). Entretanto, antes que as instruções sejam interpretadas e as unidades da CPU sejam acionadas, o processador necessita “buscar” as instruções onde elas estiverem armazenadas (memória cache ou principal) e armazená-la em seu próprio interior, em um dispositivo de memória denominado registrador.

Em seguida a este armazenamento da instrução, a CPU deverá, na maioria das vezes, buscar dados da memória (cache, principal ou mesmo da memória secundária) para serem manipulados pela ALU. Esses dados também necessitam ser armazenados em algum local do processador até serem efetivamente utilizados. Os resultados de um processamento também precisam (as vezes) ser armazenados temporariamente na CPU, ou para serem novamente manipulados pela ALU por uma outra instrução, ou para serem transferidos para uma memória externa ao processador.

Um registrador é, portanto, o elemento superior da pirâmide de memória, por possuir maior velocidade de transferência (menor tempo de acesso), menor capacidade de armazenamento e maior custo.

Memória cache

Nos computadores mais antigos os registradores eram diretamente ligados a memória principal, na execução de instruções a CPU acessava diretamente a memória principal pelo menos uma vez para buscá-las e transferi-las para um registrador interno ao processador. Considerando-se que atualmente o ciclo de memória é bem mais demorado que o período de tempo que a CPU gasta para realizar uma operação na ALU, fica evidente que a duração da execução de um ciclo de instrução é bastante afetada pela demora dos ciclos de memória.

Na tentativa de melhorar o desempenho dos computadores, os projetistas de CPUs vêm obtendo constantemente velocidades cada vez maiores na operação dessas unidades, o que não acontece na mesma proporção com a memória principal. Assim atualmente a diferença de velocidade entre a CPU e a memória principal é muito grande.

Na busca pela minimização dessa diferença foi desenvolvida uma técnica que consiste na inclusão de um dispositivo entre a CPU e a memória principal, denominado memória cache, cuja função é acelerar a velocidade de transferência entre esses dois dispositivos, e com isso melhorar o desempenho dos computadores.

Para isso, a memória cache é fabricada com tecnologia semelhante aquela empregada na CPU, e consequentemente apresenta tempos de acesso compatíveis, resultando numa considerável redução da espera da CPU para receber dados e instruções da cache.

Memória principal

A principal característica dos sistemas computacionais baseados na arquitetura “Von Neuman” consiste no fato de se encarar um computador como uma máquina de “programa armazenado”. O fato das instruções,

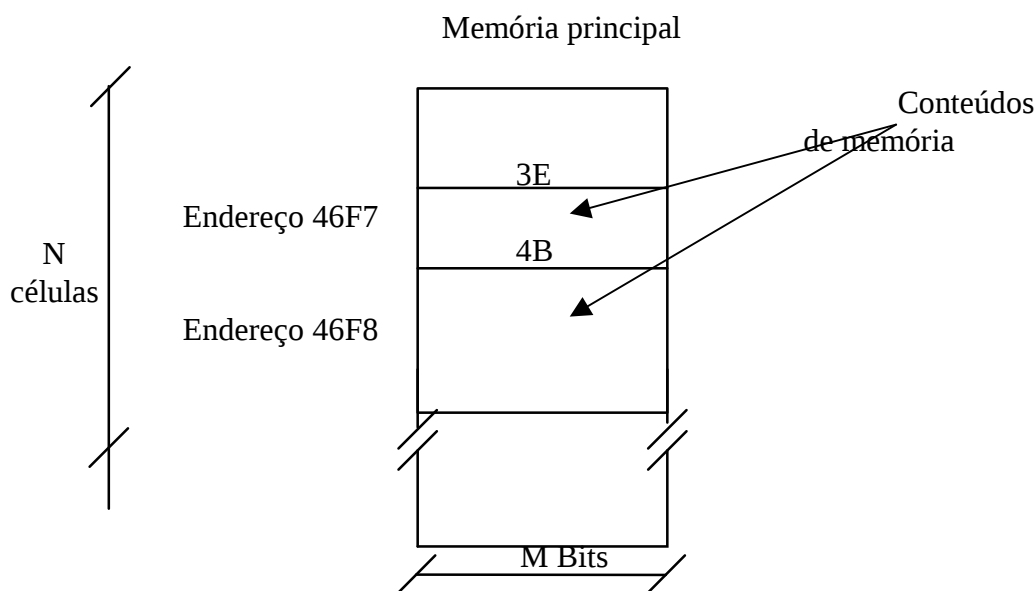
uma após a outra, serem imediatamente acessadas pela CPU é que garante o automatismo do sistema e aumenta a velocidade de execução dos programas. A CPU pode acessar uma instrução imediatamente após a outra porque ela se encontram armazenadas internamente ao computador. Esta é a importância da memória.

A memória especificada para armazenar os programas (e seus dados) a serem executados é a memória que chamamos de memória principal ou real.

A memória principal é a memória básica de um sistema de computação. É o dispositivo onde os programas (e seus dados) que serão executados são armazenados para que a CPU vá “buscando” instrução por instrução.

A memória principal é o “depósito” de trabalho da CPU, isto é, a CPU e a memória principal trabalham íntima e diretamente ligadas na execução dos programas. As instruções e os dados dos programas ficam armazenados na memória principal e a CPU vai buscando-os uma a uma, à medida que a execução vai se desenrolando. Os programas são organizados de modo que os comandos são descritos sequencialmente e o armazenamento das instruções se faz da mesma maneira.

A memória principal de qualquer sistema computacional é organizada como o conjunto de N células sequencialmente dispostas a partir da célula de endereço 0 até a última, de endereço (N-1), conforme mostrado na figura abaixo.



Cada célula armazena um grupo de M bits, que representam a informação propriamente dita.

As memórias de semicondutores são dispositivos voláteis de estado sólido e possuem várias características interessantes que as tornam extremamente vantajosas para constituírem-se na base da memória principal, por exemplo:

- São memórias de acesso aleatório (RAM- Random Access Memory);
- Ocupam relativamente pouco espaço;
- Possuem tempo de acesso pequeno.

Essencialmente, o espaço armazenado da memória principal é um grupo de N células, cada uma podendo armazenar um grupo de M bits. Esta é a memória de trabalho da CPU e, portanto, deve permitir o armazenamento de programas e dados (operação de escrita) e também a leitura destas mesmas instruções e dados. Chama-se essa característica de memória do tipo leitura e escrita. Esse tipo de memória tem uma particularidade desvantajosa, que é o fato de ser volátil.

No entanto, todo sistema precisa, para iniciar seu funcionamento regular, que um grupo de instruções esteja permanentemente armazenado na memória principal de modo que, ao ligarmos o computador, este programa inicie seu funcionamento automaticamente o funcionamento do sistema. Essas instruções vêm junto com o “hardware” e não devem sofrer um apagamento acidental se, inadvertidamente, um programa do usuário

tentar gravar por cima delas. Elas devem estar, portanto, em tipo de RAM que só permita leitura por parte dos programas comuns. A gravação(escrita) nelas deve ser realizada eventualmente e não por processos comuns. Essas memórias chamam-se memória somente de leitura (ROM- Read Only Memory).

2.5.5 - Memória secundária

Em geral é o tipo de memória que tem maior capacidade de armazenamento que os outros tipos anteriormente descritos, menor custo por “byte” armazenado e tempo de acesso superior. Conhecida como memória secundária ou memória auxiliar, ou ainda memória de massa, tem por objetivo garantir um armazenamento mais permanente aos programas e estruturas de dados, razão pela qual deve possuir maior capacidade de armazenamento que a memória principal.

A memória secundária de um computador pode ser constituída por diferentes tipos de dispositivos, alguns diretamente ligados ao sistema por acesso imediato, e outros que podem ser conectados quando desejado.

A maior característica desse tipo de memória é a sua não volatilidade.

2. **Memória Principal (RAM) e Memória Cache** – É a parte do computador onde programas (instruções) e dados são armazenados, sendo composta por unidades de acesso chamadas células, sendo que:

- A célula é a menor unidade endereçável. A maioria dos fabricantes de computador padronizam a célula em 8bits de tamanho (byte). Bytes são agrupados em palavras; um computador com palavra de 16bits tem 2bytes/palavra, enquanto que um computador com palavra de 32bits tem 4bytes/palavra. O significado de uma palavra é que a maioria das instruções operam em palavras inteiras, por exemplo somando duas palavras. Então uma máquina de 16bits terá registradores de 16bits e instruções que manipulam palavras de 16bits;
- Todas as células em uma memória possuem o mesmo número de bits;
- Cada célula da memória possui um **endereço único**; a quantidade de bits do endereço, que não tem relação com a quantidade de bits da célula, está relacionada ao número máximo de células endereçáveis.

Observe quantos números binários diferentes você pode escrever com 2 bits, com 3 bits, com 4 bits, com 5 bits. Conclua que com Nbits você pode escrever 2^N números binários diferentes.

Lembrando que $1K=2^{10}$, $1M=2^{20}$ (mega), $1G = 2^{30}$ (giga), veja quantos bits são necessários para endereçar uma memória de 1Kbyte, sendo cada célula de 1byte. Faça o mesmo para uma memória de 64Kbytes, de 32Mbytes e 4Gbytes.

Depois do processador, que é o componente que executa as instruções dos programas, a memória é o componente mais disputado pelos programas, uma vez que os programas devem estar carregados na memória principal para ser “enxergado” e executado pelo processador. Esta memória é classificada como volátil, ou seja, não tem capacidade de preservar o seu conteúdo sem uma fonte de alimentação. Por esse motivo devemos sempre gravar nossos programas/arquivos quando estes foram alterados e permanecem na memória principal.

O tempo de acesso aos dados contidos na memória principal pode ser agilizado quando coloca-se no computador uma memória auxiliar, chamada memória cache. A memória cache é uma memória volátil de alta velocidade, localizada na placa-mãe do computador e atualmente também dentro do próprio processador. O tempo de acesso a um dado nela contido é muito menor que se estivesse na memória principal. Toda vez que o processador faz referência a uma célula da memória principal, ele verifica antes na memória cache. Se a cópia da célula já estiver na cache (**acerto** ou **hit**), não há necessidade do acesso à memória principal; do contrário (**falha** ou **miss**), o acesso é obrigatório. Neste último caso, o processador, copia para a cache um bloco de células contíguas àquela referenciada. O tempo de transferência entre as memórias é pequeno, se comparado com o aumento do desempenho obtido com a utilização desse tipo de memória. Por que isto acontece ocorre o aumento de desempenho? Vejamos : se a

cada vez que o processador procurar uma cópia de célula na cache, não encontrar (falha) terá que buscar um bloco de instruções na memória principal. Sem o uso da cache a busca na memória principal é feita instrução por instrução. Conclui-se, portanto, que só será vantajoso o uso da cache se o número de acertos for muito maior que o número de falhas. E isso realmente ocorre !!! Ocorre porque na maioria dos casos, observa-se que um programa faz acessos consecutivos a endereços consecutivos da memória e, além disso, observa-se que na maioria dos casos um programa em execução passa grande parte do tempo utilizando o mesmo conjunto de instruções. À esta observação, importantíssima para grandes ganhos de performance, denomina-se **PRINCÍPIO DA LOCALIDADE**.

Cache

Objetivo - Simular a existência de uma memória de grande capacidade e alta velocidade para a CPU.

Operação - Manter em uma memória de pequena capacidade e alta velocidade cópias das posições que mais provavelmente serão acessadas na MP.

Tempo de acesso:

$$T = P_A \times T_C + (1 - P_A) \times T_P$$

Onde:

P_A = Probabilidade de acesso no cache (Hit Rate)

T_C = Tempo de acesso ao cache

T_P = Tempo de acesso à MP.

Ex.:

$P_A = 90\%$

$T_C = 50 \text{ ms}$

$T_P = 300 \text{ ms}$

$$T = 0,9 \times 50 + (1 - 0,9) \times 300 = 75$$

OPERAÇÕES DE LEITURA DA MEMÓRIA CACHE

Obs.: A busca de um bloco inteiro (e não apenas de uma palavra) objetiva minimizar a taxa de falhas nos próximos acessos (princípio da localidade). O critério de seleção do bloco a ser substituído depende da organização interna do cache.

Operações de escrita no cache

As estratégias são:

- Write back - é possível que a memória principal fique inconsistente com o cache.
- write through - A memória principal está sempre consistente.

Write Back

Em caso de acerto no cache (hit) a palavra é alterada apenas no cache.

Em caso de falha (miss), a palavra é alterada na memória e no cache.

A MP só é utilizada quando o bloco que contém a palavra é substituído no cache.

Maior velocidade nas operações de escrita.

A manutenção de caches em sistemas multiprocessados é complicado.

Write Through

Em caso de acerto (hit), a palavra é alterada no cache e na MP.

Em caso de falha (miss), 2 esquemas são possíveis:

- No Write Allocate - a palavra é escrita na MP
- Write Allocate - A palavra é escrita no cache e na MP

As operações de escrita são mais lentas

A MP está sempre consistente e é mais fácil manter a consistência das caches em sistemas multiprocessados.

ORGANIZAÇÃO INTERNA DO CACHE

- Mapeamento totalmente associativo:

Um bloco de palavras da MP pode ser armazenado em qualquer posição do cache.

- Mapeamento Direto

Cada bloco de palavras da MP só pode ser mapeado em uma única posição do cache cujo endereço é dado por: (número de blocos) módulo (tamanho do cache).

- Mapeamento Associativo por Conjunto

A cada posição do cache pode ser associado um conjunto de blocos (tipicamente 2 a 16) de palavras da MP.

CARACTERÍSTICAS DO MAPEAMENTO

- Totalmente Associativo:

- Demorado descobrir se um bloco reside ou não no cache
- A substituição de um bloco no cache só é necessária quando o cache está totalmente cheio
- O endereço do bloco de memória tem que ser armazenado integralmente no cache

- Direto

- Rápido descobrir se um bloco está presente na cache
- 2 blocos frequentemente acessados podem mapear a mesma posição do cache dada pelo índice: (end bloco) mod (tam. do cache)
- Apenas a informação de TAG, dado pelo quociente, na divisão do endereço do bloco, pelo tamanho do cache, deve ser armazenado no cache para identificar o bloco.

- Associativo por conjunto

- Reduz as chances de conflito
- Também é rápido descobrir se um bloco está no cache

TAMANHO DA MEMÓRIA CACHE

A escolha do tamanho do bloco na memória cache é uma decisão de compromisso. Blocos grandes tendem a aumentar a taxa de acerto mas possuem os seguintes inconvenientes:

- Pode diminuir a taxa de acerto global por não permitirem a convivência simultânea de muitos blocos na cache.
- Em caso de leitura com falhas o tempo de acesso em geral aumenta, pois mais informações precisam

ser lidas na MP. Perde-se tempo de transferência.

- Blocos grandes podem acarretar a ocupação desnecessária do cache com informações que nunca serão utilizadas.

Obs.: O tamanho típico de um bloco de memória varia entre 16 e 256 bytes.

POLÍTICAS PARA SUBSTITUIÇÃO DE BLOCOS NA MEMÓRIA CACHE

(Não se aplica ao mapeamento direto pois não se tem escolha onde alocar o dado)

Os esquemas de mapeamento totalmente associativo ou associativo por conjunto exige uma política de substituição de blocos exigem uma política de substituição de blocos na memória cache.

Algumas dessas políticas são:

- Randômica: Um bloco é escolhido aleatoriamente para ser substituído. Política de fácil implementação, mas gera problemas de desempenho em esquemas totalmente associativo. Pode descartar um dado que seria utilizado em seguida.
- FIFO - (First In First Out) O bloco que está a mais tempo no cache é substituído. Menos simples de implementar e pode gerar problemas de desempenho se o bloco mais antigo estiver sendo ainda muito utilizado.
- LRU - (Least Recently Used) Substitui-se o bloco que há mais tempo não é utilizado. É o esquema com o melhor desempenho e sua implementação é a mais complicada. Um bit na cache marca se ela foi ou não utilizada em cada tic do clock (Não é utilizado na prática pois consome memória cache para armazenar tais informações).

ABORDAGENS PARA MELHORAR O DESEMPENHO DA CACHE

- Reduzir a taxa de "miss" aumentando a probabilidade de "hit"
- Reduzir o tempo de acesso em caso de "hit"
- Utilizar o write-buffer (Buffer para escrita) para evitar que o processador aguarde a escrita na MP no modo write through
- Utilizar a cache de instruções e dados separados para possibilitar acessos simultâneos no cache pelo processador

Obs: Blocos são como páginas. Só se pega na MP bloco a bloco, mesmo que o primeiro bloco só contenha metade da informação que você deseja.

Write Buffer é um cache separado só para escrita. Economiza tempo de acesso.

3. **Dispositivos de Entrada/Saída** – São os componentes que permitem a comunicação entre o computador e o mundo externo. Podem ser divididos em duas categorias:

- os que servem apenas de interface homem/máquina - através deles, o processador e a memória principal podem comunicar-se com os usuários (teclado, monitor, impressoras, scanner, caneta ótica, mouse). O desenvolvimento de interfaces cada vez mais amigáveis permite que as pessoas sem conhecimento específico sobre informática possam fazer uso de computadores;
- aqueles destinados ao armazenamento de programas e dados (discos e fitas magnéticas), chamados de memórias secundárias. Seu custo é relativamente baixo, porém o tempo de acesso é maior quando comparado ao tempo de acesso à memória principal ou mesmo à memória cache.

4. **Barramento (ou Bus)** - A memória principal, o processador e os dispositivos de E/S encontram-se interligados fisicamente através de linhas de comunicação denominadas barramentos. Um barramento pode ser visto como um conjunto de fios paralelos, cada um carregando um bit, onde trafegam informações. Cada tipo de barramento carrega um tipo de informação específica, a saber: barramento de dados, barramento de endereços e barramento de controle.

➤ **LEITURA E ESCRITA NA MEMÓRIA PRINCIPAL :**

Vamos acompanhar mais de perto o processo de leitura na memória principal.

Seja, durante uma execução, em determinado instante PC=010978EF. Isto significa que este é o endereço no qual está a próxima instrução a ser executada. Quando termina a execução da instrução corrente, o processador copia para o registrador **MAR (Memory Register Address)** o valor do PC e a seguir incrementa o PC. Depois, a unidade de controle (UC) coloca no barramento de controle o código em bits informando que será feita uma leitura; e coloca no barramento de endereços o conteúdo do registrador MAR. A UC dá o sinal para a leitura. A leitura na memória principal é feita e o conteúdo da célula é colocado no barramento de dados. No processador o conteúdo do barramento de dados é copiado para o registrador **MBR (Memory Buffer Register)**.

Explicite os passos para ESCRITA na MEMÓRIA PRINCIPAL.

Conceitos de software

O *hardware* sozinho não tem utilidade, tornando-se necessária a existência de um conjunto de programas relacionado mais diretamente com os serviços do sistema operacional, possibilitando inclusive a criação de aplicativos pelos usuários desenvolvedores de sistemas (programadores). Utilizaremos o termo *utilitário* para referenciar o software mais ligado ao sistema operacional; e o termo *aplicativo* ou *aplicação* para os programas desenvolvidos pelo usuário. Isto é feito através de linguagens de programação.

Linguagem de Programação:

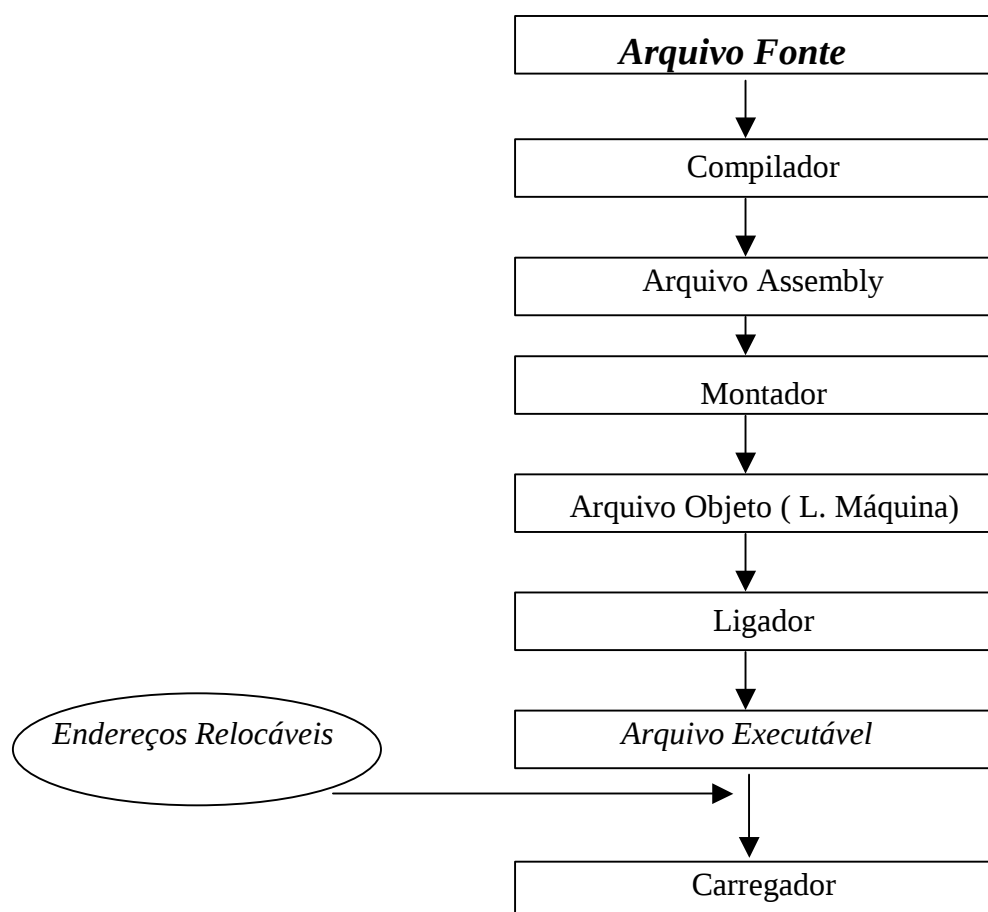
Conjunto limitado de instruções a serem executadas em um determinado computador. A linguagem de programação se divide em 3 níveis:

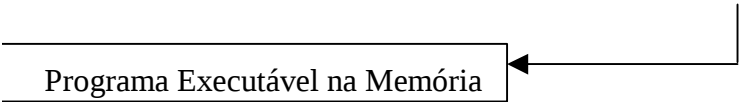
1º- Linguagem de alto nível – a linguagem que o usuário final (na maioria das vezes o programador) entende com certa facilidade. Ex.: VB, C, Delphi, Cobol, Pascal, ...

2º- Linguagem de médio nível ou linguagem de montagem – Conjunto limitado de instruções que são executados por um determinado processador. Ex.: Assembly do processador.

3º- Linguagem de baixo nível ou linguagem de máquina – Nos computadores digitais são seqüências de 0 e 1 (binárias).

Preparação para a execução de um programa:





Programa Executável na Memória

Tradutor , Montador e Compilador

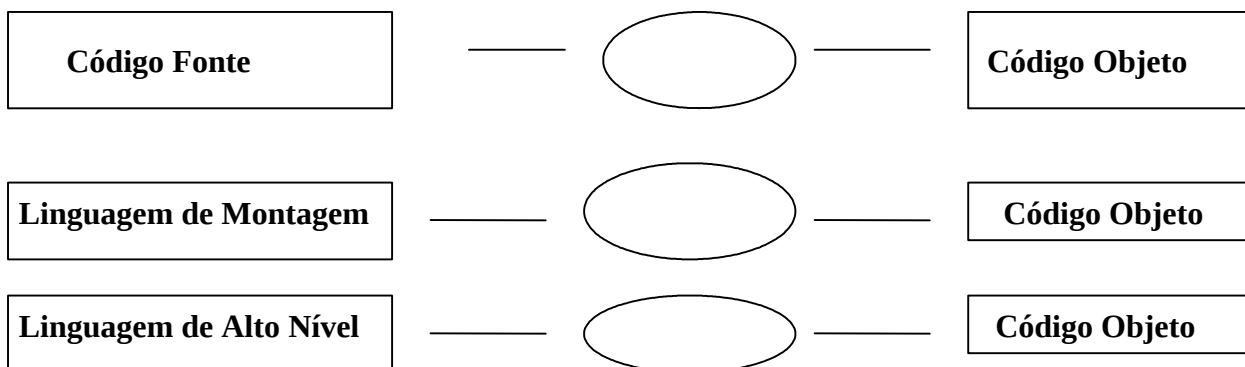
Na época do surgimento dos primeiros computadores digitais (1945 a 1955), não havia a idéia de sistema operacional. O programador tinha que saber o funcionamento do hardware para trabalhar com a máquina. Por exemplo, um programa era escrito em **linguagem de máquina** (utilizando as instruções que o processador “conhece”) e já endereçado às posições de memória que iria ocupar. Ocorreu que com o passar do tempo, mesmo depois do surgimento dos sistemas operacionais, com o objetivo de fazer a interface entre o usuário e a máquina, algumas máquinas de uso específico ainda utilizavam esta programação escrita em binário ou hexadecimal.

A linguagem de programação **Assembly** é uma **linguagem de montagem** porque as suas instruções estão ligadas diretamente às instruções do processador ==> ou seja, o Assembly está, então, ligado diretamente à *arquitetura do processador*. O software que faz a *tradução* das instruções em Assembly (**código-fonte**) para as instruções da máquina é chamado **Assembler**, e é dito **MONTADOR**, por ser o **TRADUTOR** de uma linguagem de montagem para a linguagem de máquina.

Outras linguagens de programação, chamadas de **alto nível**, não tem uma relação tão direta com a linguagem de máquina. Para dar o primeiro passo da transformação das instruções em alto nível (**código-fonte**) para a linguagem de máquina, é preciso usar um tradutor também, que neste caso (linguagens de alto-nível) é chamado de **COMPILADOR** .

O código gerado pelo tradutor é denominado **código objeto**, que, apesar de estar em código de máquina, ainda não é executável (ou seja, ainda não está pronto para execução). Então o tradutor gera um código-objeto a partir de um código-fonte. Se o código-fonte estiver escrito em linguagem de montagem (Assembly) o tradutor é chamado montador; se estiver escrito em linguagem de alto nível (Fortran, C, C⁺⁺, Cobol, Pascal, etc.) o tradutor é chamado de compilador. Os nomes dos compiladores são os mesmos da linguagem correspondente (compilador C, compilador Pascal, etc.)

Complete os quadrinhos abaixo com os nomes corretos, baseados na explicação anterior :



Linker

Também chamado de *linkage editor* (editor de ligação) ou *linkeditor*, é o utilitário responsável por gerar a partir de um ou mais módulos objeto, um único programa executável. Suas funções básicas são

resolver todas as referências simbólicas existentes entre os módulos e reservar memória para a execução do programa.

As referências simbólicas são resolvidas pelo *linker* através de pesquisa em bibliotecas do sistema ou do próprio usuário, que são arquivos que contêm diversos módulos objeto e/ou definições de símbolos.

Já a operação de reserva de memória para execução do programa é conhecida como *relocação*. Em sistemas operacionais mais antigos, a relocação era realizada uma única vez, quando todos os endereços simbólicos eram traduzidos para endereços físicos fixos e o programa executável gerado somente podia ser carregado em uma determinada região da memória (código absoluto). Porém, em sistemas multiprogramáveis esse tipo de relocação era inviável. A solução é permitir que o programa seja carregado em regiões diferentes toda vez que for trazido para a memória principal (código relocável), sendo este tipo de relocação realizado por outro utilitário denominado *loader*.

Loader

Também chamado de *carregador*, é o utilitário que carrega/leva o programa para a memória principal para que este seja então executado. Como pudemos constatar anteriormente, o procedimento de carga varia com o código gerado pelo *linker*, que em função disso é classificado como sendo do tipo *absoluto* ou *relocável*.

Interpretador

Imagine um tradutor que não gere código objeto e que a partir de um programa em código fonte, escrito em linguagem de alto nível, traduza cada instrução e a execute em seguida. Esse software é denominado interpretador e sua maior desvantagem, como podemos perceber, é o tempo gasto na tradução das instruções de um programa toda vez que este for executado, já que não existe a geração de código executável. Algumas linguagens tipicamente interpretadas: Basic, APL e dBase.

Depurador

Todos sabemos que os programadores podem cometer erros de lógica no desenvolvimento de programas. O depurador (*debugger*) permite ao usuário controlar toda a execução de um programa a fim de depurá-lo/torná-lo puro/detectar erros, através de alguns recursos:

- Acompanhamento da execução instrução por instrução;
- Monitoramento e alteração do conteúdo das variáveis (*watchpoint*);
- Implementação de pontos de parada no decorrer da execução do programa (*breakpoint*);

Linguagem de controle ou de comando

É a forma mais direta de um usuário se comunicar com o sistema operacional, possibilitando ao usuário acesso a rotinas específicas do sistema. Os comandos quando digitados pelo usuário são interpretados por um programa denominado interpretador de comandos ou *shell*, que verifica a sintaxe do comando, envia mensagens de erro e faz chamadas na rotinas do sistema.

A evolução de tais linguagens originam as interfaces gráficas e no futuro, com a introdução das linguagens naturais, os sistemas passarão a reconhecer também comandos falados.

Linguagem de máquina

É a linguagem de programação que o processador realmente consegue entender. Cada processador possui um conjunto único de instruções de máquina definido pelo fabricante, que especifica detalhes, como registradores, modos de endereçamento e tipos de dados. O programa nessa linguagem é totalmente codificado em formato binário.

Em Resumo:

Existe um conjunto de programas que permitem ao usuário gerar outros programas, que podem ser:

- Interpretadores: (Arquivo Fonte – Arquivo Objeto)

- Compiladores: (Arq. Fonte – Programa Executável)

Ambos utilizam linguagens próprias para interfacear o programador das instruções de máquina.

Uma grande diferença entre compiladores e interpretadores é que o último, para ter seu programa executando, é necessário que tenhamos um outro programa(interpretador) rodando para executá-lo passo a passo.

Já programas compilados não necessitam de mais nada para executar.

➤ Máquina de Níveis

Nos primeiros computadores, a programação era realizada em painéis, através de fios, exigindo grande conhecimento do *hardware*. Depois surgiu o sistema operacional, tornando a interação entre usuário e computador mais simples e confiável. O programador afasta-se cada vez mais da complexidade do *hardware*, deixando-a por conta do sistema operacional. Podemos então, considerar o computador como um conjunto hierárquico de níveis ou camadas, onde inicialmente existem dois níveis: o nível 0 (*hardware*) e o nível 1 (sistema operacional).

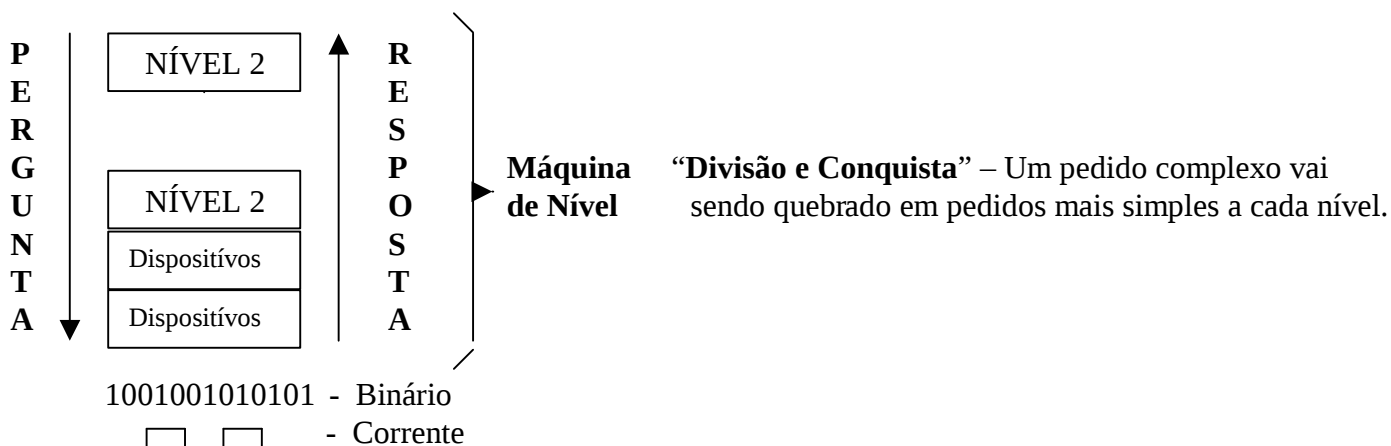
Um computador não possui apenas dois níveis, e sim tantos níveis quanto forem necessários para adequar o usuário às suas diversas aplicações. Com a evolução das arquiteturas de computadores e das linguagens de programação, atualmente a maioria dos computadores possui a estrutura mostrada na figura :

Aplicativos
Utilitários
Sistema Operacional
Linguagem de Máquina
Microprogramação
Dispositivos Físicos

Identifique na figura acima os níveis que se referem a software e os que se referem ao hardware.

De forma generica o modelo de Máquina de Níveis é utilizado no sistema de computação com a seguinte visão:

- A camada, ou nível, N faz uma chamada à camada de nível N-1 que, posteriormente, retornará um ou mais resultados à camada de nível N.



Histórico dos sistemas operacionais

Faremos um breve resumo da história dos sistemas operacionais com objetivo ilustrativo, apenas para termos uma idéia da ordem cronológica de fatos importantes. A evolução do software dos computadores, especialmente os sistemas operacionais, está relacionada ao desenvolvimento de equipamentos cada vez mais velozes, menores no tamanho e com custo cada vez menor, e à necessidade de aproveitamento e controle desses recursos.

Primeira fase (1945-1955)

- Computadores baseados em válvulas (ENIAC, criado para fins militares de cálculos balísticos, com 18 mil válvulas, 30 toneladas, consumo de cerca de 140.000 watts; EDVAC, utilizado por universidades e também órgãos militares; UNIVAC I, criado para auxiliar no censo americano de 1950, aplicação comercial);
- Ausência de sistema operacional: programação feita por painéis, através de fios, sem uso de linguagens de programação.

Segunda fase (1956-1965)

- Criação do transistor (maior velocidade e confiabilidade no processamento, menor dissipação de energia) e das memórias magnéticas (acesso mais rápido aos dados, maior capacidade de armazenamento e diminuição do tamanho dos computadores);
- Surgimento das primeiras linguagens de programação (Assembly e Fortran) – os programas deixam de ser feitos diretamente no *hardware*;
- Seqüenciamento da execução dos programas, sem intervenção do operador, conhecido como processamento *batch* (em lote);
- Importantes avanços com a linha de computadores 7094 da IBM

Terceira fase (1966-1980)

- Diminuição do tamanho e dos custos de aquisição do *hardware* com a criação dos circuitos integrados (CIs) e, posteriormente, dos microprocessadores – lançamento da série 360 de computadores da IBM e da linha PDP-8 da DEC;
- Evolução dos processadores de E/S, possibilitando a utilização da técnica de compartilhamento da memória e do processador denominada multiprogramação;
- Substituição das fitas por discos magnéticos, possibilitando a alteração na ordem de submissão dos programas em lote (*spooling*);
- Surgimento em 1969 do sistema operacional UNIX.

Quarta fase (1981-1990)

- Miniaturização e barateamento dos computadores através da integração cada vez maior dos componentes;
- Surgimento dos microcomputadores pessoais (PCs) e do sistema operacional DOS (*Disk Operating System*);
- Sistemas multiusuário e multitarefa, permitindo a execução de diversas tarefas de forma concorrente;
- Equipamentos com múltiplos processadores, processadores vetoriais e diversas técnicas de paralelismo em diferentes níveis (multiprocessamento);
- As redes de computadores se difundiram por todo mundo – software de redes intimamente relacionados ao sistema operacional e surgimento dos sistemas operacionais de rede.

Quinta fase (1991-)

- Grandes avanços de *hardware* (microeletônica), *software* e telecomunicações – processadores e memórias cada vez menores e mais baratos;
- Processamento distribuído em sistemas operacionais;
- Novas *interfaces* homem/máquina – linguagens naturais, sons e imagens;
- Sistemas multimídia, bancos de dados distribuídos e inteligência artificial.

Classificações de Sistemas Operacionais

Antes de apresentar a classificação dos sistemas operacionais, vamos introduzir **4 conceitos relativos à medida de eficiência de um SO**.

Utilização da CPU : esta medida é feita em porcentagem. Por exemplo : durante 80 ms o processador trabalhou num total de 60 ms. Qual foi a porcentagem de utilização do processador neste caso ? Na construção do projeto do SO o objetivo deve ser no sentido de maximizar ou minimizar a utilização do processador ? Por que ?

Tempo de turnaround : tempo total do programa ou tarefa desde a sua admissão/criação até o seu término. Qual deve ser o objetivo do SO em relação ao tempo de turnaround ?

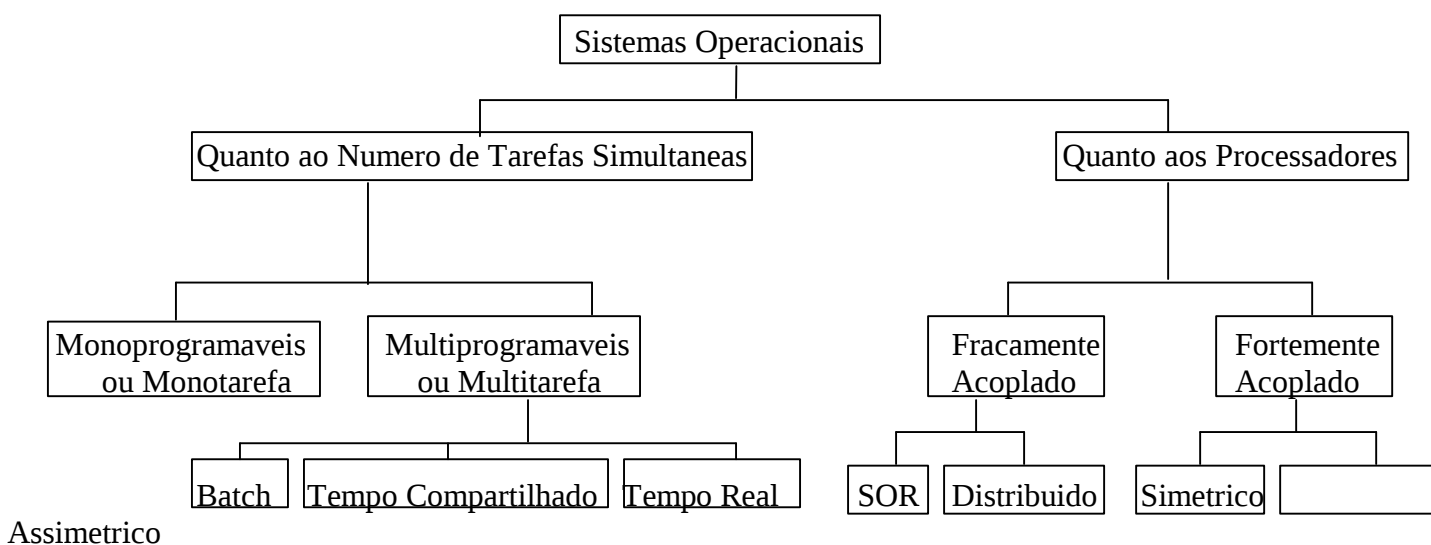
Tempo de resposta : Tempo decorrido entre o envio de um pedido ao SO e o início da resposta. Cite um exemplo de pedido ao SO. Normalmente, qual a ordem de grandeza do tempo de resposta para este pedido (s, ms, ns ?)

Throughput : É o número de tarefas e/ou programas executados em algum intervalo de tempo. No laboratório faremos a medida de throughput.

Um Sistema Operacional pode ser classificado quanto a quantidade de:

- 1- **Usuários**: Quantos usuários estão utilizando o sistema em um determinado momento. Pode ser Mono-usuário (DOS, Windows) ou Multi-usuário (VMS, Unix)
- 2- **Programas Utilizados**: Quantos programas estão utilizando o sistema em um determinado momento. Pode ser Mono-programado – executa cada programa do início ao fim sem sofrer perda de controle da CPU; ou Multi-programado – multi-task – executa vários programas, sofrendo interrupções.
- 3- **Processadores**: Número de processadores que o sistema consegue trabalhar. Pode ser Mono-processador ou Multi-processador.

O diagrama abaixo ilustra os vários tipos de sistemas operacionais de acordo com a configuração do hardware.



2.1 Sistemas Monoprogramáveis ou Monotarefas

Neste tipo de sistema operacional, todos os recursos do sistema de computação estão dedicados exclusivamente a um programa ou tarefa, de cada vez. I

Sejam três programas PROG1, PROG2 e PROG3 a serem executados. Ocorre o seguinte :

PROG1 é admitido na máquina. O SO reserva espaço na memória para o código e dados referentes ao programa. O SO aloca o programa na memória principal. O SO dá o comando para o processador iniciar a execução. A execução prossegue. Se determinada instrução fizer um pedido de leitura em disco, o SO dará o comando para a leitura. O acesso a disco é feito. Como um acesso a disco é muito lento em relação à velocidade de cálculo no processador (veja cálculos nos exercícios) , o processador ficará ocioso, durante o acesso ao disco. A execução prossegue. Se outros pedidos de E/S forem feitos, novamente o processador ficará ocioso durante o acesso aos dispositivos E/S. Mesmo que sobre espaço na memória principal, o mesmo não será preenchido, permanecendo nesta somente as rotinas do SO, o código de PROG1 e os dados de PROG até o final da execução do mesmo. Somente ao final de todo o PROG1, o PROG2 poderá ser admitido. Somente após o término de PROG2, o PROG3 poderá ser admitido.

Qual a desvantagem imediata que você percebe num sistema operacional monotarefa ?

Para que tipo de ambiente, este SO pode ser utilizado sem que esta desvantagem seja importante ?

Se a velocidade de acesso aos dispositivos de entrada e saída fosse da ordem da velocidade de execução esta desvantagem continuaria ocorrendo? Por que ?

Esses sistemas podem ser **MULTIUSUÁRIO** ?

Você acha que o projeto e código deste tipo de SO é complexo ? Por que ?

2.2 Sistemas Operacionais Multiprogramáveis ou Multitarefa

Ao contrário dos monotarefa este tipo de SO compartilha os recursos do sistema de computação (quais são mesmo estes recursos ?) entre vários programas e/ou tarefas.

O compartilhamento da **memória principal** é feito através da utilização da memória **simultaneamente** por vários programas e/ou tarefas, ou seja, em determinado instante temos o código e dados de vários programas e /ou tarefas alocados na memória principal.

Isso ocorre no monotarefa ? Qual é o tipo de segurança que o SO deve oferecer no que se refere a este item ?

O compartilhamento do processador, que é único, é feito de forma que vários programas e/ou tarefas podem ter a sua execução iniciada e mediante uma série de razões, para dar a vez a outro programa e/ou tarefa, mesmo antes de terem terminado. Ou seja, o compartilhamento **do processador** é feito de forma que temos em determinado instante **um e somente um programa ou tarefa em execução**. Ou seja o compartilhamento do processador é feito de forma **concorrente e não simultânea**. Por que ?

O compartilhamento dos dispositivos E/S é feito de forma que dois dispositivos diferentes podem estar recebendo/enviando dados relativos a tarefas e/ou programas diferentes, **simultaneamente**.

E além disso, podemos ter um acesso a disco relativo a um programa e/ou tarefa ocorrendo **simultaneamente** a uma execução, por exemplo.

Cite outros acessos simultâneos num multitarefa.

Quantas instruções podem ser executadas durante um acesso a disco ? Trabalhe com ordens de grandeza, ou seja, potências de 10.

Faça uma comparação monotarefa x multitarefa, no que se refere à complexidade do projeto e código do SO.

Faça uma comparação monotarefa x multitarefa no que se refere à segurança que o SO deve oferecer.

Faça uma comparação monotarefa x multitarefa no que se refere à eficiência que o SO deve apresentar no compartilhamento dos recursos entre os vários programas e/ou tarefas.

SISTEMAS OPERACIONAIS MULTITAREFA

Apresentamos abaixo os três tipos de multitarefa. Vamos observar ao longo do nosso estudo que primeiramente seremos apresentados a uma classificação, para depois concluir que os sistemas atuais, na verdade, implementam um *híbrido* de um ou mais dos tipos estudados.

BATCH : Inicialmente os computadores de grande porte, praticamente só processavam. Os dados a serem processados eram lidos numa máquina e gravados em uma fita magnética. Esta fita era transportada por um operador até o computador que fazia o processamento. Após o processamento os resultados eram escritos numa fita magnética e levados até uma outra máquina que procedia a impressão. O primeiro tipo de multitarefa fazia com que vários programas compartilhassem o recurso *fita magnética* entre vários programas (chamados na época *jobs*). Esta foi a primeira situação em que surgiu o termo *batch*. Vários programas e seus dados eram lidos e passados para a fita. O operador levava a fita até o computador, que processava todos os programas, um por um e ia escrevendo os resultados na fita. Depois, o operador levava a fita até o canal de saída (impressão em papel, perfuração em fita de papel). Numa segunda fase, já com os dispositivos de E/S ligados diretamente ao computador, o processamento *batch* era feito com vários programas sendo lidos diretamente pelo computador. Mas ainda executados um por um. Atualmente chamamos de *batch* as tarefas que são deixadas para execução de baixa prioridade, sem a interferência do usuário.

TEMPO COMPARTILHADO OU TIMESHARING : Os sistemas de tempo compartilhado tem como principal característica a **interação** com o usuário. Nestes sistemas todos os recursos da máquina são compartilhados por várias tarefas/programas, dando a impressão que os processamentos são simultâneos. Para atingir este objetivo, há a implementação de uma *fatia de tempo*, para o uso do processador. É este tipo de sistema operacional que é implementado nos computadores de grande porte, no qual há vários terminais ligados a um processador. Os usuários que estão on-line nos terminais trabalham como se o computador estivesse dedicado somente a cada um deles. Então esta sistema é *multiusuário*. Além disso, cada usuário pode ter ativas várias tarefas. O sistema operacional trabalha dando a impressão de que todos os recursos da máquina estão disponíveis para cada uma das tarefas.

Você conhece algum sistema multitarefa ? Qual ?

Para este tipo de sistema operacional o *tempo de resposta* é extremamente importante, por causa da interação com o usuário : o usuário não pode fazer um pedido ao SO (por exemplo abrir um aplicativo) e esperar um tempo absurdo pela resposta, porque há outras tarefas/ programas/usuários ativos. Se isto ocorrer não será cumprido o objetivo de fazer parecer a um usuário/programa/tarefa que os recursos da máquina estão dedicados a estes.

TEMPO REAL : Para esse tipo de sistema operacional, o mais importante é a prioridade e não o tempo de resposta. É utilizado em situações como navegação aérea ou marítima, controle de caldeiras de petróleo, nas quais os pedidos estão sendo processados e as respostas devem ocorrer em tempos pré-determinados. Se um programa de *prioridade maior* do que o que está sendo executado for admitido, o primeiro deverá dar lugar ao segundo. (Exemplo, no caso de navegação marítima, radar detecta um pedra no rumo atual ; deve entrar o programa de mudança de rumo)

SISTEMAS OPERACIONAIS PARA AMBIENTES COM MÚLTIPLOS PROCESSADORES

Conceitos importantes relativos a ambientes com múltiplos processadores :

Vamos definir os seguintes conceitos agora e empregá-los na explicação sobre sistemas com múltiplos processadores.

escalabilidade : refere-se ao aumento da capacidade de computação. Ou seja, uma tarefa é escalável, se puder ser dividida em várias outras tarefas que podem ser executadas independentemente. Suponha que precisamos fazer um vasto processamento sobre todas as notas da vida acadêmica dos alunos desta turma.

As notas de um aluno irão influenciar nos cálculos de outro aluno ?

Então podemos dividir a tarefa em várias outras independentes ? Então este procedimento é escalável ?

Que tipo de máquina apresenta maior , uma com 5 processadores ou uma com 30 processadores ?

reconfiguração : capacidade do sistema poder continuar o processamento mesmo se um dos processadores falhar.

Como o sistema operacional pode tornar isso possível ?

balanceamento : (de carga) : refere-se à divisão de tarefas entre os vários processadores. Se um processador, num ambiente com 4 processadores, apresenta uma *utilização de processador* de 90%, enquanto os outros no mesmo intervalo de tempo, apresentam *utilização de processador* de 5%, a carga de processamento não está bem balanceada, ou seja, o sistema com múltiplos processadores está sendo usado como um monoprocessador, praticamente.

a) Ambientes com múltiplos processadores FORTEMENTE ACOPLADOS

Os sistemas com múltiplos processadores fortemente acoplados são aqueles em que há que vários processadores dentro de uma mesma máquina, **compartilhando memória e dispositivos de E/S**. Este ambiente é gerenciado por um só sistema operacional.

Como são vários processadores executando **simultaneamente**, podemos ter várias tarefas/programas em execução simultaneamente.

Também é possível dividirmos um programa em partes que podem ser executadas independentemente umas das outras, utilizando uma *linguagem de programação paralela*. Nesse caso teremos partes diferentes de um mesmo programa sendo executadas **simultaneamente** por vários processadores. Isto é chamado **paralelismo explícito**. O contrário seria o **paralelismo implícito**, situação na qual o sistema operacional e compiladores verificariam as partes independentes do programa, direcionando-as para diferentes processadores. Isto tem sido motivo de estudo na Ciência da Computação.

Normalmente, estes sistemas com vários processadores, são utilizados em processamentos que utilizam muito o processador e fazem poucas requisições de E/S. A propósito chamamos de **CPU-bound** as aplicações que tem esta característica. Contrariamente, chamamos de **I/O-bound** as aplicações que fazem pouco processamento e muitas requisições de E/S.

SISTEMA OPERACIONAL ASSIMÉTRICO : Na organização **assimétrica** ou **mestre/escravo (master/slave)**, somente UM PROCESSADOR (mestre) EXECUTA AS ROTINAS DO SISTEMA OPERACIONAL.

O que acontece se , com esta configuração, um processador escravo precisar fazer uma leitura do teclado ?

E se forem vários processadores, cada um executando uma aplicação I/O bound ?

Então, quais são as vantagens da configuração assimétrica ?

E quais são as desvantagens ?

SISTEMA OPERACIONAL SIMÉTRICO : Nesta configuração TODOS OS PROCESSADORES EXECUTAM as rotinas do SISTEMA OPERACIONAL.

Como fica a situação do boot (inicialização) do sistema ?

O que acontece se dois processadores fizerem acesso à mesma área de memória simultaneamente ?

Quais seriam as vantagens e desvantagens deste tipo de sistema operacional em relação ao assimétrico ?

Outras questões :

Faça uma análise desses dois tipos de sistemas operacionais em relação aos conceitos de escalabilidade, reconfiguração e balanceamento de carga.

b) Ambientes com múltiplos processadores FRACAMENTE ACOPLADOS

Possuem dois ou mais **sistemas de computação** interligados por linha de comunicação. Observe que estamos tratando de dois ou mais **sistemas de computação**, que inclui todo o hardware e software básico. Cada sistema de computação, nesta configuração, terá, portanto, o seu processador central, a sua memória principal, os seus dispositivos de E/S e o seu sistema operacional, que irá gerenciar os seus próprios recursos. Observe, ainda, que cada sistema de computação tem **capacidade de processamento** própria.

Em uma rede, cada sistema de computação independente pode ser chamado de **nó, host** ou **estação**.

SISTEMA OPERACIONAL DE REDE : Cada nó é totalmente independente dos outros, podendo possuir sistemas operacionais diferentes. Cada nó compartilha os recursos com o restante da rede. Caso uma das estações sofra algum problema, os demais continuarão em funcionamento. São permitidos :

cópia remota de arquivos, emulação de terminal, impressão remota, gerência remota e correio eletrônico.

O usuário *on line* na máquina número 1 da rede, por exemplo, poderá buscar um arquivo do disco da máquina número 2. Para isso ele indicará este *direcionamento*.

SISTEMA OPERACIONAL DISTRIBUÍDO : Nesta configuração, normalmente, os sistemas operacionais são os mesmos e há uma interligação maior entre os hosts do que no caso acima. Uma aplicação está em execução na máquina número 1 da rede, por exemplo, e precisa de um arquivo que está no disco da máquina 2. Este arquivo será encontrado pelo sistema operacional e essa busca será transparente ao usuário. Outro exemplo : se um usuário *on line* na máquina número 1 enviar uma aplicação para execução e o seu processador já estiver ocupado, o SO irá verificar se há um outro processador livre em algum host da rede. Se houver o SO enviará a aplicação para este host. Mas, isso será transparente ao usuário. Uma vez escalonada para executar no processador de um determinado host, a aplicação irá até o fim neste host.

Faça a comparação do balanceamento de carga e reconfiguração entre os dois tipos de SO acima.

Qual a relação existente entre sistema fracamente acoplado distribuído e sistema fortemente acoplado ?

Tratamento de E/S em Sistemas Multiprogramáveis

Como vimos, os sistemas multiprogramáveis ou multitarefa são capazes de compartilhar os recursos de um sistema de computação, entre várias aplicações, apesar desses sistemas possuírem apenas um processador central.

Neste capítulo explicitaremos alguns detalhes das operações de E/S, que proporcionaram, cada vez mais ao longo da evolução de hardware e software, melhor performance multitarefa.

Nos sistemas mais primitivos, a comunicação entre a UCP e os periféricos era controlada por um conjunto de instruções especiais, chamadas **instruções de entrada/saída**. Estas instruções eram executadas pela UCP e continham detalhes específicos de cada periférico, como p.ex. trilhas e setores de um disco.

A implementação de um dispositivo de hardware chamado **controlador** ou **interface** fez com que a comunicação entre UCP e periféricos fosse feita através de tal dispositivo. Com esta nova implementação, surgiram **duas maneiras** de controle dos periféricos, pela UCP :

1 - E/S controlada por programa : nesta situação a UCP sincroniza com o controlador, é iniciada a transferência de uma palavra, a UCP testa o controlador para saber se terminou, senão inicia a transferência da próxima palavra. Neste caso, a UCP fica **presa** à transferência de dados, durante todo o tempo. Como a UCP executa uma instrução muito mais rápido do que a realização de uma operação de E/S, isto significa que a UCP ficava **presa** e **ociosa**, aguardando o término da transferência. Chamamos esta "espera presa" de **busy wait** ou seja **espera ocupada**.

2 - Polling : nesta situação a UCP faz o teste para saber se a transferência terminou também, mas o faz de tempos em tempos, ficando **liberada**, neste intervalos para realização de outras tarefas. Estas outras tarefas podem ser aplicações de usuário, mas antes destas, a UCP percorre **todas as portas** de E/S para verificar de já terminou alguma transferência, ou se há alguma coisa para "entrar".

Não seria melhor que a UCP fosse
AVISADA do término de uma
transferência ou de que há algo
para ser lido ?

SIM. A UCP PASSOU A SOFRER UMA INTERRUPÇÃO .

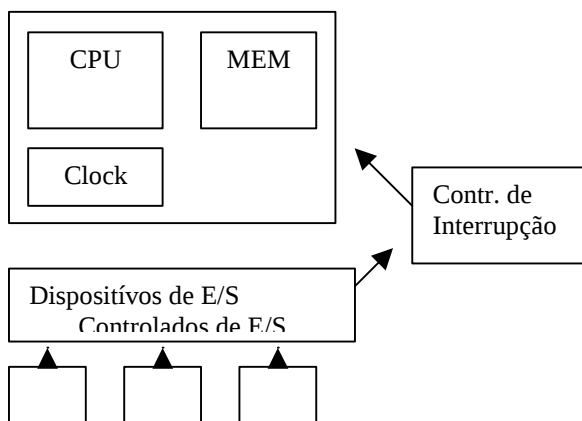
E/S controladas por Interrupção:

Ao invés do S.O. testar periodicamente o estado dos dispositivos, o próprio dispositivo, através do seu controlador, interrompe a CPU quando alterar o seu estado.

A CPU executa a operação de E/S, como já sabe que a resposta do dispositivo vai demorar, o fluxo da instrução é desviado para outro processo.

Existe dois tipos de interrupção:

- **Exceção:** (Interrupção interna ou de programa) são geradas em decorrência a falhas de execução (eventos síncronos) que podem ser previsíveis. Ex.: divisão por zero, overflow, underflow.
- **Interrupção externa:** São causadas por sinais externos a CPU (evento assíncronos). Ocorrem independentemente da execução do programa. Podendo ser:
 - **Mascaráveis:** Podem ter sua aceitação desabilitada pela CPU;
 - **Não-Mascaráveis:** Devem ser sempre atendidas e/ou tratadas.



O controlador de Interrupção sinaliza ao S.O. (CPU) qual dispositivo está pronto. Cada periférico (disp. E/S) tem uma interrupção. Daí o S.O. irá decidir, dentro de sua política, qual interrupção será tratada primeiro.

A interrupção constitui-se no seguinte :

- a UCP está executando uma instrução de um programa
- a UCP sofre uma interrupção
- a UCP salva todos os seus registradores
- a UCP identifica a origem da interrupção
- a UCP obtém o endereço da **rotina de tratamento desta interrupção**, consultando o **vetor de interrupções**
- a UCP executa as instruções da rotina adequada
- ao terminar este procedimento a UCP passará a executar algum conjunto de instruções, conforme indicação do SO. Se for continuar algum procedimento anteriormente interrompido, a UCP irá restaurar os registradores que foram guardados no momento da interrupção.

Para que serve o vetor de interrupções ?

O vetor de interrupções contém, para cada tipo de interrupção o endereço de memória da rotina de tratamento daquela interrupção :

Vetor de Interrupções (armazenado na memória principal) :

Tipo de interrupção	Endereço da Rotina de Interrupção
relógio	0015002B
teclado	0015300E
controlador de disco fixo	00161117
controlador de disco flexível	0016AE00
controlador de impressora	0016B389

Parte da memória principal :

Endereço	Conteúdo
0015002B	Rotina de Tratamento da Interrupção do Relógio
0015002C	
	. . .
0015300D	Fim da Rotina de Tratamento da Interrupção de Relógio
0015300E	Rotina de Tratamento da Interrupção de Teclado
0015300F	
	. . .
0016116	Fim da Rotina de Tratamento da Interrupção de Teclado
0016117	Rotina de Tratamento da Interrupção de Controlador de disco fixo
0016118	
	. . .

Acesso Direto à Memória (DMA):

O DMA aloca as informações diretamente na memória então o controlador irá avisar a CPU que o dispositivo está pronto.

O DMA é usado em transferência de grupos de dados nos quais a CPU pode deixar de ser sobrecarregada. Neste esquema, um volume de dados é transferido de/para memória de/para o dispositivo de E/S, sem intervenção direta da CPU. Quem realiza a transferência é o circuito de DMA. O controlador de DMA precisa saber o tamanho em bytes da transferência, a região inicial da memória, o dispositivo alvo e o sentido da transferência para realizá-la.

Técnica de Buffering:

Consiste na utilização de uma área de memória (volátil) para transferência entre os periféricos e a CPU. Com isto reduzimos o "GAP" de performance entre eles, porque a CPU poderá manipular os dados neste "Buffer", evitando consumir o tempo da operação de E/S.

Ex.: Editores que não realizam as operações de E/S diretamente, no caso de alterações em um arquivo, utilizam um buffer para tal.

No laboratório, faça uma lista dos números das interrupções e seus dispositivos associados.

Em que ponto da evolução acima descrita, foi possível a multitarefa ? Por que ?

Em que situação a multitarefa ficou mais eficiente ? Por que ?

Qual o objetivo de salvar todos os registradores da UCP, após uma interrupção ? E se isso fosse feito depois da identificação da origem da interrupção ? Isso traria alguma vantagem ou desvantagem ?

O que a UCP busca na consulta do vetor de interrupções ? Para que ?

O que você acha que acontece quando :

- dois dispositivos tentam interromper o processador ao mesmo tempo ?

[Índice](#)

Processos e Estrutura de Sistemas Operacionais

Parte I – Processos

➤ Conceito de Processo

Num ambiente monotarefa vimos que a CPU executa um programa seqüencialmente, utilizando os recursos de *hardware* e *software* na ordem em que são solicitados e aguardando o término da execução de um para começar o outro. Quando estudamos os sistemas *batch*, vimos que a CPU executava um *job* também seqüencialmente, solicitando os recursos na ordem em que eram pedidos. Já no ambiente multitarefa, entendemos que o que está executando no processador muda constantemente. Isso ocorre para dar a impressão a cada usuário (no caso multiusuário) ou a cada **tarefa** do mesmo usuário (monousuário) que a máquina está a disposição de somente aquele usuário/tarefa. Nós vimos que esses “pedaços” de programas ou tarefas que ora estavam sendo executados pela CPU, ora estavam fazendo um acesso de E/S, ora estavam aguardando para execução na CPU, etc. na verdade ficavam “disputando” os recursos (CPU, E/S, memória) entre si. Isso nos leva ao conceito de um processo → esses “pedaços” de programas ou tarefas que estão fazendo acesso concorrente aos recursos do sistema.

RESPONDA: 3.1) O que ocorre quando um programa deixa a CPU, de forma que ele possa retornar no mesmo ponto e nas mesmas condições que saiu ?

O estado atual do programa na CPU tem que ser salvo para que o programa possa retornar e continuar sua execução como se não tivesse sido interrompido. Isso inclui não só os registradores (como já estudamos no mecanismo de interrupção) como também outros detalhes como, por exemplo, quais eram os arquivos que estavam abertos. Podemos dizer que este **estado completo** do que estava em execução define um **ambiente de execução** → é o PROCESSO. Ou seja, processo é o ambiente onde se executa um programa, a estrutura responsável pela manutenção de todas as informações necessárias a execução de um programa.

➤ Materialização do Processo

Para poder definir o ambiente completo de execução e assim materializar o conceito de processo, o sistema operacional mantém uma estrutura de dados que contém os todos os detalhes necessários para esta definição. Esta estrutura chama-se PCB (*Process Control Block* = **BLOCO DE CONTROLE DO PROCESSO**). Alguns sistemas dão o nome de BLOCO DE CONTROLE DA TAREFA (*task control block*).

O conteúdo exato do PCB varia com o sistema operacional, mas basicamente contém dados como: ponteiro (uma forma de indicar algum outro processo), endereço de memória, nome do processo, tamanho, usuário que criou o processo, registradores (PC, SP, etc.),

Ponteiros
Estado do processo
Nome
Prioridade
Registradores
Limites de memória
Arquivos abertos

Figura 3.0 - Bloco de Controle de Processo (PCB)

grupo de usuários que podem acessar o processo, prioridade, pilha, classe de escalonamento, lista de arquivos abertos, quando o processo foi iniciado, estado do processo, tempo acumulado de execução, etc.

É através de várias **chamadas de sistema** (como será visto a seguir) que o sistema operacional gerencia os processos. Todos esses elementos que definem o ambiente de execução, o processo, podem ser classificados em três grandes grupos: contexto de hardware, contexto de software e espaço de endereçamento.

RESPONDA: 3.2) Você acha que a freqüente troca de contexto (troca de processos) na CPU introduz algum overhead ?

O tempo gasto com a troca de contexto vai variar com as especificações do *hardware*. Basicamente: tamanho da memória, número de registradores que devem ser copiados, existência ou não de instruções especiais (exemplo: uma instrução simples para carregar ou salvar registradores; os processadores RISC possuem essas instruções *load* e *store*), tamanho do disco (para *swapping*), velocidade da CPU.

RESPONDA: 3.3) Sobre troca (mudança) de contexto, reponda:

a) Em que consiste? b) Que “programa” está em execução durante a troca de contexto?

A troca de contexto pode produzir um **gargalo** (*bottleneck*) fazendo decair em muito a performance. Atualmente há uma solução para esse problema: a utilização de *threads*. O uso de *threads* passa pela programação, o que não é simples, e pelo suporte por parte do próprio sistema operacional, que deve suportar sua criação e controle. A idéia principal é diminuir o tempo gasto na criação/eliminação de um PCB para cada subprocesso criado. Sendo assim, subprocessos se diferenciam de *threads* pelo espaço de endereçamento independente que possuem. *Threads* compartilham o mesmo espaço de endereçamento de um processo, passando pelos mesmos estados de um processo.

Contexto de Hardware

É o contexto que é copiado para o *hardware* (CPU) para que a execução possa ocorrer. Basicamente os registradores : PC (*program counter*, que contém o endereço da próxima instrução a ser executada) , SP (*stack pointer*, o ponteiro de pilha, que é o registrador que contém o endereço de memória do topo da pilha), a pilha toda (para que a CPU, ao completar uma função ou subrotina, saiba para que endereço irá retornar), registrador de estado (PSW).

[Índice](#)

Contexto de Software

São as características que vão influir na execução. O contexto de *software* define basicamente 3 grupos de informações : identificação, cotas e privilégios.

- **Identificação** – Cada processo ao ser criado recebe uma identificação, que é um número e, normalmente, também um nome. Chamamos esse número de **PID (Process IDentification = identificação do processo)**. O processo pode receber também, de acordo com o S.O., um número relacionado à identificação do usuário que o criou **UID (User IDentification = identificação do usuário)** e uma identificação do grupo de usuários aos quais é permitido o acesso a arquivos, processos, etc.: **GID (Group identification = identificação do grupo)**. Esses dois últimos estão

ligados ao modelo de segurança implementado por alguns S.O. para permitir acessos ou não. Por exemplo, um usuário não pode normalmente deletar os processos de outros. O super-usuário pode deletar todos os processos do sistema.

Um processo pode criar um ou mais processos (estes são chamados de processos-filho = *child process*). O processo filho tem um **PPID (Parent Process Identification = identificação do processo-pai)**, para que o filho possa retornar ao pai. O PPID de um processo-filho é o UID do processo-pai.

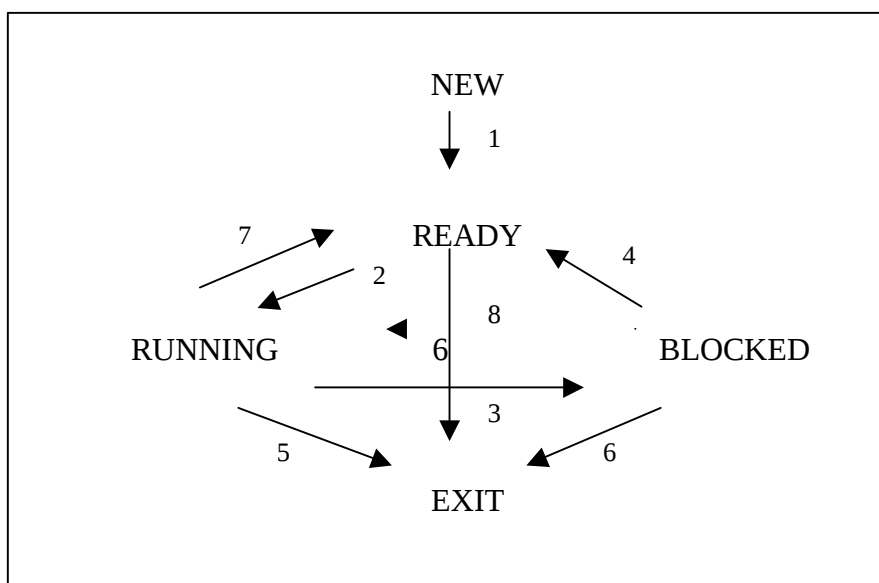
- **Cotas** – Limites de cada recurso que pode ser utilizado pelo processo. Esses limites são determinados para o processo no momento de sua criação. Alguns sistemas incluem também, dinamicamente, porcentagem do recurso já utilizado ou que ainda pode ser utilizado. Basicamente são: número máximo de arquivos abertos simultaneamente, tamanho máximo de memória que o processo pode alocar, número máximo de operações de E/S pendentes, tamanho máximo de *buffers* para o acesso de E/S, número máximo de subprocessos que podem ser criados.
- **Privilégios** – Definem o que um processo pode ou não fazer em relação ao sistema e aos outros processos. Já vimos (acima) os privilégios associados à segurança. Existem outros privilégios associados à operação e à gerência do sistema.

Espaço de Endereçamento

É a área da memória onde reside o processo. Quando um programa é colocado na memória principal, para que a sua execução seja possível, temos que as instruções estão arrumadas em ordem de execução em endereços adjacentes. Logo após ao bloco de instruções temos a seção de dados que serão utilizados pelo programa. Da mesma forma um processo reside na memória principal com o seu código, em endereços adjacentes, seguido de um bloco de dados.

➤ Estados do processo

Diagrama de Transição de estados de um processo:



New - o processo acabou de ser criado;

Ready - O processo está pronto para executar (só falta o recurso da CPU);

Running - O processo está executando (em modo usuário ou supervisor);

Blocked (sleep) - O processo aguarda disponibilidade de um recurso "lento" (não há previsão para o recebimento de um recurso que não seja a CPU);

Exit - O processo terminou.

1. Agente: S.O. - O S.O. inclui o processo dentre os prontos para executar;
2. Agente: S.O. (Rotina Dispatcher ou escalonador) - O processo ganha a CPU, troca de contexto e o processo começa a rodar. O escalonador controla essa transição através da lista de prontos (inserindo na lista);
3. Agente: O próprio processo - O processo necessita de um recurso não disponível para prosseguir a execução;
4. Agente: S.O. - O processo foi liberado;
5. Agente: O próprio processo - A imagem solicita ao S.O. ("System Call"). É a sua terminação;
6. Agente: S.O. - S.O. ou outro processo usando uma "system call". Homicídio (Kill);
7. Agente: S.O. - Preempção (troca de contexto) - ilusão que vários processos que utilizam a mesma CPU tem de parecer que tem uma própria CPU;
8. Agente: S.O. - Desespero - É utilizado em sistema de tempo real.

A política implementada, normalmente, é a de várias listas de acordo com o que o processo precisa fazer. Os nomes aqui citados e a quantidade de estados que um processo pode ter vão diferir de um SO para outro. Ao ser criado o processo vai para a lista de **PRONTO (READY)**, que indica pronto para começar a execução. Assim que houver oportunidade o SO vai selecioná-lo (escalonamento) para ocupar a CPU. Normalmente usa-se a política de listas encadeadas, ou seja, ao colocar na lista o SO já leva em conta as prioridades. Em cada lista o **ponteiro** de um processo aponta (indica) o próximo processo da lista.

Quando estiver executando, esse processo pode: terminar, fazer uma requisição de I/O, ser interrompido, ter sua fatia de tempo terminada, criar um novo processo. Enquanto está executando, ou seja, ocupando a UCP, o processo está no estado **EXECUÇÃO (RUNNING)**.

RESPONDA: 3.6) Numa máquina com um processador, quantos processos podem estar no estado EXECUÇÃO simultaneamente? E num ambiente com múltiplos processadores?

Quando faz uma requisição de I/O o processo pode ter que esperar pela liberação de algum dispositivo de I/O. Cada dispositivo tem a sua lista. Nesse caso o processo fica no estado de **ESPERA (WAIT)**. Se o recurso do sistema desejado não está disponível ainda, podemos ter uma diferenciação do estado de **ESPERA**, chamado de estado **BLOQUEADO**. Um processo pode estar no estado de **ESPERA** quando ele cria um processo-filho e tem que esperar pelo término do processo-filho para continuar a sua execução. Pode também ocorrer que um processo não precise esperar pelo término do processo-filho para voltar à sua execução, nesse caso os processos pai e filho executam concorrentemente.

O **término** de um processo envolve uma chamada de sistema, que faz com que o SO retire o processo da lista e desaloque o seu PCB. Se há processos-filho ainda não terminados, o SO “aborta” esses processos-filho.

RESPONDA: 3.7) Quantos processos podem estar no estado PRONTO ? E no estado ESPERA ?

3.8) Em que estado deve estar um processo para “entrar” na CPU ? Por que motivo?

3.9) Os processos em estado de PRONTO e ESPERA necessariamente precisam estar ocupando espaço na memória principal? Que técnica ou recurso é utilizado para solucionar esse tipo de ocorrência?

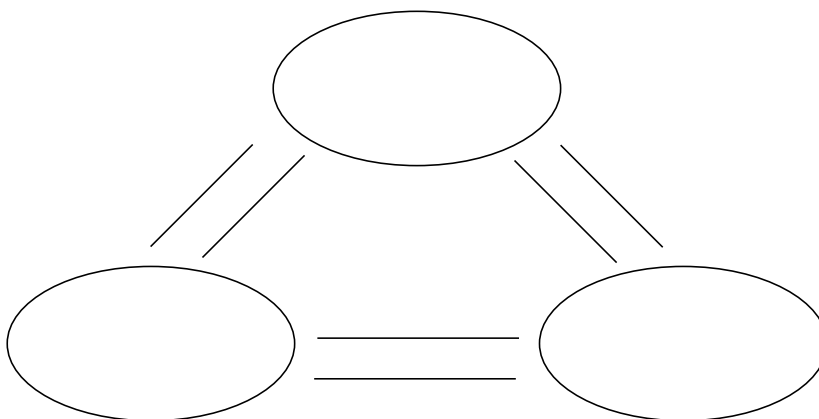
➤ Mudanças de estado

Pelo que vimos acima, diversos eventos causam uma mudança de estado em um processo. Chamamos de **eventos voluntários** aqueles gerados pelo próprio processo e de **eventos involuntários** aqueles gerados pelo sistema operacional.

Existem basicamente quatro mudanças de estado do processo, conforme será visto abaixo.

RESPONDA: 3.10) Complete o diagrama abaixo com os estados “básicos” de um processo e indicação das mudanças de estado possíveis (completando as setas e numere-as). Em seguida, responda:

- a) O que acontece ao processo em cada um dos estados?
- b) O que ocasiona cada tipo de mudança?
- c)
- d) Quais mudanças são ocasionadas por eventos voluntários? E por eventos involuntários?



➤ Tipos de processo

Os processos podem ser classificados de acordo com o tipo de processamento que realizam, a saber:

- **CPU-bound** (ligado à CPU) – passa a maior parte do tempo no estado de execução, ou seja, utilizando o processador. Esse tipo de processo realiza poucas operações de E/S.
- **I/O-bound** (ligado à E/S) – passa a maior parte do tempo no estado de espera, pois realiza um elevado número de operações de E/S.

Parte II – Estrutura**➤ Estrutura do Sistema Operacional**

Podemos criar um sistema tão grande e complexo como um sistema operacional somente dividindo-o em pequenas partes. Cada uma dessas partes deve ser uma porção bem delineada do sistema, com entradas, saídas e funções, cuidadosamente definidas. Logicamente, nem todos os sistemas têm a mesma estrutura, ou seja, não apresentam a mesma forma de ligação entre as partes. Contudo, os sistemas operacionais modernos geralmente possuem as seguintes partes:

- Gerenciamento de processos - criar e eliminar, suspender e retomar, sincronismo e comunicação entre processos;
- Gerenciamento da memória principal – manter o controle das partes da memória que estão sendo usadas e por quem, decidir que processos serão carregados para memória quando houver espaço disponível, alocar e desalocar espaço de memória quando necessário;
- Gerenciamento de memória secundária – o SO é responsável pelas atividades de alocação de espaço livre, *scheduling* de disco;
- Gerenciamento de Entrada/Saída – manter os *device drivers* para comunicação com os diferentes dispositivos, um *buffer-caching* para o sistema;
- Gerenciamento de arquivos – criar e eliminar arquivos e diretórios, manter mapeamento dos arquivos em disco;
- Proteção do sistema – se um sistema é multiusuário e permite múltiplos processos concorrentes, estes processos devem ser protegidos de outras atividades;
- *Networking* – em um sistema distribuído (fracamente acoplado) cada processador tem sua própria memória e seus processadores que se comunicam através do SO. A comunicação entre eles deve considerar roteamento e estratégias de conexão;
- Interpretador de comandos – um dos mais importantes programas do SO é o interpretador de comandos, que serve de interface entre o usuário e o SO. Alguns SO's incluem este programa no próprio núcleo (*kernel*). Já outros sistemas, como o DOS e o UNIX, tratam o interpretador de comandos como um programa especial que é executado quando uma sessão é iniciada.

Com isso, um sistema operacional fornece um ambiente para execução, melhor dizendo, fornece serviços para os programas e também para os usuários desses programas.

➤ System Calls

System Calls fornecem a interface entre os processos e o sistema operacional. Estas “chamadas” estão geralmente disponíveis como instruções da linguagem Assembly, e são normalmente encontrados nos manuais usados por programadores de linguagens Assembly. Alguns sistemas permitem que as *system calls* sejam criadas diretamente a partir de um programa em linguagem de alto nível (linguagem C, Pascal, FORTRAN). Elas podem ser agrupadas, na maioria dos sistemas, em cinco categorias principais:

- controle de processos (end, abort, load, execute, create, terminate, wait event, signal event, set attributes);
- manipulação de arquivos (create, delete, open, close, read, write, set attributes)
- manipulação de dispositivos (request, release, read, write, logically attach or detach);
- manutenção de informação (get and set time or date, get and set process or file);
- comunicação (create and delete communication connection, send and receive messages)

A partir do momento que as “chamadas ao sistema” servem de interface entre os processos e o SO, essas são o mecanismo de proteção ao núcleo do SO e também de acesso aos seus serviços, como se fossem as portas de entrada para os processos.

➤ **Modos de acesso**

- Dispositivos/recursos compartilhados (E/S de um sistema) devem ser acessados por instruções exclusivas pelo SO;
- Instruções podem ser: privilegiadas (podendo comprometer a estabilidade do sistema) ou não-privilegiadas (não oferecem perigo ao sistema);
- Para execução de instruções privilegiadas, o processador implementa através de um registrador especial, o mecanismo de modos de acesso:
 - modo usuário → permite a execução de um subconjunto do total de instruções disponíveis, ou seja, as instruções não-privilegiadas;
 - modo kernel ou supervisor → todo conjunto as as instruções podem ser executadas;
- O SO executa em modo kernel, protegendo o hardware do usuário, enquanto os demais software (editores, compiladores) executam em modo usuário;
- Quem determina o acesso, controla e alterna o modo?
As system calls e, caso o programa tente executar uma instrução privilegiada, sem o processador estar em modo kernel, uma exceção é gerada e o programa encerrado.

➤ **Estruturas**

A seguir, serão apresentadas algumas maneiras como o código do sistema é organizado e o relacionamento entre seus diversos componentes, ou em outras palavras, sua estrutura interna.

Sistemas Monolíticos

A organização mais comum aos sistemas operacionais é a monolítica. Em um sistema monolítico temos um conjunto de rotinas responsável pela interpretação dos parâmetros passados quando da chamada do sistema por parte de um programa aplicativo, pela execução do serviço solicitado e pelo retorno dos resultados. Qualquer rotina presente no sistema operacional pode vir a chamar qualquer outra das rotinas.

Um bom exemplo é o MS-DOS, originalmente escrito para fornecer o máximo de funcionalidade no menor espaço, até pela limitação do hardware no qual era executado. Como o INTEL 8088 da época não tinha registrador de modo de acesso, os desenvolvedores do MS-DOS não tinham escolha, deixando o hardware básico desprotegido (popular; não foi dividido em módulos; seus níveis de funcionalidade não são separados; programas errados ou maliciosos podem comprometer seu funcionamento).

Sistemas em Camadas

A modularização de um sistema operacional pode ser feita de diferentes formas; a mais utilizada é a aproximação em camadas, que consiste em dividir o sistema operacional em um número de camadas (níveis), hierarquicamente dispostas, cada nível construído sobre o nível imediatamente abaixo. O nível

mais baixo (nível 0) é o hardware e o mais alto é a interface com o usuário.

Módulos de uma camada oferecem funções aos módulos de camadas superiores; cada camada é implementada usando somente aquelas operações fornecidas pelas camadas de mais baixo nível, sendo que a camada não necessita saber como estas operações são implementadas; ela necessita saber o que estas operações fazem.

Sistemas Cliente-Servidor

O kernel do SO passa a ser responsável pela comunicação entre processos e pela implementação de operações que seriam difíceis de serem executadas a partir dos processos servidores. A maioria dos serviços que seriam prestados pelo SO, executado em modo supervisor em uma organização monolítica, passariam a ser prestados por um conjunto de processos servidores que seriam executados em modo usuário, sendo apenas o kernel ainda executado em modo supervisor.

No caso do Windows NT, da Microsoft, cada subsistema do ambiente e o NT Executive são implementados como sendo vários processos. Cada processo espera por uma solicitação de um cliente para um de seus serviços (por exemplo, serviços de memória, serviços de criação de processos, ou serviços de escalonamento do processador). Um cliente, que pode ser uma aplicação do usuário ou outro módulo do sistema operacional, solicita um serviço por envio de uma mensagem. A mensagem é direcionada através do NT Executive para o servidor apropriado. O servidor realiza a operação solicitada e retorna os resultados por meio de uma outra mensagem, que é direcionada através do NT Executive de volta ao cliente. [STAL98]

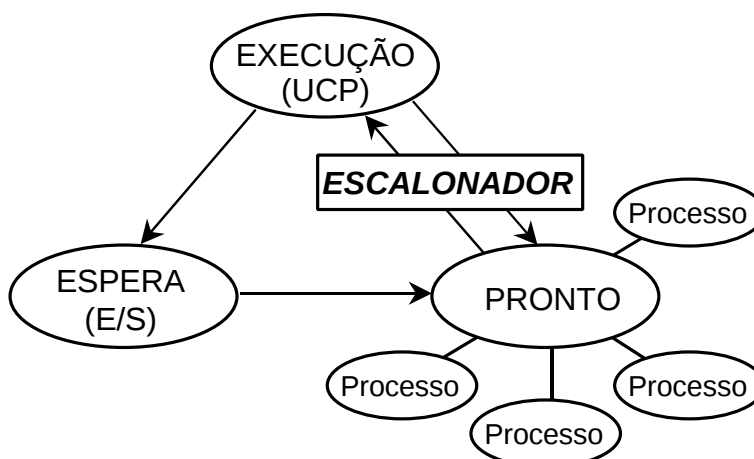
[Índice](#)

Gerência do Processador

➤ Conceito de Escalonamento

Com a possibilidade da UCP ser compartilhada entre diversos processos (multiprogramação), o SO possui critérios para determinar qual a ordem na escolha dos processos para que estes passem do estado de PRONTO para EXECUÇÃO.

O procedimento de seleção é função do SO, sendo conhecido como escalonamento (*scheduling*). A parte do código do SO responsável pelo escalonamento é o escalonador (*scheduler*). O SO deve tratar todos os processos igualmente, evitando starvation.

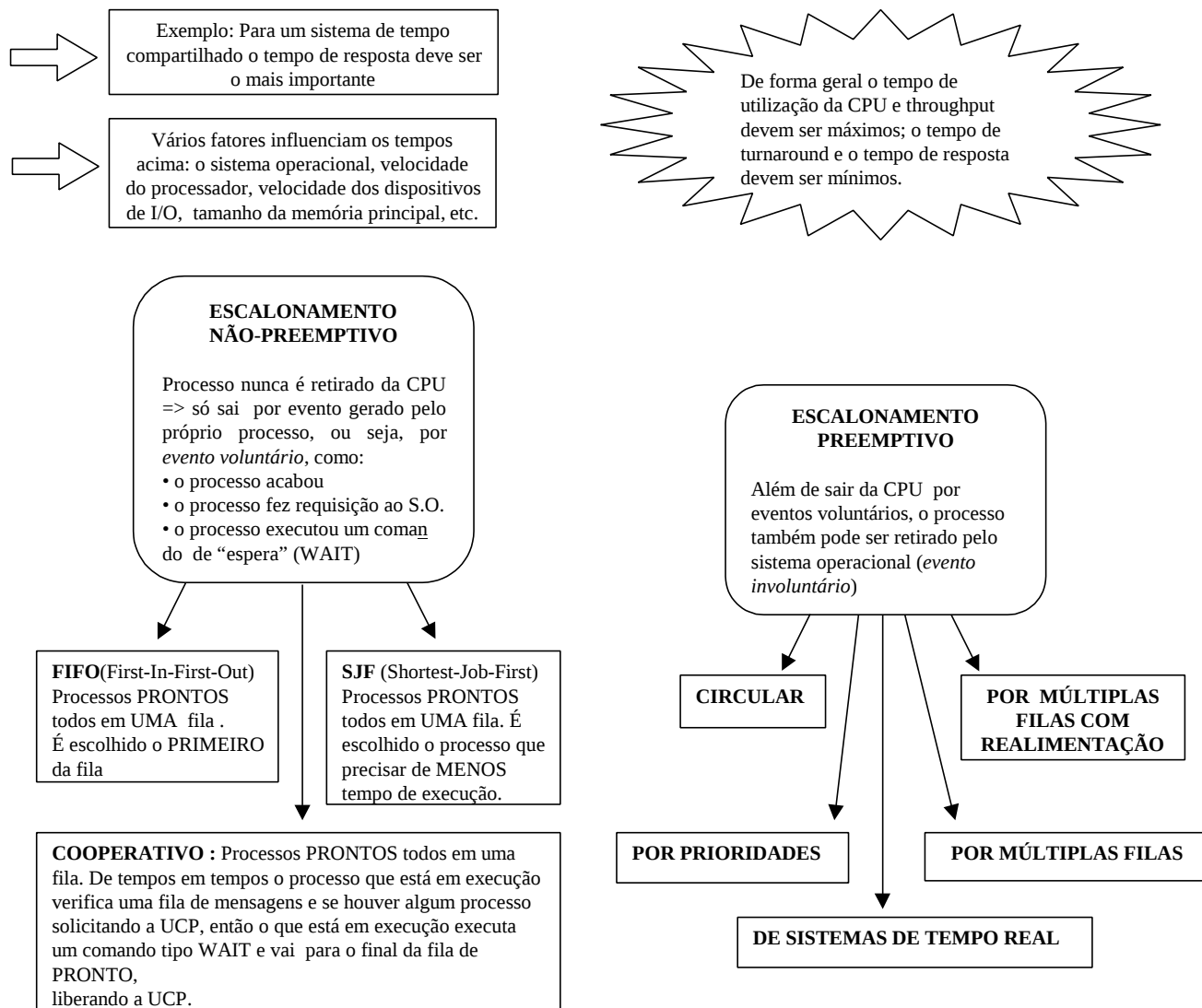


Como já foi mostrado anteriormente, um bom sistema operacional multitarefa deve levar em conta os critérios abaixo. Lembre-se que alguns podem ser mais “importantes” que outros, dependendo da atividade que o sistema operacional desempenhará com suas aplicações:

- **Utilização da CPU** : o processador principal deve estar ocupado a maior parte do tempo possível, a não ser que não haja o que executar mesmo;
- **Throughput** : é o número de processos executados por unidade de tempo; é desejável que a quantidade de processos “atendidos” pelo sistema em um determinado intervalo de tempo seja a maior possível;
- **Tempo de turnaround** : tempo decorrido desde a admissão do processo no sistema até o seu término. Então, inclui tempo na fila de espera + tempo na fila de pronto + tempo de execução;
- **Tempo de Resposta** : Em sistemas interativos, é o tempo decorrido entre o instante da submissão de um pedido (teclar um ENTER após um comando, por exemplo) e a apresentação da primeira resposta.

Algoritmos de escalonamento buscam otimizar a utilização da UCP e o throughput, enquanto tentam diminuir os tempos de turnaround e de resposta. No entanto, o escalonamento somente afeta o tempo de resposta de processos na fila de pronto.

➤ Tipos de Escalonamento



Alguns PROBLEMAS encontrados em escalonamentos não preemptivos:

- FIFO – previsão de início da execução do processo; processos CPU-bound de menor importância prejudicam processos I/O-bound mais prioritários;
- SJF – determinar quanto tempo de UCP cada processo necessita para terminar seu processamento;
- COOPERATIVO – não existe nenhuma intervenção do SO na execução do processo, o que pode ocasionar problemas caso o processo não libere o processador.

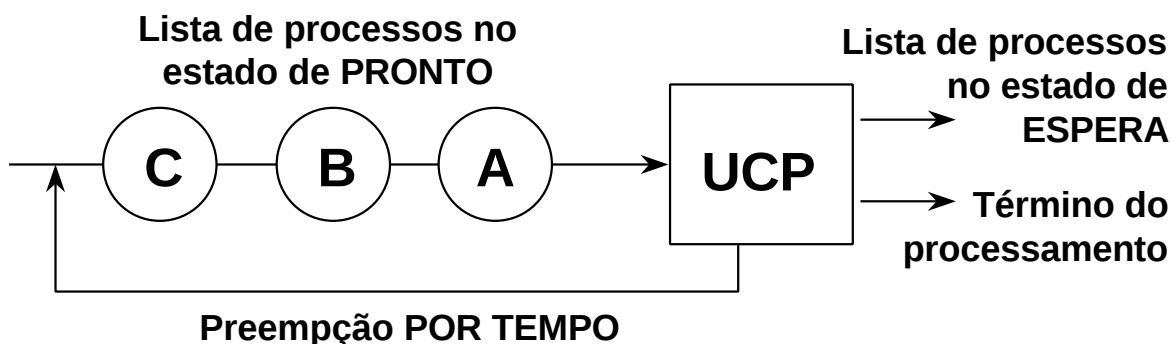
Escalonamento Circular ou Round Robin

Implementação: UMA fila de PRONTO. Os processos entram na fila na ordem de chegada e sai primeiro o que chegou primeiro (FIFO). É definida uma **fatia de tempo (time-slice)**.

Um processo em execução pode deixar a CPU pelos seguintes motivos :

- acabou → nesse caso deixa de estar ativo no sistema
- fez requisição ao SO → nesse caso fica em estado de ESPERA
- acabou a *fatia de tempo* → nesse caso fica em estado de PRONTO, entrando no final da fila de processos PRONTOS

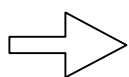
Quando termina a **fatia de tempo**, sem que o processo tenha terminado ou feito uma requisição ao SO, é o SO quem retira o processo de execução. Chamamos este evento (que é involuntário) de **preempção por tempo**.



Escalonamento por Prioridades

Implementação: UMA fila de PRONTO. Cada processo tem, associada a ele, uma **prioridade de execução**. Tal **prioridade** pode ser dada ao processo pelo usuário ou pelo SO; pode ser **estática** (quando é sempre a mesma ao longo da vida do processo) ou pode ser **dinâmica** (quando o SO a modifica durante a vida do processo, de acordo com o tipo de processamento e/ou carga do sistema).

Um clock interrompe o processador em determinados intervalos de tempo, para que seja executada a **rotina de escalonamento**. Esta rotina arruma os processos na fila de acordo com suas prioridades. Se na fila de PRONTO estiver um processo com **prioridade maior** do que a prioridade do processo que estava em execução, este que estava em execução passa a ficar em estado de PRONTO e o outro de maior prioridade é escalonado para execução. Chama-se **preempção por prioridade** o evento em que o SO retira um processo que estava em execução para dar lugar a outro processo de maior prioridade. Note que este evento é involuntário, ou seja, não depende do processo que está sendo “preemptado”.



Exemplo de **prioridade dinâmica**: para compensar o tempo que os processos ficam em estado de ESPERA, o S.O faz um acréscimo à prioridade dos processos sempre que saem do estado de ESPERA.

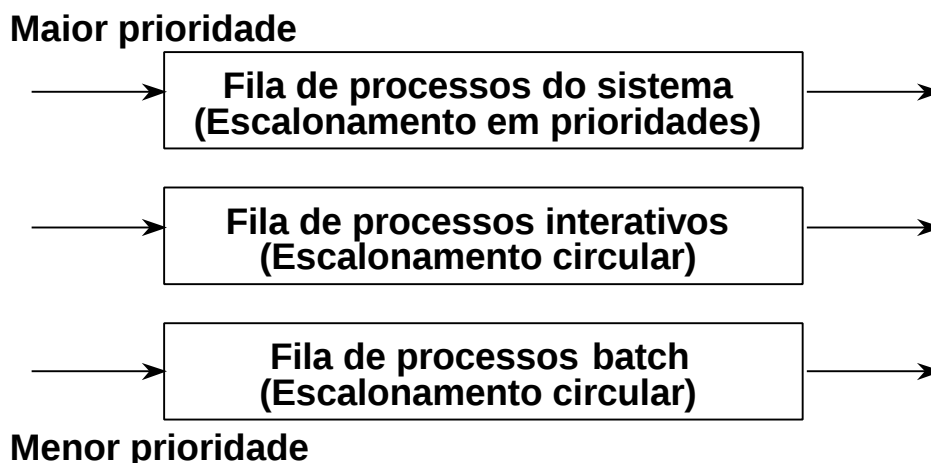
Escalonamento por Múltiplas Filas

Implementação: VÁRIAS FILAS DE PRONTO, sendo que a cada uma está associada uma prioridade. Cada processo é associado a UMA e SOMENTE UMA fila e nela PERMANECE durante toda a sua vida no sistema.

O sistema SÓ PODE ESCALONAR PROCESSOS DE UMA DAS FILAS QUANDO TODAS AS OUTRAS DE MAIOR PRIORIDADE ESTIVEREM VAZIAS.

Exemplo

Seja um sistema com três filas:



Escalonamento por Múltiplas Filas com realimentação

Implementação: VÁRIAS FILAS DE PRONTO, sendo que a cada uma está associada uma prioridade. O sistema SÓ PODE ESCALONAR PROCESSOS DE UMA DAS FILAS QUANDO TODAS AS OUTRAS DE MAIOR PRIORIDADE ESTIVEREM VAZIAS.

O SO PODE MUDAR um processo DE FILA de acordo com as características do processo e/ou carga do sistema. Chama-se este esquema de **mecanismo adaptativo**.

Quando o processo é criado, entra no final da fila de MAIOR prioridade. Quando um processo em execução deixa a CPU seja por *preempção por prioridade* ou por uma requisição ao SO, ele é reescalonado dentro da MESMA fila. Quando deixa a CPU por *preempção por tempo*, o processo é redirecionado para uma fila de MENOR prioridade. Quanto **maior** for a prioridade de uma fila **menor** será a *fatia de tempo* associada a esta fila.

Escalonamento de Sistemas de Tempo Real

É um escalonamento UNICAMENTE por prioridade ESTÁTICA.

Escalonamento com Múltiplos Processadores

- **Sistemas Fortemente Acoplados (Memória Compartilhada):** UMA única fila de PRONTO para

42

todos os processadores. Todos os processos estão presentes nesta fila e são escalonados no primeiro processador disponível. Naturalmente, cada processo só pode ser escalonado para um processador.

- **Sistemas Fracamente Acoplados (Memória Distribuída):** cada processador tem o seu próprio escalonamento, visto que cada processador tem sua memória própria, seu sistema operacional próprio e consequentemente seu algoritmo de escalonamento próprio.

[Índice](#)

Exercícios

- 4.1 Qual a parte do sistema operacional responsável pela escolha de processos no estado de pronto que passarão para o estado de execução?
- 4.2 O que é escalonamento?
- 4.3 Justifique: um algoritmo de escalonamento busca otimizar a utilização da UCP e o *throughput*, enquanto tenta diminuir os tempo de *turnaround* e de resposta?
- 4.4 O que é *starvation*? O escalonamento de processos pode levar um processo a sofrer *starvation*? Explique?
- 4.5 Cite fatores que possam influenciar no tempo total de execução de processos.
- 4.6 O que é preempção? Diferencie escalonamento preemptivo de não-preemptivo.
- 4.7 Como funciona o escalonamento FIFO? (Lembre-se de uma fila de banco)
- 4.8 Sejam os processos abaixo:

Identificação do processo	CPU (u.t.)	Característica do processo
A	52	faz requisição de I/O a cada 15u.t.
B	11	faz requisição de I/O a cada 3u.t.
C	3	faz requisição de I/O a cada 2u.t.

onde por “CPU (u.t.)” estamos representando quantas unidades de tempo são necessárias para a execução dos processos em CPU

Suponhamos que sejam necessárias 8 u.t. (oito unidades de tempo) para que se realize qualquer uma das requisições de I/O e que Vamos desconsiderar o *overhead* de criação/deleção de processos e de troca de contexto. O tempo gasto no escalonamento de processos por parte do S.O. é de 1 u.t. (uma unidade de tempo). Considere que os processos A, B e C entraram no sistema nessa ordem (A, B e depois C), mas a diferença entre tais instantes é tal que pode ser desconsiderada. Considere também que os processos foram criados no instante inicial de funcionamento do sistema (0 u.t.). Faça o papel do sistema operacional e complete os quadros abaixo:

1ª situação) Para um algoritmo de escalonamento não-preemptivo FIFO (nenhum processo fique em momento algum bloqueado).

Instante de tempo (u.t.)	Fila de Pronto	Executando na UCP	Fila de Espera

Tempo que a CPU ficou inativa = _____ % do tempo total

2ª situação) Para um algoritmo de escalonamento não-preemptivo FIFO.

Instante de tempo (u.t.)	Fila de Pronto	Executando na UCP	Fila de Espera

Tempo que a CPU ficou inativa = _____ % do tempo total

3ª situação) Para um algoritmo de escalonamento preemptivo ROUND ROBIN com *quantum* de 10u.t.

Instante de tempo (u.t.)	Fila de Pronto	Executando na UCP	Fila de Espera

Tempo que a CPU ficou inativa = _____ % do tempo total

Baseado nos resultados dos quadros acima, preencha: (1,0p)

	Tempo de turnaround			Tempo Ocioso da CPU			Avaliação
	I)	II)	III)	I)	II)	III)	
Processo A				%	%	%	
Processo B				%	%	%	
Processo A + B				%	%	%	

4.9 Considerando na fila de PRONTO de um sistema operacional que implementa um algoritmo de escalonamento de processos de forma preemptiva circular, sem prioridades e de *quantum* fixo, as seguintes situações:

1ª situação) Muitos processos *CPU-Bound* e poucos processos *I/O-Bound*

2ª situação) Somente processos *I/O-Bound*

3ª situação) Muitos processos *I/O-Bound* e poucos processos *CPU-Bound*

Classifique cada sentença como Verdadeira (V) ou Falsa (F), justificando-as quando falsas:

I. Na 2ª situação, mesmo que o único dispositivo de E/S seja rápido, a concorrência à utilização é grande o que ocasiona menor utilização de tempo da UCP. ()

II. Pode-se afirmar que o escalonamento de processos por parte do sistema operacional é o único responsável pelo tempo de *turnaround* dos processos, afetando até o tempo que os processo ficam na fila de espera. ()

III. Na 3ª situação a minoria de processos *CPU-Bound* sofreria *starvation*. ()

IV. Com exceção da 2ª situação, os processos *CPU-Bound* certamente irão monopolizar o processador, mesmo que o tempo máximo de utilização do processador seja igual ao *quantum*. ()

V. Na 1ª situação, quanto menor o tempo perdido na troca de contexto pior é, pois os processos *I/O-Bound* solicitam operações de E/S a todo instante, independente do aproveitamento do processador por parte dos processos *CPU-Bound*. ()

[Índice](#)

Sincronização na Comunicação entre processos

➤ Problemas de comunicação

A comunicação entre processos concorrentes deve ser tratada pelos sistemas operacionais. Os mecanismos que garantem a comunicação entre processos concorrentes e o acesso a recursos compartilhados são chamados **mecanismos de sincronização**;

Vejamos um exemplo onde pode ocorrer acesso concorrente a um arquivo em disco (memória secundária):

- Imagine um arquivo de contas bancárias (ARQ_CONTAS) que contenha as informações de cada cliente do banco (REG_CLIENTE), dentre elas o saldo. Quando é realizado algum lançamento por um caixa, seja depósito ou retirada, um simples programa é executado:

```
READ (ARQ_CONTAS, REG_CLIENTE);  
READLN (VALOR_DEP_RET);  
REG_CLIENTE.SALDO := REG_CLIENTE.SALDO+VALOR_DEP_RET;  
WRITE (ARQ_CONTAS, REG_CLIENTE);
```

- Se em um mesmo instante dois caixas diferentes processarem seus lançamentos no saldo de um mesmo cliente e, não houver controle do compartilhamento, o que pode acontecer?
Suponha que os dois caixas leiam do arquivo o saldo (R\$ 100,00, por exemplo), um deles realize uma operação de depósito de R\$ 50,00 (total de R\$ 150,00) e o outro de 500,00 (total de R\$ 600,00); o que realizou o maior depósito atualiza a informação do arquivo primeiro e o outro atualize a informação logo depois, no instante seguinte; o resultado seria um saldo de R\$ 150,00 para os dois depósitos.

CONCEITOS:

Recursos X RC (Regiões Críticas)

Recursos - Qualquer coisa necessária para prosseguir com sua execução

Região Crítica - Trecho de programa (processo) onde se vai fazer uso do recurso.

Postulado de Dijkstra sobre programação concorrente:

1. Garantia da Exclusão Mútua: Só um processo pode ter acesso ao recurso de cada vez (entrar na RC)
2. Um processo fora da RC (Não está usando o recurso) não deve impedir que outro processo entre na RC;
3. Nenhum processo deve sofrer adiamento indefinido - Starvation;
4. Nada pode ser assumido quanto à velocidade de execução dos processos quanto ao número de processadores;
5. Todo processo detém a posse do recurso (permanece na RC) por tempo finito.

➤ Exclusão mútua e região crítica

A solução mais simples para evitar problemas de compartilhamento de recursos é impedir que dois ou mais processos acessem este recurso no mesmo instante. Essa exclusividade de acesso é chamada **exclusão mútua**, ou seja, enquanto um processo acessa determinado recurso, todos os outros que queiram acessá-lo deverão esperar até o término deste acesso.

A **exclusão mútua** deve afetar os processos concorrentes apenas quando um deles estiver realizando acesso ao recurso compartilhado.

Somente a parte do código do programa onde é realizado o acesso ao recurso compartilhado, chamada de **região crítica**, é executada de forma mutuamente exclusiva em relação a todas as outras. Em outras palavras, se for possível evitar que dois processos entrem em suas regiões críticas ao mesmo tempo, evitam-se problemas decorrentes do compartilhamento.

➤ *Soluções de software para exclusão mútua*

a) Tranca

Variável inteira que assume dois valores que simbolizam os estados de Ocupado ou Livre, e a alteração de seus valores é indivisível.

Para tal precisa-se que o HW garanta o recurso e altere o seu valor em um único ciclo da CPU.

Operação de Test and Set - TAS:(Instrução em Assembler). A partir do momento que é ativada pede e garante o recurso. Substitui a variável que indica se está trancada ou não por uma indicando que está trancada, independentemente de qual era o seu estado anterior. Assim ela garante o recurso.

Problemas:

- Depende de instrução específica;
- Pode ocasionar starvation (3 processos: 2 revesam a posse do recurso e 1 nunca o conhece)
- Ocorre espera Ocupada.

Por que usar esta solução?

Se o recurso for rápido (alterar listas, tabelas,...) não vale a pena adormecer o processo (trocar o contexto), portanto apesar da espera ocupada, esta solução se aplica a recursos rápidos.

Spinlock - possui o TAS embutido.

Spinunlock - Destrava o recurso após sua utilização.

Tranca é um mecanismo de sincronização que permite a manipulação de um recurso simples por diversos processos. Apenas 1 processo detém a posse deste recurso em um dado instante de tempo.

2 formas de checar o valor da tranca:

- loop interno
- test an set - loop externo (2 acessos ao barramento)

Qual a vantagem de se ter um loop interno dentro do TAS?

A vantagem do loop interno é que degrada menos o barramento, pois faz apenas um acesso para fazer a consulta se o recurso está ou não disponível.

b) SEMÁFOROS

Semáforo é uma forma de sincronização que permite a utilização de múltiplos recursos por vários processos, mas apenas um único processo pode manipulá-lo em dado instante de tempo. Isso ocorre porque existe um mecanismo de tranca em sua composição que tem uma política de que apenas 1 processo pode ser executado por vez (processo de tranca).

Um **semáforo** é uma variável inteira, não negativa, que só pode ser manipulada por duas instruções: DOWN e UP. No caso da exclusão mútua, essas instruções funcionam para que um processo possa entrar e sair de sua região crítica, respectivamente.

O **semáforo** fica associado a um recurso compartilhado, indicando quando este recurso está sendo acessado por um dos processos concorrentes.

O uso de **semáforos** exige do programador muito cuidado, pois qualquer engano pode levar a problemas de sincronização imprevisíveis e difíceis de reproduzir, devido a execução concorrente dos processos.

Características

Elimina a espera ocupada

Generaliza a espera ocupada para recursos múltiplos

Tenta acabar com a starvation

Definição: O semáforo é uma variável inteira que assume valores na faixa de 0....N.

Operações:

INIT (S,X) - $S=X$, sendo $0 \leq X \leq N$

DOWN (S) - se $S > 0$ então $S=S-1$

Caso contrário o processo na lista de espera é bloqueado (Blocked)

O Down estabelece a permissão do acesso (entrada na RC)

-UP(S) - se há processos esperando na lista de S

Acorda um deles (Ready)

Caso contrário ($S=S+1$)

Chamadas:

Down (s);

RC ();

UP (s);

Obs.: As operações são indivisíveis.

Semáforo é:

- Valor (a ser incrementado/ decem.)
- Lista de espera (ponteiros p; PCB)
- Variável de tranca (garante a indivisibilidade)

DEADLOCK - É uma situação de falha na sincronização irreversível, ou seja, ocorrerá um adiamento perpétuo na execução do processo. (isto ocorre quando se esquece de dar o spinunlock)

Difere do starvation pois este adiamento é indefinido e não perpétuo. (ex. quando a região crítica é muito extensa)

Trydown - É uma função como o down dos semáforos mas esta dá uma poder decisório para se tomar uma atitude caso não haja um recurso disponível. O processo não precisa dormir automaticamente na ausência do recurso. Se houver um recurso disponível ele age exatamente como o down.

Ex.: O Trydown pode dar uma mensagem de erro ao usuário, avisando sobre a ausência de recursos. Assim o usuário poderia procurar o recurso em outra máquina.

Com esta função, você pode ter uma rotina de espera ocupada, que garante a exclusão mútua, usando semáforos, da mesma forma que com trancas. É só colocar um loop de trydown testando a disponibilidade do recurso.

c) MONITORES

O **monitor** é um conjunto de procedimentos, variáveis e estrutura de dados definido dentro de um módulo, onde a exclusão mútua é implementada automaticamente entre seus procedimentos. O mais importante é que toda a implementação da exclusão mútua nos **monitores** é realizada pelo compilador, e não mais pelo programador, como no caso dos semáforos. As regiões críticas são colocadas em forma de procedimentos no **monitor** e o compilador se encarrega de garantir a exclusão mútua desses procedimentos, sendo as chances de erro menores.

Gerência da Memória

➤ Alocação de memória

Ao executar um programa residente na memória secundária, deve-se, de alguma forma, carregá-lo para a memória principal. A organização e a gerência da memória principal são fatores importantes no projeto de sistemas operacionais modernos. Melhor dizendo, os sistemas operacionais devem ocupar pouca memória e otimizar ao máximo sua utilização.

A) **Alocação contígua simples** – comum em sistemas monoprogramáveis, onde a memória principal é dividida em duas partes: uma para o sistema operacional e outra para o programa do usuário. O programador deve se preocupar em não ultrapassar o espaço disponível de memória endereçável, ou seja, o tamanho total da memória principal menos o que está sendo ocupado pelo sistema operacional. Para proteger a área do sistema operacional alguns desses sistemas implementavam proteção através de um registrador, que delimita as áreas.

Nesse esquema de fácil implementação e código reduzido, apenas um usuário poderia dispor do processador e da memória, que ficava sem utilização caso o programa do usuário não o preenchesse totalmente. Como os programas estavam limitados ao tamanho da memória principal disponível, a situação foi contornada dividindo-se o programa em partes (módulos) que pudessem executar independentemente uma da outra, utilizando uma mesma área de memória (*overlay*);

B) **Alocação particionada estática** – para que a multiprogramação fosse eficiente, era necessário que vários programas estivessem na memória principal ao mesmo tempo. Por esse motivo, a memória foi dividida em pedaços de tamanho fixo, chamados partições. O tamanho das partições era estabelecido na fase de inicialização do sistema, em função do tamanho dos programas que iriam executar no sistema. Sempre que fosse necessária a alteração do tamanho de uma partição, o sistema deveria ser desativado e reinicializado com a nova configuração.

Quando os compiladores e linkeditores geravam CÓDIGO ABSOLUTO, o código gerado possuía endereços físicos de memória (observação: vamos trabalhar com endereços expressos em decimal). O formato geral das instruções de máquina é:

endereço	instrução
----------	-----------

Para facilitar, vamos supor que cada instrução de alto nível gera 1 (uma) instrução de máquina. Com código absoluto as instruções de máquina de um trecho do programa ficariam:

ENDEREÇO	CÓDIGO DA INSTRUÇÃO
501	
502	Ler B, C e H
503	$A = B + C$
504	$D = H - A$
505	Se $D < 0$ então vá para 502
506	$J = V * H$
507	Escreve J
508	

Este programa SÓ PODE SER carregado na memória nos endereços físicos 502 a 507.

Com a evolução dos compiladores, *linkeditores* e *loaders*, a geração de **CÓDIGO RELOCÁVEL** foi possível, e os programas podiam então ser carregados em qualquer região disponível da memória. Ou seja, as instruções são geradas com um **endereço virtual** e não com o endereço físico.

ENDEREÇO VIRTUAL	CÓDIGO DA INSTRUÇÃO
0	Ler B, C e H
1	$A = B + C$
2	$D = H - A$
3	Se $D < 0$ então vá para 0
4	$J = V * H$
5	Escreve J

No momento em que o programa é carregado na memória, um **registrador de relocação** recebe o endereço inicial da região de memória que o programa irá ocupar (relocação dinâmica). Toda vez que ocorrer uma referência a algum endereço virtual, o endereço contido na instrução será somado ao conteúdo do registrador de relocação, obtendo-se desta forma o endereço físico. Então esse programa (como qualquer outro) PODERÁ RESIDIR EM QUALQUER PARTE da memória.

Para o exemplo anterior:

REGISTRADOR DE
RELOCAÇÃO

721

ENDEREÇO VIRTUAL	ENDEREÇO FÍSICO	CÓDIGO DA INSTRUÇÃO
0	$0+721=721$	Ler B, C e H
1	$1+721=722$	$A = B + C$
2	$2+721=723$	$D = H - A$
3	$3+721=724$	Se $D < 0$ então vá para $0+721=721$
4	$4+721=725$	$J = V * H$
5	$5+721=726$	Escreve J

Como os programas normalmente não preenchiam totalmente as partições onde eram carregados, eles deixavam pedaços de memória que ficariam impedidos de ser utilizados por outros programas (fragmentação). E mesmo que existissem duas ou mais partições adjacentes que, somadas, totalizassem o tamanho do programa, ele ficaria aguardando uma única que o acomodasse;

A) **Alocação particionada dinâmica** – Como foi visto no item anterior, havia a necessidade de diminuir o problema da fragmentação, aumentando o compartilhamento da memória principal. Na alocação particionada dinâmica foi eliminado o conceito de partições de tamanho fixo, de forma que cada programa utilizasse o espaço de que necessitasse, passando esse pedaço a ser uma partição. A fragmentação no entanto, começaria a ocorrer quando os programas fossem terminando e deixando espaços cada vez menores na memória, não permitindo ingresso de novos programas.

Existiam duas possíveis soluções para este problema: na primeira, apenas os espaços adjacentes são reunidos, produzindo um único espaço de tamanho maior; a segunda maneira envolve a relocação de todas

as partições ocupadas, eliminando todos os espaços entre elas e criando uma única área livre contígua (alocação dinâmica com relocação). Porém a complexidade do seu algoritmo e o consumo de recursos do sistema, como processador e área em disco, podem tornar esse esquema inviável;

➤ **Estratégias para escolha de partição**

Na tentativa de evitar, ou mesmo reduzir, a fragmentação antes que ela ocorra, algumas estratégias são implementadas pelos sistemas para determinar em qual partição livre um programa será carregado para execução:

- ⇒ BEST-FIT – escolhe a partição em que o programa deixa o menor espaço sem utilização; a lista de áreas livres está ordenada por tamanho; desvantagem: com a alocação de partições que deixem as menores áreas livres, a tendência é que cada vez mais a memória fique com pequenas áreas não contíguas, aumentando o problema da fragmentação;
- ⇒ WORST-FIT – é escolhida a partição em que o programa deixa o maior espaço sem utilização; apesar de utilizar as partições maiores, os espaços livres maiores permitem o aproveitamento da memória por um maior número de programas;
- ⇒ FIRST-FIT – esse mecanismo escolhe a primeira partição livre, que tenha tamanho suficiente para carregar o programa; a lista de áreas livres está ordenada por endereços crescentemente; é mais rápida, consumindo menos recursos do sistema.

➤ **Swapping**

É uma técnica utilizada para tentar solucionar o **problema de falta de memória**. Seja uma memória com área livre para programas de usuário de 500Kb. Imagine que há 4 programas A, B, C e D, cada um com 200Kb, que poderiam estar todos ativos se coubessem na memória. Então aloca-se os programas A e B somente. Se o programa A tiver que aguardar por algum evento, podemos retirá-lo da memória e levá-lo para o disco, enquanto aguarda pelo evento. Enquanto isso o programa C, por exemplo, pode ir para a memória para que sua execução seja iniciada.

SWAP OUT → retirada de um programa da memória para o disco

SWAP IN ← retorno do programa do disco para a memória

➤ **Memória Virtual**

A utilização da memória virtual é uma outra técnica para tentar solucionar o **problema de falta de memória**. Esta técnica consiste em:

- utilizar os endereços virtuais
- o disco com continuação da memória principal

sem que o usuário perceba.

Se um programa de 200Kb só pode utilizar 100Kb da memória, porque não há mais área livre, os outros 100Kb ficarão em disco. Somente quando os endereços virtuais que estão no disco forem referenciados é que a 2ª parte do programa vai para a memória.

➤ **Mapeamento**

Mecanismo que permite ao SO traduzir os endereços virtuais em endereços reais.

Implementação por software: o SO tem uma tabela com a tradução, que é mantida na memória principal. Esse mecanismo é custoso, pois envolve percorrer tabelas

End. Real	Nº processo	End. Virtuais
210	3174	20 a 133
323	2022	74 a 191
431	3174	0 a 20

Implementação por hardware: **Translation Lookside buffer** ou **memória associativa**
Qual o tamanho desses pedaços de programa?

➤ **Paginação**

O espaço de endereçamento virtual e o espaço de endereçamento real são divididos em blocos do mesmo tamanho.

O princípio da localidade é a tendência dos programas de fazerem referência às posições consecutivas da memória. Baseado nisso surgiu a idéia de se dividir os processos em blocos de tamanhos fixos, chamados PÁGINAS. Se dividirmos também o espaço real em blocos do mesmo tamanho, teremos na memória principal, por exemplo, algo assim:

Página 4 do processo A
Página 10 do Processo B
Página 1 do processo A

Com isso não haverá espaços livres entre os espaços ocupados, isto é, **fragmentação**. As páginas são levadas para a memória principal de acordo com a sua necessidade. Ao ser referenciado um endereço qualquer de uma página, toda a página é colocada na memória.

Nesse esquema o endereço virtual é:

nº da página	deslocamento dentro da página
--------------	-------------------------------

Cada processo tem sua tabela de páginas:

nº da página	end. físico	Bit de validade
000		0
001	7024	1
002	810	1

O bit de validade informa se a página está na memória (igual a 1) ou não (igual a 0).

Por exemplo: o endereço virtual 001 3 do processo A está no endereço real $7024+3=7027$. Se um endereço referenciado não está na memória principal a página é alocada e a tabela de páginas é atualizada. Esse

mecanismo de só levar para a memória principal as páginas que são necessárias chama-se PAGINAÇÃO POR DEMANDA. E chama-se FALHA DE PÁGINA (page fault) o evento de o processo precisar de 1 página que não está na memória.

WORKING SET de um processo é o conjunto de páginas referenciadas por ele durante um intervalo de tempo. Uma segunda definição: o conjunto das páginas constantemente referenciadas pelo processo, devendo permanecer na memória principal. Caso contrário, o processo poderá sofrer com a elevada taxa de paginação (TRASHING) comprometendo seu desempenho. O *working set* do processo deve ter um limite máximo de páginas permitidas; quanto maior a *working set*, menor a chance de ocorrer FALHA DE PÁGINA.

A paginação envolve uma operação de E/S, então é aconselhável que haja uma política que diminua a taxa de paginação. A escolha da página “certa” a ser retirada da memória principal para que outra seja carregada em seu lugar é importante. A página que está sendo retirada não deverá ser usada posteriormente, o que ocasionaria nova troca. As estratégias para retirar páginas de um processo da memória (realocação de páginas) são:

- ⇒ Aleatória: retira uma página aleatoriamente; algoritmo fácil → pouco overhead → pouca eficiência;
- ⇒ FIFO (*First In First Out*): a 1ª página referenciada é a 1ª página a sair; algoritmo simples utilizando fila → pouca eficiência;
- ⇒ LRU (*Least Recently Used*): retira a página menos recentemente utilizada, ou seja, a que está há mais tempo sem ser referenciada;
- ⇒ NRU (*Not Recently Used*): parecida com a anterior, um *flag* de referência indica se a página foi ou não utilizada. Inicialmente todas as páginas tem 0 (zero). Quando a página é utilizada, seu *flag*=1. De tempos em tempos, todos os *flags* são zerados. *Overhead* menor que a anterior, boa eficiência;
- ⇒ LFU (*Least Frequently Used*): retira a menos frequentemente utilizada. Utiliza um contador que recebe um acréscimo cada vez que a página for referenciada. Grande eficiência; *overhead* razoável.

Para qualquer estratégia: ANTES DE RETIRAR UMA PÁGINA DA MEMÓRIA é preciso verificar o seu bit de modificação. Se a página foi modificada tem que ser atualizada no disco.

O tamanho da página é determinado pelo SO de acordo com o hardware (geralmente entre 512bytes e 64Kb). Se a página for . . . então . . .

MUITO PEQUENA: tabelas de mapeamento muito grandes → grande *overhead* para percorrer tabela → muita paginação → menor fragmentação;

MUITO GRANDE: memória ocupada sem utilização → custo maior a cada swap in, swap out → maior fragmentação;

➤ Segmentação

Com o uso da paginação tentou-se deixar na memória apenas as páginas mais utilizadas de cada processo, ao invés de deixar residente o processo inteiro. Mas o sucesso dessa técnica vai depender da estrutura do programa. Uma outra idéia é dividir o programa pela sua estrutura e não em tamanhos fixos. Os blocos tem então tamanhos diferentes e são chamados segmentos. O endereço virtual é composto pelo número do segmento e o deslocamento dentro do segmento. O endereço físico é calculado a partir do endereço físico do segmento + o deslocamento dentro do segmento. E somente os segmentos referenciados são alocados na memória principal.

A eficiência desta técnica vai depender da modularização da aplicação: módulos grandes (cada módulo será

um segmento) nos quais somente uma parte do código é referenciada vai retornar ao problema de ocupação desnecessária na memória.

SEGMENTAÇÃO COM PAGINAÇÃO

Os segmentos são estabelecidos de acordo com a estrutura do programa e os segmentos são divididos em páginas. As páginas tem tamanho único no sistema. Os segmentos tem tamanho variável. O endereço virtual é formado assim:

nº do segmento	nº de página dentro do segmento	deslocamento dentro da página
----------------	---------------------------------	-------------------------------

Exercícios

- Qual a diferença da alocação particionada estática e dinâmica?
 - Como funciona a alocação dinâmica com relocação e quais suas desvantagens?
 - O que é e qual a importância da análise do *working set* dos programas?
 - Qual a relação entre *working set* e *trashing*?
 - O que é falha de página e quais as formas de se evitá-la?
 - Diferencie paginação de segmentação com relação aos seguintes aspectos:
 - lógica dos programas X divisão dos espaços de endereçamento
 - fragmentação (quando e como ocorre)
1. *Working Set* de um processo A tem seu tamanho limitado a quatro *frames* e encontra-se ocupado conforme quadro abaixo:

Página	Última referência	Flag de Referência	Entrada no sistema	Contador de referências
1	260 u.t.	1	200 u.t.	3
5	150 u.t.	0	150 u.t.	0
2	200 u.t.	1	100 u.t.	1
4	250 u.t.	1	180 u.t.	2

Tabela 1 – Frames do processo A

Às 300 u.t. ocorre uma falha de página referente à página 3, pertencente ao processo A. Responda qual página será escolhida para cada estratégia de relocação abaixo, **justificando sua resposta**:

- a) LRU b) FIFO c) NRU d) LFU

8. O *working set* de um processo possui quatro *frames*. O momento da carga (em segundos), o momento do último acesso (em segundos), o contador de referências e o *bit* de modificação (M) para cada uma das páginas na memória são mostrados abaixo:

PÁGINA	CARGA	ÚLTIMA REFERÊNCIA	CONTADOR	M
0	126	279	2	0
1	230	260	4	0
2	120	272	3	1
3	160	280	1	1

- Qual página será substituída pelo algoritmo FIFO? Justifique.
- Qual página será substituída pelo algoritmo LFU? Justifique.
- Qual página será substituída pelo algoritmo LRU? Justifique.

Outra visão de GERÊNCIA DE MEMÓRIA

No momento da execução o processador faz a busca das instruções e dos dados na memória principal. Além disso, no caso de escrita, o processador também o faz na memória principal. Sabemos que os sistemas de computação atuais utilizam a memória cache como uma memória intermediária entre a memória principal e o processador, com o objetivo de aumentar a velocidade de acesso aos dados e instruções. Neste capítulo vamos tratar da gerência de memória feita pelo sistema operacional, ou seja, vamos estudar como o sistema operacional trata de fazer com que o código e os dados de um processo estejam na memória principal no momento de sua execução da forma mais rápida e eficiente possível. Também estudaremos as técnicas utilizadas pelo sistema operacional para ter em memória principal os códigos e dados referentes ao maior número de processos possível, o que melhora a performance do sistema de computação.

Vamos dividir o nosso estudo em partes :

Parte I - Gerenciamento de Memória : Mecanismos fundamentais utilizados em gerenciamento de memória.

Parte II - Memória Virtual : Criação de um espaço virtual em disco => parte deste espaço é trazido para a memória real quando é necessário => maior número de processos compartilhando a memória real, sendo que apenas uma pequena parte (a mais necessária) pertencendo a cada processo.

Parte III - Gerenciamento de Memória no Windows NT : Utilizaremos as ferramentas do Windows NT para observarmos os aspectos desta gerência no Windows NT

PARTE I - GERENCIAMENTO DE MEMÓRIA

Em sistemas monotarefa a memória principal é dividida em duas partes : uma para o sistema operacional e a outra para a aplicação que está em execução.

----> (Prática + Gabriel Torres)

Nos sistemas multitarefa a parte relativa às aplicações deve ser "dividida" entre várias tarefas. **Por que ?**

R : Sabemos que nos sistemas multitarefa há várias tarefas ativas em um determinado instante. O sistema operacional utiliza o seu algoritmo de escalonamento para determinar que processo estará ativo em um determinado momento. Lembremos que nos **sistemas de tempo compartilhado**, por exemplo, o sistema operacional dedica uma **fatia de tempo** para a execução de cada processo, dando a impressão ao usuário de que várias tarefas estão em execução. Na verdade, sabemos que num sistema de computação com um único processador central, apenas um processo estará em execução de cada vez. Se a cada vez que o S.O. dedicar o tempo do processador para outro processo, o conjunto de instruções e dados deste processo tiver que ser buscado no disco, o tempo de troca passará a ser da ordem do tempo de acesso a disco, o que tornará o processador ocioso e o sistema como um todo extremamente lento, caindo por terra toda a vantagem da multitarefa.

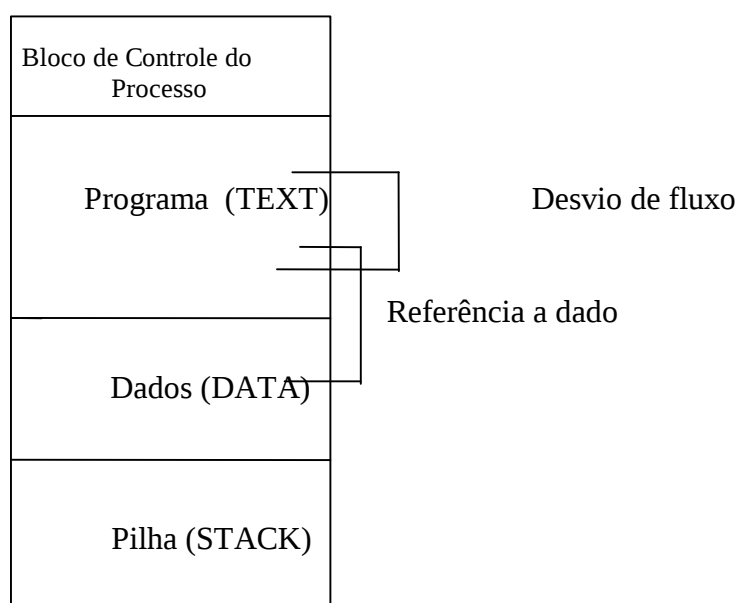
Esta "divisão" dinâmica da memória principal, é tarefa do sistema operacional e é conhecida como **gerenciamento de memória**.

I. 1 Requisitos necessários ao Gerenciamento de Memória

relocação compartilhamento organização física
proteção organização lógica

Relocação : Se o conjunto de código e dados de um processo passa para disco (**swap out**) enquanto o processo ainda está ativo, que região da memória física este conjunto irá ocupar quando retornar a esta (**swap in**) ? Se o código criado para cada processo for **absoluto**, todas as referências feitas serão aos endereços físicos nos quais as informações desejadas estão alocadas. Nesse caso, a alocação terá que ser sempre no mesmo espaço físico. É desejável que a alocação possa ser feita em qualquer lugar da memória. Esta questão levou à idéia de **endereçamento virtual** que veremos adiante.

Lembremos a **imagem de um processo** :



Proteção : Um processo não pode fazer referência para leitura ou escrita em outro processo, sem permissão. Assim todas as referências a endereços de memória devem ser verificados em tempo de execução.

Compartilhamento : O mecanismo de proteção deve ter flexibilidade para permitir que alguns processos possam acessar alguma porção da memória principal. Por exemplo, vários processos estão executando o mesmo programa. É desejável que estes processos possam acessar a mesma cópia do programa, ao invés de cada processo ter a sua cópia.

Organização lógica : A memória de um sistema e computação é normalmente organizada de forma linear, consistindo de uma sequência de bytes ou palavras. Os programas são organizados tipicamente em módulos, alguns dos quais são read only(somente para leitura), outros execute only (somente para execução) e outros contêm dados que podem ser modificados. Se o sistema de computação puder lidar com estes programas e dados de forma modular, teremos as vantagens :

- módulos podem ser escritos e compilados independentemente, com todas as referências de um módulo para outro resolvidas em tempo de execução
- diferentes graus de proteção (read only, execute only) podem ser dados a diferentes módulos

- é possível introduzir mecanismos para compartilhar módulos entre processos.

Veremos como a técnica de **segmentação** resolve essas questões.

Organização Física : Sabemos que num sistema de computação a memória é classificada hierárquicamente quanto à velocidade de acesso. É tarefa do S.O. mover dados entre memória secundária e memória principal.

I. 2 Particionamento da Memória

A principal operação do gerenciamento de memória é levar os programas para a memória principal para que possam ser executados. Os sistemas operacionais modernos trabalham com **memória virtual**, que envolve duas técnicas básicas : **paginação** e **segmentação**. Observemos abaixo um resumo das técnicas de partição da memória principal, inclusive as que não envolvem memória virtual.

Partições fixas

Memória principal dividida em um número fixo de partições de tamanho fixo, determinados quando da inicialização do sistema.

Vantagem : Simples implementação; pequeno overhead do sistema operacional.

Desvantagem : Uso ineficiente da memória, por causa da **fragmentação interna**, ou seja, áreas livres de memória dentro das partições que abrigam programas menores que o tamanho da partição na qual estão alocados.

Partições Dinâmicas

As partições são criadas de acordo com os tamanhos dos programas alocados.

Vantagem : nenhuma fragmentação interna

Desvantagem : grande overhead do sistema operacional, principalmente no que concerne à compactação necessária para contornar a **fragmentação externa**, ou seja, áreas de memória livres tão pequenas que não comportam nenhum programa.

Estratégias de alocação : **first-fit** : escolhe a primeira partição livre que comporte o programa
best-fit : escolhe a partição livre cujo tamanho seja o mais próximo possível do tamanho do programa.
worst-fit : escolhe a partição cujo tamanho seja o maior possível

Paginação Simples

Memória principal é dividida em um número fixo de quadros de tamanhos iguais, chamados **molduras**. Cada processo é dividido em **páginas** do mesmo tamanho da moldura da memória principal. Todas as páginas de cada processo são alocadas em molduras não necessariamente contíguas da memória principal.

Vantagem : Nenhuma fragmentação externa.

Desvantagem : Uma pequena fragmentação interna

Segmentação Simples

Cada processo é dividido logicamente em partes, de tamanho diferentes, chamados **segmentos**. Todos os segmentos de um processo são alocados em partições dinâmicas, não necessariamente contíguas.

Vantagem : nenhuma fragmentação interna

Desvantagem : Fragmentação externa: quando as áreas livres são tão pequenas que não comportam nenhum segmento. Overhead comparado ao da partição dinâmica.

Memória Virtual com Paginação

Mesmo da paginação simples, com a importante diferença que nem todas as páginas do processo precisam estar na memória principal : **as páginas dos processos são alocadas a medida que são referenciadas**. Veremos que há vários algoritmos para a retirada das páginas da memória principal, de forma a deixar nesta as mais necessárias.

Vantagem : nenhuma fragmentação externa; alto grau de multiprogramação (ou multitarefa) pois há um grande número de processos na memória principal, com um pequeno espaço desta destinado a cada um; grande espaço de endereçamento virtual para cada processo.

Desvantagem : pequena fragmentação interna; overhead devido ao complexo gerenciamento de memória

Memória Virtual com Segmentação

Mesmo da segmentação simples, com a importante diferença que nem todos os segmentos do processo precisam estar na memória principal : **os segmentos são alocados a medida que são referenciados**. Existem algoritmos para manter na memória os segmentos mais necessários.

Vantagem : Nenhuma fragmentação interna; alto grau de multiprogramação (ou multitarefa); grande espaço de endereçamento virtual para cada processo; suporte à proteção e compartilhamento.

Desvantagem : Fragmentação externa; overhead do sistema operacional devido ao complexo gerenciamento de memória, incluindo a compactação.

Memória Virtual com Segmentação com Paginação

Mesmo da memória virtual com segmentação, sendo que cada segmento é dividido em páginas. A memória física é dividida em molduras, que abrigam as páginas do processo.

Vantagem : nenhuma fragmentação externa; alto grau de multiprogramação (ou multitarefa); suporte à proteção e compartilhamento.

Desvantagem : pequena fragmentação interna; overhead devido ao complexo gerenciamento de memória.

I. 3 Endereçamento Físico versus Endereçamento Virtual

Endereço Físico : é o endereço da memória real. É único no sistema de computação, ou seja, cada célula da memória possui um e somente um endereço físico e a recíproca é verdadeira: cada endereço físico refere-se a uma e somente uma célula de memória.

Endereço lógico : é uma referência a um endereço independente da sua localização na memória real. Como vimos, na figura relativa à imagem de um processo, o código deste faz referência a outra parte do código quando há um desvio de fluxo. O código também faz referência aos dados. Estas referências são feitas através de endereços. Se o código criado for **código absoluto**, os endereços serão endereços físicos e este código só pzação na memória real. Como vimos, na figura relativa à imagem de um processo, o código deste faz referênci, **os endereços ser**, os endereços serão endereços físicos e este código só pzação na memória r, **os endereços serão**, os endereços serão endereços físicos e este código só pzação na memória real. Como vimos, na figura relativa à imagem de um processo, o código dest, **os endereços serão endereços físicos e este código só pzação na m**, os endereços serão endereços físicos e este código só pzação na memória real. Como vimos, na figura relativa à imagem de um processo, o código deste faz re(**registrador base ou registrador de relocação**) recebe o endereço físico do início do processo. Um outro registrador recebe o endereço final do processo.

As referências são feitas aos endereços lógicos.

Para ir buscar um dado ou instrução na memória principal o endereço lógico é levado a um hardware que consiste de uma unidade lógica e aritmética, do registrador de relocação e de um registrador que contém o último endereço dedicado ao processo. Nessa unidade é feita a soma do registrador base com o endereço lógico, obtendo assim o endereço físico. Antes de fazer a busca na memória principal é verificado se este endereço físico obtido encontra-se no intervalo dedicado ao processo, em caso negativo é feita uma chamada ao sistema operacional.

PARTE II - MEMÓRIA VIRTUAL

Memória virtual é uma técnica sofisticada de gerenciamento de memória, cujo objetivo é aumentar a memória principal, fazendo de uma parte do disco a continuação da memória, para alocação do processo inteiro.

61

Baseada no **Princípio da Localidade**, aloca na memória principal somente as páginas ou segmentos do processo que forem referenciadas. Veremos alguns algoritmos para a troca de páginas ou segmentos da memória para o disco, cujo objetivo é manter na memória principal as páginas ou segmentos mais utilizados.

II. 1 Princípios básicos :

Todas as referências de memória são feitas aos endereços lógicos, que são transformados nos endereços físicos em tempo de execução (ver **Mapeamento**).

Uma página ou segmento só é alocado na memória principal quando for referenciado (**swapping in**). Se não houver mais espaço na memória principal para alocar a página ou segmento desejado, alguma página ou segmento deverá ir para o disco (**swapping out**). Para escolher que página ou segmento deve ser retirado, dos **Algoritmos de Troca** deve ser utilizado. Esta idéia tem fundamento na observação do **Princípio da Localidade**.

O Princípio da Localidade :

Localidade é a tendência que os programas tem de fazer referências a posições de memória de forma quase uniforme, ou seja, a instruções próximas. Isso significa que um processo tenderá a concentrar suas referências em um mesmo conjunto de páginas durante determinado período de tempo. Naturalmente, a forma como a aplicação foi escrita pode aumentar ou diminuir a sua localidade.

Por exemplo um programa composto de 30 páginas de código e 5 páginas de dados, com um loop cujo código ocupe duas páginas e fazendo acesso a dados que ocupem duas páginas. Durante todo o tempo de execução deste loop, somente essas quatro páginas precisam estar na memória principal. Essas quatro páginas **não necessariamente** são alocadas em molduras contíguas da memória principal.

II . 2 SWAPPING

É uma técnica auxiliar no gerenciamento de memória virtual.

Todo o código e dados dos processos ficam alocados numa parte do disco, destinada especificamente para esta finalidade (**área de swap**).

- páginas ou segmentos que não foram referenciadas ficam em disco, sendo somente alocadas quando forem referenciadas (**swapping in**)
- páginas ou segmentos são retiradas da memória principal (**swapping out**), quando é necessário esvaziar uma moldura para alocar páginas ou segmentos do mesmo processo ou de outro processo. A escolha é feita pelos algoritmos de troca. São vários os motivos, inclusive, situações nas quais o processo ao qual estas páginas ou segmentos pertencem estejam em estado de espera.
- antes de uma página ou segmento ser retirado da memória é verificado se houve modificação. Se SIM a página ou segmento correspondente no disco deve ser ATUALIZADA.

II. 3 MAPEAMENTO

Já vimos que apenas algumas páginas ou segmentos de um processo podem estar alocadas na memória principal em um dado instante. E mais ! as páginas ou segmentos não precisam estar alocadas em regiões contíguas da memória. Como isso é possível ?

Chamamos de **mapeamento** ao mecanismo utilizado pelo sistema operacional para fazer a tradução do endereço localizado no espaço virtual para o endereço localizado no espaço físico.

Nota : Vamos fazer o estudo para paginação e depois faremos a extensão para segmentação e paginação com segmentação

Toda a imagem do processo é criada num **espaço de endereçamento virtual** : ou seja, todos os processos criados começam no endereço virtual zero. Se não houvesse a divisão do processo em páginas e consequentemente o processo fosse alocado inteiro na memória principal, para obter qualquer endereço físico, bastaria somar a cada endereço virtual o primeiro endereço físico do processo, como já vimos anteriormente. Mas, para utilizar a técnica da paginação, outro mecanismo terá que ser empregado.

Se imaginássemos que cada página do processo é um "pequeno processo", poderíamos utilizar a técnica acima para transformar qualquer endereço virtual dentro da página no seu correspondente endereço físico, bastando para isso somar o endereço virtual em questão ao primeiro endereço físico da página. Assim todas as páginas começariam no endereço virtual zero.

O mecanismo utilizado baseia-se nesta idéia :

- Cada processo é dividido em páginas.
- O endereço virtual é composto do pelo **número da página virtual (NPV)** e pelo *deslocamento* dentro da página .
- Cada processo possui uma tabela, a **tabela de páginas do processo**, contendo :
 - os números das suas páginas,
 - um **bit de validade**, que indica se a página está ou não na memória
 - o endereço físico do início da página
 - um **bit de modificação**, que indica se a página foi ou não modificada
- Cada página virtual possui uma única **entrada na tabela de páginas** (ETP) do processo

Quando um processo é colocado em execução, além dos registradores da CPU serem restaurados (caso o processo já tenha entrando antes em execução), passa a fazer parte da troca de contexto, a gravação da **tabela de páginas do processo**, num hardware específico, denominado **memória associativa ou translation lockside buffer (TLB)**.

Assim, quando a CPU faz referência a um endereço, o está fazendo a um endereço virtual e não físico. Antes de buscar este endereço na memória principal, este endereço virtual passa pela TLB para fazer a transformação :

- Como a primeira parte do endereço virtual é formada pelo NPV (número da página virtual), com este número, é feita a entrada na tabela de páginas (ETP)
- É verificado, através do bit de validade, se a página correspondente está na memória. Se não estiver, é gerada uma chamada ao sistema operacional, chamada **page fault (falha de página)**, para que este faça a alocação. O processo fica em estado de espera, até que a página seja alocada.
- Se a página está na memória é encontrado, o endereço físico da página.
- A segunda parte do endereço virtual é formada pelo deslocamento dentro da página. Somado-se este valor ao obtido no item anterior, obtém-se o endereço físico correspondente ao virtual referenciado.

Agora ficou possível entender porque as páginas do processo não precisam estar todas alocadas na memória principal ao mesmo tempo e, além disso, as que o estão, não necessariamente ocupam molduras contíguas.

II. 4 ALGORITMOS DE TROCA

Quando uma página faz-se necessária, mas não está alocada na memória principal, o sistema operacional deve alocá-la. Se houver alguma moldura livre na memória principal, basta alocar nesta a página desejada. Se não houver, o sistema operacional deverá **retirar alguma página da memória principal**, para alocar a página desejada. Mas, **que página o sistema operacional escolhe para retirar ?**

Há várias estratégias para retirar uma página da memória principal :

- **Aleatória** : Nenhum critério é utilizado, escolhe uma página aleatoriamente, para a retirada. Pouco overhead, pouca ineficiência, pois todas as páginas possuem a mesma probabilidade de serem retiradas, inclusive a que está sendo referenciada, ultimamente, com maior frequência.
- **First-in-First-out (FIFO)** : A página que primeiro foi referenciada (first-in), será a primeira a ser escolhida para sair (first-out). Pouco overhead, pois a implementação é simples : consiste apenas de uma fila. Pouca eficiência pois a primeira página a ser referenciada, pode estar sendo constantemente referenciada.
- **Least-Recently-Used (LRU)** : Seleciona a página utilizada menos recentemente, ou seja, a que está há mais tempo sem ser referenciada. Grande overhead, pois a cada referência, deve haver o registro do momento desta referencia. Além disso o algoritmo de busca também causa grande overhead.
- **Not-Recently-Used (NRU)** : Semelhante ao LRU, porém com outro tipo de implementação, cujo objetivo é diminuir o overhead : implementa-se um **flag de referência**, para indicar se a página foi referenciada ou não. Inicialmente todas as páginas estão com o flag com valor zero, indicando que Não foram referenciadas. À medida que as páginas são referenciadas, o flag associado a cada página é modificado para 1. Depois de um certo tempo é possível saber que páginas foram referenciadas ou não.
- **Least-Frequently-Used (LFU)** : Seleciona a página menos referenciada, ou seja, a que foi referenciada com menor frequência. Neste caso, implementa-se um contador que é incrementado a cada referência da página. Depois de um certo tempo estão na memória as páginas mais utilizadas. A única desvantagem é que as páginas que entrarem mais recentemente podem ser sempre escolhidas para serem retiradas; e pode acontecer que essas páginas sejam frequentemente utilizadas.

VALE PARA QUALQUER ESTRATÉGIA UTILIZADA

- Antes de retirar uma página da memória principal, o sistema operacional verifica, através do **bit de modificação (dirty bit ou modify bit)** se a página foi ou não modificada. Se SIM : deve ser feita uma cópia para o arquivo de paginação.
- Para retirar um processo da memória principal o estado do processo e a sua prioridade são observados.

II.5 WORKING SET

Chamamos de **taxa de paginação** ao número de páginas que tem que ser alocadas na memória, para um determinado processo, por tempo de observação. Quando um processo inicia a sua execução a taxa de paginação é alta e depois vai baixando tendendo para um baixo valor, que deve estabilizar-se, devido ao Princípio da Localidade e do algoritmo de troca, se for eficiente.

Dessa análise resulta o conceito de **working set**, que é o conjunto de páginas constantemente utilizadas pelo processo ao longo da sua vida ativa. Para melhor eficiência é desejável que estas páginas permaneçam na memória principal, a fim de evitar uma alta taxa de paginação.

Assim, passa-se a trabalhar com mais uma cota para cada processo : working set

Quanto maior o tamanho do working set, menor será a taxa de paginação. Entretanto, quanto maior o working set , menor será o número de processos que poderão compartilhar a memória principal. Para resolver este dilema, o sistema operacional pode implementar um tamanho de working set não fixo para cada processo. Pode comparar evolutivamente o tamanho do working set com a taxa de paginação e fixar, depois de algum tempo o tamanho do working set para determinado processo.

II. 6 CONSIDERAÇÕES SOBRE O TAMANHO DA PÁGINA

Páginas pequenas provocam menos fragmentação interna, entretanto resultam maiores taxas de paginação e maiores tabelas de páginas, sendo que estas últimas provocam maior overhead na troca de contexto.

Páginas grandes ocasionam menor compartilhamento da memória e maior fragmentação externa.

II. 7 EXTENSÃO PARA SEGMENTAÇÃO

Neste caso, os programas são divididos de acordo com a sua estrutura, em partes de tamanhos desiguais entre si. O **mapeamento** é semelhante ao da paginação : são criadas as **tabelas de mapeamento dos segmentos (TMS)** e os endereços são compostos pelo número do segmento e pelo deslocamento dentro do segmento. O número do segmento indica a **entrada na tabela e segmentos (ETS)**. O endereço físico de qualquer referência é calculado pela soma do endereço físico do segmento e do deslocamento dentro do segmento.

A tabela de mapeamento dos segmentos contém uma informação adicional : o tamanho do segmento.

A alocação é feita através de uma das estratégias utilizadas no caso de partições dinâmicas.

Somente os segmentos referenciados são alocados na memória principal.

Consequências : Com esta técnica a performance do sistema depende da modularização das aplicações. Se as aplicações não forem bem modularizadas haverá desperdício na utilização da memória principal, ou seja, teremos partes dos segmentos sem uso, ocupando a memória desnecessariamente.

Como no caso das partições dinâmicas, haverá fragmentação externa.

II. 8 EXTENSÃO PARA SEGMENTAÇÃO COM PAGINAÇÃO

Esta técnica faz a divisão lógica dos programas em segmentos. Além disso, cada segmento é dividido em páginas.

O endereço é formado pelo

número do segmento ==> entrada na tabela de segmentos ==> contém informações das páginas do segmento

65

número da página dentro do segmento ==> entrada na tabela de páginas => endereços físicos das páginas
deslocamento dentro da página ==> somado ao endereço físico da página => endereço físico

Consequências : não há fragmentação externa.

há fragmentação interna (normalmente bem menor que a externa)

II. 9 PROTEÇÃO

É preciso haver um mecanismo que proteja a área de memória de cada um dos vários processos que a compartilham e, sobretudo que proteja a área do sistema operacional.

Como descrito acima, a técnica de memória virtual implica num mapeamento dentro do processo : o endereço referenciado pela UCP é o endereço virtual. Através das entradas nas tabelas das páginas/segmentos os endereços virtuais são associados aos endereços físicos que pertencem aquele processo. Como cada processo tem a sua tabela de páginas/segmentos é impossível que um processo faça referência à área de memória de outro processo, a menos que haja **compartilhamento explícito** de páginas/segmentos entre processos, como veremos no próximo item.

A proteção nos sistemas que utilizam paginação/segmentação é feita a nível de página/segmento, através de bits que especificam os acessos permitidos nas tabelas de mapeamento. Se o bit de leitura (read) for 0, não será permitido o acesso para leitura; e o mesmo para escrita (write).

II. 10 COMPARTILHAMENTO EXPLÍCITO

Código reentrante : parte de código de um processo que não pode ser modificada e pode ser compartilhada por vários processos. Esse compartilhamento traz economia no uso da memória. Ex : utilitários e aplicativos do sistema.

Com o uso das tabelas de páginas/segmentos este compartilhamento é feito simplesmente através do endereço físico. Ex : uma página que pode ser compartilhada (acesso somente para leitura) pertencente ao processo A já se encontra na memória. Um outro processo B precisa alocar esta mesma página na memória. Não é necessário fazer a locação, mas somente escrever o endereço físico desta página já alocada na tabela de páginas do processo B.

[Índice](#)