**ARQUITETURA DE SISTEMAS OPERACIONAIS**

(5ª edicação – LTC)

Francis Berenger Machado

Luiz Paulo Maia

1. Visão Geral

Sistema Operacional pode ser definido como um conjunto de rotinas executadas pelo processador de maneira assíncrona, ou seja, de maneira concorrente, não executável como os softwares tradicionais com início, meio e fim. O Sistema Operacional é a interface entre o usuário e o computador.

FUNÇÕES BÁSICAS

\*Facilitar a interação e o acesso dos usuários aos diversos dispositivos conectados ao computador de maneira simples, segura e eficiente.

\*Permitir o compartilhamento dos recursos computacionais entre diversos usuários, controlando o uso concorrente destes recursos.

MAQUÍNA DE NÍVEIS

USUÁRIO

APLICAÇÕES

SISTEMA OPERACIONAL

Nível 1

HARDWARE

Nível 0

A aplicação do usuário interage diretamente com o SO como se o hardware não existisse - MÁQUINA VIRTUAL.

Atualmente, a maioria dos computadores possui diversos níveis e o programador pode interagir com cada um deles como se os outros acima e abaixo não existissem. A linguagem utilizada varia de linguagem de mais baixo nível às mais sofisticadas e parecidas com a linguagem humana (alto nível).

Estrutura multicamada básica:

Aplicações

Utilitários

Sistema Operacional

Linguagem de Máquina

Microprogramação

Circuitos Eletrônicos

# TIPOS DE SISTEMAS OPERACIONAIS

Os Sistemas Operacionais foram evoluindo de acordo com os avanços de tecnologia nos hardwares em conjunto com o avanço das aplicações por ele suportadas.

Existem três tipos principais de Sistemas Operacionais:

\*Sistemas Operacionais Monoprogramáveis (Monotarefa)

\*Sistemas Operacionais Multiprogramáveis (Multitarefas)

\*Sistemas Operacionais com Múltiplos Processadores

< Sistemas Operacionais Monoprogramáveis (Monotarefa) >

É um Sistema Operacional que permite a execução de um único programa por vez, qualquer outra aplicação para ser executada deverá esperar a finalização do programa que está em processamento; desta maneira neste tipo de sistema não há concorrência entre os programas. Todos os recursos do sistema ficam dedicados unicamente a uma tarefa.

Neste tipo de sistema enquanto algum dado está sendo inserido (entrada do sistema), o processador fica ocioso aguardando para realizar o processamento deste dado, o que é uma desvantagem já que poderia estar sendo utilizado para processar outra aplicação; além disso, a memória também fica subutilizada caso a aplicação em processamento não necessite de toda a sua capacidade. Por fim, os dispositivos de saída, como a impressora, são dedicados a um único usuário e nem sempre utilizados de forma completa.

<Sistemas Operacionais Multiprogramáveis (Multitarefas)>

Nestes sistemas os recursos computacionais são compartilhados entre diferentes aplicações concorrentes. Enquanto uma aplicação está sendo gravada no disco outra pode estar sendo processada - no mesmo período de tempo. O Sistema Operacional neste caso irá gerenciar a concorrência entre as aplicações para que o processo ocorra de maneira ordenada e segura.

A maior vantagem deste tipo de sistema operacional é a redução do tempo necessário para a execução das aplicações, além do óbvio de poder executar mais de uma tarefa ao mesmo tempo – otimização dos processos. Também possibilita uma redução de custos já que há o compartilhamento dos diversos recursos existentes.

Podem ser monousuários e multiusuários. Para este quando há a possibilidade de interação entre diversos usuários se conectarem ao mesmo tempo e simultaneamente utilizando conjuntamente os recursos computacionais; para aquele quando somente um usuário interage com o sistema e utiliza os recursos.

De acordo com o gerenciamento das aplicações pelo sistema operacional, os sistemas Multiprogramáveis podem ser classificados em:

\*Sistema Batch: Primeiros Sistemas Operacionais Multiprogramáveis caracterizados por não haver interação com o usuário. Este sistema é baseado na entrada de cartões perfurados nos quais estavam gravados os programas, chamados de Jobs; estes programas eram armazenados em fitas nos primeiros modelos e posteriormente em discos, os quais aguardavam para serem processados em função da disponibilidade de memória; por fim quando executados a saída também se dava através da impressão em disco ou fita. (processamento Spooling)

\*Sistema de Tempo Compartilhado (time-sharing): há uma divisão de tempo do processador em pequenos intervalos, chamada de time-slice (definidos pelo próprio Sistema Operacional), permitindo que diversos programas sejam executados compartilhando cada um desses intervalos. Se uma dessas fatias de tempo não é suficiente para executar um programa, ele é interrompido pelo sistema e substituído por outro programa, enquanto fica aguardando uma nova fatia de tempo. Esta interrupção é considerada um desvio da execução normal do programa, enquanto a execução deste programa para, outro é colocado no lugar, ou seja, só há uma mudança no estado do programa.

Este sistema permite uma comunicação direta entre o usuário e o Sistema Operacional.

\*Sistema de Tempo Real (real-time): utilizados em aplicações para as quais o tempo de processamento é fator crucial; neste tipo de Sistema Operacional não há a divisão em intervalos de tempo para cada um dos programas. Um único programa utiliza integralmente o processador pelo tempo que for necessário, tempo este definido pelo próprio programa, não pelo Sistema Operacional. Este programa só será interrompido ou finalizado quando este tempo definido acabar ou quando outra aplicação com maior prioridade precisar ser executada, tal prioridade também é definida pela própria aplicação.

<Sistemas Operacionais Multiprocessadores)>

Caracteriza-se pela presença de duas ou mais Unidades Centrais de Processamento ligadas entre si e trabalhando conjuntamente. Este tipo de Sistema Operacional permite que diversos programas sejam executados ao mesmo tempo ou que um mesmo programa seja dividido em partes para serem executados simultaneamente em mais de um processador.

Às características dos Sistemas Operacionais Multitarefas como a possibilidade de Concorrência e a Interrupção de programas, foram acrescentadas outras características vantajosas: escalabilidade, disponibilidade e balanceamento de carga. Escalabilidade se refere à capacidade de aumentar o poder de processamento apenas aumentando o número de processadores. Disponibilidade se refere à manutenção do sistema mesmo em caso de falha de um dos processadores que, poderá ser prontamente substituído por outro. Balanceamento de Carga é a distribuição do processamento entre os diversos processadores aumentando o desempenho geral.

Em função da forma de comunicação entre os diversos processadores e o grau de compartilhamento entre as memórias e os dispositivos de entrada e saída esse tipo de Sistema Operacional pode ser dividido em:  
\*Fortemente Acoplados (tightly-coupled): Comunicação entre as UCPs é feita através de barramento; os dispositivos de entrada e saída são gerenciados apenas por um sistema operacional.

\*Fracamente Acoplados (loosely coupled): possuem sistemas computacionais conectados por linhas de comunicação, formando uma rede. Cada sistema é independente, possuindo o próprio sistema operacional, o qual gerencia seus próprios recursos como a Unidade Central de Processamento, memória e dispositivos de entrada e saída; são conhecidos como Multicomputadores.

2. Conceitos de Hardware e Software

Sistema Computacional = conjunto de circuitos eletrônicos interligados. (dispositivos físicos que manipulam dados na forma digital). Todos os componentes de um sistema computacional são agrupados em três grandes grupos ou unidades funcionais:

* Unidade Central de Processamento;
* Memória Principal;
* Dispositivos de Entrada/Saída;

# PROCESSADOR

Gerencia todo o sistema computacional controlando e executando as operações de cada unidade funcional; sincronizando todas essas funções através de um sinal de *clock* (pulso elétrico gerado ciclicamente por um cristal de quartzo).

Unidade de Controle: responsável por gerenciar todas as atividades de todos os componentes do computador (exemplo: gravação de dados no disco e a busca de instruções na memória).

Unidade Lógica e Aritmética: responsável pela realização de operações lógicas e aritméticas.

Registradores: são dispositivos com a função principal de armazenar dados temporariamente. Funciona como uma memória interna de alta velocidade do processador, contudo com capacidade de armazenamento reduzida.

Registradores de Uso Geral: podem ser manipulados diretamente através de instruções.

Registradores de Uso Específico: armazenam dados de controle específico do processador e do sistema operacional. Como:

- contador de instruções (CI) / Program Counter (PC) = contém o endereço da próxima instrução a ser buscada e executada pelo processador. Toda vez que o processador buscar uma nova instrução, o contador é atualizado com o endereço de memória da próxima instrução.

-apontador de pilhas (AP) / Stack Pointer (SP) = contém o endereço de memória do topo da pilha (estrutura de dado), que é a estrutura em que o sistema armazena informações sobre programas que estão sendo executados.

- registrador de instruções (RI) = armazena a instrução que será decodificada e executada pelo processador.

- registrador de status / Program status word (PSW) = armazena informações sobre a execução de instruções; a maioria das instruções quando executadas, altera o registrador de status conforme o resultado.

Ciclo de busca de instruções:

1)Processador busca na memória principal a instrução armazenada no endereço indicado pelo CI e armazena no RI;

2)Processador incrementa o CI com o endereço de memória da próxima instrução a ser executada;

3)A informação armazenada no RI é decodificada;

4)Processador busca os operandos na memória, se houver;

5)A instrução decodificada é executada;

6)Ciclo reiniciado

# MEMÓRIA

Local onde são armazenados dados e instruções. Composta por unidades de acesso chamadas de células; sendo cada uma composta por um determinado número de bits. (BIT=unidade básica de memória, podendo assumir o valor de 0 ou 1).[1byte=8bits]. A memória é formada por um conjunto de células, onde cada célula possui um determinado número de bits.

O acesso ao conteúdo de uma célula é realizado através da especificação de um número chamado de endereço. O endereço é uma referência única que podemos fazer a uma célula de memória.

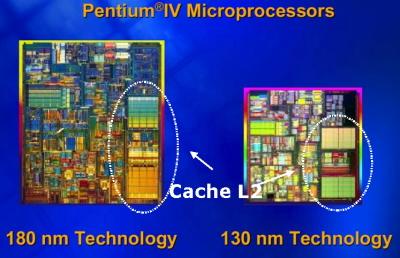
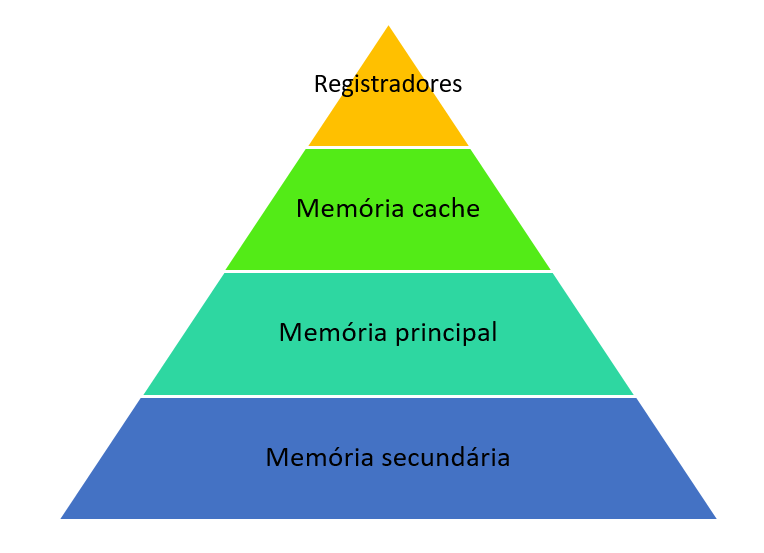
A especificação do endereço é realizada por um registrador de endereço denominado Memory Adress Register (MAR).

CLASSIFICAÇÃO: as memórias são classificadas de acordo com a sua volatilidade, ou seja, a capacidade de manter o armazenamento mesmo sem uma fonte de energia.

Memória Cache: é uma memória volátil de alta velocidade e pequena capacidade de armazenamento. Ela serve para minimizar a diferença entre a velocidade com que o processador executa instruções e a velocidade de leitura e gravação de dados na memória principal. O tempo de acesso a um dado contido na cache é muito menor do que o tempo de acesso a um dado na memória principal.

A memória cache armazena uma pequena quantidade de informações da memória principal, desta forma quando o processador requisitar leitura ou gravação ocorre uma verificação se o dado solicitado está na memória cache (cache hit), se sim o tempo de processamento será menor; caso a informação solicitada pelo processador não esteja na memória cache (cache miss) o acesso à memória principal é obrigatório.

Em caso de cache miss, o processador transfere um bloco de dados da memória principal para a memória cache, há neste processo um gasto de tempo adicional devido à transferência de dados entre as memórias, mas tal gasto é compensado pela melhora de desempenho. (Princípio da Localidade = é a tendência do processador, ao longo da execução de um programa, de referenciar dados/instruções da memória principal localizados em endereços próximos. Este princípio garante, então, que após a transferência de um novo bloco da memória principal para a cache haverá uma alta probabilidade de cache hits em futuras referências).



Maior custo e velocidade de acesso

Maior capacidade de armazenamento

Memória Secundária: é um meio permanente, isto é, não volátil de armazenamento de programas e dados. O acesso à memória secundária é lento, se comparado com o acesso à memória principal, porém seu custo é baixo e sua capacidade de armazenamento é bem superior. Enquanto a velocidade de acesso à memória principal é da ordem de nanossegundos, o acesso à memória secundária é de ordem de milissegundos.

-- Barramento/Bus – é um meio de comunicação entre as unidades funcionais do sistema computacional.

Barramento processador-memória – curta extensão e alta velocidade.

Barramento entrada/saída – maior extensão e mais lentos

Barramento backplane – existente em sistemas de alto desempenho. Nesta organização o barramento de E/S não se conecta diretamente ao barramento processador-memória, diminuindo a necessidade do número de adaptadores (para compatibilizar as diferentes velocidades de barramento).

-- Pipelining – é uma técnica que permite ao processador executar múltiplas instruções paralelamente em estágios diferentes. A execução de uma instrução pode ser dividida em subtarefas, como as fases de busca da instrução dos operandos, execução e armazenamento dos resultados. O processador através de suas várias unidades funcionais pipeline, funciona de forma a permitir que, enquanto uma instrução se encontra na fase de execução, uma oura instrução possa estar na fase de busca simultaneamente. (Pode ser empregado em sistemas com um ou mais processadores).

# SOFTWARE

-- Utilitário – softwares relacionados mais diretamente com serviços complementares do SO

-- Aplicações ou Aplicativos – softwares mais relacionados à utilização do usuário

Tradutor: As linguagens de alto nível não podem ser executadas diretamente pelo processador; para que isso ocorra é necessário que essas linguagens sejam convertidas em linguagem de máquina. Esta conversão é realizada pelo utilitário chamado de tradutor. O módulo gerado pelo tradutor é denominado de módulo-objeto, que apesar de estar em código de máquina, na maioria das vezes não pode ser ainda executado. Existem dois tipos de tradutores: montador (assembler) e compilador.

Compilador – é o utilitário responsável por gerar, a partir de um programa escrito em uma linguagem de alto nível, um programa em linguagem de máquina não executável.

Interpretador: é considerado um tradutor que não gera um módulo-objeto. A partir de um programa-fonte escrito em linguagem de alto nível, o interpretador, durante a execução do programa, traduz cada instrução e a executa imediatamente.

A maior desvantagem na utilização de interpretadores é o tempo gasto na tradução das instruções de um programa toda vez que este for executado, já que não existe a geração de um código executável. A vantagem é permitir a implementação de tipos de dados dinâmicos, ou seja, que podem mudar de tipo durante a execução do programa, aumentando, assim, sua flexibilidade.

COMPILADOR X INTERPRETADOR

Um compilador é um programa com o objetivo principal de traduzir todas as suas linhas de código para outra linguagem, normalmente uma de alto nível para uma de baixo nível.

Um interpretador também é um programa, mas, ao contrário do compilador, ele não converte o código todo para linguagem de máquina de uma vez. Ele executa diretamente cada instrução, passo a passo.

Enquanto um compilador analisa todo o código a fim de traduzi-lo de uma vez (muitas vezes, o resultado é um arquivo executável ou uma biblioteca), o interpretador faz esse trabalho de conversão aos poucos, sempre que uma declaração ou função é executada.

Uma vez que o código foi compilado, ele não precisará mais ser traduzido, a menos que o código seja alterado. Uma das grandes vantagens dos compiladores é sua velocidade de execução (não precisa fazer a conversão toda vez que o sistema é executado, provê uma eficiência muito maior do que um interpretador). A compilação costuma dar resultados mais confiáveis graças às suas diversas etapas de validação e otimização.

Contudo, uma linguagem compilada precisa fazer essa tradução para cada plataforma destinada, enquanto a interpretação, por poder rodar em tempo de execução, é independente, não importa se é Linux ou Mac.

Além disso, para linguagens interpretadas, a capacidade de execução em runtime permite utilizar reflexão (ou seja, examinar e modificar sua própria estrutura em tempo de execução) e tipagem dinâmica (capacidade de escolher dinamicamente o tipo de uma variável, não exigindo uma declaração). Verificar e modificar o código de uma linguagem interpretada também é mais fácil, já que basta abrir o arquivo e ver o que tem escrito. Para fazer o mesmo com uma linguagem compilada é necessário utilizar um descompilador. Isto, no entanto, pode ser visto como uma desvantagem, pois qualquer pessoa com um mínimo de conhecimento pode ver a implementação de um JS embutido numa página web ou até mesmo realizar uma injeção de código.

Compilação just-in-time é uma compilação feita em tempo de execução, ao invés de antes de rodar a aplicação (ahed-of-time).

Sistemas web frequentemente usam linguagens interpretadas para serem feitos, pela facilidade de manutenção e pela necessidade de portabilidade. Aplicações compiladas são bem mais seguras e rápidas, e são recomendadas para quem exige um desempenho considerável. Mas também trazem alguns incômodos, como ter que reinstalar o sistema quando uma versão mais atualizada fica disponível.

Linker / Editor de Ligação: 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. Outra função importante do linker é a relocação, que determina a região da memória na qual o programa será carregado para execução.

Loader / Carregador: é o utilitário responsável por carregar na memória principal um programa para ser executado. O procedimento de carga varia com o código gerado pelo linker e, em função deste, o loader é classificado como do tipo absoluto ou relocável.

Loader absoluto – quando o código realocado pelo linker estiver vinculado a um endereço fixo de memória (código absoluto), o loader só necessita saber o tamanho do módulo e a sua posição inicial transferindo o programa da memória secundária para a principal.

Loader relocável – para códigos realocados pelo linker do tipo realocável, ou seja, aquele que pode ser carregado a partir de diferentes locais da memória,

Depurador / Debugger: é o utilitário que permite ao usuário acompanhar toda a execução de um programa a fim de detectar erros na sua lógica.

3. Concorrência

Concorrência pode ser conceituada como a disputa pelos recursos computacionais por diferentes softwares.

Sistemas Operacionais podem ser vistos como um conjunto de rotinas executadas de forma concorrente e ordenada. O conceito de concorrência é o princípio básico para o projeto e a implementação dos sistemas multitarefas/multiprogramáveis.

A utilização concorrente da UCP deve ser implementada de maneira que, quando um programa perde o uso do processador e depois retorna para continuar o processamento, seu estado deve ser idêntico ao do momento em que foi interrompido. O programa deverá continuar sua execução exatamente na instrução seguinte àquela em que havia parado, aparentando ao usuário que nada aconteceu. Em sistemas de tempo compartilhado (time-sharing) existe a impressão de que o computador está inteiramente dedicado ao usuário, ficando esse mecanismo totalmente transparente.

# INTERRUPÇÕES E EXCEÇÕES

Desvios do fluxo normal de execução de um programa: estes tipos de eventos são chamados de Interrupções e Exceções e podem ser consequência de alguma interferência de hardware externo ao processador ou então da execução de instruções do próprio programa.

A interrupção é o mecanismo básico que tornou possível a implementação da concorrência nos computadores, sendo o fundamento básico dos sistemas Multiprogramáveis. Uma interrupção é gerada por algum evento externo ao programa e, nesse caso, independe da instrução que está sendo executada.

Ao final da execução de cada instrução, a unidade de controle verifica a ocorrência de algum tipo de interrupção. Nesse caso, o programa em execução é interrompido e o controle desviado para uma rotina responsável por tratar o evento ocorrido denominada rotina de tratamento de interrupção. Para que o programa possa posteriormente voltar a ser executado é necessário que, no momento da interrupção, um conjunto de informações sobre a sua execução seja preservado.

As interrupções são decorrentes de eventos assíncronos, ou seja, não relacionados à instrução do programa corrente. Esses eventos, por serem imprevisíveis, podem ocorrer múltiplas vezes, como no caso de diversos dispositivos de E/S informarem ao processador que estão prontos para receber ou transmitir dados.

As interrupções sempre são tratadas pelo SO.

Uma exceção é semelhante a uma interrupção, sendo a principal diferença o motivo pelo qual o evento é gerado. A exceção é resultado direto da execução de uma instrução do próprio programa, como a divisão de um número por 0 ou a ocorrência de overflow em uma operação aritmética.

Na maioria das vezes as exceções provocam um erro fatal no programa, ocasionando o seu fechamento (término anormal).

A exceção é um evento síncrono, ou seja, é resultado direto da execução do programa corrente. Tais eventos são previsíveis e, por definição só podem ocorrer um de cada vez.

Da mesma forma que na interrupção, sempre que uma exceção é gerada o programa em execução é interrompido e o controle é desviado para uma rotina de tratamento de exceção. Para cada tipo de exceção existe uma rotina de tratamento adequada, para a qual o fluxo de execução deve ser desviado. O mecanismo de tratamento de exceções, muitas vezes, pode ser escrito pelo próprio programador. Desta forma, é possível evitar que um programa seja encerrado no caso de ocorrer, por exemplo, overflow.

# BUFFERING

A técnica de buffering consiste na utilização de uma área na memória principal, denominada buffer, para a transferência de dados entre os dispositivos de E/S e a memória. Esta técnica permite que em uma operação de leitura, os dados sejam transferidos primeiramente para o buffer, liberando imediatamente o dispositivo de entrada para realizar uma nova leitura. Nesse caso, enquanto o processador manipula o dado localizado no buffer, o dispositivo realiza outra operação de leitura no mesmo instante. Este mecanismo também pode ser aplicado nas operações de gravação.

O buffer deve permitir armazenar diversos registros, de forma que existam dados lidos, mas ainda não processados (operação de leitura), ou processados, mas ainda não gravados (operação de gravação). Desta forma, o dispositivo de entrada poderá ler diversos registros antes que o processador manipule os dados, ou o processador poderá manipular diversos registros antes de o dispositivo de saída realizar a gravação. Isso é extremamente eficiente, pois, dessa maneira, é possível compatibilizar a diferença existente entre o tempo em que o processador executa instruções e o tempo em o dispositivo de E/S realiza as operações de leitura e gravação.

# SPOOLING

A técnica de spooling, semelhante à técnica de buffering, utiliza uma área em disco como se fosse um grande buffer. Neste caso, dados podem ser lidos ou gravados em discos enquanto programas são executados concorrentemente.

Atualmente esta técnica está presente na maioria dos SO, sendo utilizada no gerenciamento de impressão. No momento em que um comando de impressão é executado, as informações que serão impressas são gravadas antes em um arquivo em disco, conhecido como arquivo de spool, liberando imediatamente o programa para outras atividades. Posteriormente, o SO encarrega-se de direcionar o conteúdo do arquivo spoll para a impressora.

O uso do spooling permite desvincular o programa do dispositivo de impressão, impedindo que um programa reserve a impressora para uso exclusivo.

# REENTRÂNCIA

Reentrância é a capacidade de um código executável (código reentrante) ser compartilhado por diversos usuários, exigindo que apenas uma cópia do programa esteja na memória. A reentrância permite que cada usuário possa estar em um ponto diferente do código reentrante, manipulando dados próprios, exclusivos de casa usuário. É comum, em sistemas Multiprogramáveis, vários usuários utilizarem os mesmo aplicativos simultaneamente. Se para cada um desses usuários for necessário utilizar espaço da memória para a mesma aplicação (para armazenar o código executável) há um desperdício de espaço na memória que teria que abrir o mesmo código diversas vezes, por isso a reentrância é uma técnica de processamento que aperfeiçoa a utilização do espaço de memória.

4. Estrutura do Sistema Operacional

O Sistema Operacional é formado por um conjunto de rotinas (executadas de maneira concorrente e ordenada) que oferece serviços aos usuários e às suas aplicações. Esse conjunto de rotinas é denominado núcleo do sistema, ou Kernel. A maioria dos SO é fornecida acompanhada de utilitários e linguagem de comandos, que são ferramentas de apoio ao usuário, porém não são parte do núcleo do sistema. A linguagem de comandos é particular de casa sistema, com estrutura e sintaxe próprias.

# FUNÇÕES DO KERNEL

Diferentemente de uma aplicação convencional, com início, meio e fim, as rotinas do sistema são executadas concorrentemente sem uma ordem predefinida, com base em eventos dissociados do tempo (eventos assícronos). Muitos desses eventos estão relacionados ao hardware e a tarefas internas do próprio sistema operacional.

\*Tratamento de interrupções e exceções;

\*Criação e eliminação de processos e threads;

\*Sincronização e comunicação entre processos e threads;

\*Escalonamento e controle dos processos e threads;

\*Gerência de memória;

\*Gerência do sistema de arquivos;

\*Gerência de dispositivos de E/S;

\*Suporte a redes locais e distribuídas;

\*Contabilização do uso do sistema;

\*Auditoria e segurança do sistema;

Em sistemas multiprogramáveis com mais de um usuário, como vários programas ocupam a memória simultaneamente, casa usuário deve possuir uma área reservada onde seus dados e códigos são armazenados. O sistema operacional implementa mecanismos de proteção, de forma a preservar estas informações de maneira reservada. Caso um programa tente acessar uma área de memória fora de sua área ocorrerá um erro indicando violação de acesso. Para que diferentes programas tenham o direito de compartilhar uma mesma área de memória, o sistema operacional deve oferecer mecanismos para que a comunicação seja feita de forma sincronizada e controlada, evitando, desta forma, problemas de inconsistência.

De modo semelhante ao compartilhamento de memória, um disco pode armazenar arquivos de diferentes usuários. Mais uma vez o sistema operacional deve garantir a integridade e a confidencialidade dos dados, permitindo ainda que dois ou mais usuários possam ter acesso simultâneo ao mesmo arquivo.

# MODO DE ACESSO

Caso uma aplicação, que tenha acesso ao núcleo, realize uma operação que altere sua integridade, todo o sistema poderá ficar comprometido e inoperante. Muitas das principais implementações de segurança de um SO utilizam um mecanismo presente no hardware dos processadores conhecido como modo de acesso.

Em geral, os processadores possuem dois modos de acesso: modo usuário e modo Kernel. Quando o processador trabalha no modo usuário, uma aplicação só pode executar instruções conhecidas como não privilegiadas, tendo acesso a um número reduzido de instruções do processador. O modo de acesso é determinado por um conjunto de bits, localizado no registrador de status do processador, que indica o modo de acesso corrente. Por intermédio desse registrador, o hardware verifica se a instrução pode ou não ser executada.

Em relação ao disco, por ser um recurso compartilhado, sua utilização deverá ser gerenciada unicamente pelo sistema operacional, evitando que a aplicação possa ter acesso a qualquer área do disco sem autorização, o que poderia comprometer a segurança e integridade do sistema de arquivos.

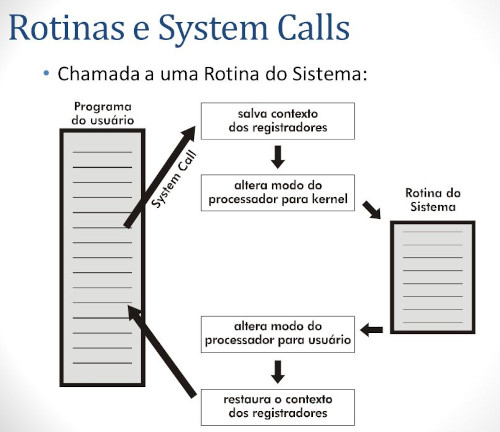
Outro exemplo do uso dos modos de acesso é a proteção do próprio núcleo do sistema residente na memória principal. Caso uma aplicação tenha acesso a áreas de memória onde está carregado o SO, um programador mal-intencionado ou um erro de programação poderia gravar nesta área, violando o sistema. Com o mecanismo de modo de acesso, para uma aplicação escrever numa área onde resida o SO o programa deve estar sendo executado no modo Kernel.

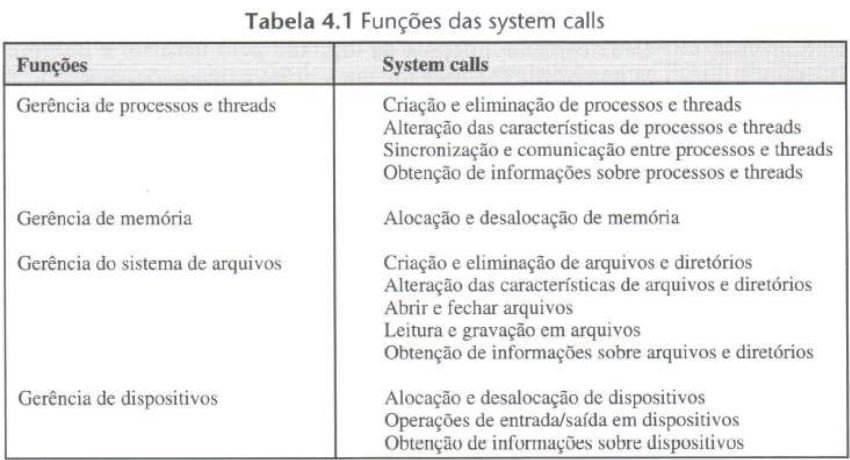
# ROTINAS DO SO E SYSTEM CALLS

Todas as funções do núcleo são implementadas por rotinas do sistema que necessariamente possuem em seu código instruções privilegiadas. A partir desta condição, para que estas rotinas possam ser executadas o processador deve estar obrigatoriamente em modo de acesso Kernel, o que exige a implementação de mecanismos de proteção para garantir a confiabilidade do sistema.

Todo controle de execução de rotinas do SO é realizado pelo mecanismo conhecido como System Call. Toda vez que uma aplicação desejar chamar uma rotina do SO, este mecanismo é acionado. Inicialmente, o SO verificará se a aplicação possui privilégios necessários para executar a rotina desejada. Em caso negativo, o SO impedirá o desvio para a rotina do sistema, sinalizando que a operação não é possível – mecanismo de proteção por software.

Considerando que a aplicação possua o devido privilégio para chamar a rotina desejada, o SO primeiramente salva o conteúdo corrente dos registradores, troca o modo de acesso do processador de usuário para Kernel e realiza o desvio para a rotina alterando o registrador PC com o endereço da rotina chamada. Ao término da execução da rotina do sistema, o modo de acesso é alterado de Kernel para usuário e o contexto dos registradores restaurados para que a aplicação continue a execução a partir da instrução que chamou a rotina do sistema.





Uma aplicação sempre deve executar com o processador no modo usuário. Numa situação em que a aplicação tenta executar uma instrução privilegiada em modo usuário e sem a supervisão do SO, o próprio hardware do processador sinalizará um erro. Uma exceção é gerada e a execução do programa é interrompida, protegendo desta forma o núcleo do sistema.

Os mecanismos de System Call e proteção de hardware garantem a segurança e a integridade do sistema.

As rotinas e o mecanismo de System Call podem ser entendidos como uma porta de entrada para o núcleo do SO e a seus serviços. Sempre que uma aplicação desejar algum serviço do sistema, deve ser realizada uma chamada a uma de suas rotinas através de uma System Call.

O termo System Call é tipicamente utilizado em sistemas Unix, porém em outros sistemas o mesmo conceito é apresentado com diferentes nomes, como System Services, no OpenVMS, e Application Program Interface (API) no MS Windows.

Cada SO possui seu próprio conjunto de rotinas, com nomes, parâmetros e formas de ativação específicos. Consequentemente, uma aplicação desenvolvida utilizando serviços de um determinado SO não pode ser portada diretamente para outro sistema, exigindo algumas correções no código-fonte.

# LINGUAGEM DE COMANDO

A linguagem de comando, ou linguagem de controle, permite que o usuário se comunique de uma forma simples com o SO, capacitando-o a executar diversas tarefas específicas do sistema como criar, ler ou eliminar arquivos, consultar diretórios ou verificar a data e a hora armazenadas no sistema. Dessa forma, o usuário dispõe de uma interface direta com o SO.

Na maioria dos SO, as linguagens de comando evoluíram no sentido de permitir a interação mais amigável com os usuários, utilizando interfaces gráficas como janelas e ícones, a exemplo dos sistemas MS Windows. Na maioria dos casos, a interface gráfica é apenas mais um nível de abstração entre o usuário e os serviços do SO.

# BOOT – ATIVAÇÃO E DESATIVAÇÃO DO SISTEMA

Incialmente, quando um computador é ligado não há SO carregado na memória principal. Em geral, o SO reside em um HD, podendo também estar armazenado em outras memórias secundárias como CDs e Pen-drivers. Toda vez que o computador é ligado os arquivos do SO devem ser carregados para a memória principal, isto ocorre por intermédio de um procedimento chamado de ativação do sistema ou boot.

O procedimento de ativação se inicia com a execução de um programa chamado boot loader. Este programa chama a execução de outro programa conhecido como POST (Power-On Self Test), que identifica possíveis problemas de hardware no equipamento. Após esta fase, o procedimento de ativação verifica se há no sistema computacional algum dispositivo de armazenamento onde haja um SO. Caso nenhum dispositivo seja encontrado, uma mensagem de erro é apresentada e o procedimento de ativação é interrompido. Se o SO é encontrado, um conjunto de instruções é carregado para memória e localizado em um bloco específico do dispositivo conhecido como setor de boot (boot setor). A partir da execução deste código, o SO é finalmente carregado para a memória principal.

Na maioria dos sistemas também existe o processo de shutdown. Este procedimento permite que as aplicações e componentes do SO sejam desativados ordenadamente, garantindo, desta forma, sua integridade.

# ARQUITETURAS DO NÚCLEO

<Arquitetura Monolítica>

Formada por vários módulos que são compilados separadamente e depois linkados, formando um grande e único programa executável.

Os primeiros SO foram desenvolvidos com base neste modelo, o que tornava seu desenvolvimento e, principalmente, sua manutenção bastante difíceis.

<Arquitetura de Camadas>

O sistema é dividido em níveis sobrepostos. Cada camada oferece um conjunto de funções que podem ser utilizadas apenas pelas camadas superiores. Neste tipo de implementação, as camadas mais internas são mais privilegiadas que as mais externas.

A vantagem da estruturação em camadas é isolar as funções do SO, facilitando sua manutenção e depuração, além de criar uma hierarquia de níveis de modos de acesso, protegendo as camadas mais internas. Uma desvantagem para o modelo de camadas é o desempenho. Cada nova camada implica uma mudança no modo de acesso.

Atualmente, a maioria dos sistemas comerciais utiliza o modelo de duas camadas, onde existem os modos de acesso usuário (não privilegiado) e Kernel (privilegiado). A maioria das versões do Unix e Windows está baseada neste modelo.

<Máquina Virtual>

Um sistema computacional é formado por níveis, onde a camada de nível mais baixo é o hardware. Acima desta camada encontramos o SO que oferece suporte para as aplicações. O modelo de Máquina Virutal (Virtual Machine), cria um nível intermediário entre o hardware e o SO, denominado gerência de máquinas virtuais. Este nível cria diversas máquinas virtuais independentes, onde cada uma oferece uma cópia virtual do hardware, incluindo os modos de acesso, interrupções, dispositivos de E/S, etc.

Esse modelo cria o isolamento total entre cada VM, oferecendo grande segurança para cada máquina virtual. Se, por exemplo, uma VM executar uma aplicação que comprometa o funcionamento do seu SO, as demais máquinas virtuais não sofrerão qualquer problema.

<Máquina Microkernel>

Uma tendência nos SO modernos é tornar o Kernel menor e mais simples possível. Para implementar esta ideia, os serviços do sistema são disponibilizados através de processos, onde cada um é responsável por oferecer um conjunto específico de funções, como gerência de arquivos, gerência de processos, gerência de memória e escalonamento.

Sempre que uma aplicação deseja algum serviço, é realizada uma solicitação ao processo responsável. Neste caso, a aplicação que solicita o serviço é chamada de cliente, enquanto o processo que responde é chamado de servidor. Um cliente, que pode ser uma aplicação de um usuário ou outro componente do SO, solicita um serviço enviando uma mensagem para o servidor. O servidor responde ao cliente através de outra mensagem. A principal função do núcleo é realizar a comunicação, ou seja, a troca de mensagens entre cliente e servidor.

Apenas o núcleo do sistema, responsável pela comunicação entre clientes e servidores, executa no modo Kernel. Como consequência, se ocorrer um erro em um servidor, este poderá parar, mas o sistema não ficará inteiramente comprometido, aumentando assim a sua disponibilidade.

5. Processo

A gerência de processos é uma das principais funções de um SO. Para isso, um programa ao ser executado deve estar sempre associado a um processo. O conceito de processo é a base para a implementação de um sistema Multiprogramável.

A gerência de processos é uma das principais funções de um SO, possibilitando aos programas alocar recursos, compartilhar dados, trocar informações e sincronizar suas execuções.

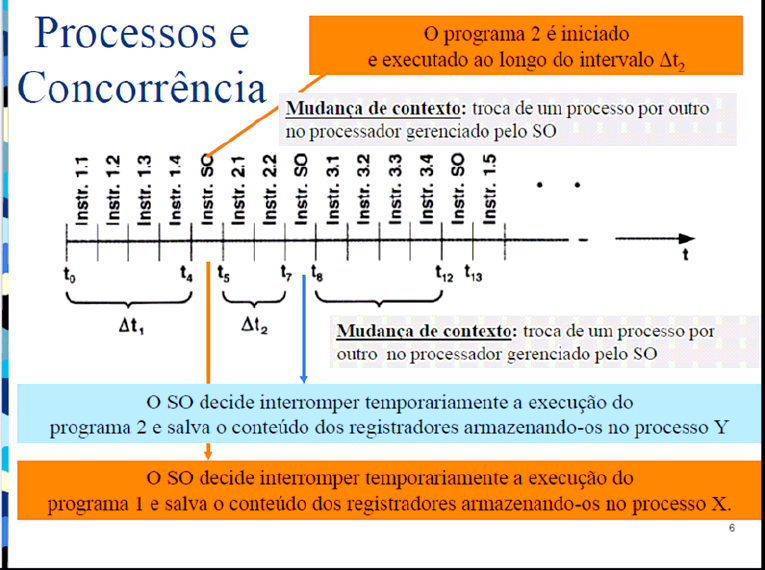
Nos sistemas multiprocessadores não só existe a concorrência de processos pelo uso do processador como também a possibilidade de execução simultânea de processos nos diferentes processadores.

# ESTRUTURA DO PROCESSO

Recapitulação: o processador é projetado para executar instruções a partir do ciclo de busca e execução. Neste ciclo, o processador busca a instrução na memória principal, armazena-a no registrador de instruções, para finalmente decodifica-la e executa-la. O registrador PC tem a função de armazenar o endereço da próxima instrução a ser executada, e as alterações do seu conteúdo determinam o sequenciamento de execução das instruções armazenadas na memória principal. Na visão da camada de hardware, o processador executa instruções sem distinguir qual programa encontra-se em processamento. Desta forma, é de responsabilidade do SO, através da implementação do mecanismo de concorrência, gerenciar a alternância da execução de instruções pelo processador.

Um processo pode ser definido incialmente como um programa em execução pelo SO. Para que a concorrência ocorra é necessário que todas as informações do programa interrompido sejam guardadas para que, quando voltar a ser executado, não lhe falte nenhuma informação necessária a continuação do processamento. Estas informações são fundamentais para que o SO possa gerenciar a execução concorrente de programas. Desta forma, o conceito de processo pode ser definido como sendo o conjunto necessário de informações para que o SO implemente a concorrência de programas.

A troca de um processo por outro no processador, comandada pelo SO, é denominada mudança de contexto. É desta maneira que o SO implementa e gerencia um ambiente Multiprogramável.

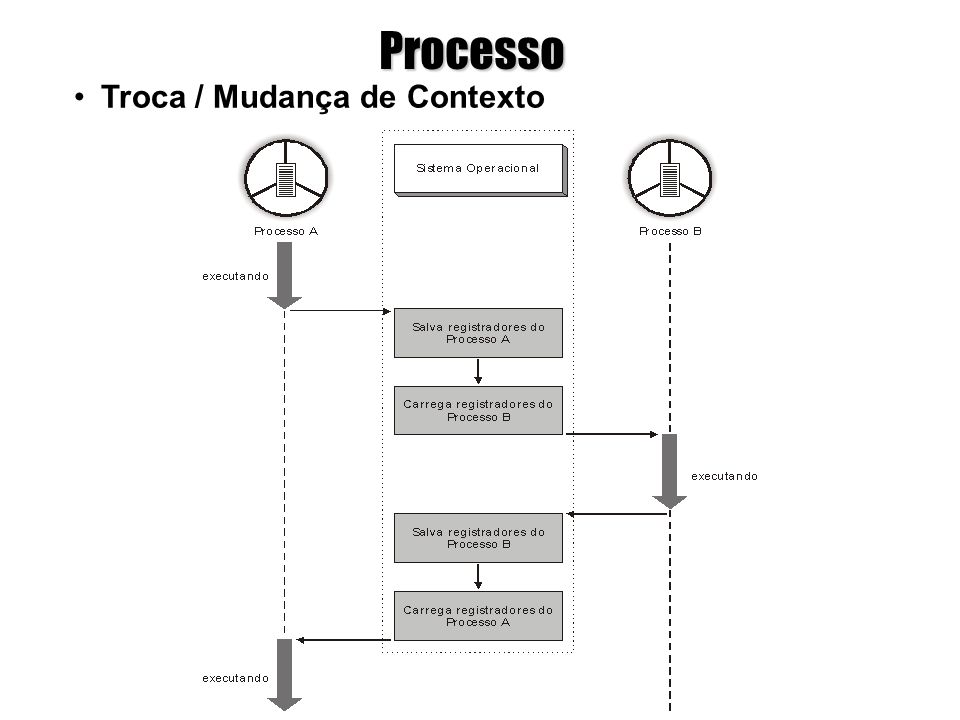


Um processo é formado por três partes, conhecidas como Contexto de Hardware, Contexto de Software e Espaço de Endereçamento, que juntos mantêm todas as informações necessárias à execução de um programa.

<Contexto de Hardware>

Neste contexto são armazenados o conteúdo dos registradores gerais e também dos registradores específicos como PC, SP e registrador de status. Quando um processo está em execução, o seu contexto de hardware está armazenado nos registradores do processador. Quando ele é interrompido - perde a utilização da UCP, o sistema salva as informações no contexto de hardware do processo.

Na concorrência há a substituição de um contexto de hardware de um processo por outro de um processo concorrente.



<Contexto de Software>

No contexto de software de um processo são especificados limites e características dos recursos que podem ser alocados pelo processo, como o número máximo de arquivos abertos simultaneamente, prioridade de execução e tamanho do buffer para operações de E/S. Muitas destas características são determinadas no momento da criação do processo, enquanto outras podem ser alteradas durante sua existência.

-- Identificação – cada processo criado pelo sistema recebe uma identificação numérica única (PID - process identification). Através da PID, o SO e outros processos podem fazer referência a qualquer processo existente, consultando seu contexto ou alterando uma de suas características. O processo também possui a identificação do usuário ou processo que o criou (owner). Cada usuário possui uma identificação única no sistema (UID – user identification), atribuída ao processo no momento da criação. A UID permite implementar um modelo de segurança, onde apenas os objetos (processos, arquivos, áreas de memórias, etc) que possuem a mesma UID podem ser acessados.

-- Quotas – são os limites de casa recurso do sistema que um processo pode utilizar. São exemplos: número de arquivos que podem ser abertos pelo processo simultaneamente, tamanho máximo de memória principal e secundária que o processo pode utilizar, número máximo de operações de E/S, tamanho do buffer para operações de E/S e número máximo de processos, subprocessos e threads que podem ser criados.

-- Privilégios – definem as ações que um processo pode fazer em relação a ele mesmo, aos demais processos e ao SO.

<Espaço de Endereçamento>

O espaço de endereçamento é área da memória pertencente ao processo onde instruções e dados do programa são armazenados para execução. Cada processo possui seu próprio espaço de endereçamento, que deve ser devidamente protegido do acesso dos demais processos.

# BLOCO DE CONTROLE DO PROCESSO

O processo é implementado pelo SO através de uma estrutura de dados chamada bloco de controle do processo ( process control block – PCB). A partir do PCB, o SO mantém todas as informações sobre o contexto de hardware, contexto de software e espaço de endereçamento de cada processo.

Os PCBs de todos os processos ativos residem na memória principal em uma área exclusiva do SO.

# ESTADOS DOS PROCESSOS

Os processos passam por diferentes estados ao longo do seu processamento, em função de eventos gerados pelo SO ou pelo próprio processo. Um processo ativo pode encontrar-se em três diferentes estados:

<Estado de Execução - Running>

Um processo está neste estado quando está sendo executado pelo processador. Em sistema Multiprogramáveis somente um processo pode estar neste estado por vez. Em sistema multiprocessadores existe a possibilidade de mais de um processo ser executado ao mesmo tempo (balanceamento de carga).

<Estado de Pronto - Ready>

Um processo está neste estado quando aguarda apenas para ser executado. O SO é responsável por determinar a ordem e os critérios pelos quais os processos em estado de pronto devem fazer uso do processador (escalonamento). Em geral existem vários processos neste estado, organizados em listas encadeadas por ordem de prioridade.

<Estado de Espera - Wait> (em alguns SO este estado por ser chamado de Bloqueado -Blocked)

Um processo neste estado está aguardando por algum evento externo ou por algum recurso para prosseguir seu processamento.

# MUDANÇAS DE ESTADO DO PROCESSO

Um processo muda de estado durante seu processamento em função de eventos originados por ele próprio (eventos voluntários) ou pelo SO (eventos involuntários).

<PRONTO 🡪 EXECUÇÃO>

Após a criação de um processo, o sistema o coloca em uma lista de processos no estado de pronto, onde aguarda por uma oportunidade para ser executado (Política de Escalonamento).

<EXECUÇÃO 🡪 ESPERA>

Um processo em execução passa para o estado de espera por eventos gerados pelo próprio processo, como uma operação de E/S, ou por eventos externos, como por exemplo, quando o SO suspende por um período de tempo a execução do programa.

<ESPERA 🡪 PRONTO>

Essa mudança de estado ocorre quando a operação solicitada é atendida ou o recurso esperado é concedido. Um processo no estado de espera sempre terá que passar pelo estado de pronto antes de ir para execução. Não existe a mudança direta de estado de espera para execução.

<EXECUÇÃO 🡪 PRONTO>

Um processo em execução passa para o estado de pronto por eventos gerados pelo sistema, como o término da fatia de tempo que o processo possui para sua execução.

Um processo em estado de espera ou de pronto pode não se encontrar na memória principal. Esta condição ocorre quando não existe espaço suficiente para todos os processos na memória principal e parte do contexto do processo é levado para a memória secundária. A técnica conhecida como SWAPPING, na condição citada, retira processos da memória principal (SWAP OUT) e os traz de volta (SWAP IN) seguindo critérios de cada SO.

Além dos estados de execução, pronto e espera a maioria dos SO estabelece mais dois estados:

<ESTADO DE CRIAÇÃO>

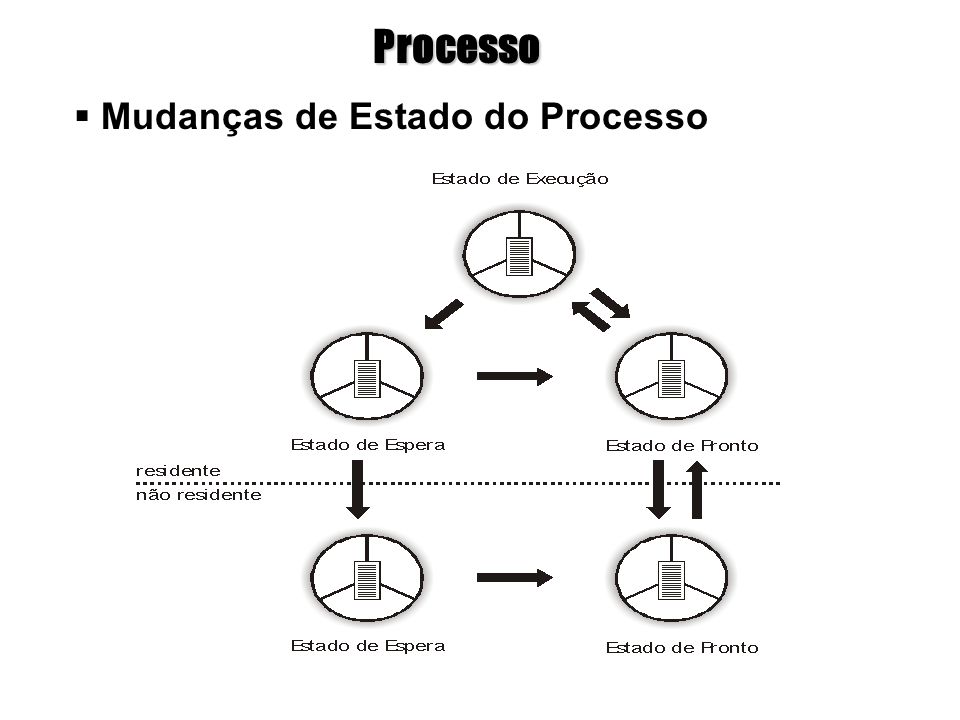
Ocorre quando o SO criou um novo PCB (contexto de hardware, contexto de software e memória de endereçamento), porém ainda não pode coloca-lo na lista de processos do estado de pronto.

<ESTADO DE TERMINADO>

Um processo no estado de terminado não poderá ter mais nenhum programa executado no seu contexto, porém o SO ainda mantém suas informações de controle presentes em memória. Um processo neste estado não é considerado mais ativo, mas como o PCB ainda existe, o SO pode recuperar informações sobre a contabilização de uso de recursos do processo.

No caso de eliminação de um processo, todos os recursos associados ao processo são desalocados e o PCB eliminado pelo SO.

O término de um processo pode ocorrer, por exemplo, devido ao término normal de execução, a eliminação por outro processo ou a eliminação forçada por ausência de recursos.



# PROCESSOS CPU-BOUND E I/O-BOUND

Essas classificações são feitas de acordo com a utilização do processador e dos dispositivos de E/S.

Um processo CPU-bound é aquele que passa a maior parte do tempo em estado de execução ou pronto. Utiliza-se muito o processador com poucas interações com dispositivos de E/S (leitura e gravação). É encontrado em aplicações científicas que efetuam muitos cálculos.

Um processo I/O-bound é aquele que passa a maior parte do tempo em estado de espera, pois realiza muitas operações com dispositivos de E/S (leitura e gravação de dados). Esse tipo de processo é encontrado, por exemplo, em aplicações comerciais, assim como em processos interativos.

# PROCESSOS FOREGROUND E BACKGROUND

Um processo foreground é aquele que permite a comunicação direta do usuário com o processo durante o seu processamento. Neste caso, tanto o canal de entrada como o de saída estão associados a um terminal com teclado, mouse e monitor, permitindo a interação.

Um processo background é aquele onde não existe a comunicação com o usuário durante o processamento. Neste caso, os canais de E/S não estão associados a nenhum dispositivo de E/S interativo, mas em geral a arquivos de E/S.

# FORMAS DE CRIAÇÃO DE PROCESSOS

<Logon Interativo>

O usuário por intermédio de um terminal fornece ao sistema um nome de identificação (username ou logon) e uma senha (passaword). O SO autentica estas informações verificando se estão corretamente cadastradas no arquivo de usuários. Em caso positivo, um processo foreground é criado.

<Linguagem de Comandos>

Um usuário pode, a partir do seu processo, criar novos processos por intermédio de comando. O principal objetivo para que um usuário crie diversos processos é a possibilidade de execução de programas concorrentemente. Esta é uma maneira de criar um processo a partir de outro existente.

<Rotina do SO>

Um processo pode ser criado a partir de qualquer programa executável com o uso de rotinas do SO. A criação deste processo possibilita a execução de outros programas concorrentemente ao programa chamador. A rotina de criação de processos depende do SO e possui diversos parâmetros, como nome do processo a ser criado, nome do programa executável que será executado dentro do contexto do processo, prioridade de execução, estado do processo, se o processo é do tipo foreground ou background, etc.

# PROCESSOS INDEPENDETES, SUBPROCESSOS E THREADS

Processos independentes, subprocessos e threads são formas de implementar concorrência dentro de uma aplicação.

O uso de processos independentes é a maneira mais simples. Neste caso, não existe vínculo do processo criado com seu criador. A criação de um processo independente exige alocação de um PCB, possuindo contextos de hardware, contextos de softwares e endereçamentos de memória próprios.

Subprocessos são processos criados dentro de uma estrutura hierárquica. Neste modo, o processo criador é denominado processo-pai, enquanto o novo processo é chamado de subprocesso ou processo-filho. O subprocesso, por sua vez, pode criar outras estruturas de subprocessos. Uma característica desta implementação é a dependência existente entre o processo criador e os subprocessos. De modo semelhante aos processos independentes, subprocessos possuem seu próprio PCB.

Além da dependência hierárquica entre processos e subprocessos, uma outra característica neste tipo de implementação é que subprocessos podem compartilhar quotas com o processo-pai. Neste caso, quando um subprocesso é criado as quotas do processo-pai são divididas com o processo-filho.

O uso de processos independentes e subprocessos no desenvolvimento de aplicações concorrentes demanda consumo de diversos recursos do sistema devido à criação de inúmeros PCBs. Sempre que um processo é criado o sistema deve alocar recursos, consumindo tempo de UCP. No momento do término dos processos, o SO também dispensa tempo para desalocar recursos previamente alocados.

O conceito de Thread foi introduzido na tentativa de reduzir o tempo gasto em criação, eliminação e troca de contexto de processos nas aplicações concorrentes, bem como economizar recursos do sistema como um todo. Threads compartilham o processador da mesma maneira que um processo, ou seja, enquanto um thread espera por uma operação de E/S, outro thread pode ser executado.

Cada Thread possui seu próprio contexto de hardware, porém compartilha o mesmo contexto de software e espaço de endereçamento com os demais threads do processo.