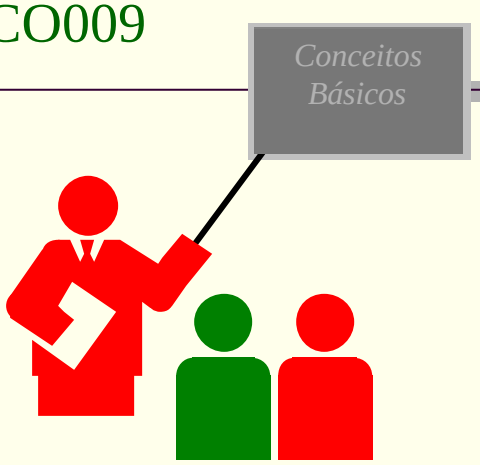
Modulo I
Introdução aos Sistemas Operacionais

UniverCidade - Prof. Ismael H F Santos

Ementa

- **Introdução aos Sistemas Operacionais**
  - *Conceitos Básicos*
  - *Ambientes Computação*
  - *Histórico*
  - *Classificação*
  - *SOs Multiprogramáveis*
    - *Interrupção x Exceção*
    - *Sistemas Batch Mono e Multitarefa*
  - *Estrutura do SO*
    - *System Calls*
    - *Kernel*
    - *Subsistemas do Kernel*

SOP – CO009



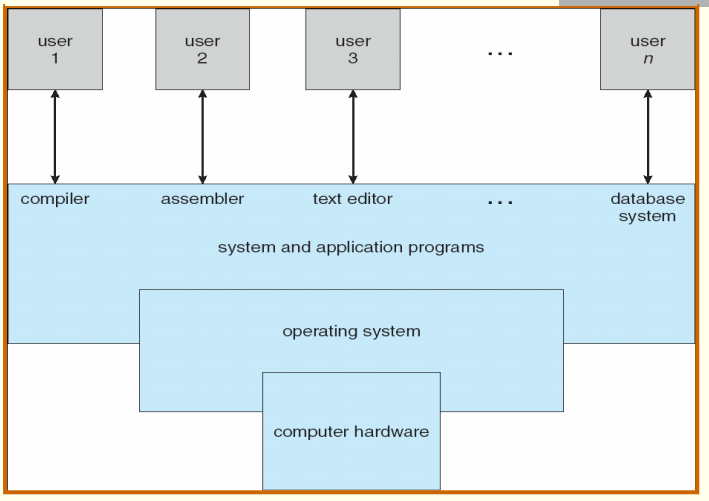
O que é o SO

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- Operating system goals:
  - Execute user programs and make solving user problems easier.
  - Make the computer system convenient to use.
- Use the computer hardware in an efficient manner

Estrutura de um Sistema de Computação

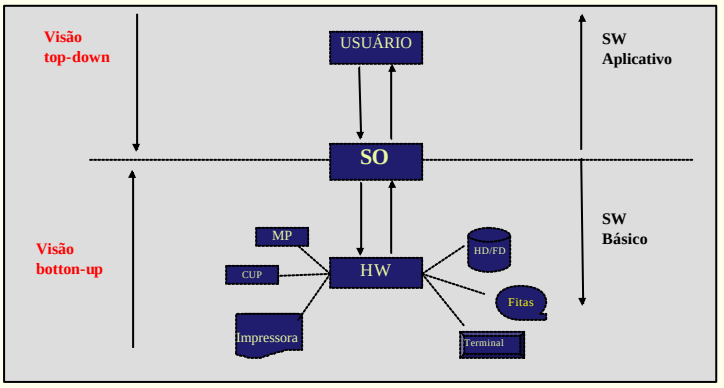
- Computer system can be divided into four components
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

Estrutura de um Sistema de Computação



Conceitos Básicos

- **Visão do Sistema Operacional**



Conceitos Básicos

- **Visão Top-down:** ***Máquina Virtual***

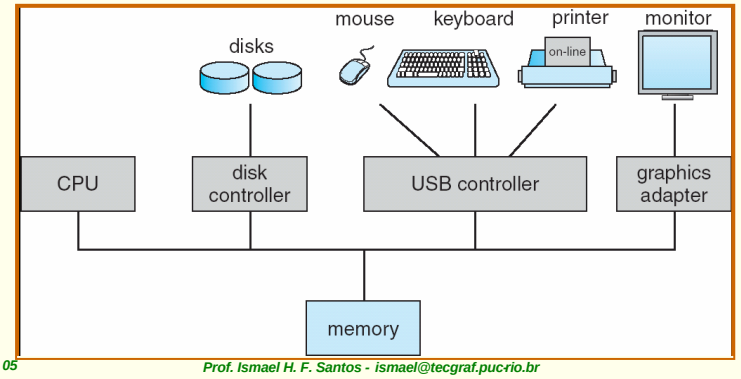
  O SO deve facilitar e padronizar o acesso aos recursos do sistema, servindo de interface entre o usuário e os recursos do sistema computacional, tornando esta comunicação transparente, eficiente e menos suscetível a erros.

- **Visão Botton-up:** ***Gerenciador de Recursos***

  O SO deve permitir o compartilhamento de recursos, entre os diversos usuários, de forma organizada e protegida. Tal compartilhamento além de dar impressão ao usuário de ser o único a utilizar um determinado recurso permite a diminuição de custos, na medida que mais de um usuário passa a utilizar as mesmas facilidades concorrentemente e de forma segura.

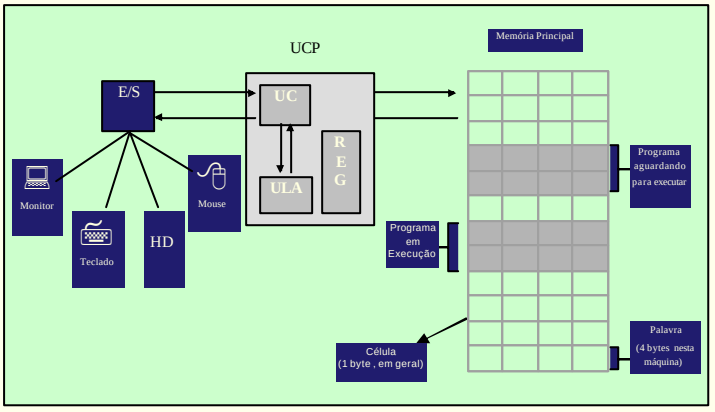
Estrutura de um Sistema de Computação

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory



Conceitos Básicos

- **Modelo de Máquina de Von-Newman**



Conceitos Básicos

- **Computador em Camadas (Tanenbaunn)**

Conceitos Básicos

Dispositivos Físicos - nível mais primitivo da máquina de Níveis, constituído pelos circuitos integrados, fonte de alimentação, memórias, periféricos e controladoras

Microprogramação - representa uma camada de SW primitiva que controla diretamente os Dispositivos Físicos, fornece uma interface clara para a camada superior e é usualmente gravada em memória ROM (Read Only Memory).

Linguagem de Máquina - é definido pelo conjunto de todas as instruções que serão executadas pelo nível de Microprogramação.

Firmware - designação que se dá a todo e qualquer SW que seja implementado em dispositivos semicondutores. O nível de Microprogramação é implementado com Firmware. O programa de Boot (inicialização) do PC também.

Estrutura de um Sistema de Computação

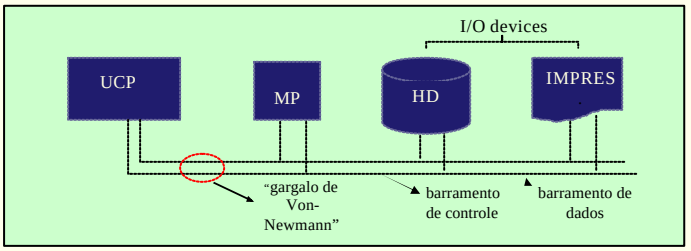
- **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EEPROM, generally known as **firmware**
  - Initializates all aspects of system
  - Loads operating system kernel and starts execution

Estrutura de um Sistema de Computação

- Computer-system operation
  - Concurrent execution of CPUs and devices competing for memory cycles I/O devices and the CPU can execute concurrently.
  - Each device controller is in charge of a particular device type.
  - Each device controller has a local buffer.
  - CPU moves data from/to main memory to/from local buffers
  - I/O is from the device to local buffer of controller.
  - Device controller informs CPU that it has finished its operation by causing an *interrupt*

Conceitos Básicos

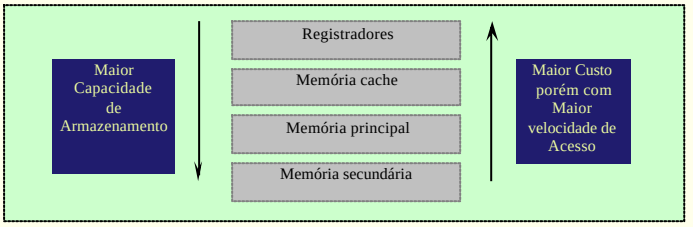
- **O Gargalo de Von-Newmann (GVN)**



  - **Modelo de Von-Newman apresenta um problema estrutural chamado Gargalo de Von-Newman. Este problema acontece devido ao elevado tráfego de dados e informações entre a UCP e a MP.**

Conceitos Básicos

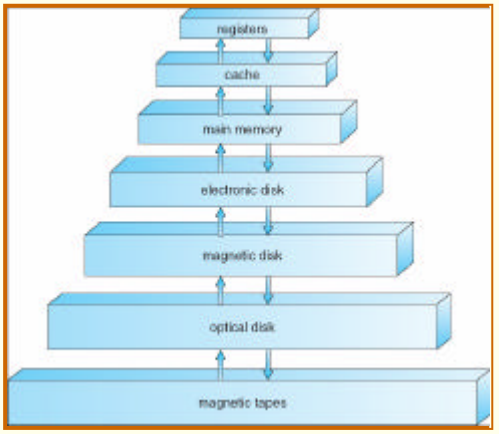
- **Memória Cache e o problema GVN**



- **Para amenizar o problema GVN, criou-se a Memória Cache, que é uma memória volátil de alta velocidade e cujo tempo de acesso a um dado nela contido é muito menor que o tempo de acesso caso o dado estivesse na memória principal.**

Hierarquia de Memória

- Storage systems organized in hierarchy.
  - Speed
  - Cost
  - Volatility
- *Caching* – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage.



Técnica de Cache

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

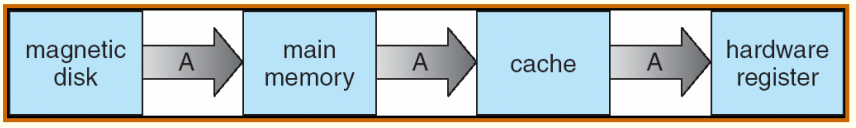
Performance dos diferentes níveis de Memória

- Movement between levels of storage hierarchy can be explicit or implicit

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Migration of an Integer "A" from Disk to Register inside the CPU

- Multitasking environments must be careful to use most recent value, not matter where it is stored in the storage hierarchy

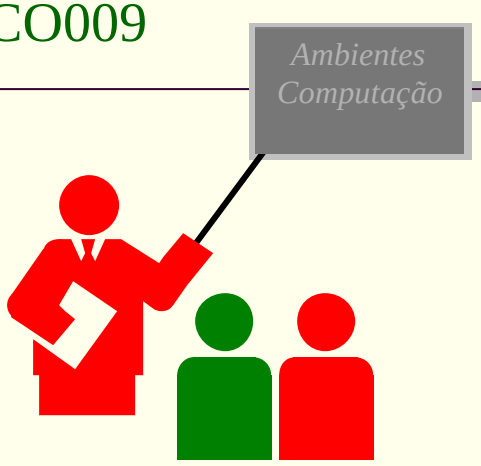


- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist
  - Various solutions covered later ….

System Boot

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- *Booting* – starting a computer by loading the kernel
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution
- System Boot
  - Operating system must be made available to hardware so hardware can start it
    - Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it
    - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
    - When power initialized on system, execution starts at a fixed memory location
      - Firmware used to hold initial boot code

SOP – CO009

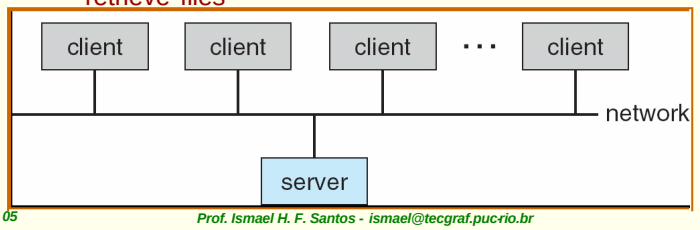


Computador Tradicional

- Traditional computer
  - Blurring over time
  - Office environment
    - PCs connected to a network, terminals attached to mainframe or minicomputers providing batch and timesharing
    - Now portals allowing networked and remote systems access to same resources
  - Home networks
    - Used to be single system, then modems
    - Now firewalled, networked

Cliente - Servidor

- Client-Server Computing
  - Dumb terminals supplanted by smart PCs
  - Many systems now servers, responding to requests generated by clients
    - Compute-server provides an interface to client to request services (i.e. database)
    - File-server provides interface for clients to store and retrieve files



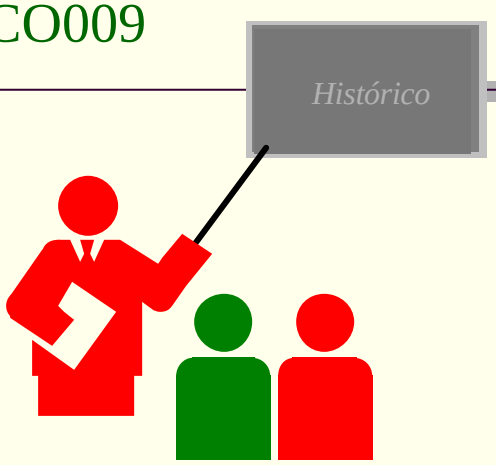
Peer-to-Peer

- Another model of distributed system
- P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for service and respond to requests for service via *discovery protocol*
  - Examples include *Napster* and *Gnutella*

Web-Based

- Web has become ubiquitous
- PCs most prevalent devices
- More devices becoming networked to allow web access
- New category of devices to manage web traffic among similar servers: **load balancers**
- Use of operating systems like Windows 95, client-side, have evolved into Linux and Windows XP, which can be clients and servers

SOP – CO009



Histórico do HW

- *A evolução dos SOs está, em grande parte, relacionada ao desenvolvimento dos computadores (HW) que a cada vez se tornam mais velozes, compactos e baratos. A evolução do HW pode ser dividida em cinco fases.*
  - ***PRIMEIRA GERAÇÃO - Válvulas (tubos de vácuo)***
  - ***SEGUNDA GERAÇÃO – Transistores***
  - ***TERCEIRA GERAÇÃO - Circuitos Integrados (CI)***
  - ***QUARTA GERAÇÃO – CIs em larga escala (LSI, VLSI)***
  - ***QUINTA GERAÇÃO - CIs em ultra larga escala (ULSI)***

Histórico dos SOs

- ***PRIMEIRA GERAÇÃO - 40 ~ 52***
  - *Não há sistema operacional;*
  - *Necessidade de conhecimento profundo do HW, pois a programação era feita em painéis, através de fios, utilizando linguagem de máquina.*

Histórico dos SOs

- ***SEGUNDA GERAÇÃO - 53 ~ 62***
  - *Com o surgimento das primeiras Linguagens de Programação, como o **Assembly e Fortran**, os programas deixaram de ser feitos diretamente no HW, facilitando o processo de desenvolvimento de SW.*
  - *Sistemas operacionais funcionavam com **processamento batch**. Programas passam a ser perfurados em cartões, que submetidos a uma leitora, os gravava em uma fita de entrada. Esta fita, pode então posteriormente ser lida pelo computador, que executava um programa de cada vez, gravando o resultado em do processamento em uma fita de saída.*

Histórico dos SOs

- *SOs passam a ter o seu próprio conjunto de rotinas de entrada/saída (**IOCS- input/output control system**). O IOCS eliminou a necessidade de os programadores terem que desenvolver suas próprias rotinas de leitura/gravação específicas para cada dispositivo periférico, criando o conceito de **independência de dispositivos E/S;***
- *Avanços em nível de HW, principalmente na linha 7094 da IBM, permitiu a criação do **processador de canal**, que veio permitir a transferência de dados entre dispositivos de entrada/saída e memória principal de forma independente da UCP.*

Histórico dos SOs

- ***TERCEIRA GERAÇÃO - 63 ~ 80***
  - *Evolução dos processadores de entrada/saída permitiu que, enquanto um programa esperasse por uma operação de leitura/gravação, o processador executasse um outro programa, criando o conceito de* **multiprogramação** *(SO* **multiprogramado** *ou* **multitarefa***).*
  - *Com o objetivo de reduzir o tempo de resposta no atendimento aos requisitos dos usuários (através de terminais on-line) a multiprogramação evoluiu até a criação dos* **sistemas time-sharing** *(***tempo compartilhado***).*

Histórico dos SOs

- **Sistemas de propósito geral** *(IBM/360, sistema operacional "OS" em 1964). Os sistemas de propósito geral são em geral associados a grande sobrecarga de utilização, de aprendizado demorado, grande tempo de depuração de erros e de difícil manutenção - grandes e caros ! Exceção - sistema UNIX, desenvolvido para máquinas de pequeno porte (minicomputador PDP-7 da Digital Equipment Corporation - DEC).*
- **Sistemas de tempo real** *foram criados para controle de processos.*

Histórico dos SOs

- ***QUARTA GERAÇÃO – 81 ~ 90.***
  - *Surgimento dos microprocessadores e do* **sistema operacional DOS (Disk Operation System);**
  - *No final dos anos 80, aplicações que exigiam um enorme volume de cálculos, puderam ser desenvolvidas com o suporte fornecido pelo SO ao* **multiprocessamento.**
  - *Assim, foi possível a execução de mais de um programa simultaneamente, ou até a execução de um mesmo programa por mais de um computador O surgimento de processadores vetoriais e técnicas de paralelismo em diferentes níveis tornou os computadores ainda mais poderosos.*

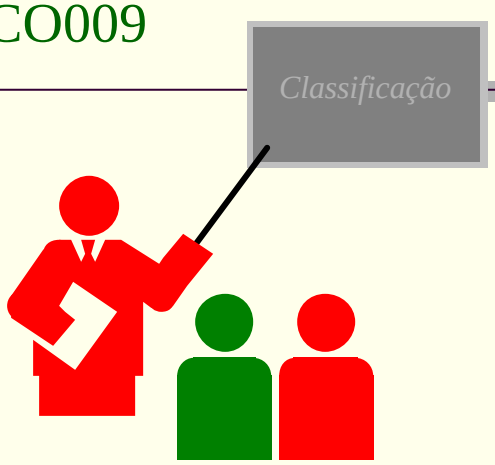
Histórico dos SOs

- ***QUINTA GERAÇÃO - 91 até hoje***
  - *Grandes evoluções em HW, SW e telecomunicações, vêm permitindo o desenvolvimento de sistemas multimídia, bancos de dados distribuídos, inteligência artificial (sistemas especialistas e redes neuronais) com cada vez maior capacidade.*
  - *A forma de interação com os computadores sofrerá também modificações diversas: novas interfaces homem-máquina (MMI - Mem Machine Interface) serão utilizadas com o uso de linguagens naturais, reconhecimento de voz e de assinaturas.*

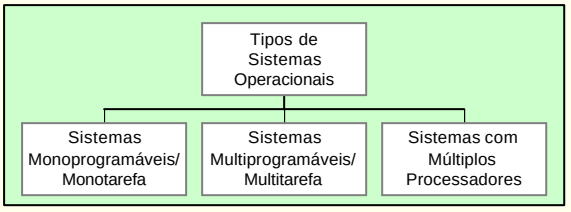
Histórico dos SOs

- ***QUINTA GERAÇÃO - 91 até hoje***
  - *A evolução das telecomunicações, com a crescente capacidade de transmissão de dados, tornará a WEB (World Wide Web da Internet) uma realidade na interação entre usuários, permitindo inclusive a transmissão de shows ao vivo e interação entre* **ambientes 3D** *(VRML).*

SOP – CO009



Classificação dos SOs



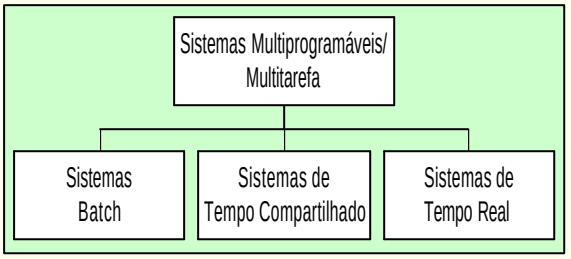
- **SOs Monoprogramáveis ou Monotarefa**
  - *Se caracterizam por permitir que o processador, a memória e os periféricos permaneçam exclusivamente dedicados à execução de um único programa. Recursos são mal utilizados, entretanto é fácil de ser implementado.*

Classificação dos SOs

- **SOs Multiprogramáveis ou Multitarefa**
  - *Nestes SOs vários programas dividem os do sistema. As vantagens do uso destes sistemas são o aumento da produtividade dos seus usuários e a redução de custos, a partir do compartilhamento dos diversos recursos do sistema.*
  - *Podem ser* **Multiusuário** *(mainframes, mini e microcomputadores) ou* **Monousuário** *(PCs e estações de trabalho). É possível que ele execute diversas tarefas concorrentemente ou mesmo simultaneamente (***Multiprocessamento***) o que caracterizou o surgimento dos SOs* ***Multitarefa****.*

Classificação dos SOs Multiprogramados

- *Os SOs* **Multiprogramáveis/Multitarefa** *podem ser classificados pela forma com que suas aplicações são gerenciadas, podendo ser divididos conforme mostra o gráfico abaixo.*

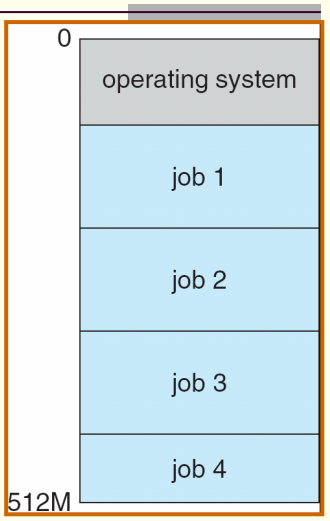


Classificação dos SOs Multiprogramados

- **SOs Multiprogramáveis Batch**
  - *Foram os primeiros SOs Multiprogramáveis a serem implementados e caracterizam-se por terem seus programas, quando submetidos, armazenados em disco ou fita, onde esperam ser carregados para execução seqüencial pelo monitor (embrião do SO).*
  - *Quando bem projetados, podem ser bastante eficientes, devido à melhor utilização do processador. Entretanto, podem oferecer tempos de resposta longos, em face do processamento puramente seqüencial e com uma variação alta dos seus* **tempos de execução** *(**tempo de parede** ou **wall clock time** ou **elapsed time**).*

Layout da Memória de um SO Multiprogramado

- **Multiprogramming** is needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job



Classificação dos SOs Multiprogramados

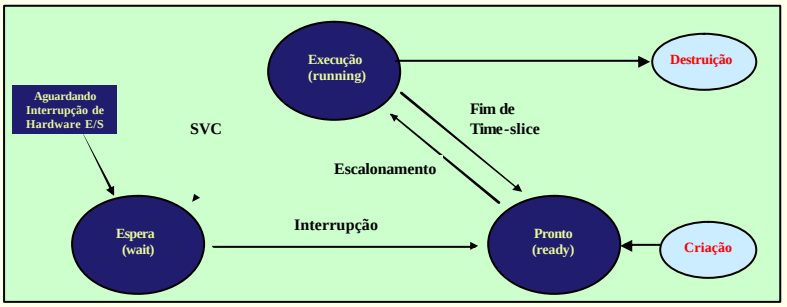
- **SOs Tempo Compartilhado – Time Sharing**
  - *Permitem a interação dos usuários com o sistema, basicamente através de terminais de vídeo e teclado (interação on-line). Dessa forma, o usuário pode interagir diretamente com o sistema em cada fase do desenvolvimento de suas aplicações.*
  - *Esses sistemas possuem uma* ***Linguagem de Controle*** *que permite ao usuário comunicar-se diretamente com o SO para obter e dar informações diversas.*

SOs Tempo Compartilhado

- *O SO aloca para cada usuário uma fatia de tempo de execução do processador o qual é chamada de* ***time-slice*** *(ou quantum de tempo). A execução do programa do usuário é interrompida após este tempo ou caso o programa peça a execução de uma* **SVC** *(chamada ao supervisor* - supervisor call*) para a leitura/gravação em algum periférico de E/S.*

- *O* ***processo*** *(programa do usuário executando) cuja execução foi suspensa por fim do time-slice entra numa "***fila de execução de processos prontos***" para ser processado mais tarde.*

SOs Tempo Compartilhado



- *O processo que pediu uma **SVC** entra em* **estado de espera** *(até que a SVC tenha sido executada).*

SOs Tempo Compartilhado

- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨**process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

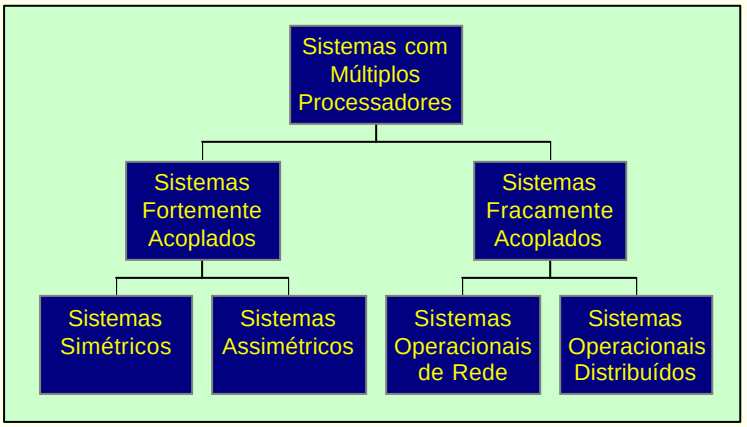
SOs de Tempo Real

- **SOs Multiprogramados de Tempo Real**
  - *Os SOs de Tempo Real são semelhantes aos sistemas Time-Sharing. A maior diferença é o* **tempo de resposta**, *exigido no processamento das aplicações, que não pode variar para não comprometer a segurança do sistema.*
  - *Neste sistema não existe o conceito de time-slice. Um programa executa o tempo que for necessário, ou até que outro programa com maior prioridade tome o seu lugar. Esta importância ou prioridade de execução é controlada pela própria aplicação e não pelo sistema operacional.*

SOs com Múltiplos Processadores

- **SOs com Múltiplos Processadores**
  - *Os SOs com Múltiplos Processadores caracterizam-se por possuírem duas ou mais UCPs interligadas, trabalhando em conjunto. Um fator chave na classificação de SOs com Múltiplos Processadores é a forma de comunicação entre as UCPs e o grau de compartilhamento da memória e dos dispositivos de entrada e saída. Em função destes fatores, podemos classificar os SOs com* **Fortemente Acoplados** *ou* **Fracamente Acoplados** *conforme a figura a seguir:*

Classificação dos SOs com Múltiplos Processadores

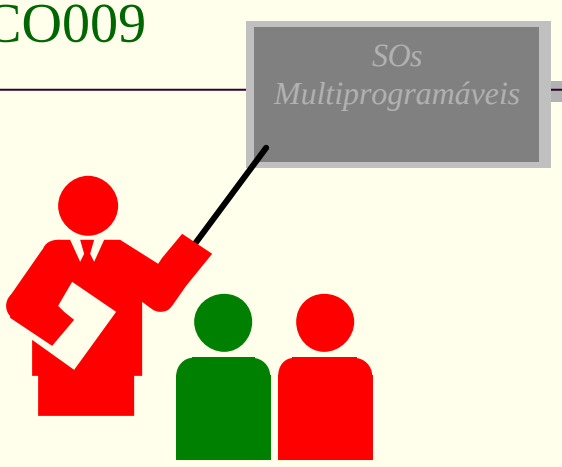


Classificação dos SOs com Múltiplos Processadores

- **SOs Fracamente Acoplados** *permitem que as máquinas e os usuários de um sistema distribuído sejam completamente independentes. Este é o caso, por exemplo, de uma rede de PCs que compartilham alguns recursos como impressoras laser, servidores de Banco de Dados, via uma rede local (LAN).*

- *SOs* **Fortemente Acoplados** *permitem que os programas dos usuários sejam divididos em sub-programas para execução simultânea em mais de um processador.*

SOP – CO009



SOs Multiprogramáveis

- ***Motivação***
  - *Os SOs Multiprogramáveis surgiram devido a baixa utilização dos recursos do sistema presente nos SOs Monoprogramáveis.*

SOs Monoprogramáveis		***SOs Multiprogramáveis***	
UCP	***MP***	***UCP***	***MP***
30%	*má utilização existência de áreas de memória livre*	*90%*	*boa utilização*

SOs Multiprogramáveis

- **Funções desejadas**
  - **Flexibilidade**
    - facilidade para carregar programas para execução;
    - interpretar linguagem de comandos e de controle - Shell;
    - suportar interação com usuário através de terminais - "Time-Sharing";
    - facilidade para controle, administração, contabilização do uso dos recursos do HW;
  - **Concorrência**
    - compartilhar memória entre os diversos programas dos usuários;
    - compartilhar o uso da UCP para a realização das tarefas dos usuários;
    - sobrepor E/S e processamento;

SOs Multiprogramáveis

- **Funções desejadas (cont.)**
  - **Compartilhamento**
    - compartilhar a UCP selecionando o próximo programa a executar - "**Scheduling**";
    - compartilhamento de dados;
    - reutilização de programas criados por outros - "**código reentrante**";
    - eliminação de redundâncias para tratamento de E/S;
  - **Proteção**
    - tratamento de erros e exceções de HW;
    - tratamento de interrupções;
    - proteger o programa contra violações de outros programas - "**Memory Protection**";

SOs Multiprogramáveis

- **Principais Problemas**
  - Como revezar a UCP entre os programas dos usuários ?
  - Como proteger os programas entre si ?
  - Como sincronizar atividades que são mutuamente exclusivas (Ex.: impressão, leitura e/ou gravação em dispositivos de acesso seqüencial)
  - *A concorrência é fundamental para entendermos o funcionamento de um sistema operacional Multiprogramável. A possibilidade de periféricos e dispositivos funcionarem simultaneamente com a UCP, permitiu a execução de tarefas concorrentes*

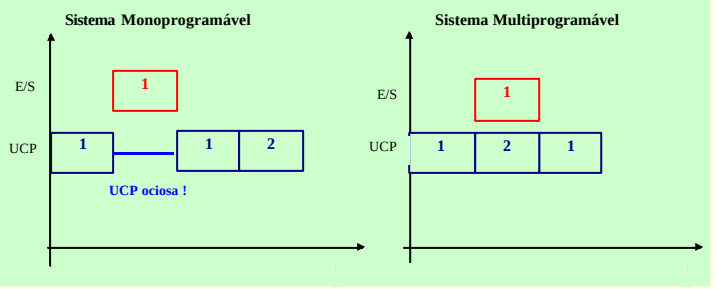
SOs Multiprogramáveis

- **Enquanto nos SOs Monoprogramáveis, somente um programa pode estar residente na memória (com conseqüente dedicação exclusiva da UCP à execução desse programa), nos SOs Multiprogramados, vários programas podem estar residentes em memória, concorrendo pela utilização da UCP.**

- **Dessa forma quando um programa solicita uma operação de entrada/saída, outros programas poderão estar disponíveis para utilizar o processador.**

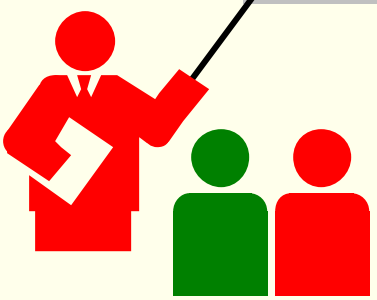
SOs Multiprogramáveis

- **Conseqüência** -> UCP permanece menos tempo ociosa e a MP é utilizada de forma mais eficiente



SOP – CO009

Interrupção
X
Exceção



Interrupção e Exceção

Conceito de Interrupção e Exceção

- *Uma **Interrupção** ou **Exceção** é a ocorrência de algum evento durante a execução de um programa obrigando a intervenção do SO.*

- *Tal evento pode ser resultado de:*
  - *execução de instruções do próprio programa seja devido a um erro ou a um pedido do usuário. Neste caso é chamada de **trap** (sw-generated interrupt);*
  - *gerado pelo SO;*
  - *por algum dispositivo de HW.*

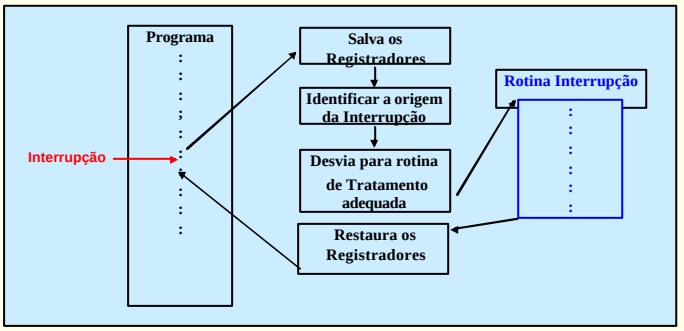
Interrupção e Exceção

Conceito de Interrupção e Exceção (cont.)

- *Uma **Interrupção** é gerada pelo SO ou por algum dispositivo e, neste caso, independe do programa que está sendo executado. Um exemplo é quando um periférico avisa à UCP que está pronto para transmitir algum dado. Neste caso, a UCP deve interromper o programa para atender a solicitação do dispositivo. Este tratamento é feito pelo **Mecanismo de Interrupção**, executado pelo HW, definido a seguir.*

Mecanismo de Interrupção

- **Mecanismo de Interrupção**



1. Unidade de controle detecta a ocorrência de interrupção, ela interrompe a execução do programa e salva o PC e a PSW do processo corrente na sua área de STACK.

Mecanismo de Interrupção

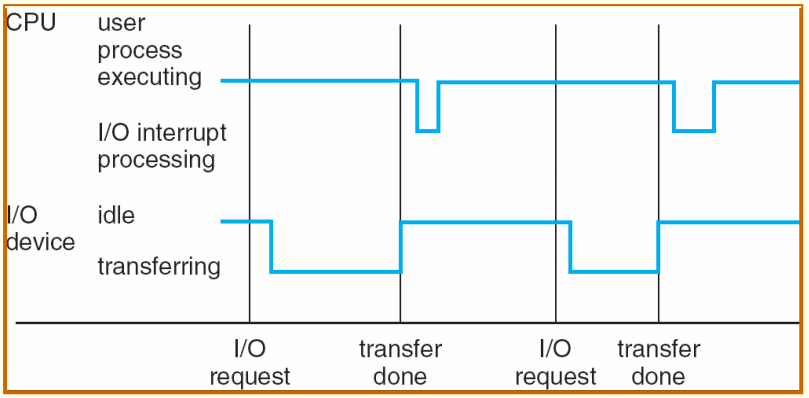
- **Mecanismo de Interrupção (cont.)**
  2. A seguir o controle da execução é passado para o SO que salva o restante das informações do contexto do processo que estava executando.
  3. Feito isto o controle é então desviado para a rotina do SO responsável pelo tratamento da interrupção.
- **Vetor de Interrupção**
  - Existem diferentes tipos de interrupção que são atendidas por diversas rotinas de tratamento. No momento que uma interrupção acontece, a UCP deve saber para qual rotina de tratamento deverá ser desviado o fluxo de execução. Essa informação está em uma estrutura do SO chamado ***Vetor de Interrupção***, que contém a relação de todas as rotinas de tratamento existentes, associadas a cada interrupção.

Mecanismo de Interrupção

- Todo o procedimento para detectar a interrupção, salvar o contexto do programa e desviar para a rotina de tratamento é denominado **Mecanismo de Interrupção.** Este mecanismo é realizado, na maioria das vezes, pelo HW dos computadores, e foi implementado pelos projetistas para criar uma maneira de sinalizar ao processador eventos assíncronos que possam ocorrer no sistema.

- No caso de múltiplas interrupções ocorrerem, o processador deve saber qual interrupção será tratada primeiro, o que é feito através da prioridade que é atribuída pelo sistema operacional a cada interrupção. Normalmente o HW possui um dispositivo denominado **Controlador de Pedidos de Interrupção** que avalia, ordena os pedidos e salva o endereço da instrução interrompida.

Linha de tempo de Interrupção



Exceção x Interrupção

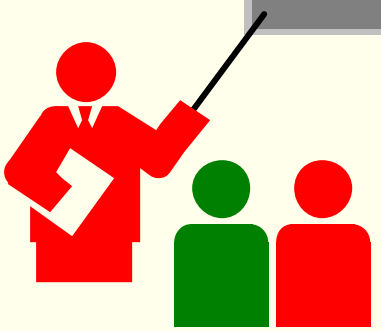
- O que diferencia uma **Interrupção** de uma **Exceção** é o tipo de evento que gera esta condição. Uma exceção é resultado direto da execução de uma instrução do próprio programa. Situações como a divisão por zero ou a ocorrência de um overflow caracterizavam essa situação.

- A principal diferença entre **Exceção** e **Interrupção** é que a primeira é gerada por um **evento síncrono**, enquanto que a segunda é gerada por **eventos assíncronos**.

- A interrupção é o mecanismo que torna possível a implementação da concorrência nos computadores, sendo o fundamento básico dos sistemas Multiprogramáveis.

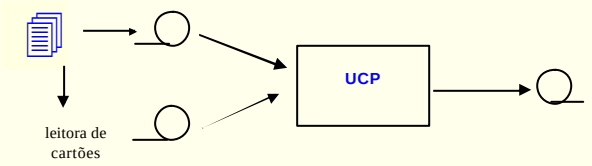
SOP – CO009

Sistemas Batch
Mono e Multitarefa



Sistema Batch Simplificado - Monotarefa

- **Sistema Batch Simplificado - Monotarefa**



  - SO tem sua execução baseada em fitas magnéticas. Vários passos eram necessários para executar os programas. Assim para facilitar as atividades do programador/operador foi criada a **Linguagem de Controle de Programas** (**JCL - Job Control Language**) para automatizar as etapas necessárias para a execução de um programa.

Sistema Batch Simplificado - Monotarefa

- **Sistema Batch Simplificado - Monotarefa**
  - Mais de um lote (batch) para execução na UCP, acelera o ciclo de execução, cria a figura do operador de computador e libera o programador.
  - UCP ociosa entre a carga de diferentes lotes.
  - **Monitor** residente: programa para carregar os programas a serem executados a partir das unidades de fita.
  - Cartões de controle: servem para instruir o monitor na carga dos programas. Ex.: $JOB, $FTN, $END.

Sistema Batch Simplificado - Monotarefa

- ***Modos Monitor/Usuário***
  - Melhoria em HW possibilita maior robustez ao SO.
  - Uso opcional das rotinas de E/S para manipulação de periféricos => maior eficiência e segurança na execução dos programas - estas rotinas fazem parte do monitor.
  - Portanto agora parte do programa é executada pelo monitor (**Modo Monitor**) e parte pelo programa propriamente dito (**Modo Usuário**).
  - Funcionando desta forma, a interceptação de erros de execução (através do "trapeamento de erros") pode ser identificada pelo monitor, o que aumentava a segurança do sistema.

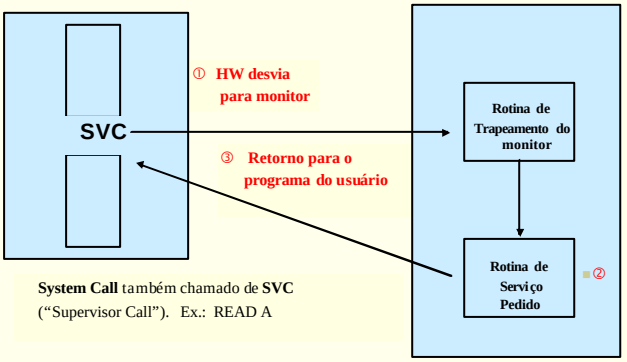
Sistema Batch Simplificado - Monotarefa

- ***Instruções Privilegiadas e Modos do Processador***
  - Apenas em **modo monitor** o HW permite executar determinadas instruções, as chamadas **instruções privilegiadas**.
  - Então surge o problema de como se passar do estado usuário para o estado monitor ?
  - Por exemplo: como se consegue executar uma instrução privilegiada de E/S ?

Sistema Batch Simplificado - Monotarefa

Execução de uma SVC

1. Programa do usuário executa uma SVC para executar uma rotina de monitor. (Ex.: READ).



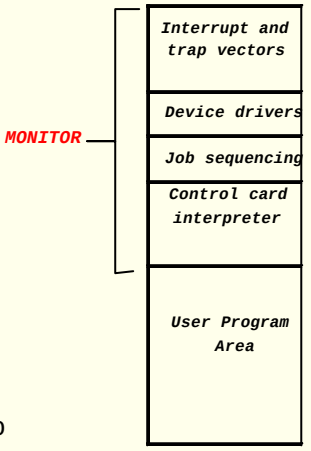
System Call também chamado de **SVC** ("Supervisor Call"). Ex.: READ A

Sistema Batch Simplificado - Monotarefa

Execução de uma SVC

2. Parâmetros na chamada indicam o serviço pedido. No momento do desvio do programa do usuário para Rotina de Trapeamento o estado do processo (programa executando) passa para **monitor** (HW).

3. Na volta da Rotina de Serviço o modo passa de monitor para usuário (HW) Exemplo de Instruções Privilegiadas: Instruções de E/S; Parada do processador (HALT); Mudança no modo do processador para monitor.

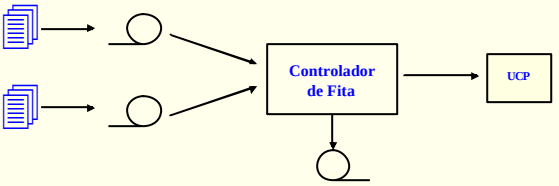


Sistema Batch Sofisticado - Multitarefa

- **Sistema Batch Sofisticado – Multitarefa**
  - Os periféricos de E/S são dispositivos eletromecânicos e portanto lentos em relação a UCP e memória, que são dispositivos eletrônicos. O tempo de switching (chaveamento - ciclo de máquina) da UCP é da ordem de microssegundos enquanto o tempo médio de acesso a disco é da ordem de dezenas de milissegundos. Para resolver este problema de diferença de velocidades de acesso diversas técnicas foram sendo introduzidas para reduzir o Blocked-Time.

Sistema Batch Sofisticado - Multitarefa

- Operações “Off-line”

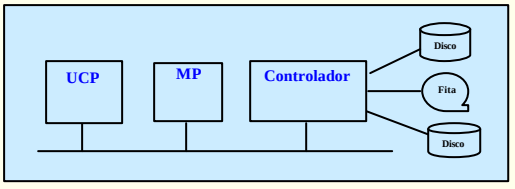


  - cartões de entrada e imagem de relatórios gravados em fita;
  - independência de dispositivos (E/S lógica). O SO torna transparente ao programa do usuário o acesso à fita e entrega/recebe um registro por vez com a imagem do cartão ou linha do relatório de saída;
  - cartões de controle mapeiam dispositivos lógicos em físicos; mais de um sistema leitora “cartões-fita” ou “fita-impressora” otimizava a operação, porém apenas quando a fita estivesse cheia é que então a UCP pode continuar o processamento.

Sistema Batch Sofisticado - Multitarefa

- **Operações de Entrada/Saída**
  - *Nos primeiros SOs, a comunicação entre a UCP e os periféricos era controlada por um conjunto de instruções especiais, denominadas **instruções de entrada/saída**, executadas pela própria UCP. Essas instruções continham detalhes específicos de cada periférico.*
  - *A implementação de um dispositivo chamado **controlador** permitiu à UCP agir de maneira independente dos dispositivos de E/S. A partir daí a UCP não se comunicava mais diretamente com os periféricos, mas sim através do controlador, conforme mostra a figura a seguir.*

Sistema Batch Sofisticado - Multitarefa



- *Existiam agora duas maneiras para UCP controlar operações de E/S:*
  - ***E/S controlada por programa** - UCP sincronizava-se com o periférico para o início da transferência de dados e, após iniciada a transferência, o sistema ficava permanentemente testando o estado do periférico para saber quando a operação tinha chegado ao seu final.* **Problema: busy-waiting**.

Sistema Batch Sofisticado - Multitarefa

- ***E/S por Polling*** *- após o início da transferência dos dados a UCP fica livre para se ocupar de outras tarefas e em determinados intervalos de tempo se realiza um teste para saber do término ou não da operação de E/S em cada dispositivo.* **Problema*:* o SO tem que freqüentemente interromper o processamento dos programas para testar os diversos periféricos.**

- *A criação do* ***Mecanismo de Interrupção*** *permitiu que as operações de E/S pudessem ser realizadas de uma forma mais eficiente, criando o que se chamou de*

Sistema Batch Sofisticado - Multitarefa

- ***E/S controlada por Interrupção*** *- em vez do sistema verificar periodicamente o estado de uma operação pendente, o próprio controlador interrompe a UCP para avisar o término da operação.*

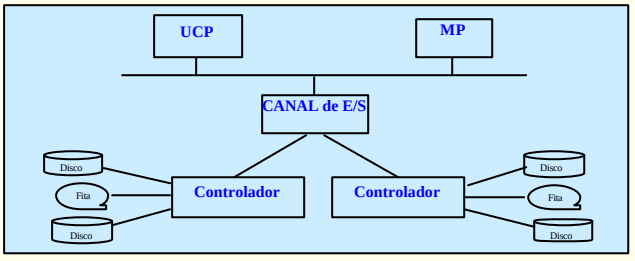
- *Apesar da* ***E/S controlada por interrupção*** *ser mais eficiente que a* ***E/S controlada por programa****, toda transferência de dados entre memória e periféricos exigia a intervenção da UCP. Para liberar a UCP, permitiu-se então ao controlador fazer a transferência dos dados direto para a memória, dando origem a técnica* ***DMA - Direct Memory Access.***

Sistema Batch Sofisticado - Multitarefa

- A técnica de **DMA** permite que um bloco de dados seja transferido entre memória e periféricos, sem a intervenção da UCP, exceto no início e no final da transferência.
- Quando o sistema deseja ler ou gravar um bloco de dados, são passadas da UCP para o controlador informações como: onde o dado está localizado, qual o dispositivo de E/S envolvido na operação, posição inicial da memória de onde os dados serão lidos ou gravados e o tamanho do bloco de dados.
- Com estas informações, o controlador realiza a transferência entre o periférico e a memória principal, e a UCP é somente interrompida no final da operação. A área de memória utilizada pelo controlador na técnica de **DMA** é chamada **buffer de E/S**, sendo reservada exclusivamente para este propósito.

Sistema Batch Sofisticado - Multitarefa

- A extensão do conceito de **DMA** possibilitou o surgimento dos **canais de E/S**. O **canal de E/S** é um processador com capacidade de executar programas de E/S cujas instruções são armazenadas na memória principal pela UCP. Assim, a UCP realiza uma operação de E/S, instruindo o canal para executar um programa localizado na memória (programa de canal). Este programa especifica os dispositivos para transferência, buffers e ações a serem tomadas em caso de erros.

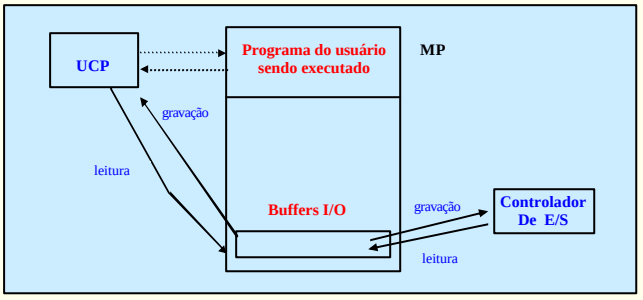


Sistema Batch Sofisticado - Multitarefa

- **Técnica de Buffering**
  - A técnica de **Buffering** consiste na utilização de uma área de memória para a transferência de dados entre os periféricos e a memória principal denominada Buffer de E/S.
  - O **Buffering** veio permitir que, quando um dado fosse transferido para o buffer após uma operação de leitura. o dispositivo de entrada pudesse iniciar uma nova leitura (**READ-AHEAD**).

Sistema Batch Sofisticado - Multitarefa

- **As principais características da técnica de Buffering**
  - paralelismo entre UCP e E/S, já que a UCP é muito mais rápida que E/S
  - leitura/gravação antecipada de vários registros;



Sistema Batch Sofisticado - Multitarefa

- registros colocados no buffer para posterior gravação;
- detecção do fim de E/S via interrupção de HW;
- o SO gerencia buffers de cada periférico através de device drivers;
- o programa do usuário recebe/envia um registro a cada "chamada ao sistema" (SVC) para E/S;

- A finalidade da técnica de **Buffering** é minimizar o problema da disparidade da velocidade de processamento existente entre a UCP e os dispositivos de E/S, mantendo na maior parte do tempo UCP e dispositivos de E/S ocupados.

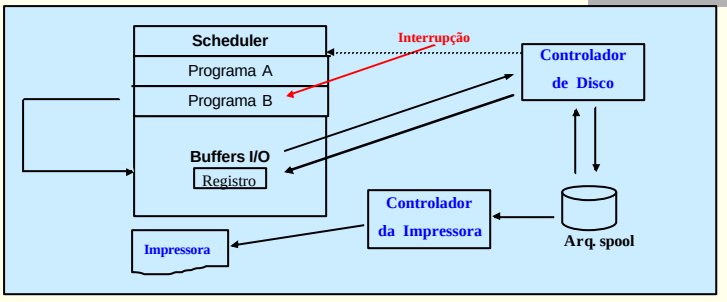
Sistema Batch Sofisticado - Multitarefa

- **Técnica de Spooling**
  - A técnica de **Spooling** tem a finalidade de liberar a UCP das tarefas de impressão. No momento em que um comando de impressão é executado, as informações a serem impressas são gravadas em um arquivo em disco (arquivo de spool), para ser impresso posteriormente pelo sistema conforme mostra a figura. Dessa forma, situações como a de um programa reservar a impressora. imprimir uma linha e ficar horas para continuar a impressão não acontecerão.

Sistema Batch Sofisticado - Multitarefa

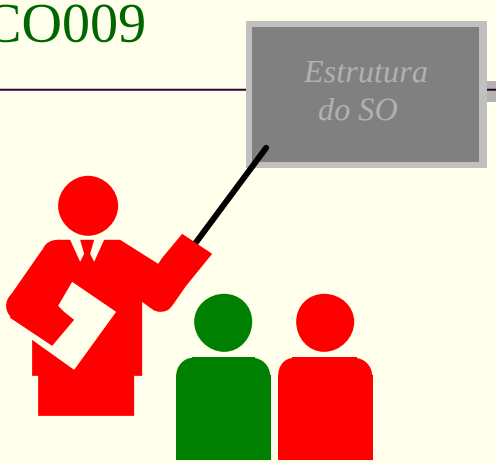
- **Principais características da técnica de Spooling**
  - "**spool**" = carretel; "**spooling**" uso do disco para enfileirar imagens de cartões e relatórios para serem gravados/lidos posteriormente;
  - o SO controla a localização de cada "job" e cada "relatório" no disco através de uma estrutura de dados chamada "tabela de spooling";
  - há concorrência entre submissão, execução e impressão de diferentes serviços;
  - **Job-spool** - grupo de serviços aguardando execução seguindo algum critério de escalonamento ("Scheduling").

Sistema Batch Sofisticado - Multitarefa



- A técnica de **Buffering** permite que um job utilize um buffer de memória concorrentemente com um dispositivo de E/S. Já a técnica de **Spooling**, utiliza o disco como um grande buffer, permitindo que dados sejam lidos e gravados em disco, enquanto outros jobs são processados.

SOP – CO009



Estrutura do SO

- **Introdução**
  - O SO é formado por um conjunto de rotinas que oferecem serviços aos usuários do sistema e suas aplicações. Este conjunto de rotinas é chamado **Núcleo** ou **Kernel**.
- **Principais funções do SO**
  - *Interface com o Usuário;*
    - *CLI – command-Line Interface;*
    - *GUI – Graphical User Interface*
    - *Batch*
  - *Criação e Eliminação de Processos;*
    - *Carga de um programa para a memória e controle de sua execução;*
  -

Estrutura do SO

- **Principais funções do SO (cont.)**
  - *Sincronização e Comunicação entre Processos;*
    - *Permitir a troca de informações entre processos na mesma maquina ou em maquinas diferentes. A comunicação pode ser via memória compartilhada (shared memory") ou troca de mensagens (message passing);*
  - *Escalonamento Processos(Gerência de Processos)*
  - *Gerência do Sistema de Arquivos;*
    - *Criar, remover, ler e escrever arquivos e diretórios*
  - *Gerência de Memória;*
  - *Tratamento de Interrupções;*

Estrutura do SO

- **Principais funções do SO (cont.)**
  - *Operações de E/S (Gerência de Periféricos);*
    - *O programa em execução pode solicitar uma operação de E/S que pode envolver um arquivo ou um device E/S*
  - *Detecção e correção de erros;*
    - *O SO deve estar constantemente controlando a ocorrência de possíveis erros:*
      - *Podem ocorrer na CPU, Memória, dispositivos de E/S ou no programa do usuário*
      - *Para cada tipo de erro, o SO deve tomar a ação apropriada para garantir a a correção e consistência da computação executada.*
      - *Facilidades de depuração permitem ao usuário mais facilmente identificar e consertar os erros.*

Estrutura do SO

- **Principais funções do SO (cont.)**
  - *- Contabilização;*
    - *Controlar e contabilizar o uso dos recursos utilizados pelos usuários;*
  - *- Segurança do Sistema - Proteção e Segurança;*
    - *As informações armazenadas em um sistema Multiusuário devem ter as suas informações sigilosas preservadas, de tal forma que processos concorrentes não interfiram uns com os outros:*
      - *Proteção – garantir que o acesso a todos os recursos seja feita de forma controlada;*
      - *Segurança – requer o uso de mecanismos de autenticação para defender o sistema contra a entrada de intrusos;*

SO - CLI

- CLI allows direct command entry
  - Sometimes implemented in kernel, sometimes by systems program
  - Sometimes multiple flavors implemented – **shells**
  - Primarily fetches a command from user and executes it
    - Sometimes commands built-in, sometimes just names of programs
      - If the latter, adding new features doesn't require shell modification

SO - GUI

- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI “command” shell
  - Apple Mac OS X as “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

Kernel do SO

- **Kernel ou Núcleo do SO (cont.)**
  - *A estrutura do **Kernel** do SO, isto é, a maneira como o código do SO é organizado e o inter-relacionamento entre os seus diversos componentes, pode variar conforme a concepção de projeto do SO. Existem basicamente três abordagens: **Monolítico**, **Em Camadas** e **Micro-Núcleo** (**Micro-Kernel**)*

Estrutura do SO

- **Kernel ou Núcleo do SO (cont.)**
  - O núcleo roda com interrupções inibidas (desabilitadas) portanto, deve limitar-se às funções essenciais para minimizar o tempo em que a máquina fica insensível as modificações do ambiente.
  - Os demais gerentes do SO são processos interrompíveis, assim como os processos dos usuários. Todos os gerentes e o núcleo rodam em modo **SUPERVISOR**. As rotinas de tratamento de interrupção e SVC residem no núcleo.

Estrutura do SO

- **Kernel ou Núcleo do SO (cont.)**
  - O SO mantém a estrutura de dados que reflete "o estado" dos recursos do sistema em um dado momento. Esta estrutura é composta por:
    - descritores de processos - *recursos lógicos*
    - descritores de memória - *recursos físicos*
    - descritores de arquivo - *recursos lógicos*
    - descritores de periféricos - *recursos físicos*
  - *Quando um gerente se comunica com outro gerente a comunicação passa pelo núcleo. Os descritores são implementados como listas encadeadas em memória.*

Estrutura do SO

- **Proteção em Sistemas Multiprogramados**
  - *A fim de garantir a integridade dos dados pertencentes a cada usuário o SO deve implementar algum tipo de proteção aos diversos recursos que são compartilhados no sistema, como memória, dispositivos de E/S e UCP.*
  - ***Proteção a memória** : quando o programa tenta acessar uma posição de memória fora de sua área endereçável, um erro do tipo violação de acesso ocorre e o programa é encerrado.*

Estrutura do SO

- **Proteção em Sistemas Multiprogramados**
  - ***Compartilhamento de dispositivos de E/S** : é controlado de forma centralizada pelo SO. Em geral o SO disponibiliza rotinas para trancamento (lock) de arquivos e/ou registros de arquivos para permitir o acesso exclusivo ou compartilhado por diversos usuários. No UNIX temos a **system call** flock.*

Estrutura do SO

- ***Compartilhamento da UCP*** *: Para evitar que um programa em loop aloque o processador por tempo indeterminado, a UCP possui um* ***relógio de tempo real*** *que gera interrupções periódicas (time-slice). A cada interrupção é executada a rotina de tratamento de interrupções de relógio, que entre outras funções, tem a responsabilidade de suspender a execução do processo corrente.*

Estrutura do SO

- Para garantir a proteção do sistema exige-se que toda vez que um usuário desejar utilizar um recurso, ele deverá solicitar o uso do recurso ao SO. O pedido é realizado através de chamadas a rotinas especiais denominadas **system calls**

Chamadas ao Sistema - System Call

- Uma **system call** permite o acesso a recursos do SO, e por isso possuí um mecanismo de proteção para sua execução denominado **estado de Execução** (**PSW** process status word). O **estado de execução** é uma característica associada ao programa em execução, que determina se ele pode ou não executar outras instruções ou rotinas.

Estrutura do SO

- No **estado usuário**, um programa só pode executar instruções que não afetam diretamente outros programas (**instruções não-privilegiadas**). No **estado supervisor**, qualquer instrução pode ser executada. As instruções que só podem ser executadas por programas no **estado supervisor** são denominadas **instruções privilegiadas**.

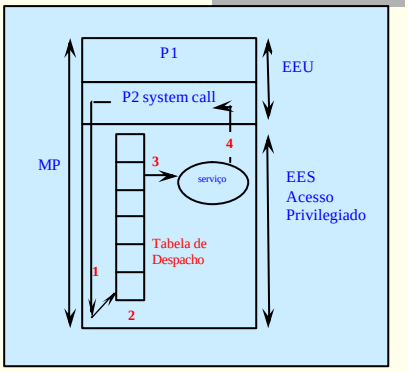
- O **estado de execução** de um processo é determinado por um conjunto de bits, localizado em um registrador especial da UCP que indica o estado corrente (PSW process status word) o qual o HW do sistema acessa para verificar o estado do processo, e se a instrução pode ou não ser executada. Quando um programa que está sendo processado no estado usuário executa uma **system call** o seu estado é alterado pela própria rotina do sistema que se encarrega, ao seu término, de restaurar o estado de execução anterior do processo. Caso um programa tente executar uma **instrução privilegiada**, sem estar no estado supervisor, uma interrupção é gerada e o programa é encerrado.

Estrutura do SO

- *O SO divide a MP em dois espaços de enderecamento:*

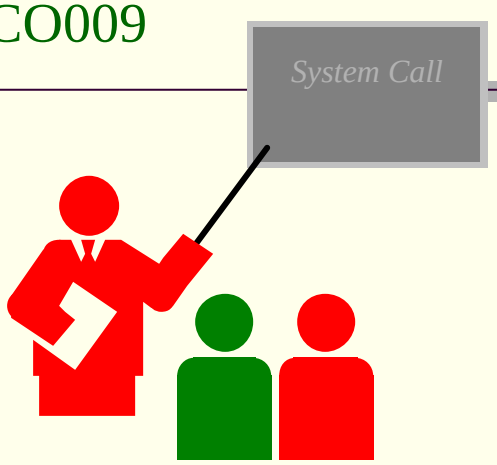
  EES – EE do Sistema

  EEU – **EE do Usuário**



- *Programas dos usuários só podem ter acesso ao **EES** via mecanismos de* **system call**, *que é invocado sempre que um processo precisa ter acesso a algum serviço disponível.*

SOP – CO009

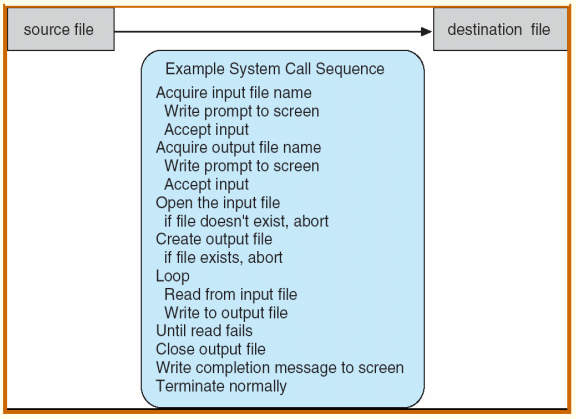


System Calls

- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

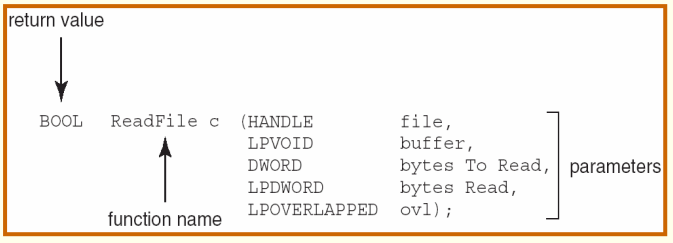
Exemplo de System Call

- System call sequence to copy the contents of one file to another file



Exemplo Standard API

- Consider the ReadFile() function in the
- Win32 API - a function for reading from a file

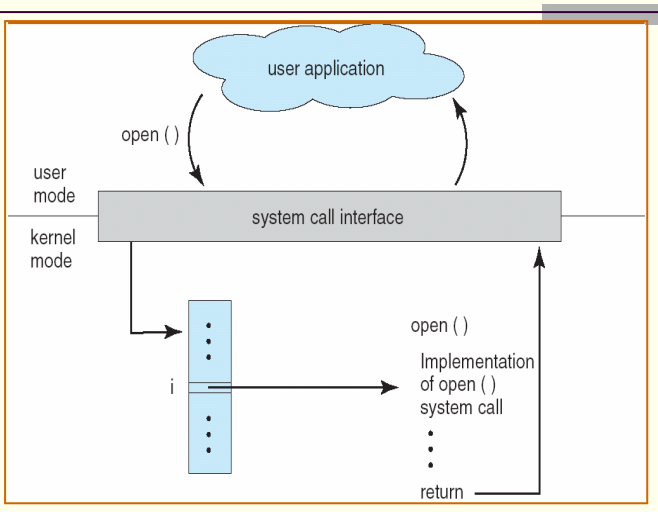


- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

System Call Implementation

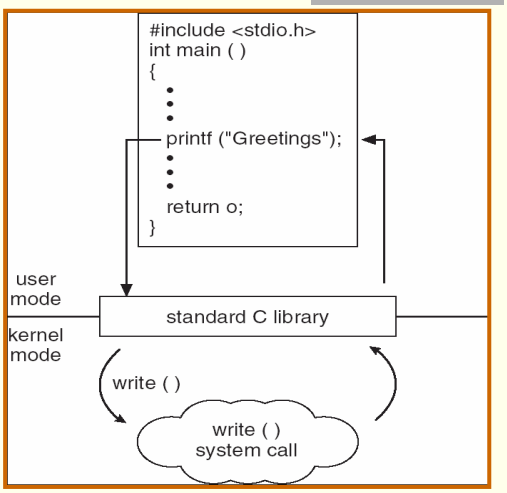
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call



Standard C Library Example

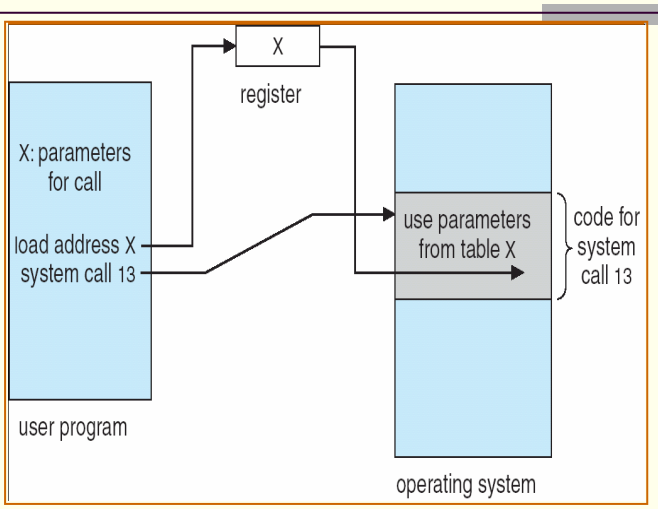
- C program invoking printf() library call, which calls write() system call



Passagem de Parâmetros para System Call

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - Parameters stored in a *block,* or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

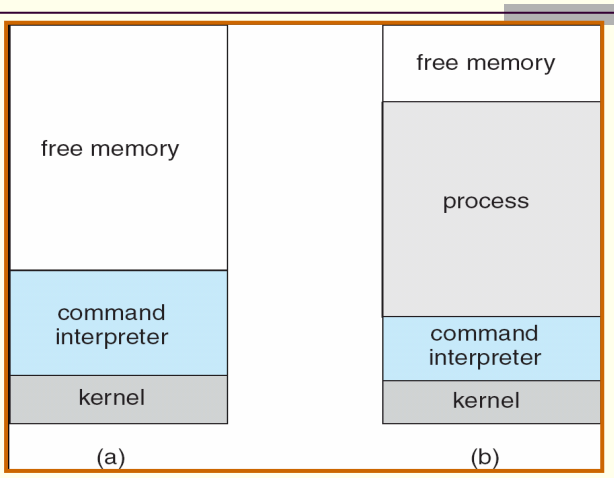
Passagem de Parâmetros via Tabela



Tipos de System Calls

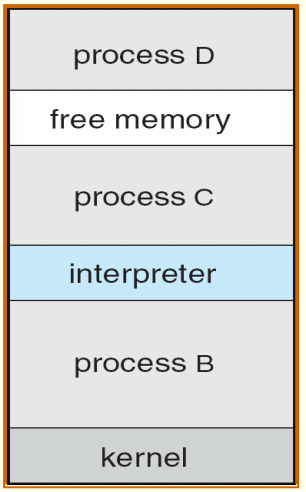
- *As* **system calls** *podem ser divididas em 5 categorias:*
  - *Controle de processos – criar, terminar, sinalizar, etc.*
  - *Manipulação de arquivos – criar, remover, ler, gravar, etc.*
  - *Gerência de dispositivos*
  - *Comunicação entre processos*
  - *Informações gerais – obter informações de contabilização, data e hora do sistema, etc.*

MS-DOS execution



- (a) At system startup (b) running a program

FreeBSD – Running Multiple Programs



System Programs

- System programs provide a convenient environment for program development and execution. The can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls

System Programs (cont'd)

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- Status information
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a registry - used to store and retrieve configuration information

System Programs (cont'd)

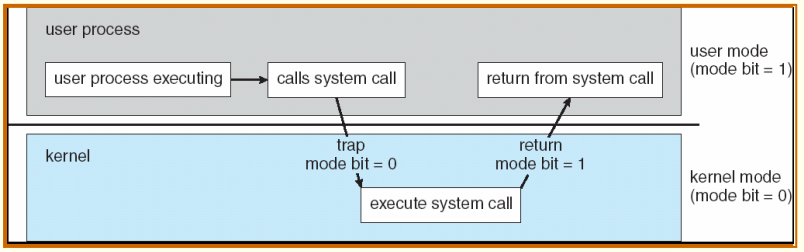
- File modification
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided
- Program loading and execution- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Operação do SO

- Interrupt driven by hardware
- Software error or request creates **exception** or **trap**
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** in the **PSW** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user

Transição User-Mode to Kernel-Mode

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time



Tratamento de E/S

- After I/O starts, control returns to user program only upon I/O completion.
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access).
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing.
- After I/O starts, control returns to user program without waiting for I/O completion.
  - *System call* – request to the operating system to allow user to wait for I/O completion.
  - *Device-status table* contains entry for each I/O device indicating its type, address, and state.
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt

E/S Síncrona e Assíncrona

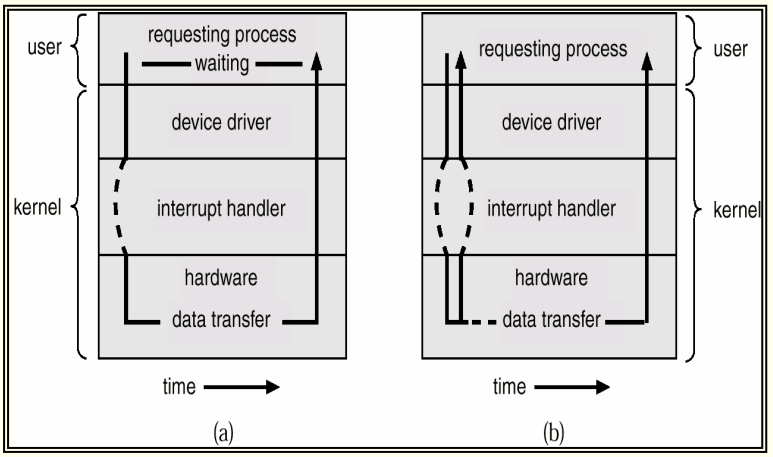
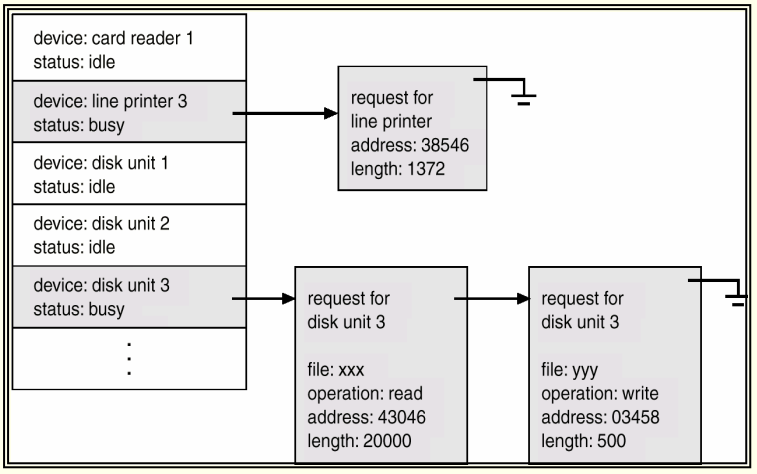


Tabela de Status de cada Device



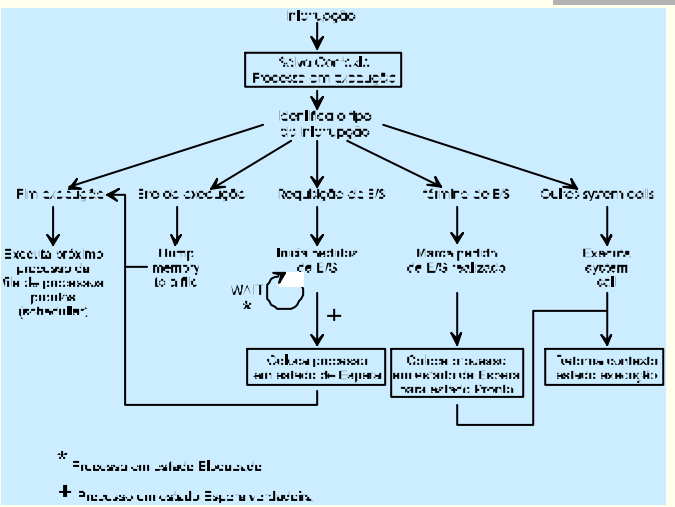
DMA – Direct Memory Access

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only on interrupt is generated per block, rather than the one interrupt per byte.

Conclusão

- *Concluímos então que o SO é um conjunto de programas direcionados por eventos (**event-driven programs**): se não há jobs para executar, nenhum dispositivo de E/S para atender e nenhum usuário para atender o SO permanecerá parado esperando até que algum evento ocorra para ser atendido.*

Operação do SO



SOP – CO009

Estrutura do Kernel

Estrutura do Kernel do SO

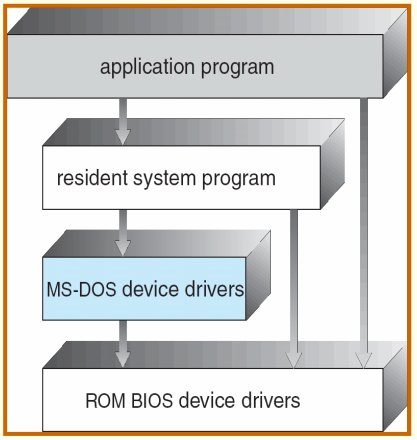
- *O Projeto e a Implementação de um SO é uma tarefa complexa e por isso a sua estrutura interna varia de acordo com a escolha do HW e o tipo de arquitetura.*
- *User* goals and *System* goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- **Policy:** What will be done?
  Mechanism: How to do something ?
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

Estrutura do Kernel do SO

- *A estrutura do **Kernel** do SO pode variar conforme o projeto do SO. Existem basicamente nas seguintes categorias:*
  - ***Simples***
  - ***Em Camadas***
  - ***Monolítico***
  - ***Micro-Núcleo (Micro-Kernel)***
  - ***Máquinas Virtuais***

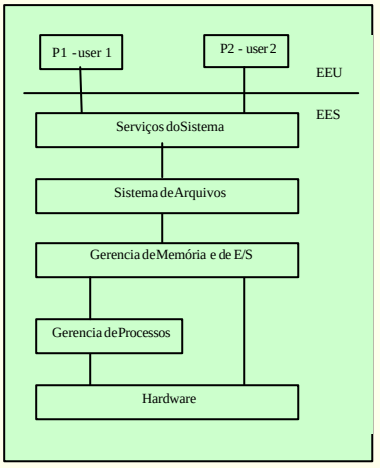
Estrutura do Kernel do SO

- Sistemas Simples
  - MS-DOS – written to provide the most functionality in the least space
    - Not divided into modules
    - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



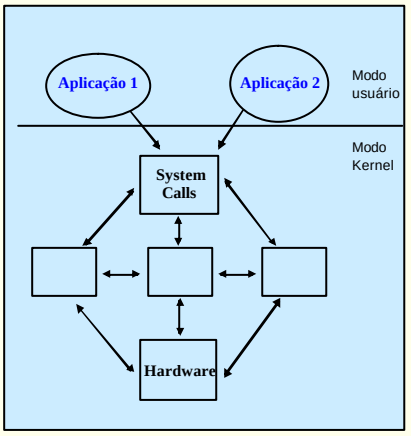
Estrutura do Kernel do SO

- Sistemas Em Camadas
  - *Neste sistema, os programas dos usuários fazem chamadas às camadas mais altas do núcleo, as quais por sua vez, passam adiante o pedido para o serviço apropriado.*
  - *Neste sistema um usuário não pode, ter acesso a uma posição arbitrária de memória ou a uma interface de hardware.*
  - *A vantagem deste sistema é isolar as funções do SO facilitando sua alteração e depuração, além de criar uma hierarquia de níveis de modos de acesso, protegendo as camadas mais internas.*



Estrutura do Kernel do SO

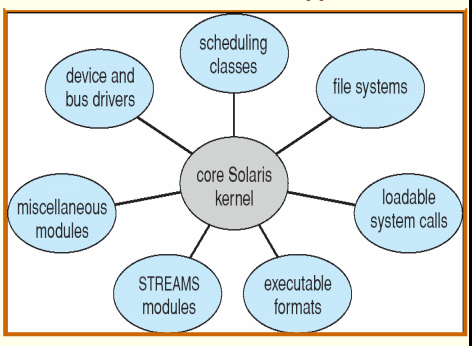
- Sistemas Monolíticos
  - *Nestes sistemas o código do SO reside no EES, que é protegido, e os programas dos usuários residem no EEU.*
  - *A estrutura monolítica consiste de um conjunto de rotinas, organizadas na forma de um grande módulo objeto, que podem interagir livremente umas com as outras. A ausência de uma estrutura hierárquica no Kernel é uma característica.*



Estrutura do Kernel do SO

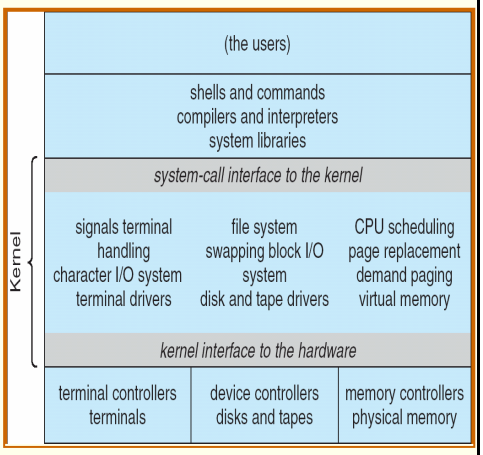
- Sistemas Monolíticos(cont)
  - Most modern operating systems implement kernel modules
    - Uses object-oriented approach
    - Each core component is separate
    - Each talks to the others over known interfaces
    - Each is loadable as needed within the kernel
  - Overall, similar to layers but with more flexible

Solaris Modular Approach



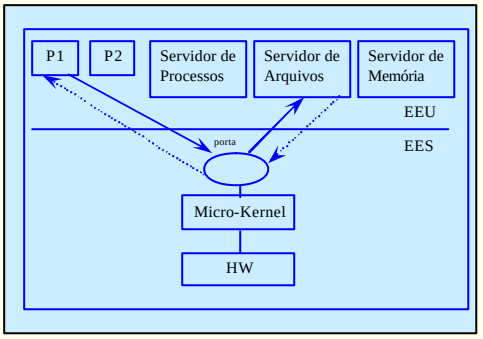
Estrutura do Kernel do SO

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level



Estrutura do Kernel do SO

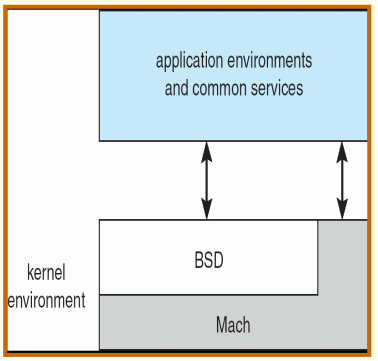
- Sistemas Micro-Kernel
  - *Nestes sistemas a função do Kernel é fornecer uma interface para todo o hardware e gerenciar toda a comunicação entre os serviços que agora residem no **EEU**, e não mais no **EES** como nos outros casos.*
  - *Um processo cliente obtém um serviço pela troca de mensagens com processos servidores através de portas de comunicação (mailbox), mantidos no EES.*



Estrutura do Kernel do SO

- Sistemas Micro-Kernel (cont)
  - Os Micro-Núcleos disponibilizam uma quantidade mínima de serviços chamados serviços essenciais:
    - comunicação entre processos;
    - gerência de memória básica;
    - gerencia de processos
    - escalonamento;
    - entrada/saída de baixo nível.
  - Benefits:
    - Easier to extend a microkernel
    - Easier to port the operating system to new architectures
    - More reliable (less code is running in kernel mode)
    - More secure
  - Detriments:
    - Performance overhead of user space to kernel space communication

Mac OS X Structure



Estrutura do Kernel do SO

- Sistemas Micro-Kernel (cont)
  - Os **serviços não essenciais** são implementados como processos em nível de EEU. Tal enfoque conduz ao projeto de um um núcleo menor, mais confiável, que fornece uma maior facilidade para a adição, alteração e teste de novos serviços.
  - A única vantagem potencial dos **sistemas monolíticos** é em relação a performance, já que a execução de uma system call (que envolve o desvio para a área do SO e execução da rotina de serviço no modo kernel) é mais rápida que enviar mensagens para servidores remotos.

Estrutura do Kernel do SO

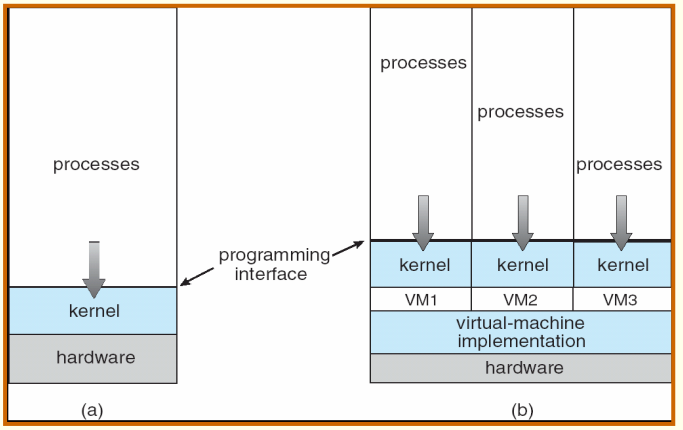
- Máquinas Virtuais
  - A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
  - A virtual machine provides an interface *identical* to the underlying bare hardware
  - The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory

Estrutura do Kernel do SO

- Máquinas Virtuais (cont'd)
  - The resources of the physical computer are shared to create the virtual machines
    - CPU scheduling can create the appearance that users have their own processor
    - Spooling and a file system can provide virtual card readers and virtual line printers
    - A normal user time-sharing terminal serves as the virtual machine operator's console

Estrutura do Kernel do SO

- Máquinas Virtuais (cont'd)

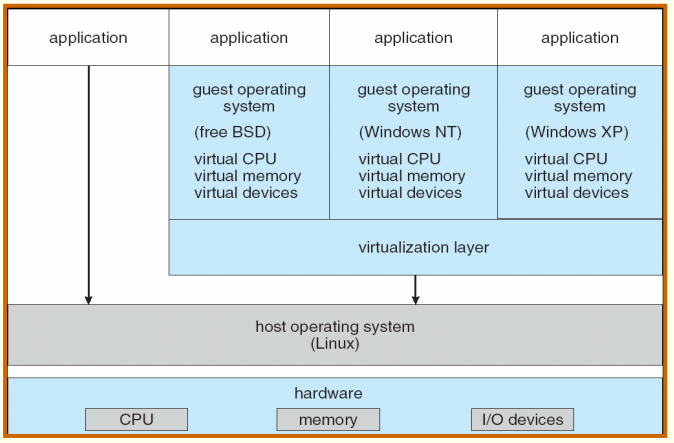


Estrutura do Kernel do SO

- Máquinas Virtuais (cont'd)
  - The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
  - A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
  - The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine

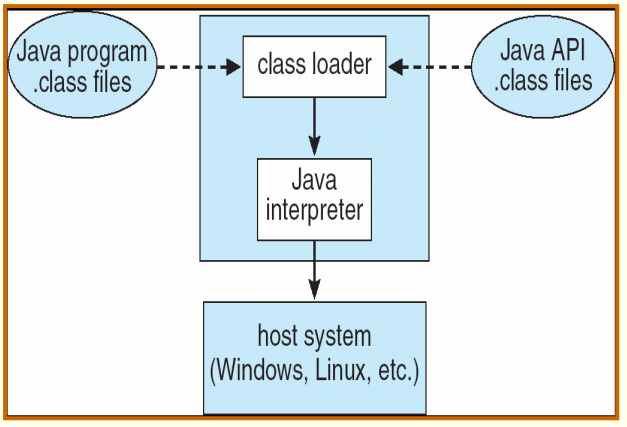
Estrutura do Kernel do SO

- Máquinas Virtuais (cont'd)

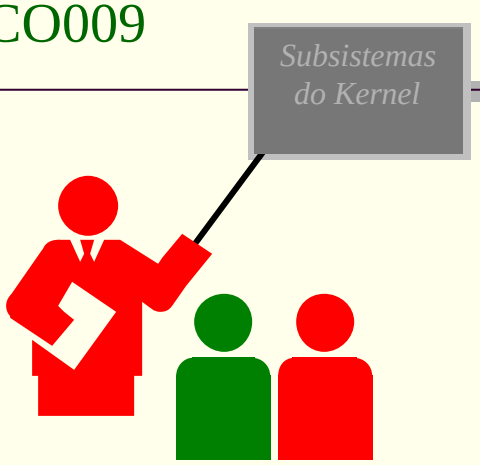


Estrutura do Kernel do SO

- Máquina Virtual Java



SOP – CO009



Subsistemas do Kernel

- Gerência de Processos - The operating system is responsible for the following activities in connection with process management:
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication
  - Providing mechanisms for deadlock handling

Subsistemas do Kernel

- Gerência de Memória
  - All data in memory before and after processing
  - All instructions in memory in order to execute
  - Memory management determines what is in memory when
    - Optimizing CPU utilization and computer response to users
  - Memory management activities
    - Keeping track of which parts of memory are currently being used and by whom
    - Deciding which processes (or parts thereof) and data to move into and out of memory
    - Allocating and deallocating memory space as needed

Subsistemas do Kernel

- Gerência de Memória
  - All data in memory before and after processing
  - All instructions in memory in order to execute
  - Memory management determines what is in memory when
    - Optimizing CPU utilization and computer response to users
  - Memory management activities
    - Keeping track of which parts of memory are currently being used and by whom
    - Deciding which processes (or parts thereof) and data to move into and out of memory
    - Allocating and deallocating memory space as needed

Subsistemas do Kernel

Gerência de Sistema de Arquivos

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and dirs
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media

Subsistemas do Kernel

Gerência de Sistema de E/S

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time.
- Proper management is of central importance. Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

Subsistemas do Kernel

- Gerência de Sistema de E/S
  - One purpose of OS is to hide peculiarities of hardware devices from the user
  - I/O subsystem responsible for
    - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
    - General device-driver interface
    - Drivers for specific hardware devices

Proteção e Segurança

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights