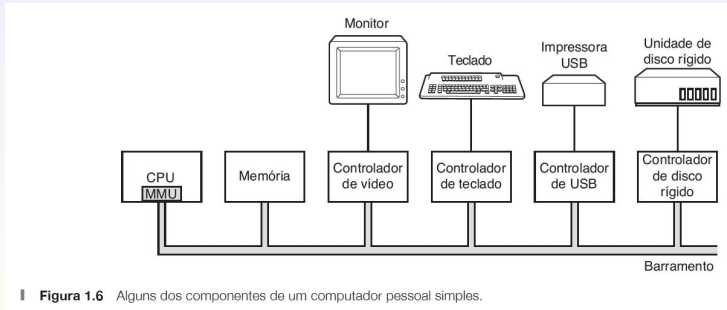






# Sistema Computacional

- Um sistema computacional moderno consiste em:
  - Um ou mais processadores;
  - Memória principal;
  - Diversos dispositivos de entrada e saída.











# Implementando programas

- Programas são implementados para utilizar os recursos do *hardware*;
- Por exemplo, um programa de execução de vídeo:
  - Deve ler arquivos em formato de vídeo
    - Disco rígido;
    - Pen drive;
    - DVD;
    - Disquete;
    - ...
  - Enviá-los para um dispositivo de saída
    - Monitor (CRT, LCD, Tv Plasma, ...);
    - Datashow;
    - Dispositivo de rede (*streaming*);
    - ...



# Implementando programas

- Programas são implementados para utilizar os recursos do *hardware*;
- Por exemplo, um programa de execução de vídeo:
  - Deve ler arquivos em formato de vídeo
    - Disco rígido;
    - Pen drive;
    - DVD;
    - Disquete;
    - ...
  - Enviá-los para um dispositivo de saída
    - Monitor (CRT, LCD, Tv Plasma, ...);
    - Datashow;
    - Dispositivo de rede (*streaming*);
    - ...

## Problema 1

Como garantir o correto funcionamento do programa independente do recurso de hardware disponível?

# Implementando programas

- Além disso, os recursos do sistema são compartilhados:
- Outros programas podem querer acessar o disco enquanto o vídeo está executando:
  - Fazendo um *download*;
  - Convertendo um vídeo de .avi para .rmvb;
  - ...

# Implementando programas

- Além disso, os recursos do sistema são compartilhados:
- Outros programas podem querer acessar o disco enquanto o vídeo está executando:
  - Fazendo um *download*;
  - Convertendo um vídeo de .avi para .rmvb;
  - ...

## Problema 2

Como garantir o correto funcionamento do programa nesse verdadeiro “ecossistema computacional”?



# Gerenciando um sistema computacional

Como seria o correto funcionamento de um sistema computacional?

- ① A execução de programas de usuários;
- ② Permitindo a solução de problemas
  - Compartilhando recursos;
  - Resolução de conflitos;
  - Controle de permissões;
  - Controle de acesso;
  - ...

Para gerenciar tudo isso, existe uma camada de software:

**Sistema Operacional**

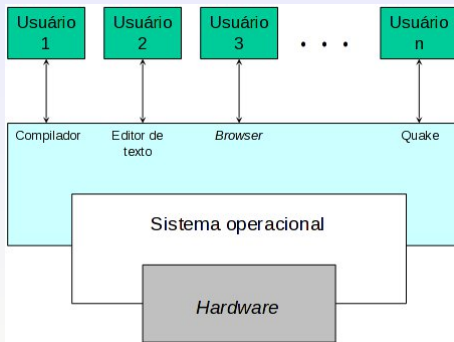
# Sistema Operacional

Programas
Sistema Operacional
Hardware



# O que é um Sistema Operacional?

Uma visão mais detalhada:





# O que é um Sistema Operacional?

Com a palavra, Mr. Linus Torvalds.



Fonte: Vídeo *Revolution OS*, 2001.  
<http://www.revolution-os.com/>

# Sistema operacional: conceito

- Um programa que controla a execução de programas aplicativos
- Interface entre aplicativos e o hardware
- Duas formas de ver um sistema operacional:
  - Alocador de recursos
    - Gerencia todos os recursos
    - Decide entre requisições conflitantes para uso eficiente e justo dos recursos
  - Programa de controle
    - Controla execução de programas para prevenir erros e usos indevidos do computador

# Sistema operacional: conceito

- Não há um termo universalmente definido
  - “Tudo aquilo que um vendedor entrega quando você pede um sistema operacional” é uma boa aproximação
- Uma corrente afirma que o SO é o **Kernel**
  - “O” programa que fica executando durante todo o funcionamento do computador
  - O coração do sistema
  - Tudo o mais que executa são programas de sistema ou programas aplicativos
- Outra define um SO como um conjunto de aplicativos, dentre eles, o kernel

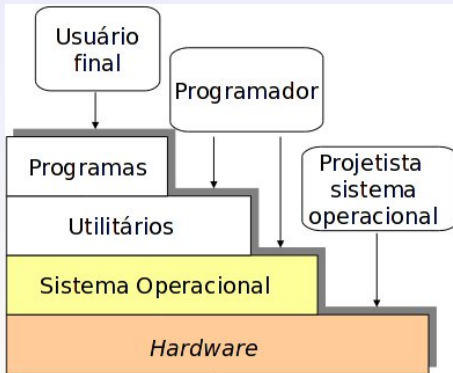
# Funções de um Sistema Operacional

- Máquina estendida:
  - Interface entre programas e o hardware;
  - Permitindo uma *interface* genérica de acesso a diversos dispositivos.
- Gerenciador de recursos:
  - Um programa que controla a execução de outros programas;
  - Permite que múltiplos programas sejam executados ao mesmo tempo;
  - Gerencia e protege a memória, os dispositivos de entrada e saída e outros recursos;



# Interfaces do Sistema

Uma visão das interfaces com o sistema computacional:











# Breve Histórico dos Sistemas Operacionais

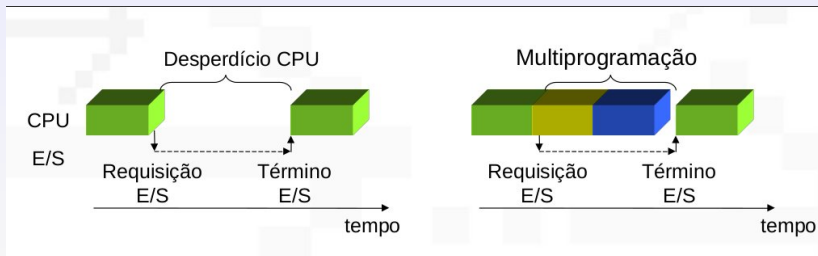
- Monitor residente
  - Evolução:
    - Sequenciamento automático de jobs, transferindo o controle de um job a outro
    - Primeiro sistema operacional (rudimentar)
  - Monitor residente:
    - Programa que fica permanentemente em memória
    - Execução inicial
    - Controle é transferido para o job (Cartões de controle)
    - Quando o job termina, o controle retorna ao monitor
    - Centraliza as rotinas de acesso a periféricos disponibilizando aos programas de usuário

# Breve Histórico dos Sistemas Operacionais

- Sistema batch multiprogramados (multitarefa)
  - Monitor residente permite a execução de apenas um programa a cada vez
  - Desperdício de tempo de CPU com operações de E/S
  - Evolução:
    - Manter diversos programas na memória ao mesmo tempo
    - Enquanto um programa realiza E/S, outro pode ser executado
- Multiprogramação:
  - Manter mais de um programa em “execução” simultaneamente
  - Duas inovações de hardware possibilitaram o surgimento da multiprogramação
    - Interrupções (Sinalização de eventos)
    - Discos magnéticos: Acesso randômico a diferentes jobs (programas) no disco. Melhor desempenho em acessos de leitura e escrita

# Breve Histórico dos Sistemas Operacionais

- Exemplo de multiprogramação:





# Sistemas monousuário e multiusuário

- Sistemas monousuário
  - Projetados para serem usados por um único usuário
    - e.g.: MS-DOS, Windows 3.x, Windows 9x, Millenium
- Sistemas multiusuário
  - Projetados para suportar várias sessões de usuários em um computador
    - e.g.: Windows NT (2000), UNIX

# Sistemas multitarefa e monotarefa

- Sistemas monotarefa
  - Capazes de executar apenas uma tarefa de cada vez
    - e.g.: MS-DOS
- Sistemas multitarefas:
  - Capazes de executar várias tarefas simultaneamente
  - Existem dois tipos de sistemas multitarefa:
    - Não preemptivo/cooperativo (e.g.: Windows 3.x, Windows9x (aplicativos 16 bits))
    - Preemptivo (e.g.: Windows NT, OS/2, UNIX, Windows9x (aplicativos 32 bits))

# Serviços do Sistema Operacional

- Um conjunto de serviços do sistema operacional fornece funções que são úteis ao usuário:
  - **Interface com o Usuário** – Quase todos os sistemas operacionais possuem uma interface com o usuário (UI)
  - **Execução de Programas** – O sistema deve estar apto a carregar um programa na memória e executá-lo, terminar a execução, seja normalmente ou de forma anormal (indicando o erro)
  - **Operações de E/S** – Um programa em execução pode requisitar E/S, o que poderá envolver um arquivo ou um dispositivo de E/S.
  - **Manipulação de Sistemas de Arquivos** – O sistema de arquivo é de especial interesse. Obviamente, programas necessitam ler e escrever arquivos e diretórios, criar e deletar, procurar, listar informações de arquivos e gerenciar permissões.



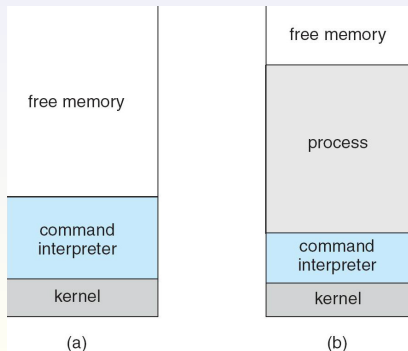
# Os serviços do Sistema Operacional

- Criação de programas:
  - Editores, depuradores, compiladores.
- Execução dos programas:
  - Carga de programas em memória.
- Acesso a dispositivos de E/S.
- Controle de acesso a arquivos.
- Acesso a recursos de sistema:
  - Proteção entre usuários.
- Estatísticas.
- Detecção de erros:
  - *Hardware*: erros de memória, falha em dispositivos de E/S, ...
  - *Software*: overflow, acesso não autorizado a posições de memória, segmentation fault ...

# Execução de um SO

- Inicialização do sistema

- Programa de *Bootstrap*: É carregado durante a ligação(boot)/reinicialização(reboot) do sistema
- Em geral é armazenado em ROM, conhecido como Firmware (BIOS)
- Inicializa todos os aspectos do sistema
- Carrega o kernel do SO e inicia sua execução

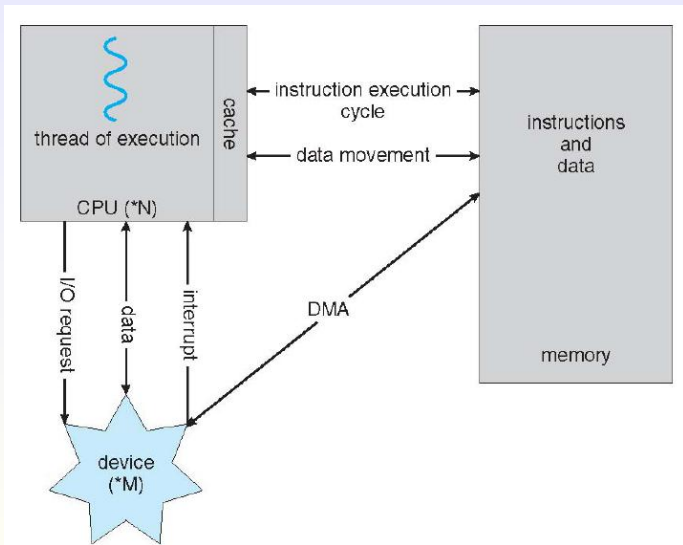


# Execução de um SO

- Sistema em execução

- CPU e controladores de dispositivos se comunicam através de um barramento comum, tendo acesso à memória compartilhada
- Dispositivos de I/O e CPU podem executar concorrentemente
- Cada controlador se encarrega de um tipo de dispositivo
- Cada controlador de dispositivo possui um *buffer* local
- CPU move dados de/para a memória principal de/para os buffers locais
- I/O move do dispositivo para o buffer local do controlador
- Controladores informam à CPU que suas operações terminaram por meio de interrupções
- Alguns dispositivos possuem acesso direto à memória (DMA)

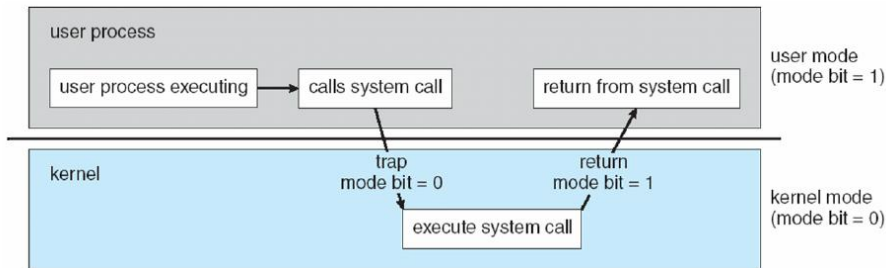
# Como funciona um sistema computacional moderno



# Operações de um sistema

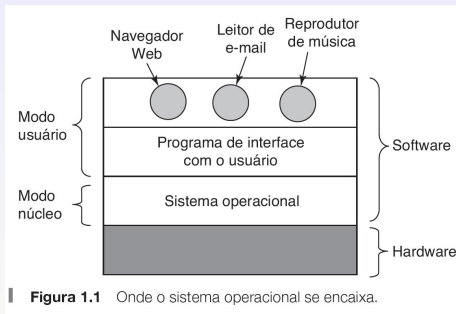
- Interrupções são tratadas por hardware
- O SO opera em *dual-mode* para sua proteção
  - Modo de **usuário** e modo **kernel**
  - Modo usuário: executa código de usuário e possui menor prioridade
  - Modo kernel: executa códigos de sistema, permitindo operações privilegiadas
    - Acesso a dados no disco
    - Gerenciar memória
    - Liberar espaço de memória utilizado por um processo
    - ...
  - Algumas instruções (Assembly) são definidas como privilegiadas, outras não

# Transição dos modos de operação



# Onde está o Sistema Operacional?

O Sistema Operacional na memória do computador:



## O Sistema Operacional na memória do computador:



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

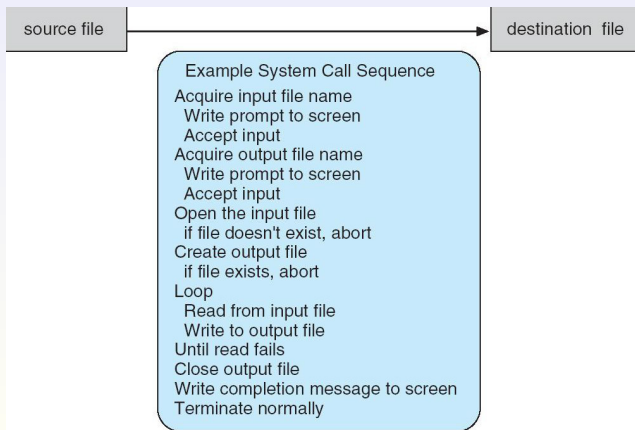


# Chamadas de Sistema

- Forma que programas solicitam **serviços** ao sistema operacional;
- Interface de programação aos serviços fornecidos pelo SO
- Tipicamente escritos em uma linguagem de alto nível (C or C++)
- Análogo a sub-rotinas (funções);
  - Transferem controle para o sistema operacional,
  - Ao invés de transferir para outro ponto do programa.
- É o núcleo (*kernel*) do sistema operacional que implementa as chamadas de sistema:
  - As chamadas de sistema mudam o modo de operação para modo kernel
  - Executam suas instruções
  - Com o retorno da chamada retorna o modo para usuário
- APIs mais comuns são Win32 API para Windows, POSIX API para sistemas baseados em POSIX (incluindo virtualmente todas as versões de UNIX, Linux, e Mac OS X)

# Chamadas de Sistema

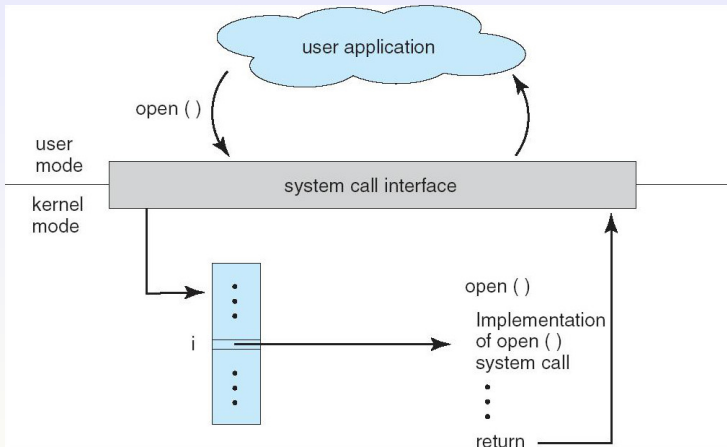
- Sequência de eventos em uma chamada de sistema para copiar o conteúdo de um arquivo em outro



# Chamadas de Sistema

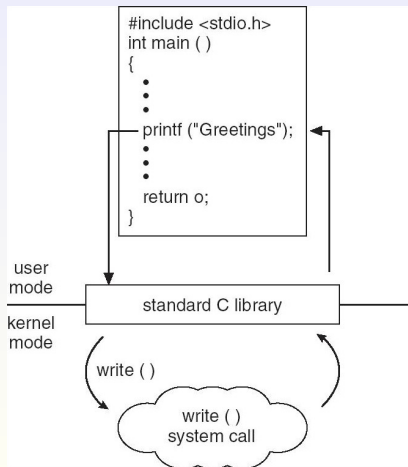
- Tipicamente, um número é associado com cada chamada de sistema
  - A interface das chamadas de sistema mantém uma tabela indexada de acordo com esses números
- A interface das chamadas de sistema evoca a chamada de sistemas pretendida no kernel do SO e retorna o status e quaisquer valores de retorno
- O chamador não precisa saber nada sobre a implementação da chamada de sistemas
  - Só precisa obedecer a API e entender o que o SO irá realizar em resposta à chamada
  - Grande parte dos detalhes da interface do SO são escondidas dos programadores pela API

# Chamadas de Sistema



# Chamadas de Sistema

- Programa em C evocando a chamada de biblioteca *printf()*, que executa a chamada de sistemas *write()*



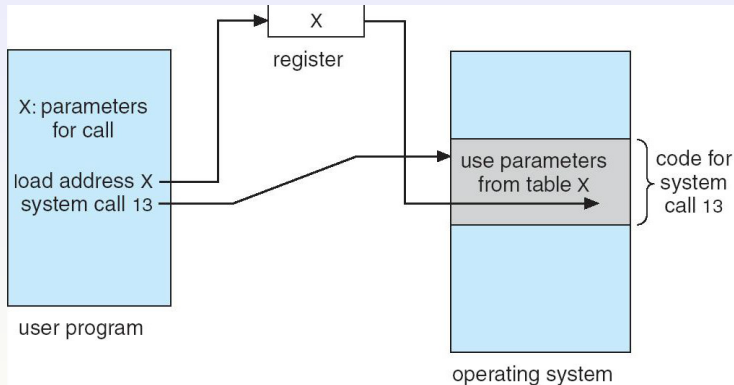
# Chamadas de Sistema

## Passagem de Parametros nas Chamadas de Sistema

- Três métodos gerais são usados para passar parâmetros ao SO
  - Mais simples: passar parâmetros em registradores
    - em alguns casos, pode existir mais parâmetros que registradores
  - Parâmetros armazenados em um bloco, ou tabela, na memória, e o endereço do bloco é passado como parâmetro em um registrador
    - Essa abordagem é utilizada pelo Linux e Solaris
  - Parâmetros colocados na pilha (empilhados / push) pelo programa e removidos (desempilhados / pop) desta pelo sistema operacional
  - Métodos de bloco e pilha não limitam o número ou tamanho dos parâmetros que estão sendo passados

# Chamadas de Sistema

- Passagem de Parametros via Tabela



- Tipos de chamadas de sistema:

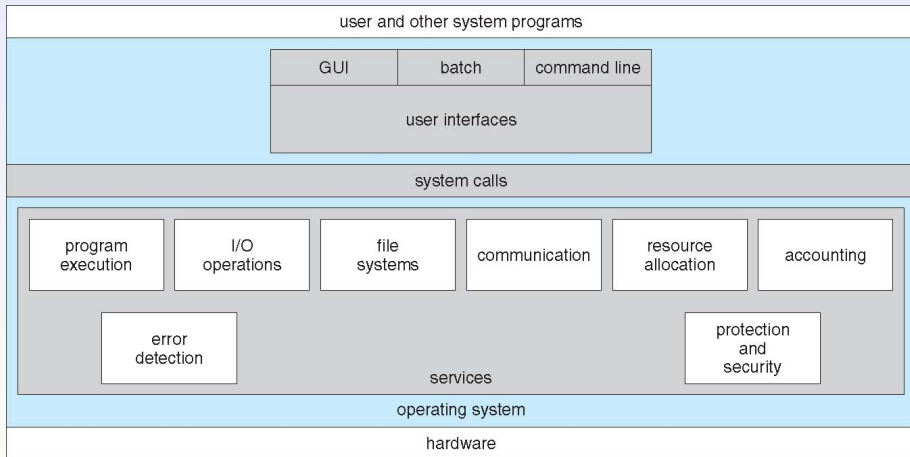
- Controle de processos
  - Associadas à gerência do processador
- Gerência de memória
- Gerenciamento de arquivos
- Gerenciamento de dispositivos de entrada/saída.
- Manutenção de Informações
- Comunicações
- Proteção



# Chamadas de Sistema

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

# Interface entre Usuário e Sistema Operacional



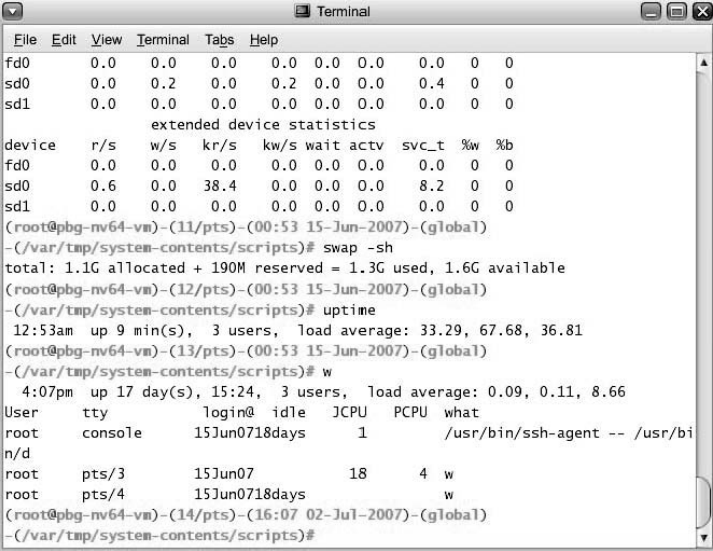
# Interface entre Usuário e Sistema Operacional

- CLI (*Command Line Interface* – Interface de Linha de Comando) permite a entrada de comandos diretos
  - Algumas vezes implementada no kernel, outras por programas de sistemas
  - Algumas vezes várias alternativas implementadas – shells
  - Basicamente obtém um comando do usuário e o executa
    - Algumas vezes comandos internos, Algumas vezes somente nomes de programas (externos)
    - No último caso, a adição de novas características não requer modificação do shell

# Interface entre Usuário e Sistema Operacional

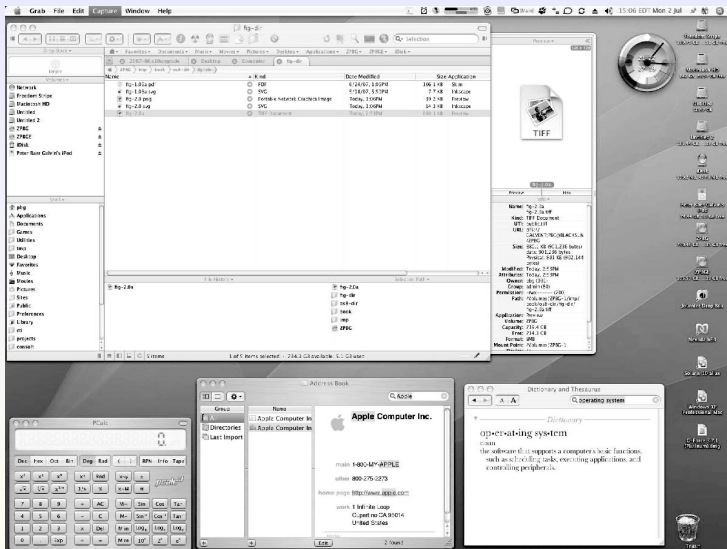
- GUI (*Graphical User Interface* – Interface Gráfica com o Usuário)
  - Interface com área de trabalho amigável (User-friendly desktop)
  - Normalmente mouse, teclado e monitor
  - Ícones representando arquivos, programas, ações, etc.
  - Cliques no mouse em objetos da interface causam ações variadas (obter informações, opções, funções de execução, abertura de diretório – conhecido como pasta)
  - Inventado no Xerox PARC
- Muitos sistemas hoje incluem tanto interface CLI como GUI
  - Microsoft Windows é GUI com CLI “command” shell
  - Apple Mac OS X tem a interface “Aqua” GUI com um kernel UNIX abaixo e shells disponíveis
  - Solaris é CLI com interfaces GUI opcionais (Java Desktop, KDE)

# Interface entre Usuário e Sistema Operacional



```
File Edit View Terminal Tabs Help
fd0      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
sd0      0.0    0.2    0.0    0.2  0.0  0.0    0.4  0  0
sd1      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
          extended device statistics
device   r/s    w/s    kr/s    kw/s wait actv  svc_t  %w  %b
fd0      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
sd0      0.6    0.0   38.4    0.0  0.0  0.0    8.2  0  0
sd1      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
12:53am up 9 min(s), 3 users, load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
4:07pm up 17 day(s), 15:24, 3 users, load average: 0.09, 0.11, 8.66
User      tty          login@ idle   JCPU   PCPU   what
root      console      15Jun07 18days    1      /usr/bin/ssh-agent -- /usr/bi
n/d
root      pts/3        15Jun07    18        4    w
root      pts/4        15Jun07    18        4    w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```

# Interface entre Usuário e Sistema Operacional



# O papel dos temporizadores

- Todo o sistema baseia-se na contagem de um relógio (*clock*)
- O SO faz uso de temporizadores para manter o controle da CPU
- Interrupções de relógio:
  - Interrompem a operação do computador, dando o processamento ao SO (kernel)
  - Utiliza um mecanismo de *watchdog*
    - Um temporizador é decrementado a cada intervalo de tempo
    - Quando chegar a 0, uma interrupção é gerada
  - Útil para impedir que um programa de usuário fique em looping
  - Permite que o SO possa gerenciar os programas em execução

# Tipos de Sistemas Operacionais

O zoológico de Sistemas Operacionais:

- Sistemas operacionais de computadores de grande porte;
- Sistemas operacionais de servidores;
- Sistemas operacionais de multiprocessadores;
- Sistemas operacionais de computadores pessoais;
- Sistemas operacionais de computadores portáteis;
- Sistemas operacionais embarcados;
- Sistemas operacionais de nós sensores;
- Sistemas operacionais de tempo real;
- Sistemas operacionais de cartões inteligentes (smart card);
- Sistemas operacionais distribuídos;
- ...



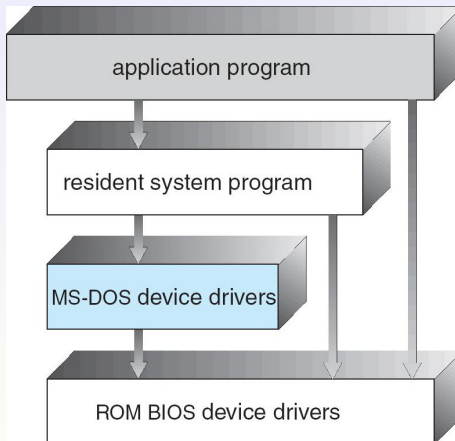
# Exemplos de SOs

## MS-DOS (*Disk Operating System*)

- Originalmente projetado e implementado por pessoas que não imaginavam que ele se tornaria tão popular
- Escrito para fornecer a maior funcionalidade no menor espaço
  - Não é dividido em módulos
  - Apesar do MS-DOS ter alguma estrutura, sua interface e seus níveis de funcionalidade não são bem separados
- Sistema estruturado em camadas
  - Um sistema operacional é dividido em um número de camadas (ou níveis), cada uma construída no topo das camadas abaixo. A camada mais inferior (camada 0) é o hardware; A camada de mais alto nível (camada N) é a interface com o usuário.
  - Com modularidade, camadas são selecionadas de forma que cada uma use as funções (operações) e serviços somente das camadas de mais baixo nível.

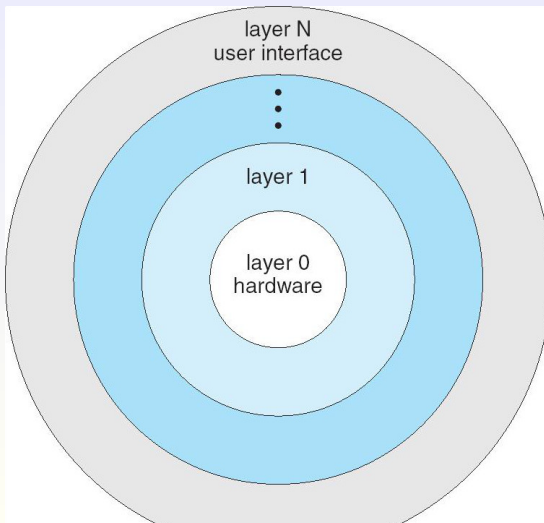
# Exemplos de SOs

- Estrutura em Camadas do MS-DOS



# Exemplos de SOs

## Sistema operacional em camadas



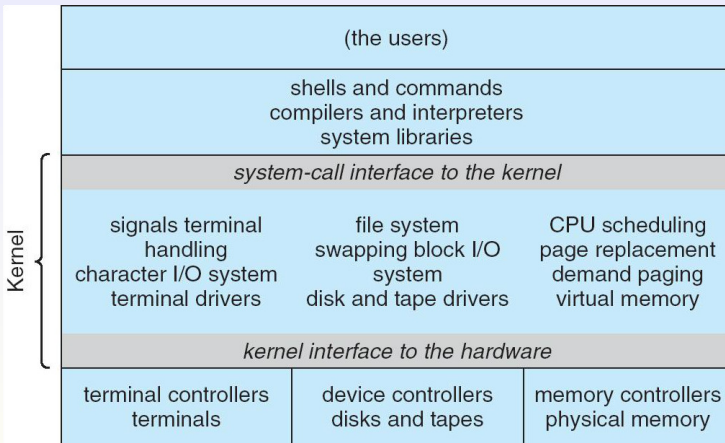
# Exemplos de SOs

## UNIX

- Limitado pela funcionalidade do hardware, o sistema operacional UNIX original tinha estrutura limitada.
- O SO UNIX consiste de duas partes separáveis
  - Programas de Sistemas
  - O kernel (**monolítico**)
    - Consiste de tudo abaixo da interface de chamadas de sistemas e acima do hardware físico
    - Fornece o sistema de arquivos, escalonamento da CPU, gerência de memória e outras funções do sistema operacional
    - Incorpora um grande número de funções

# Exemplos de SOs

## Estrutura do Sistema UNIX



# Exemplos de SOs

## Estrutura Microkernel

- Move tanto quanto possível do kernel para o espaço do “usuário”
- Comunicação ocorre entre módulos em nível usuário usando troca de mensagens (message passing)
- Benefícios:
  - Facilidade de estender um microkernel
  - Facilidade de portar o sistema operacional para novas arquiteturas
  - Mais confiabilidade (menos código está executando em modo kernel)
  - Mais seguro
- Desvantagem:
  - Sobrecarga causada pela comunicação entre o modo usuário e o modo kernel

# Exemplos de SOs

## Módulos

- Grande parte dos sistemas operacionais modernos implementam módulos no kernel
  - Usa a abordagem orientada a objetos
  - Cada componente chave é separado
  - Cada módulo se comunica com outra através de interfaces conhecidas
  - Cada módulo é carregado no kernel quando necessário
  - Resumindo, similar à estrutura em camadas porém mais flexível

## Estrutura Modular do Solaris

