Engenharia de Computação

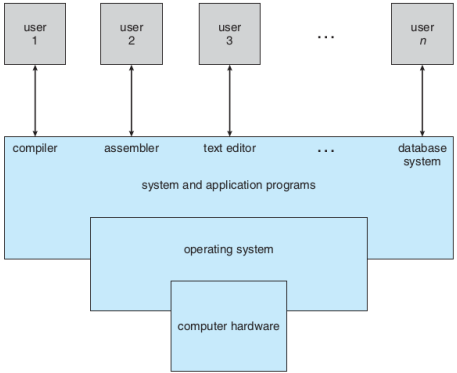
ECM225 – Sistemas Operacionais

Introdução aos Sistemas Operacionais

Slides da disciplina ECM225 – Sistemas Operacionais
Curso de Engenharia de Computação
Instituto Mauá de Tecnologia
Prof. Marco Antonio Furlan de Souza

- Um **sistema operacional provê** um **meio** para a **utilização apropriada** dos **recursos** de um **sistema computacional**;
- Pode-se entender um **sistema computacional** como um **sistema composto** por:

  - **Hardware**
  - **Sistema Operacional**
  - **Programas de aplicação**
  - **Usuários**



- **Visão do usuário**
  - **Usuário de PC ou computador desktop**: o sistema é projetado para um usuário **monopolizar** seus **recursos**, maximizando o trabalho que o usuário está executando – o **sistema operacional** é **projetado principalmente** para **facilidade de utilização** e depois para **desempenho**;
  - **Usuário de mainframe/minicomputador**: podem existir **muitos usuários** usando o sistema **simultaneamente**, por **terminais diferentes**. Esses usuários **compartilham recursos** e podem **trocar informações**. O sistema operacional é projetado para **maximizar a utilização de recursos**, e garantir que o **tempo de CPU**, **memória** e **entrada e saída (E/S)** sejam utilizados **eficientemente** e que **nenhum usuário** tenha **mais recursos** que sua **parte justa**.

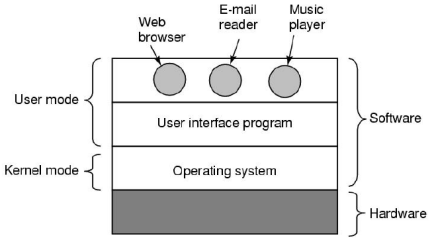
- **Visão do usuário**
  - **Usuário de computação móvel**: são unidades que operam sozinhas para **usuários individuais**. Normalmente estão **conectadas** a uma **rede** (com ou sem fio) e executam **computações mais simples** por causa de **limitações de energia**, **desempenho** e **interface**. Seus sistemas operacionais são projetados para **maximizar a usabilidade** e **desempenho do consumo de energia**.
  - **Computadores sem interface de usuário**: são computadores que **operam** normalmente **sem a interferência de um usuário**. São os **sistemas embarcados** em dispositivos residenciais, automotivos e aviônica, por exemplo.

- **Visão do sistema**
  - Sob o **ponto de vista do computador**, um **sistema operacional** está intimamente **ligado** ao **hardware**;
  - Desse modo, ele pode ser entendido como um **gerenciador de recursos**: tempo de CPU, espaço de memória, espaço de sistema de armazenamento, dispositivos de E/S, etc;
  - Ele recebe inúmeras **requisições** de recursos – muitas vezes **conflitantes** – e **decide** como **alocá-las** a **usuários** e **programas**, de modo que o sistema possa **operar** de modo **eficiente** e **justo**;
  - Um sistema operacional também pode ser visto como um **programa de controle**, que **gerencia** a **execução concorrente de programas** e **controla** a utilização dos **diversos dispositivos** de **E/S**;
  - Isso inclui a **multiplexação** (compartilhamento) de **recursos** de duas formas:
    - **No tempo**: permite que **programas** ou **usuários** utilizem recursos **alternadamente**;
    - **No espaço**: permite o **acesso simultâneo** a "partes" de um **mesmo recurso** (por exemplo, a memória principal).

- Por causa da **variabilidade** de **programas** que são incluídos por **produtores** de sistemas operacionais, **não há um consenso** de como **definir unicamente** o que é um sistema operacional;
- Uma **definição comum** é a seguinte:

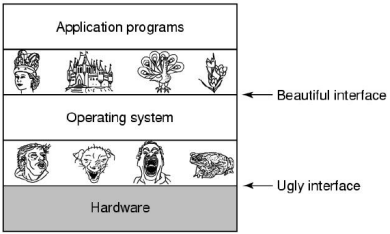
  Um sistema operacional é o programa que sempre é executado em um computador e é denominado de ***kernel****.*
- Além do *kernel*, existem duas outras **categorias de programas**:
  - **Programas de sistema**: são programas **associados ao sistema operacional**, mas **não** são **parte** integrante do ***kernel***;
  - **Programas de aplicação**: inclui **todos** os programas que **não são** parte do ***kernel*** **e** também não são **programas** de **sistema**.

- O sistema operacional abrange softwares que podem operar de **dois modos de operação**:
  - **Modo *kernel***: programas neste modo possuem **acesso completo ao hardware**;
  - **Modo usuário**: programas (aplicação e sistema) neste modo possuem **restrições** ao **acesso completo ao hardware**.

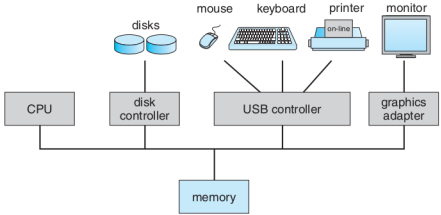


- **Vantagens** da separação em modos:
  - Programas do **usuário** devem seguir uma **interface de funções** provida pelo sistema operacional para **acessar corretamente o hardware**, evitando erros;
  - **Não** é **necessário escrever código** para **acessar** o **hardware** – ele já está pronto no *kernel* (na forma de *device drivers*).

- A separação de modos acaba por definir **dois tipos de interface**:
  - **Provida pelo sistema operacional**: funções que o *kernel* disponibiliza para os **programadores** de **aplicação** e **sistema** para poder realizar **operações** que necessitam **acesso** ao **hardware**, por exemplo, operar com arquivos, alocar memória, ler teclado, escrever na tela etc, bem como bibliotecas de linguagens de programação alto nível;
  - **Interface com o hardware**: o *kernel* precisa **acessar** o **hardware** de modo a **utilizar** os **recursos** do computador. Para isso, existem **funções** internas ao **kernel** que são **padronizadas** para **acessar** diversas **categorias** de **hardware** (E/S, memória, disco, rede, etc). Essas funções dependem de *device drivers* específicos para concluir suas operações.



- Um **computador de propósito geral** consiste em uma ou mais CPUs (CPU – *central processing unit*) e um conjunto de **controladores de dispositivos** conectados por um ***bus* comum** (vias que trafegam sinais de controle e dados) que permite acesso a uma **memória compartilhada**:

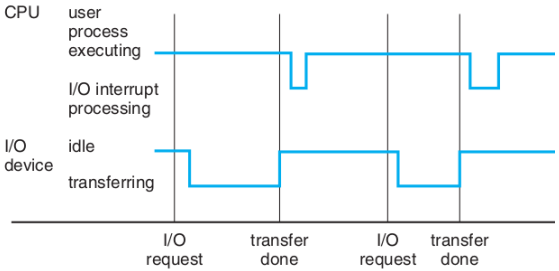


  - **Controlador de dispositivo**: cada controlador de dispositivo é **responsável** por um **tipo** de **dispositivo** (*drive* de disco, dispositivo de áudio, controlador de vídeo, etc);
  - **Controladores** de dispositivo **operam concorrentemente** com a **CPU** – eles **competem** pela utilização de **ciclos de acesso à memória** e necessitam ser **sincronizados** nesse acesso.

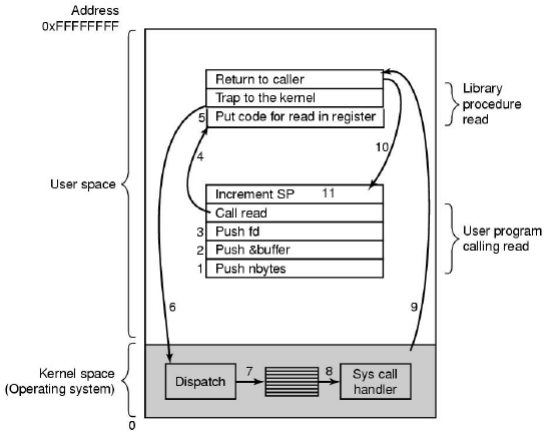
- Quando um **computador** é **ligado**, um **programa inicial** precisa ser executado – o ***programa de bootstrap***. É um programa simples, armazenado em uma memória **não volátil ROM** (*read-only memory*) ou em uma memória **não volátil EEPROM** (*electrically erasable programmable read-only memory*), denominado genericamente de ***firmware***;
- O programa de ***bootstrap*** deve saber como **carregar** o ***kernel*** na memória e como **iniciar a execução do sistema**. **Este primeiro processo a ser executado** é denominado de **init** (em UNIX/Linux é este o nome mesmo!);
- O **init** então **aguarda** a **ocorrência** de algum **evento**.

- A ocorrência de um **evento** é **sinalizada** por uma **interrupção**, que pode ser de **software** ou **hardware**. Uma **Interrupção de software** também é conhecida por **trap**;
- Um **hardware** pode **desencadear** uma **interrupção** a qualquer momento **enviando um sinal à CPU**;
- Um **software** pode **desencadear** um ***trap*** a qualquer momento **executando um procedimento especial** denominado **chamada de sistema** (*system call*), que permite executar com segurança algum código do *kernel* para acessar o hardware (executar **instruções privilegiadas**);

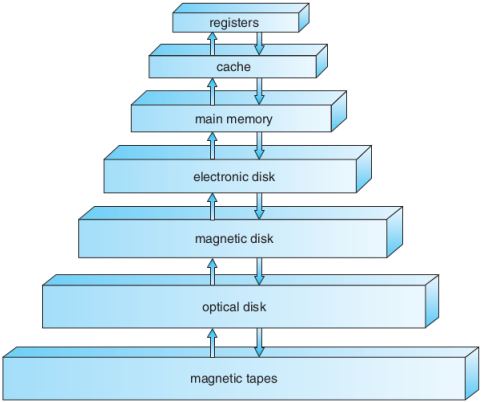
- Quando ocorre uma **interrupção de hardware**, a CPU **suspende** o que está **executando** e então **transfere** a execução para uma **localização fixa** que contém o **endereço** onde a **rotina de serviço para a interrupção** (ISR – *interrupt service routine*) está localizada;
- A **ISR** é então executada e ao **terminar** o **controle** é **retornado** para o **execução** que **anteriormente** havida sido **suspensa**;
- As localizações das **interrupções dos dispositivos** normalmente são armazenadas em um **vetor de interrupções**.



- Um ***trap*** ocorre quando uma **chamada de sistema é realizada**. Por exemplo, no Linux, quando um programa deseja ler dados de um arquivo, ele executa uma função da biblioteca padrão tal como `read()`;
- A função `read()` executa uma **chamada de sistema**, que efetivamente realiza a leitura do arquivo;
- Uma função **ISR** é então no espaço do kernel e, ao **terminar**, o **controle** é **retornado** para a **execução** que **anteriormente** havida sido **suspensa**.



- A **estrutura de armazenamento** de um computador segue uma **hierarquia**, onde no **topo** está o dispositivo **mais rápido e mais caro (custo/byte)** e na **base** está o dispositivo **mais lento e mais barato (custo/byte)**:



- **Quantidades de armazenamento em Computação**[1]

Nome	Unidade	Equivale a	Tamanho (em bytes)
Bit	bit	1 Bit	1/8
Nibble	–	4 Bits	1/2
Byte	B	8 Bits	1
Kilobyte	kB	1.024 Bytes	1.024
Megabyte	MB	1.024 Kilobytes	1.048.576
Gigabyte	GB	1.024 Megabytes	1.073.741.824
Terabyte	TB	1.024 Gigabytes	1.099.511.627.776
Petabyte	PB	1.024 Terabytes	1.125.899.906.842.624
Exabyte	EB	1.024 Petabytes	1.152.921.504.606.846.976
Zetabyte	ZB	1.024 Exabytes	1.180.591.620.717.411.303.424
Iotabyte	YB	1.024 Zettabytes	1.208.925.819.614.629.174.706.176

[1] O correto seria utilizar prefixos já em potências de 2: KiB, MiB etc. Consulte: <https://pt.wikipedia.org/wiki/Prefixo_binário>

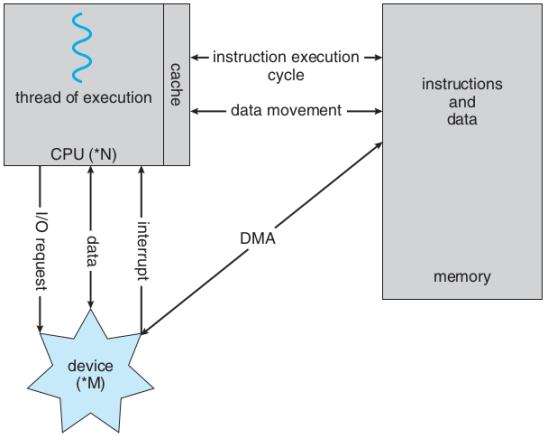
- Os **registradores** são elementos de armazenamento presentes **internamente** na **CPU**. A **capacidade** de **armazenamento** de um registrador depende da CPU, mas tipicamente são 64 bits e o **tempo de acesso** é da ordem de **poucos nanossegundos**;
- As **memórias cache** são utilizadas para **intermediar** a **transferência** de dados entre a **memória principal** (mais lenta) e os **registradores** (mais rápidos). Possuem **tempo de acesso** maior que a dos registradores, mas ainda na **ordem** de **nanossegundos**. São **organizadas em níveis** (L1, L2 e L3), sendo que os caches L1 e L2 normalmente são embutidos na CPU e o cache L3 pode ser ou não embutido na CPU. O **tamanho** dos caches **varia** de poucos **kilobytes** a alguns **megabytes**.

- A **memória principal** é uma **memória volátil** que utiliza tecnologia de semicondutor **DRAM** (dynamic random access memory);
- Este tipo de memória (e de outros tipos também) pode ser considerada como um **vetor de palavras** (uma **palavra** é um **conjunto contíguo de bytes** que depende da arquitetura do computador – **de 1 a 8 bytes**), sendo que cada **palavra** possui um **único endereço** (cada **endereço representado** tipicamente por um **número** de **32 ou 64 bits** – depende da arquitetura);
- Durante a **execução de um programa**, **dados** são **lidos** de **registradores da CPU para endereços da memória principal** (operação `store`) e de **endereços da memória principal para registradores da CPU** (operação `load`);
- Em um tipo de **arquitetura de computador** chamada de **Von Neumann** a **memória principal** contém tanto **dados** quanto **instruções** de um programa. Em outro tipo de arquitetura denominada de **Harvard**, as **memórias** para **dados** e **instruções** de programas são **separadas**.

- A memória principal possui tamanhos que variam tipicamente de alguns gigabytes (para computadores desktop e notebooks) até centenas de gigabytes para servidores. No entanto, por causa da **volatilidade dos dados** e também do **espaço consumido** pelos programas e dados, é **necessário ampliar** o espaço de **armazenamento** com algum **armazenamento secundário**, composto por dispositivos não voláteis de **grande capacidade**;
- **discos magnéticos** são os dispositivos de armazenamento secundário **mais comuns**. Permitem **armazenar** tanto **dados** quanto **programas**. Os programas que estão em um disco magnético precisam ser **carregados** na **memória principal** para serem **executados**;
- Por serem **baratos** em **relação à memória principal**, não há **limites** na **quantidade** de discos que se deseja instalar em um sistema computacional – hoje é muito comum que um notebook tenha um dico de 1 TB de espaço total.

- **Dispositivos de entrada e saída** (**E/S**) se comunicam com a CPU por meio das **vias de dados comuns** (*bus*);
- Cada **controlador de dispositivo trata** de **um tipo específico** de **dispositivo**;
- É muito comum que **um mesmo controlador** consiga controlar **vários dispositivos**. Para se obter isso, os dispositivos e o controlador devem seguir algum **protocolo**;
- Por exemplo, existem **controladores SCSI** (***small computer-systems interface***) que permitem a conexão de diversos dispositivos simultaneamente;
- Outro exemplo é o **protocolo USB** (***universal serial bus***) que permite interligar pelo mesmo controlador diversos dispositivos de natureza distinta;

- **Funcionamento de um sistema computacional**



- A transferência de dados ocorre quando um ***device driver*** **ajusta** um **controlador** para que este obtenha os dados do dispositivo e os disponibilize em algum ***buffer*** do controlador. Depois, o *device driver* **retorna** esses **dados ao sistema operacional**, **devolvendo-lhe** o **controle** da operação;
- Alguns controladores (principalmente os de disco) utilizam a técnica **DMA** (***direct memory access***) que permite que o **controlador negocie diretamente** com a CPU a **transferência direta** de um **volume grande de dados** para a **memória**, **sem** a interferência da **CPU** (que é avisada quando se termina a operação).

- Existe uma **única CPU** capaz de **executar instruções** de máquinas definida no conjunto **ISA** (instruction set architecture) da CPU em questão;
- É comum haver **outros processadores específicos**, além da CPU, que executam **instruções próprias** do **controlador** que estão associados (disco, vídeo, teclado, rede etc.);
- As instruções da CPU são **executadas** em um **ciclo** denominado **busca-decodificação-execução** (*fetch-decode-execute cycle*), que pode ser assim resumido:

(1) **Buscar** a próxima **instrução** da **memória** e **armazenar** no **registrador** de **instrução**.

(2) Alterar o **contador de programa** para a **próxima instrução**.

(3) **Determinar** o **tipo** da instrução buscada.

(4) Se a instrução **depende** de uma **palavra na memória**, **descobrir** seu **endereço**.

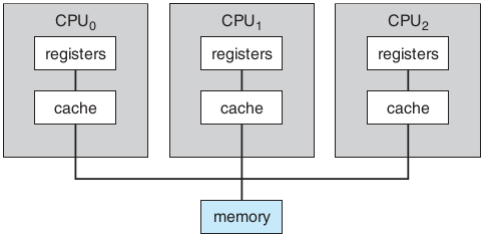
(5) **Buscar** a **palavra** e **armazenar** em um **registrador** da CPU.

(6) **Executar** a **instrução**.

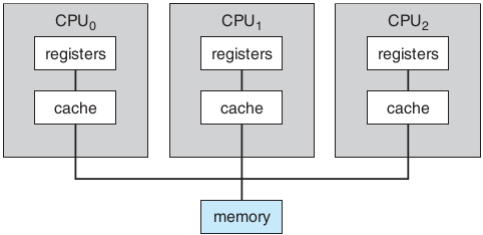
(7) **Voltar** ao **passo** (1).

- **Sistemas multiprocessados** (ou **sistemas paralelos** ou **sistemas com múltiplos núcleos**) são sistemas que possuem **dois ou mais processadores** que operam próximos entre si;
- Compartilham o ***bus*** de dados, **relógio** (muitas vezes) e **dispositivos** periféricos;
- **Vantagens**:
  - **Maior rendimento**: há um aumento da velocidade por um fator menor que o número de processadores, mas é melhor que o fator unitário;
  - **Economia de escala**: um sistema multiprocessado é mais barato do que vários sistemas monoprocessados em conjunto;
  - **Aumento na confiabilidade**: distribuindo as funções entre os diversos processadores, caso um deles falhe, este fato não impedirá que os demais continuem a operar, somente reduzirá a velocidade.

- **Dois tipos de multiprocessamento**:
  - **Multiprocessamento simétrico** (**SMP** – *symmetric multiprocessing*): **todos os processadores executam** as **tarefas do sistema operacional** operando como **pares**, sem a necessidade de um elemento de coordenação;
  - Se existem N **processadores**, nesta organização podem ser **executados** N **processos em paralelo**, sem degradação.

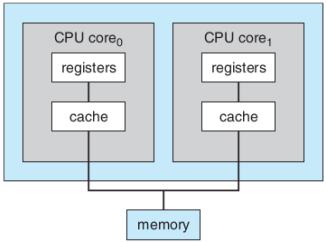


- **Dois tipos de multiprocessamento**:
  - **Multiprocessamento simétrico** (**SMP** – *symmetric multiprocessing*): **todos os processadores executam** as **tarefas do sistema operacional** operando como **pares**, sem a necessidade de um elemento de coordenação;
    - Se existem N **processadores**, nesta organização podem ser **executados** N **processos em paralelo**, sem degradação.



- **Dois tipos de multiprocessamento**:
  - **Multiprocessamento assimétrico**: a cada **processador** é **atribuído** uma **tarefa específica**;
  - Um **processador** – **o chefe** – controla o sistema;
  - Os **outros processadores** – **trabalhadores** – **requisitam instruções** ao chefe executando **tarefas específicas**;
  - Define-se, portanto, uma relação tipo **"mestre-escravo"**: o **mestre** agenda e aloca trabalho para os **escravos**.

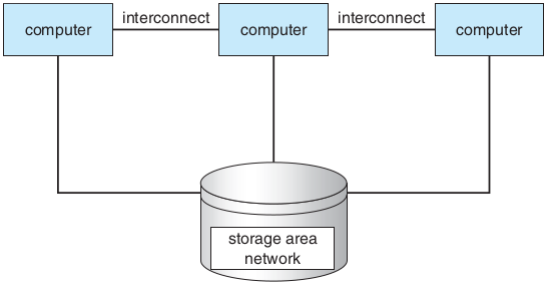
- **Sistemas com múltiplos núcleos** (*multicore*)
  - Uma tendência recente no design de CPU é **incluir** vários **núcleos de computação** em um **único chip**;
  - Esses sistemas multiprocessadores são denominados **multicore**;
  - Eles são **mais eficientes** do que vários chips com núcleos únicos porque dentro do chip a **comunicação é mais rápida** do que a comunicação entre os chips;
  - Além disso, um chip com múltiplos núcleos **usa significativamente menos energia** do que múltiplos chips com núcleos únicos.



Sistemas em *cluster*

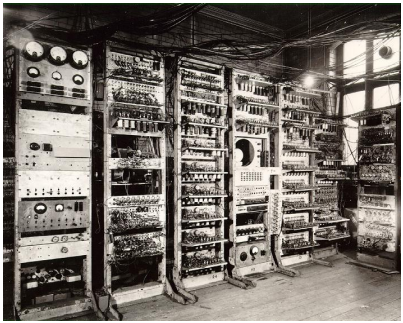
- **Diferem** dos **sistemas multiprocessadores** apresentados **pois** são **compostos** por **dois ou mais sistemas individuais** – denominados por **nós** – unidos;
- Tais sistemas são considerados **fracamente acoplados**;
- Cada **nó** pode ser um **sistema de processador único ou multicore**;
- Os nós **compartilham armazenamento** e estão conectados por meio de uma **rede local** (LAN – *local area network*).
  Vantagens:
  - Provêm um **serviço de alta-disponibilidade**: a falha de um nó do *cluster* não impede que o serviço continue;
  - Pode ser **configurado** para operar de **modo simétrico** ou **assimétrico**;
  - Permite criar um **ambiente de computação de alto desempenho**, com aplicações **paralelizadas**.

- **Sistemas em *cluster***



Primeira Geração: (1945–55) Válvulas

- Antes de 1940: dispositivos de computação para **tarefas espcíficas**;
- **John von Neumann** e **Alan Turing**: contribuições teóricas para definir um **computador** com **programa armazenado** – **mais geral**;
- **ABC** (*Atanasoff-Berry Computer*) (1939);
- **II Guerra Mundial**: **Colossus** (GB – 1943/1945); **Zuse Z3** (AL – 1941/1943));
- **Primeiro computador** de **uso geral** de **programa armazenado**: **Manchester Mark 1** (1949);
- **Primeiro computador comercial**: Ferranti Mark 1 (1951).



- **Segunda geração: (1955-1965) Transistores e sistemas em lote**
  - Os programas eram agrupados (**lote**) para melhor aproveitamento da máquina (muito caro). **Exemplo**:

(a) O operador leva seus cartões ao 1401;

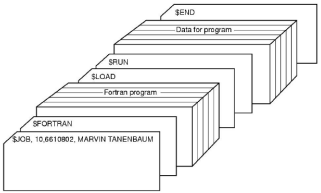
(b) O 1401 lê um lote de *jobs* (programas) e grava em uma fita magnética;

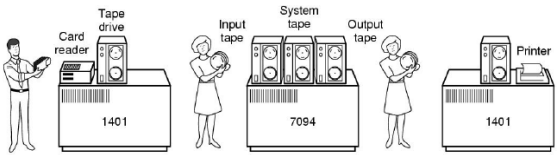
(c) O operador carrega a fita de entrada no 7094;

(d) O 7094 executa as computações;

(e) O operador carrega a fita de resultado no1401;

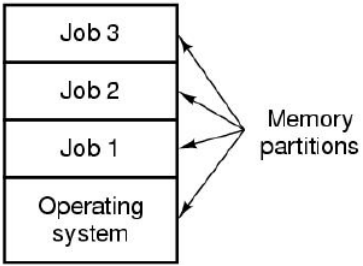
(f) O 1401 imprime a saída dos programas.





Terceira geração: (1965-1980): CIs e multiprogramação

- Criação de **família de produtos escaláveis e compatíveis** – IBM 360;
- Sistema operacional **OS/360** era **grande**, **complexo** e **cheio de *bugs***;
- Surgimento da técnica de **multiprogramação**: particionar a memória em várias partições, com um *job* diferente em cada partição;
- Enquanto um *job* aguarda pelo término de uma operação de E/S, outro *job* pode usar a CPU.



- **Terceira geração: (1965-1980): CIs e multiprogramação**
  - Técnica de ***spooling***: cartões poderiam ser lidos diretamente para fita/disco, economizando uma etapa no processo;
  - Surgimento da técnica de ***timesharing*** (divisão de tempo – variação de multiprogramação): se 20 usuários estão logados e 17 deles não estão executando nada na CPU, esta pode ser alocada para executar os três *jobs* que precisam ser executados;
  - Surgimento dos **minicomputadores**: menores e mais baratos que os mainframes (o UNIX foi escrito em um minicomputador PDP-7).

A quarta geração (1980-presente): computadores pessoais

- Viabilizados pela tecnologia **LSI** (*Large-Scale Integration*);
- **Microprocessadores Intel** a partir do 8080 de 1974; **Zilog** Z-80;
- Sistema operacional **CP/M** (1974–1977);
- Sistema operacional **DOS** (pior que o CP/M!) desenvolvido pela **Microsoft** para o IBM PC no início de 1980 – a IBM não confiava no sucesso do PC, abriu a arquitetura e a Microsoft se tornou um gigante produtor de software;
- Interfaces gráficas GUI (*Graphical User Interface*)– Apple Lisa e Macintosh (início 1980);
- **Sistemas operacionais em rede**.

A quinta geração (1990-presente): computadores móveis

- Primeiro **telefone móvel** na década de 1970 (pesava cerca de um quilo!);
- Hoje sua **utilização** está em **próxima** de **90%** da população global e engloba **funções** de **telefonia** e **computador**;
- Não se restringe a apenas telefones – tem os **dispositivos vestíveis**: óculos, relógios, etc;
-
- **PDAs** (*personal digital assistant*) surgiram em 1998;
- Sistema operacional dominante: **Android** (**kernel Linux!**) seguido de **iOS** e **Windows Phone**.

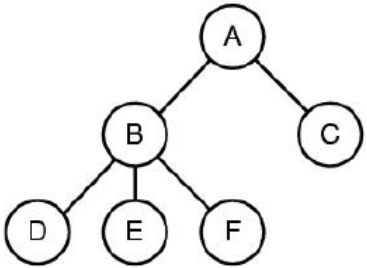
Categorias de sistemas operacionais

Tipos mais importantes de sistemas operacionais

- Sistemas operacionais de **servidor** (Linux, Free BSD, Solaris, HP-UX, Windows Server)
- Sistemas operacionais de **mainframe** (OS/390);
- **Sistemas operacionais multiprocessadores** (Linux, Free BSD, Solaris, HP-UX, Windows);
- Sistemas operacionais para **computadores pessoais** (Linux, Free BSD, Windows, OS X);
- Sistemas operacionais para **dispositivos móveis** (Android, iOS, Windows);
- Sistemas operacionais **embarcados** (Linux, Windows, OpenWrt);
- Sistemas operacionais em **tempo-real** (Free RTOS, RT Linux)
- Sistemas operacionais para **robótica** (ROS).

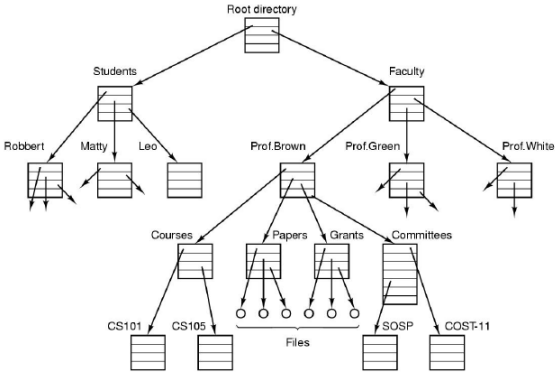
Conceitos

- **Processo** é um programa em execução;
- Por exemplo, no Linux/UNIX existe um **processo inicial**, `init` que sempre é **executado** quando o **computador inicializa**;
- Depois, por meio do **comando** `fork`, **novos processos** são **criados**, formando uma estrutura **hierárquica** de processos – **árvore de processos**;
- Além disso, o sistema deve prover meios para **controlar processos** (criar, terminar, suspender, reativar).



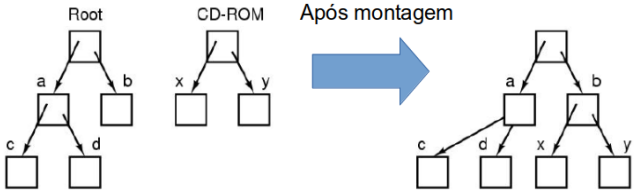
Conceitos

- Todo sistema operacional deve prover um **sistema de gerenciamento de arquivos**;
- No **Linux/UNIX**, por exemplo, a organização das pastas e arquivos toma a forma de uma **grande** (e **única**) **árvore**;
- O sistema deve prover meios de **gerenciar** não apenas **arquivos**, como também **diretórios** (criar, remover, alterar, acessar).



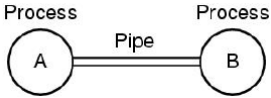
Montagem de um sistema de arquivos

- No caso do Linux/UNIX como o sistema de arquivos é representado por uma única árvore, o **que acontece** quando se **anexa** um **novo dispositivo** de **armazenamento** externo?
- Ele é **montado na árvore** e se torna parte dela (até ser desmontado).



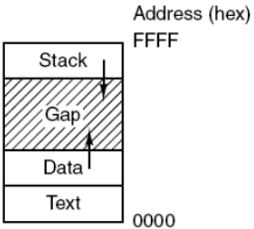
Arquivos especiais

- No caso do Linux/UNIX existem tipos especiais de arquivos (**pseudo arquivos**) denominados **pipes** que podem ser utilizados para **conectar** a **saída** de um **processo** à **entrada** de outro;
- Este é uma forma simples de **comunicação interprocesso**.

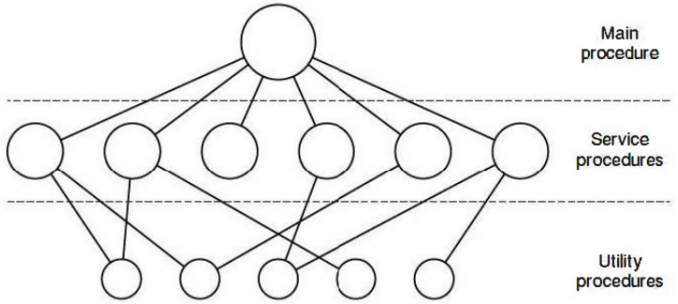


- Em sistemas operacionais, como o código do usuário é proibido de acessar o hardware diretamente (questão de proteção), existem **funções do *kernel*** denominadas de **chamada de sistema** que são implementadas corretamente para realizar tal tarefa;
- As chamadas de sistemas produzem uma **interrupção de software** ou ***trap*** (consulte o slide 13 para maiores detalhes).

- Antes de um processo entrar em execução, um **programa executável** presente no disco deve ser **carregado** na memória;
- Um programa **compilado** possui normalmente uma área de código e uma área de dados globais e instruções;
- O **processo** relacionado a este programa deverá ser organizado na **memória** de modo que a **área de dados** seja **distinta** da **área de código** e ainda tenha espaço para as variáveis locais e parâmetros de funções (que fica em uma **pilha** que pode crescer e diminuir);
- Por exemplo, no Linux o leiaute de um processo na memória possui três segmentos:



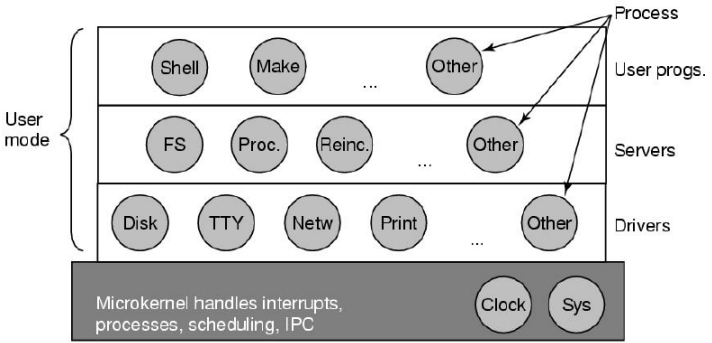
- O **sistema operacional inteiro** executa no **modo *kernel*** como um **único processo principal**;
- Este processo do sistema operacional possui uma **coleção de procedimentos de serviço**;
- Esses **procedimentos** de serviço **realizam as chamadas de sistema** quando necessário;
- Um conjunto de **procedimentos utilitários auxiliam** esses procedimentos de serviço.



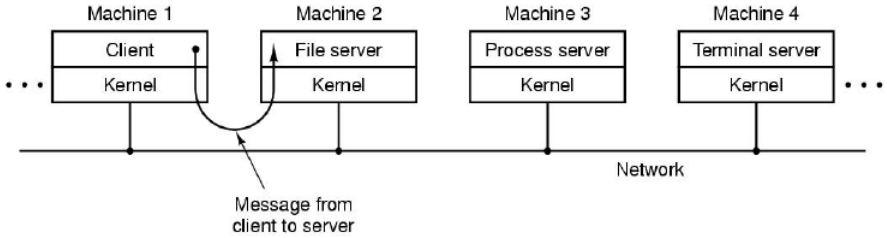
- É uma generalização da abordagem monolítica. O sistema operacional é organizado **como uma hierarquia de camadas**, cada uma construída sobre a camada abaixo dela;
- Cada camada possui uma tarefa bem definida. Por exemplo, o sistema **THE** era assim organizado:

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

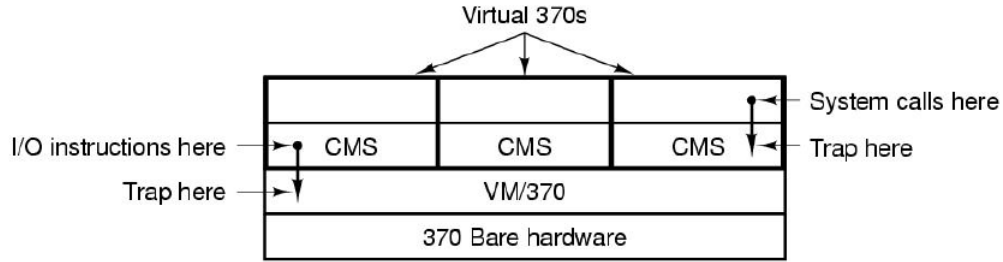
- A ideia é atingir uma **alta confiabilidade** através da divisão do sistema operacional em **módulos pequenos** e **bem definidos**, apenas um dos quais – o **microkernel** (micronúcleo) — é executado em modo *kernel* e o resto é executado como processos de usuário comuns relativamente sem poder (inclusive *drivers* de dispositivo):



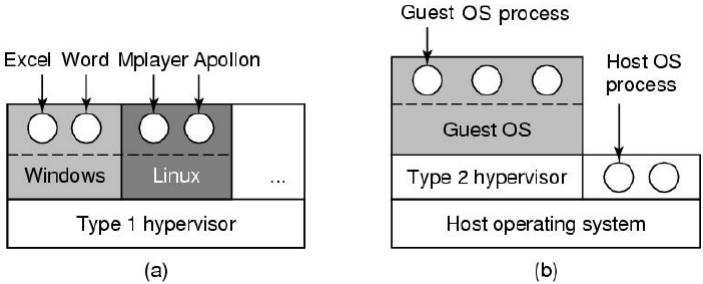
- É uma **variação** da ideia do **microkernel** onde se distinguem **duas classes de processos**, os **servidores**, que **prestam algum serviço**, e os **clientes**, que usam esses serviços;
- A **comunicação** entre **clientes** e **servidores** é realizada muitas vezes pela **troca de mensagens**:



- São **softwares que emulam hardwares** (reais ou fictícios);
- Na década de 70 foi projetado o **VM/370** (executava em >= IBM 360) com CMS (*Conversational Monitor System*):



- Atualmente o nome máquina virtual tem como sinônimo a palavra ***hypervisor*** (hipervisor);
- Existem dois tipos de hipervisores:
  - **Tipo 1:** permite a **execução** de **um ou mais sistemas operacionais simultaneamente** no **hardware**;
  - **Tipo 2**: **depende** de um **sistema operacional hospedeiro** que então **implementa um simulador** de máquina que **executa** um **ou mais sistemas operacionais**.



Referências bibliográficas

[1] SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating system concepts**. 8. ed. [s.l.] John Wiley & Sons, Inc., 2009.

[2] TANENBAUM, A. S.; BOS, H. **Modern operating systems**. 4. ed. [s.l.] Pearson, 2015.

[3] TANENBAUM, A. S. **Structured Computer Organization**. 5. ed. Upper Saddle River, N.J.: Pearson Prentice Hall, 2006.

[4] **Operating System Concepts**. Disponível em: <https://www.cs.rutgers.edu/~pxk/416/notes/03-concepts.html>. Acesso em: 18 jan. 2021.