Redes e Segurança

Universidade de Mogi das Cruzes (UMC)

Parte 1:
Introdução aos Sistemas Operacionais

Sistema **Computacional**

Antes de definir o SO, um Sistema Computacional consiste de:

- Um ou mais processadores
- Memória principal
- Discos
- Teclado
- Monitor
- Mouse
- Interfaces de redes
- Impressora
- Outros dispositivos

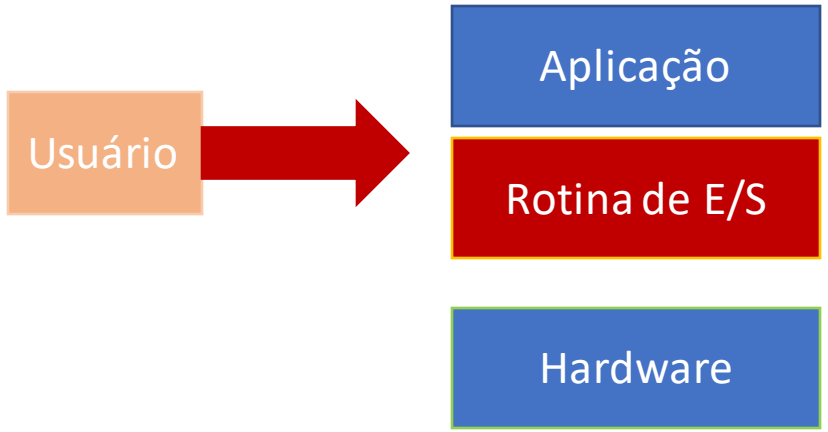
Sistema Computacional

Programas precisam saber lidar com todos os elementos que compõem um **sistema computacional**.

Importância do **Sistema Operacional**

Aplicação sem SO:

- Gasto maior de tempo de programação
- Aumento da dificuldade
- Usuário preocupado com detalhes de hardware.

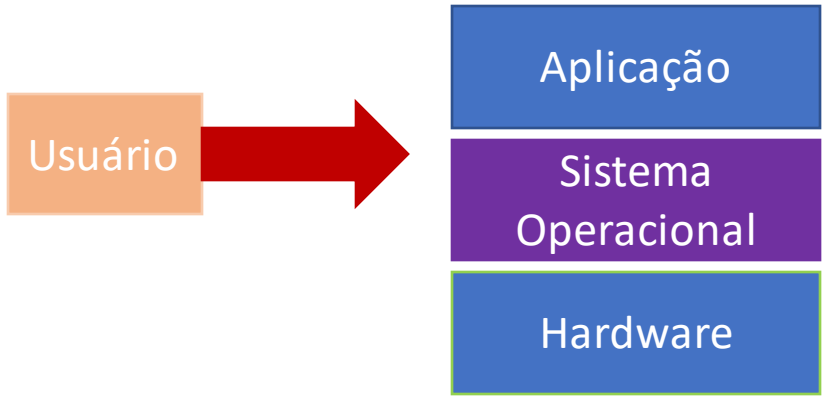


E/S = Entrada / Saída
I/O = Input / Output

Importância do **Sistema Operacional**

Aplicação com SO:

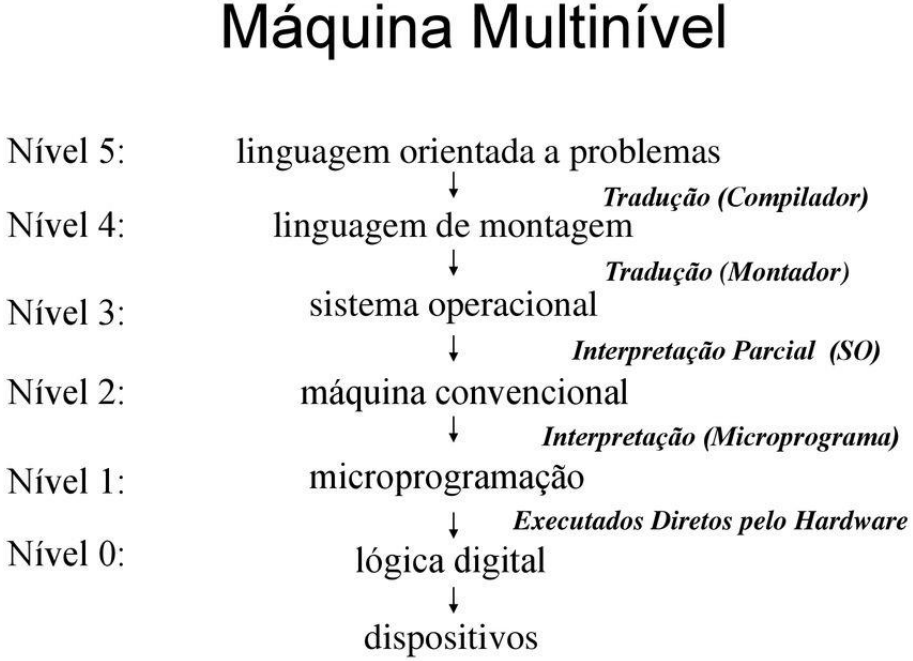
- Maior racionalidade
- Maior dedicação aos problemas de alto nível
- Maior portabilidade



Máquina **Multinível**

Uma máquina multinível pode ser vista como tendo vários níveis, cada um capaz de executar um conjunto de instruções específicas. Isto é, cada nível possui linguagens apropriadas para descrever as instruções que nele podem ser executadas.

Máquina **Multinível**



Definição de Sistema Operacional

- Um programa ou software;
- **Um conjunto de programas interrelacionados cuja finalidade é agir como**:
  - Intermediário (ou interface) entre o usuário e o hardware
  - Gerenciador de recursos

Definição de Sistema Operacional

- O sistema operacional é uma interface HW/SW aplicativo e um gerenciador de recursos.
- Duas formas de vê-lo:
  - É um **fiscal** que controla os usuários
  - É um **juiz** que aloca os recursos entre os usuários

Definição de Sistema Operacional

Objetivos contraditórios:

- Conveniência
- Eficiência
- Facilidade de evolução
- A melhor escolha sempre depende de alguma coisa...

Vantagens do Sistema Operacional

- Apresentar uma máquina mais flexível
- Permitir o uso eficiente e controlado dos componentes de hardware
- Permitir o uso compartilhado e protegido dos diversos componentes de HW e SW por diversos usuários

Funções do Sistema Operacional

- O SO deve fornecer uma interface aos programas do usuário.
  - Quais recursos de HW?
  - Problema? (Segurança, falha…?)
  - Chegou um email?
  - Entre outros…
  - Chamadas de sistema [e.g. malloc()] – programas dos programas do SO

Interação com o Sistema Operacional

Usuário pode interagir com o SO através de uma linguagem de comunicação especial, chamada de linguagem de comando.

Três mecanismos:

- JCL (Job Control Language)
- GUI (Graphic User Interface)
- Chamadas de Sistema

Interação com o Sistema Operacional

JCL (Job Control Language)

- O usuário pode interagir com o SO através de uma linguagem de comunicação especial, chamada **linguagem de comando**.
- Linguagem de Comando são específicas para cada sistema.

Linux/Unix	*Windows*
ls	dir
cp	copy
rm	del

Interação com o Sistema Operacional

Chamadas de Sistema (system calls)

- Elas permitem um controle mais eficiente sobre as operações do sistema

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Formas de processamento do Sistema Operacional

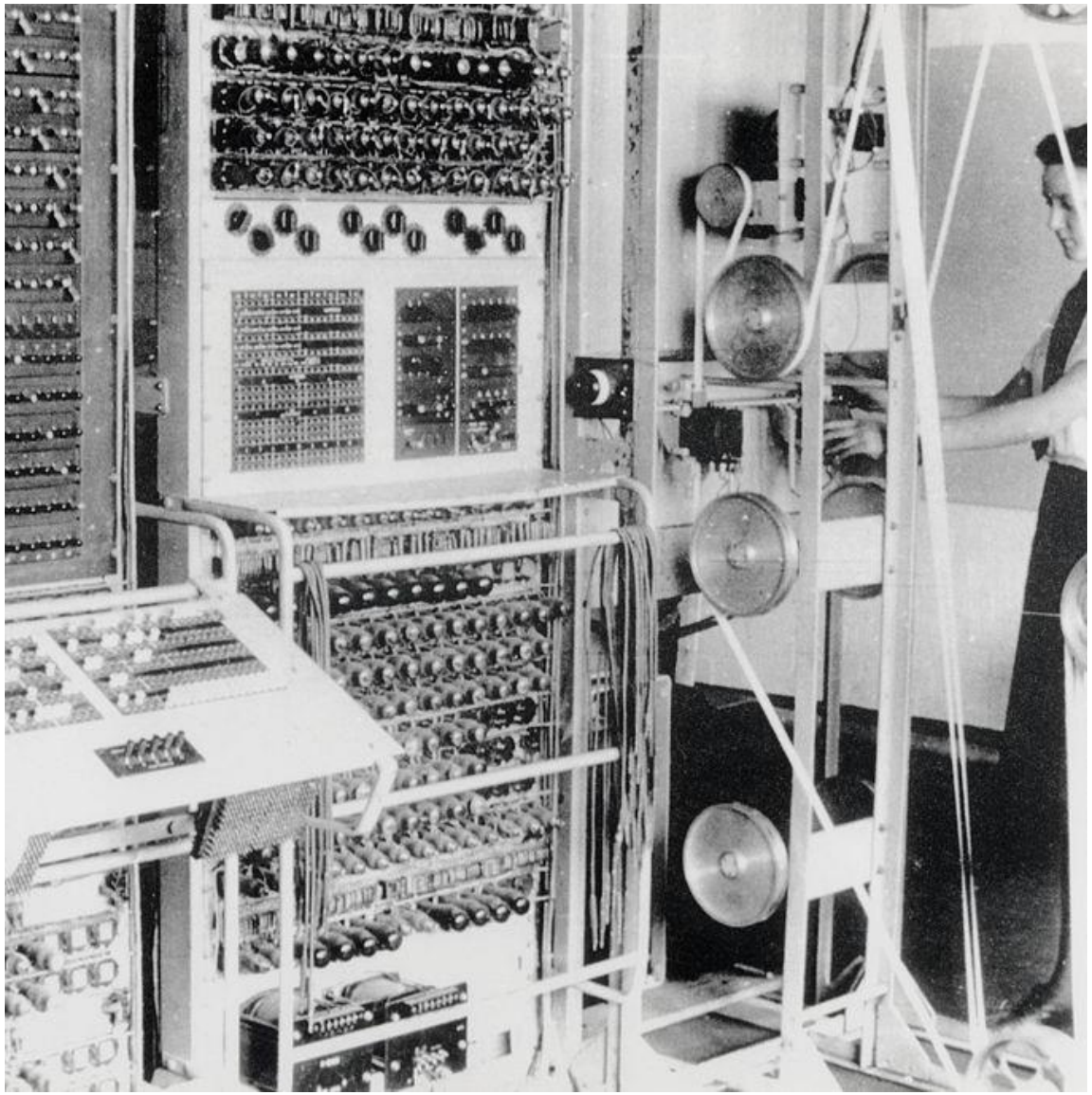
- **Serial (Monoprogramada)**
  - Recursos alocados a um único programa
- **Concorrente Multiprogramada**
  - Recursos dinamicamente reassociados entre uma coleção de programas em diferentes estágios

Histórico: Sistema Operacional

- 1ª Geração
- 1945/1955
- Computadores a Válvula
- Ausência de um S.O
- Programação em linguagem de máquina

Ex: Colossus Mark I

COLOSSUS MARK I



Histórico: Sistema Operacional

- 2ª Geração
- 1955/1965
- Invenção e uso dos transistores
- Uso da linguagem Assembly e FORTRAN
  - SOs do tipo lote (batch)
  - Programas são executados em fila

Histórico: Sistema Operacional

- 3ª Geração
- 1965/1980
- Circuitos integrados
- Multiprogramação
  - Vários programas concorrentes
- Time-sharing
  - SO compartilhado com vários programas

Ex: Sistema OS/360 (IBM)

- 1º a usar circuitos SSI

IBM 360 - MAINFRAME

Histórico: Sistema Operacional

- 4ª Geração
- 1980/1990
- Circuitos integrados com alta escala de integração (LSI – Large Scale Integration - Circuito integrado em larga escala)
- SO para micros (MS-Dos e Windows)
- Difusão da internet

Histórico: Sistema Operacional

- 5ª Geração
- 1990/Atual
- Difusão da internet
- SO com suporte para TCP/IP
- Diferentes tipos:
  - Cliente/Servidor
  - Sistemas de tempo-real
  - Computação ubíqua
  - Internet das Coisas
  - Entre outros

Parte 2:
Introdução ao Tipo e Estrutura dos Sistemas Operacionais

Tipos de Sistema Operacional

- Classificação pelo compartilhamento de HW:

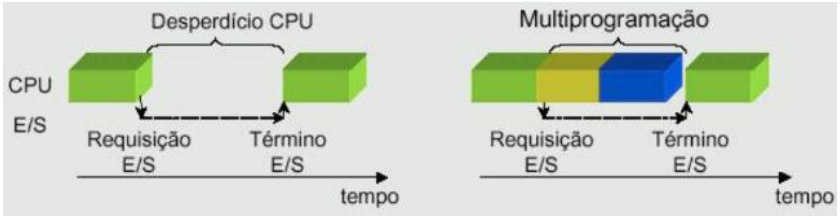
- **Sistemas Operacionais Monoprogramados ou Monotarefa:**
  - Só permitem um programa ativo em um dado período, o qual permanece na RAM até seu fim (Ex: MS-DOS)
- **Sistemas Operacionais Multiprogramados ou Multitarefa:**
  - Mantêm mais de um programa na memória, para permitir o compartilhamento do tempo de CPU e demais recursos (Ex.: UNIX, Windows)

Sistema Operacional: **Monotarefa**

- Caracterizam-se por permitir que o processador, a memória e os periféricos permaneçam exclusivamente dedicados à execução de um único programa.
- Recursos são mal utilizados, entretanto, são implementados com facilidade.
- Pode-se pensar que o processo estará em um destes estados.

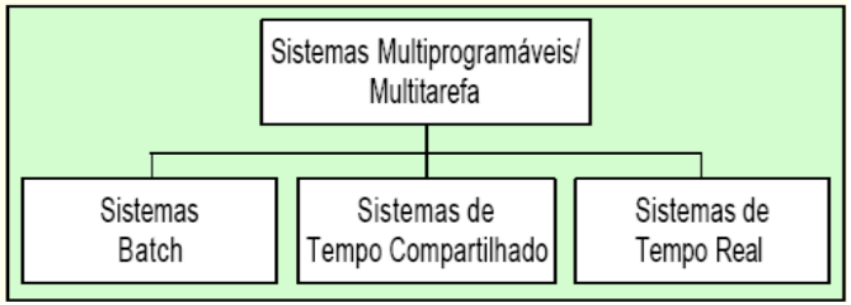
Sistema Operacional: **Multitarefa**

- A ideia é manter vários programas em memória ao mesmo tempo.
- Há várias tarefas simultâneas, em um único processador: enquanto uma espera, a outra roda.
- Demandam mecanismo de trocas rápidas de processos.



Sistema Operacional: **Multitarefa**

- Podem ser classificados quanto à interação permitida com as aplicações:



Sistema Operacional: **Multitarefa em Batch**

- Sistema Batch (lote) consiste em submeter ao computador um lote (batch) de programas de uma só vez.
- Os jobs (script com lote de programas) dos usuários são submetidos em ordem sequencial para a execução.
- Não existe interação entre o usuário e o job durante sua execução.

Sistema Operacional: **Multitarefa em Batch**

- Exemplo de Sistema Batch atualmente:

```
#
# makefile for 2ndYrep
#

pdf::
        latex  thesis.tex
        dvipdf  thesis.dvi thesis.pdf
        acroread thesis.pdf &

ref::
        latex  thesis.tex
        latex  thesis.tex
        dvips  thesis.dvi -o thesis.ps
        ggv thesis.ps &

bib::
        latex  thesis.tex
        bibtex thesis
        latex  thesis.tex
        latex  thesis.tex
        dvips  thesis.dvi -o thesis.ps
        ggv thesis.ps &
```

Sistema Operacional: **Multitarefa Interativo**

- O sistema permite que os usuários interajam com suas computações na forma de diálogo.
- Pode ser projetado como sistema monousuário ou multiusuário (usando conceitos de multiprogramação e time sharing).

Estrutura dos Sistema Operacional

- SOs são normalmente grandes e complexas coleções de rotinas de software.
- Projetistas devem dar grande ênfase à sua organização interna e estrutura:
  - **Monolítica**
  - **Micronúcleo**
  - Camadas
  - Máquina virtual

Estrutura dos Sistema Operacional: **Monolítica**

- SO é um único módulo.
- Consiste de um conjunto de programas que executam sobre o hardware, como se fosse um único programa.
- Os programas de usuário invocam rotinas do SO.

- Ex: MS-DOS, Windows, Unix e Linux

Estrutura dos Sistema Operacional: **Microkernel**

- Busca tornar o núcleo do SO o menor possível.
- A principal função do núcleo é gerenciar a comunicação entre esses processos.
- Núcleo fornece serviços de alocação de CPU e de comunicação aos processos (IPC).
- Ex: Minix, Symbian

Estrutura dos Sistema Operacional: **Camadas**

- A ideia é criar um SO:
  - **Modular -** divisão de um programa complexo em módulos de menor complexidade.
  - **Hierárquico -** a cada nível, os detalhes de operação dos níveis inferiores podem ser ignorados.
- As interfaces são definidas para facilitar a interação entre os módulos hierárquicos.
- Ex: MULTICS, OpenVMS

Estrutura dos Sistema Operacional: **Máquina Virtual**

- Essa estrutura cria um nível intermediário entre o hardware e o SO, denominado Gerência de VM.
- Esse nível cria diversas VMs independentes.
- Cada VM oferece uma cópia virtual do hardware, incluindo modos de acesso, interrupções, dispositivos de E/S, etc.

- Ex: VirtualBox, VMSphere, VM/370

Estrutura dos Sistema Operacional: **Máquina Virtual**

- Um outro exemplo de utilização dessa estrutura ocorre na linguagem Java.
- Para executar um programa Java é necessário uma Java Virtual Machine.

Parte 3:
Introdução as Chamadas de Sistema e Interrupções

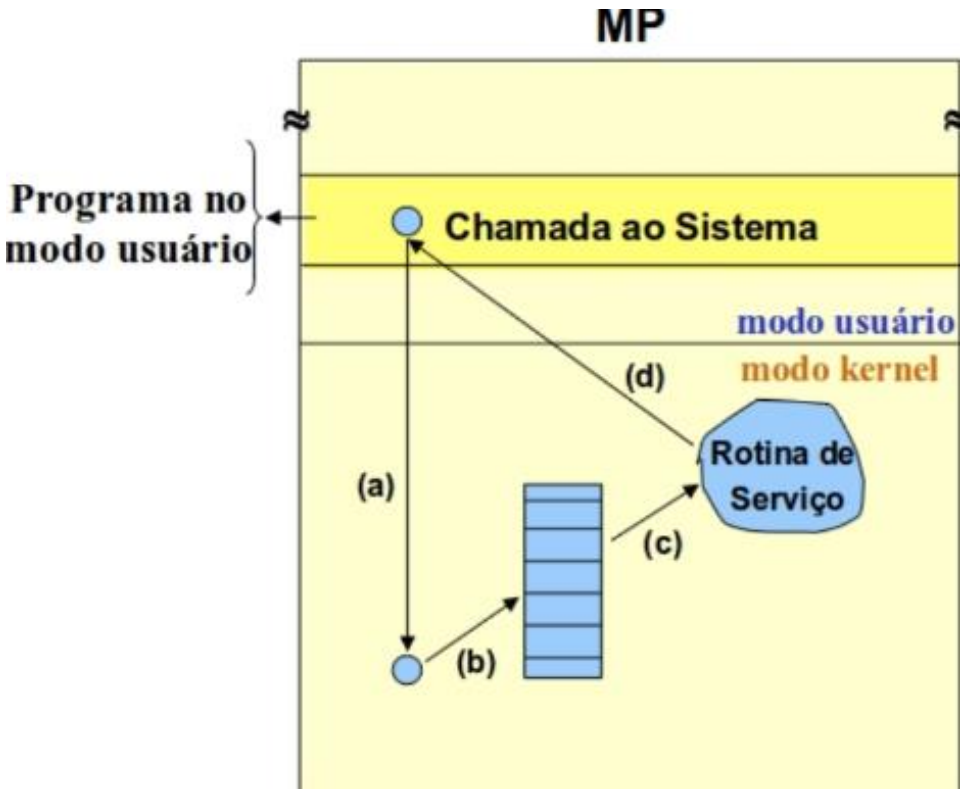
Chamadas de Sistema

- Se uma aplicação precisa realizar alguma instrução privilegiada (imprimir um arquivo), ela realiza uma chamada de sistema, que altera do modo usuário para o modo kernel;
  - Ex: Ler um arquivo
- Chamadas de sistemas são a porta de entrada para o modo kernel;

Chamadas de Sistema

- As chamadas de sistemas são realizadas através de instruções Traps.
- **Traps** são conhecidos como **interrupções de software**.
- Após o término da chamada (ex.: ler um arquivo), a execução **continua após a chamada de sistema**.

Chamadas de Sistema



Passos para chamada de sistema:

a) Aplicativo faz chamada ao sistema (TRAP).
b) Através de uma tabela, o SO determina o endereço da rotina.
c) Rotina de Serviço é acionada (rotina compartilhada).
d) Serviço solicitado é executado e o controle retorna ao aplicativo.

Interfaces das **SysCalls**

- Interface para esconder a complexidade das syscalls.
- Interface de programação fornecida pelo SO.
- Geralmente escrita em linguagem de alto nível (C, C++ ou Java).
- Normalmente as aplicações utilizam uma Application Program Interface (API).
- Interface que encapsula o acesso direto às chamadas ao sistema.

Interfaces das **SysCalls**

- Interface das Chamadas de Sistema (Wrappers) mais utilizadas:
  - Win32 API para Windows
  - POSIX API para praticamente todas as versões de UNIX
  - Java API para a Java Virtual Machine (JVM)

Interfaces das **SysCalls**

- Motivos para utilizar APIs em vez das chamadas ao sistema diretamente
  - Portabilidade - independência da plataforma
  - Esconder complexidade inerente às chamadas ao sistema
  - Acréscimo de funcionalidades que otimizam o desempenho

Interrupções

- Um software pode interromper seu próprio processo (ao fazer uma chamada ao sistema):
  - Usando traps (Interrupções de software ou Exceções).
  - Para isso, a aplicação tem que estar rodando.
- Mas ocorrem interrupções que não são causadas por aplicações em execução:
  - Interrupções de hardware (eventos externos).
  - Um sinal elétrico no hardware.
  - Causa: dispositivos de E/S ou o clock.

Interrupções **vs** Traps

- **Interrupção**
  - Evento externo ao processador.
  - Gerada por dispositivos que precisam da atenção do SO.
  - Pode não estar relacionada ao processo que está rodando.
- **Traps**
  - Evento inesperado vindo de dentro do processador.
  - Causados pelo processo corrente no processador (seja por chamada ao SO, seja por instrução ilegal).

Parte 4:
Introdução aos Processos de Sistema

Definição de Processo

- Um processo é caracterizado por um **programa em execução**.
- Diferença entre processo e programa?
  - Um processo é uma instância de um programa e possui dados de entrada, dados de saída e um estado (executando, bloqueado, pronto).

Programa **vs** Processo

- **Programa**
  - Pode ter várias instâncias em execução (em diferentes processos).
  - Algoritmo codificado.
  - Forma como o programador vê a tarefa a ser executada.
- **Processo**
  - É único.
  - Código acompanhado de dados e estado.
  - Forma pela qual o SO vê um programa e possibilita sua execução.

Processo em **Primeiro Plano**

- Interage com o usuário.
- Exemplos:
  - Ler um arquivo.
  - Iniciar um programa (linha de comando ou um duplo clique no mouse).

Processo em **Segundo Plano**

- Processos com funções específicas que independem de usuários – daemons:
- Exemplos:
  - Recepção e envio de emails.
  - Serviços de impressão.

Elementos dos Processos

- **Conjunto de instruções**
- **Espaço de endereçamento** (espaço reservado para que o processo possa ler e escrever – 0 até max)
- **Contexto de hardware** (valores nos registradores, como PC, ponteiro de pilha, e reg. de prop. gerais)
- **Contexto de software** (atributos em geral, como lista de arquivos abertos, variáveis, etc.)

Espaço de **Endereçamento**

- **Texto:** código executável do(s) programa(s);
- **Dados:** as variáveis;
- **Pilha de Execução:**
  - Controla a execução do processo;
  - Empilhando chamadas a procedimentos, seus parâmetros e variáveis locais, etc.

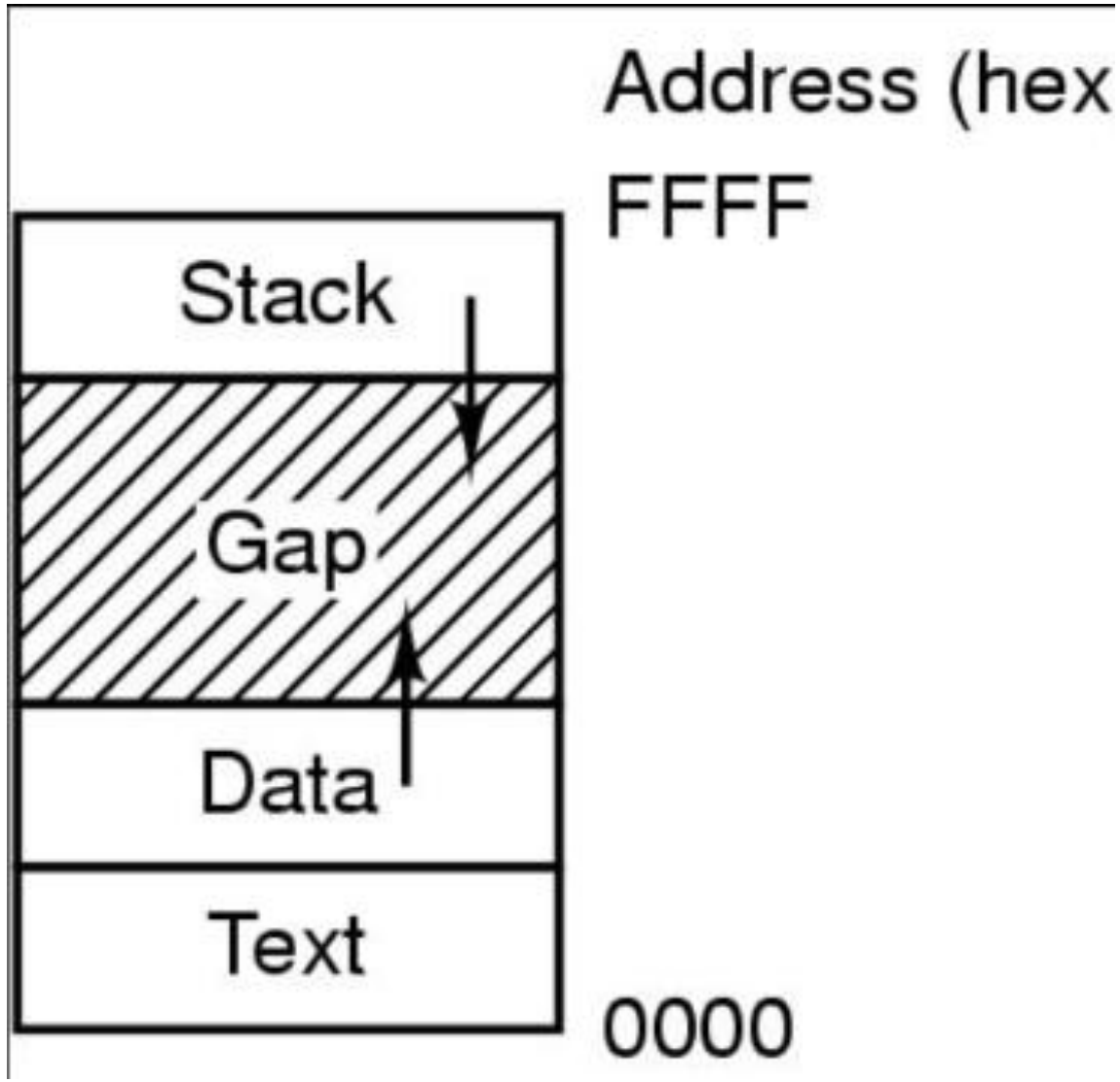


Tabela de Processos

- Também chamada de **BCP** (Bloco Controle de Processo).
- Contém informações de contexto de cada processo (ex. **ponteiros de arquivos abertos**, posição do próximo byte a ser lido em cada arquivo, etc.)
- Contém **informações necessárias para trazer o processo de volta**, caso o SO tenha que tirá-lo de execução.
- Contém **estados de um processo em um determinado tempo**.

Tabela de Processos

- O BCP só não guarda o conteúdo do espaço de endereçamento do processo.
- Assim, um processo é **constituído de seu espaço de endereçamento e BCP** (com seus registradores, etc.), representando uma entrada na tabela de processos.

Características dos Processos

- Processos **CPU-bound** (orientados à CPU): processos que utilizam muito o processador;
  - Tempo de execução é definido pelos ciclos de processador;
- Processos **I/O-bound** (orientados à E/S): processos que realizam muito E/S;
  - Tempo de execução é definido pela duração das operações de E/S;
- **IDEAL:** existir um balanceamento entre processos CPU-bound e I/O-bound;

Criação dos Processos

a. Inicialização do sistema
b. Execução de uma chamada de sistema para criação de processo, realizada por algum processo em execução
c. Requisição de usuário para criar um novo processo (duplo clique do mouse, etc.)
d. Inicialização de um processo em batch (em sistemas mainframes com proc. em batch).

Processos **criando outros** Processos

- No UNIX com a função fork()
  - Cria clone do processo Pai: cópias exatas na memória, mas com identificadores diferentes.
- No Windows com CreateProcess
  - Cria processo Filho, já carregando novo programa nele.

Finalizando Processos

- **Término normal** (voluntário):
  - A tarefa a ser executada é finalizada;
  - Ao terminar, o processo executa uma chamada (comunicando ao SO que terminou): exit (UNIX) e ExitProcess (Windows).
- **Término por erro** (voluntário):
  - O processo sendo executado não pode ser finalizado. Ex: gcc filename.c; o arquivo filename.c não existe.

Finalizando Processos

- **Término com erro fatal** (involuntário);
  - Erro causado por algum erro no programa (bug); Ex: Divisão por 0 (zero); Referência à memória inexistente; Execução de uma instrução ilegal.
- **Término causado por algum outro processo** (involuntário), via chamada a:
  - kill (UNIX)
  - TerminateProcess (Windows)

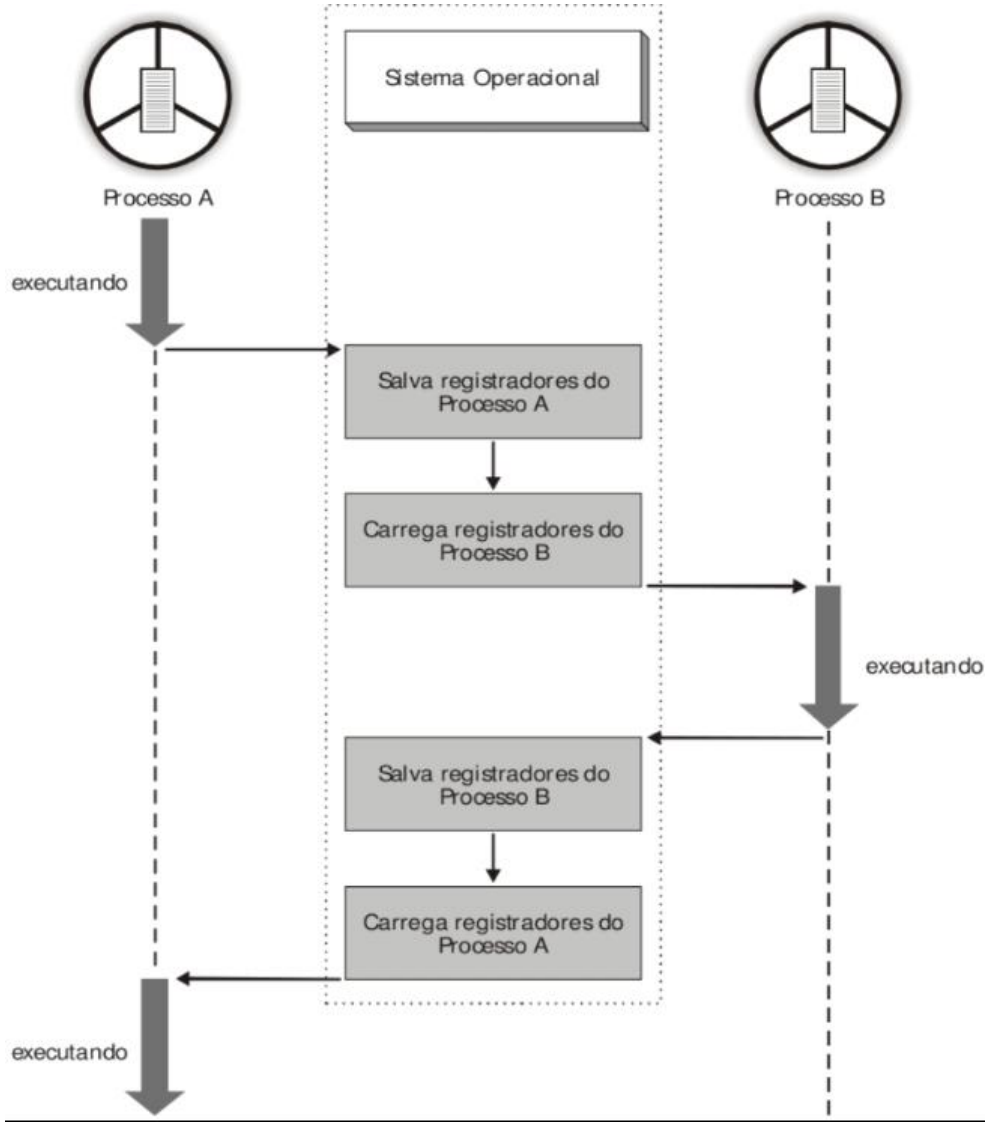
Estados
dos Processos

- **Executando:** realmente usando a CPU naquele momento.
- **Bloqueado:** incapaz de executar enquanto um evento externo não ocorrer.
- **Pronto:** em memória, pronto para executar (ou para continuar sua execução), apenas aguardando a disponibilidade do processador.

Parte 5:
Introdução aos Processos e Escalonamento

O que é um **Escalonador**?

- A troca de processos é feita pelo Escalonador de Processos.
- **Escalonador é o processo que escolhe qual será o próximo processo a ser executado.**
- Existem diversas técnicas/algoritmos para escalonamento de processos.
- Nível mais baixo do SO.



Mudança de **Contexto**

- A mudança de contexto leva a um overhead de tempo (tarefa cara):
- É preciso salvar as informações do processo que está deixando/entrando a/na CPU em seu BCP.
- Salvar o conteúdo dos registradores.

Componentes Envolvidos

- **Despachante (Dispatcher):**
  - Armazena e recupera o contexto;
  - Atualiza as informações no BCP;
  - Processo relativamente rápido (0,1ms).
- **Escalonador (Scheduler):**
  - Escolhe a próxima tarefa a receber o processador;
  - Parte mais demorada.

Quando o Escalonador é chamado?

- Um novo processo é criado.
- Quando um processo cria outro, qual executar? Pai ou filho?
- Um processo chegou ao fim e um processo pronto deve ser executado.
- Quando um processo é bloqueado (dependência de E/S), outro deve ser executado.

Quando E/S ocorre, o Escalonador deve:

- Executar o **processo que estava esperando esse evento**;
- **Continuar executando o processo** que já estava sendo executado; ou
- **Executar um terceiro processo** que esteja pronto para ser executado.

Categorias do Escalonador

- **Preemptivo:**
  - Quando um processo pode, por algum motivo, perder seu uso da CPU.
  - Provoca uma interrupção forçada de um processo para que outro possa usar a CPU.
- **Não preemptivo:**
  - Permite que o processo sendo executado continue executando.

Categorias do Escalonador

- Condições de parada do Não Preemptivo:
  i. termine de executar;
  ii. solicite uma operação de entrada/saída;
  iii. libere explicitamente o processador, voltando à fila de prontos.

Algoritmos de **Escalonamento**

- **Sistemas Batch (Lote)**
- Sistemas Interativos
- Sistemas de Tempo Real

Algoritmos de **Escalonamento**

- First-Come First-Served (ou FIFO);
- Shortest Job First (SJF);
- Shortest Remaining Time Next (SRTN);

Algoritmos de Escalonamento: FIFO

- Não preemptivo;
- Processos são executados na CPU seguindo a ordem de requisição;
- Fácil de entender e programar;
- **Desvantagem:**
  - Ineficiente quando há processos que demoram na sua execução.

Algoritmos de Escalonamento: **SJF**

- Não preemptivo;
- Deve-se prever o tempo de execução do processo;
- Menor processo da lista é executado primeiro;
- Menor turnaround (médio).
- **Desvantagens:**
  - Todos os jobs precisam ser conhecidos de antemão.
  - Se jobs curtos começarem a chegar, os longos podem demorar a ser executados.

Algoritmos de Escalonamento: **SRTN**

- Preemptivo;
- Versão preemptiva do Shortest Job First;
- Processos com menor tempo de execução são executados primeiro;
- Se um processo novo chega e seu tempo de execução é menor do que o do processo corrente na CPU, a CPU suspende o processo corrente e executa o processo que acabou de chegar.

Algoritmos de Escalonamento: SRTN

- **Desvantagens:**
  - Processos que consomem mais tempo podem demorar muito para serem finalizados se muitos processos pequenos chegarem (inanição ou starvation).
  - Exceto que, pela preempção no Shortest Remaining Time Next, o processo, mesmo rodando, seja interrompido.
  - No Shortest Job First, se der a sorte de começar a rodar, vai até o fim.

Parte 6:
Introdução ao Escalonamento de Processos

Algoritmos de **Escalonamento**

- Sistemas Batch (Lote)
- Sistemas Interativos
- Sistemas de Tempo Real

Revisando **Sistemas Interativos**

- Sistemas que tem interação com o usuário, diferente dos sistemas batch (lote) que só tem interação no momento da submissão do batch.
- Regido pelo time-sharing (tempo compartilhado)
- Algoritmos buscam garatir um fair-sharing (compartilhamento justo)

Escalonamento em **Sistemas Interativos**

- Round-Robin;
- Prioridade;
- Múltiplas Filas;
- Shortest Process Next;
- Garantido;
- Lottery;
- Fair-Share.

Algoritmo de Escalonamento: **Round-Robin**

- Antigo, mais simples e mais utilizado;
- Preemptivo;
- Cada processo recebe um tempo de execução chamado **quantum**;
- Ao final desse tempo, o processo é suspenso e outro processo é colocado em execução;
- Também suspenso em caso de interrupção;
- Escalonador mantém uma fila de processos prontos.

Algoritmo de Escalonamento: **Round-Robin**

- Os processos são colocados em uma fila circular (de prontos) e executados um a um.
- Quando seu tempo acaba, o processo é suspenso e volta para o final da fila.
- Outro processo (primeiro da fila) é então colocado em execução.
- Quando um processo solicita E/S, vai para a fila de bloqueados e, ao terminar a operação, volta para o final da fila de prontos.

Algoritmo de Escalonamento: **Round-Robin**

- **Problema:**
  - Tempo de chaveamento de processos;
- **Quantum:**
  - Se for muito pequeno, ocorrem muitas trocas diminuindo, assim, a eficiência da CPU;
  - Se for muito longo, o tempo de resposta é comprometido;

Algoritmo de Escalonamento: **Prioridade**

- Cada processo possui uma prioridade;
- Os processos prontos com maior prioridade são executados primeiro;
- Prioridades são atribuídas dinamicamente (pelo sistema) ou estatisticamente;
- Preemptivo;
- Round-Robin pressupõe igual prioridade para todos os processos.

Algoritmo de Escalonamento: **Prioridade**

- Enquanto houver processos na classe maior: rode cada um de seus processos usando Round-Robin (quantum fixo de tempo).
- Se essa classe não tiver mais processos: passe a de menor prioridade.
- Não esqueça de ajustar as prioridades de alguma forma. Do contrário, as menos prioritárias podem nunca rodar (**inanição**).

Algoritmo de Escalonamento: **Múltiplas Filas**

- Cada vez que um processo é executado e suspenso, ele recebe mais tempo para execução;
- Inicialmente recebe 1 quantum, e é suspenso;
- Então muda de classe e recebe 2, sendo suspenso;
- Reduz o número de trocas de processo;
- Os mais curtos terminam logo;
- Aos mais longos é dado mais tempo, progressivamente;
- Preemptivo.

Algoritmo de Escalonamento: **Múltiplas Filas**

- Um processo precisa de 100 quanta para ser executado;
- Inicialmente, ele recebe um quantum para execução;
- Das próximas vezes ele recebe, respectivamente, 2, 4, 8, 16, 32 e 64 quanta (7 chaveamentos) para execução;
- Quanto mais próximo de ser finalizado, menos frequente é o processo na CPU;
- Mais ele desce na fila de prioridade;
- Eficiência – **os pequenos ainda têm vez**.

Algoritmo de Escalonamento: **Shortest Process**

- Mesma ideia do Shortest Job First
- Processos Interativos: não se conhece o tempo necessário para execução;
- Como empregar esse algoritmo: **ESTIMATIVA de TEMPO**!
- Com base em execuções antigas da mesma tarefa;
- Verificar o comportamento passado do processo e estimar o tempo.

Algoritmo de Escalonamento: **Garantido**

- Garantias são dadas aos processos dos usuários:
- n usuários (ou processos, em sistemas monousuário) → 1/n do tempo de CPU para cada usuário.

Algoritmo de Escalonamento: **Loteria**

- Cada processo recebe “bilhetes” que lhe dão direito a recursos do sistema (inclusive processador);
- **Fatias de processamento iguais** por bilhete;
- Quando um escalonamento deve ser feito, escolhe-se **aleatoriamente** um bilhete.
- Processos mais importantes podem receber mais bilhetes;
- Processos podem doar bilhetes para colaboração com outros.

Algoritmo de Escalonamento: **Loteria**

- Precisa garantir que todos terão sua vez de rodar.
- Um modo é manter **duas filas**:
  - **Bilhetes já sorteados**
  - **Bilhetes ainda não sorteados**
- Quando a lista de não sorteados se esvazia, os bilhetes da lista de sorteados são transferidos a ela, reiniciando o processo.

Algoritmo de Escalonamento: **Fair-Share**

- O dono do processo é levado em conta;
- Se um usuário A possui 9 processos e um usuário B apenas 2 processos: o usuário A ganha 90% do uso da CPU, com **round-robin (injusto)**;
- Se a um usuário foi prometido uma certa fatia de tempo, ele a receberá, independentemente do número de processos (ex.: **50% para A e 50% para B**)

Algoritmo de Escalonamento: **Fair-Share**

- Usuário 1 → A, B, C, D
- Usuário 2 → E
- Foi prometido 50% da CPU a cada um, e foi usado Round-Robin:
  - A, E, B, E, C, E, D, E, A, E, ...
- Se 2/3 devem ir ao Usuário 1:
  - A, B, E, C, D, E, A, B, E, ...

Algoritmo de Escalonamento: **Tempo Real**

- Tempo é um fator crítico;
- Tipicamente, um ou mais aparatos externos ao computador geram estímulos;
- O computador deve reagir apropriadamente **dentro de um intervalo fixo de tempo**.
- Sistemas críticos:
- Piloto automático de aviões;
- Monitoramento de pacientes em hospitais;
- Controle de automação em fábricas

Algoritmo de Escalonamento: **Tempo Real**

- Eventos causam a execução de processos;
- Quando um evento externo (sensor?) é detectado, cabe ao escalonador arrumar os processos de modo que todos os prazos sejam cumpridos;
- Eventos podem ser classificados como:
- Periódicos: ocorrem em intervalos regulares de tempo;
- Aperiódicos: ocorrem em intervalos irregulares de tempo.

Algoritmo de Escalonamento: **Tempo Real**

- Algoritmos podem ser estáticos ou dinâmicos.
  - **Estáticos:** decisões de escalonamento antes do sistema começar a rodar;
- Informação disponível previamente sobre tarefas e prazos.
  - **Dinâmicos:** decisões de escalonamento em tempo de execução.

Parte 7:
Introdução aos Threads

Modelo de **Processos**

- Utilizado para agrupar recursos;
- Um espaço de endereço (0 até algum endereço máximo do processo) e **uma única linha de execução (Thread)**;
- Agrupamento de recursos (espaço de endereço com texto e dados do programa, arquivos abertos, processos filhos, tratadores de sinais, alarmes pendentes, etc.)

Modelo de **Thread**

- **Um espaço de endereço** e múltiplas linhas de controle;
- Conjunto de threads compõe as linhas de execuções de um processo;
- Threads **compartilham um mesmo espaço de endereço** (sendo menos independentes que processos);
- Possuem recursos particulares (PC, registradores, pilha).

Modelo de Thread: **Vantagens**

- Em muitas aplicações há múltiplas atividades ao mesmo tempo.
- Podemos decompô-las em atividades paralelas.
- Algumas tarefas precisam do compartilhamento do espaço de endereçamento.
- **CPU-bound** e **I/O-bound** podem se sobrepor, acelerando a aplicação (fica a dica ao programador).

Modelo de Thread: **Vantagens**

- São mais rápidas de criar e destruir que processos.
- Em algumas ocasiões, até 100 vezes mais rápidas.
- Úteis em sistemas com múltiplas CPUs -> **paralelismo real**.

Modelo de Thread: **Exemplos**

- Processador de texto
- Processos separados não funcionam - **o documento tem que estar compartilhado**
- Threads para: Identação, fonte, correção, mudança de linha, etc.

Modelo de Thread: **Servidor Web**

- O despachante (i) lê as requisições de trabalho que chegam, (ii) escolhe uma thread operário ociosa e (iii) entrega a requisição. A thread operário (iv) lê a cache e, caso não encontre a informação, (v) inicializa uma leitura de disco.

Modelo de Thread: **Servidor Web**

- O despachante (i) lê as requisições de trabalho que chegam, (ii) escolhe uma thread operário ociosa e (iii) entrega a requisição. A thread operário (iv) lê a cache e, caso não encontre a informação, (v) inicializa uma leitura de disco.

Processos **vs** Threads

- Cada thread tem sua própria pilha de execução (pois chamam rotinas diferentes), embora **compartilhe o espaço de endereçamento e todos os seus dados**.

Problemas com as Threads

- **Como cada thread pode ter acesso a qualquer endereço de memória dentro do espaço** de endereçamento do processo:
  - uma thread pode ler, escrever ou apagar a pilha ou as variáveis globais de outra thread.
  - Exemplo:
    - a = b + c;
    - x = a + y.
- Necessidade de **sincronizar** a execução (assunto das próximas aulas)

Tipos
de Threads

- No Modo Usuário
- No Modo Núcleo
- Híbrido

Tipos de Threads: **Modo Usuário**

- Implementadas totalmente no espaço do usuário.
- Por meio de uma biblioteca (criação, exclusão, execução etc., não necessariamente gerenciamento).
- Criação e escalonamento são realizados sem o conhecimento do kernel.
- Para o kernel, é como se rodasse um programa monothread.
- Gerenciadas como processos no kernel.

Tipos de Threads: **Modo Usuário**

- Cada processo possui sua própria tabela de threads.
- Como uma tabela de processos, gerenciada pelo runtime.
- Controla apenas as propriedades da thread (PC, ponteiro da pilha, registradores, estado, etc.).

Tipos de Threads: Modo Usuário Escalonamento

- O núcleo escolhe um processo e passa o controle a ele, que **escolhe uma thread**.
- A gerência da thread fica no espaço do usuário e o núcleo só escalona em nível de processo.

Tipos de Threads: **Modo Núcleo**

- Suportadas diretamente pelo SO.
- Criação, escalonamento e gerenciamento são feitos pelo kernel.
- O núcleo possui tabela de threads (com todas as threads do sistema) e tabela de processos separadas.
- As tabelas de threads estão agora no kernel.
- Os algoritmos mais usados são Round-Robin e Prioridade.

Tipos de Threads: **Modo Núcleo**

- Agora, as tabelas de threads estão no núcleo.
- Gerenciar threads em modo kernel é mais caro devido à alternância entre modo usuário e modo kernel.
- Mudança de contexto pode envolver threads.
- Criar e destruir threads no núcleo é mais caro.
  - **Exemplo:** Linux, Família Windows, OS/2, Solaris 9 (mapeia 1 thread usuário para 1 de kernel, i.e. 1:1)

Tipos de Threads: Modo Núcleo Escalonamento

- O núcleo escolhe a thread diretamente.
- A thread é quem recebe o **quantum**, sendo suspensa se excedê-lo.
- **Thread bloqueada por E/S não bloqueia o processo**.
- Permite múltiplas threads em paralelo.

Tipos de Threads: **Híbridas**

- Seguem o modelo N para M:
- N threads de usuário são mapeadas em M <= N threads de núcleo
  - Ex.: Solaris até versão 8, HP-UX, Tru64 Unix

Parte 8:
Introdução ao Gerenciamento de Memória

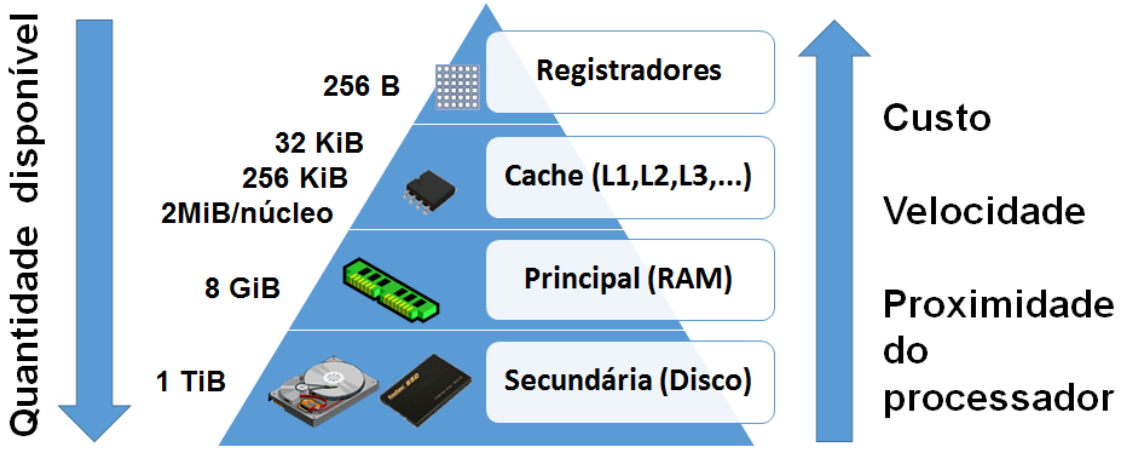
Gerenciamento de Memória

- Idealmente os programadores querem uma memória que seja:
  - Grande
  - Rápida
  - Não volátil
  - De baixo custo
- Infelizmente a tecnologia atual não comporta tais memórias.

Gerenciamento de Memória

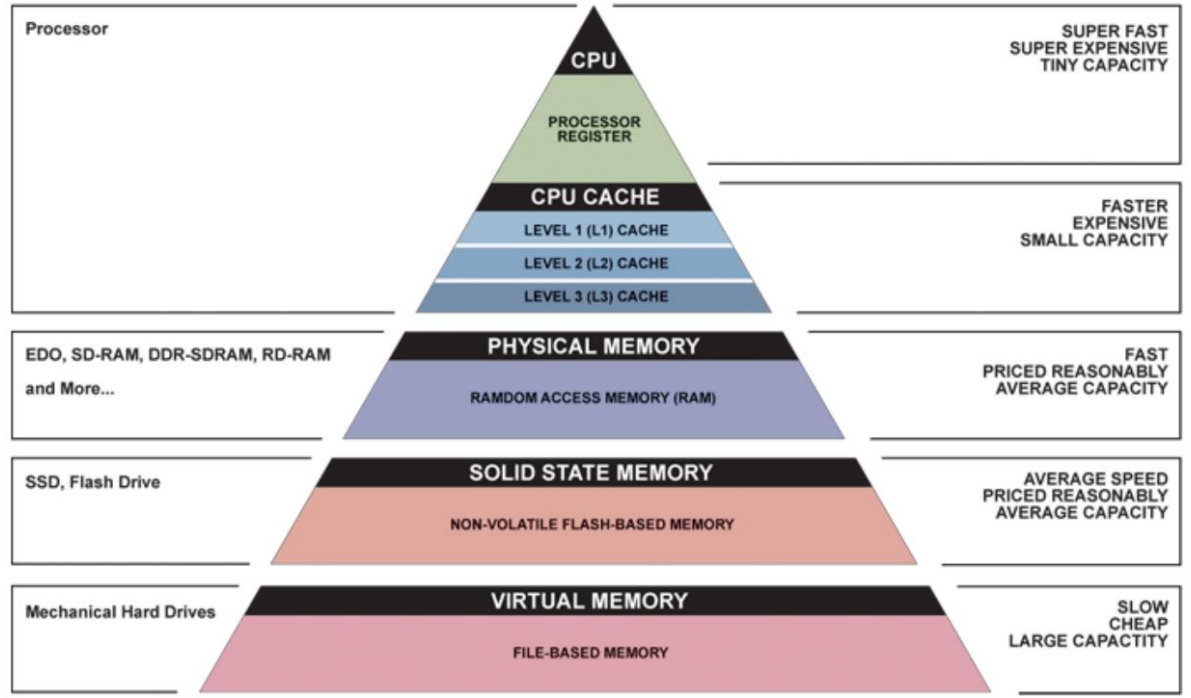
- Idealmente os programadores querem uma memória que seja:
  - Grande
  - Rápida
  - Não volátil
  - De baixo custo
- Infelizmente a tecnologia atual não comporta tais memórias.

Hierarquia de Memória



Hierarquia de Memória

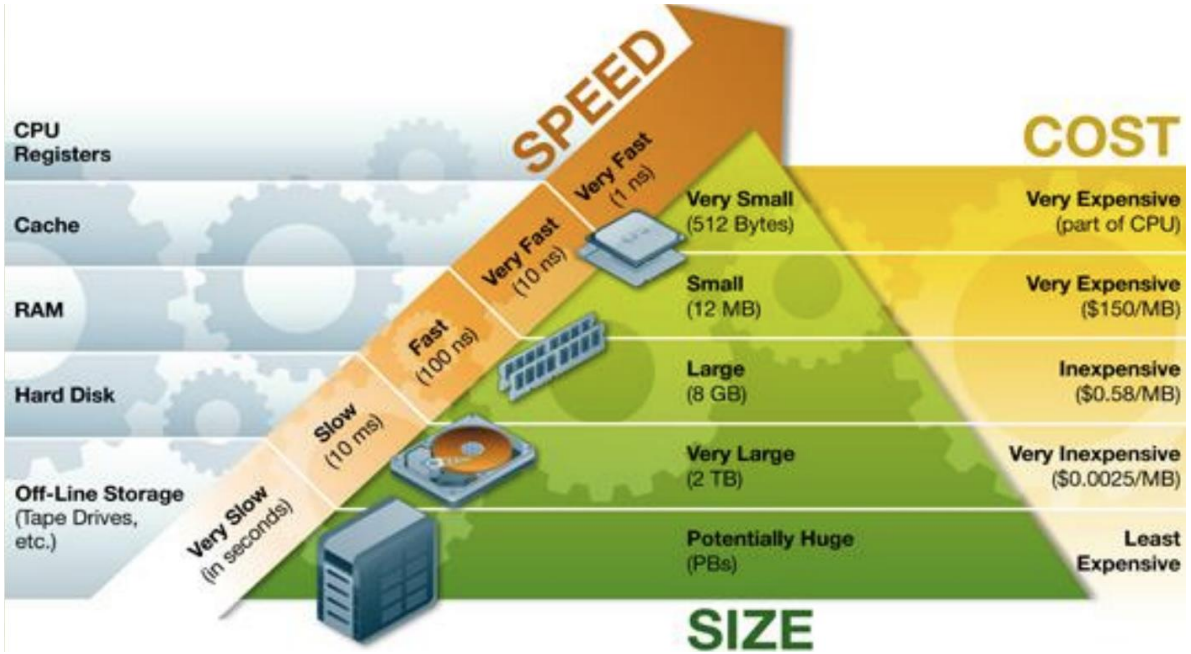
- Níveis intermediários usados para amortizar a diferença de velocidade entre processador e memória.



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Hierarquia de Memória

- Comparação de tempo de acesso, tamanho e custo, entre diferentes níveis da hierarquia de memória.



Tarefas de Gerenciamento de Memória

- Gerenciar a hierarquia de memória
  - Gerenciar espaços livres/ocupados.
  - Alocar e localizar processos/dados na memória.
- Controlar as partes que estão em uso, e as que não, para:
  - **Alocar** memória aos processos, quando estes precisarem.
  - **Liberar** memória quando um processo termina.

Tarefas de Gerenciamento de Memória

- Controlar as partes que estão em uso, e as que não, para:
  - Tratar do problema do swapping.
  - Responsável por gerenciar **chaveamento entre a memória principal e o disco** e memória principal e memória cache.
- Swap?
  - Veremos mais adiante.

Gerenciamento de Memória: **Multiprogramação**

- Como fazer para armazenar n processos na memória?
- Divida a memória em n partições, de tamanho fixo.
- Não necessariamente iguais.
- Ao chegar um job, coloque-o na fila.
- O espaço que sobrar não será utilizado.

Gerenciamento de Memória: **Multiprogramação Endereçamento**

- Como dar a cada programa seu próprio espaço de endereços, de modo que o endereço 28 em um seja diferente, na memória física, do 28 em outro?
- **2 registradores → base e limite.**
- A CPU adiciona o valor base ao endereço.
- Verifica se o endereço é maior ou igual ao limite.
- Obsoleto

O que é **MMU**?

- **Dispositivo de hardware que transforma endereços virtuais em endereços físicos**.
- Na MMU (Memory Management Unit):
- **O valor no registro de realocação é adicionado a todo o endereço lógico**.
- O programa manipula endereços lógicos; ele nunca vê endereços físicos reais.

Tipos de **Memória Particionada**

- Partições fixas (ou alocação estática)
  - Tamanho e número de partições são fixos (estáticos).
  - Tendem a desperdiçar memória.
  - Mais simples.
- Partições variáveis (ou alocação dinâmica)
  - Tamanho e número de partições variam.
  - Otimiza a utilização da memória, mas complica a alocação e liberação.
  - Partições são alocadas dinamicamente.
  - < fragmentação interna e > fragmentação externa.

O que é **Swapping**?

- Chaveamento de processos entre a memória e o disco.
- Swap-out
  - Da memória para o disco para uma área de “swap”.
- Swap-in
  - Do disco para a memória.
- Fragmentação interna e externa.

Estruturas para Gerenciar a Memória

- Usando Mapa de Bits (Bitmaps)
  - Memória é dividida em unidades de alocação.
  - Unidade pode conter vários KB.
  - Cada unidade corresponde a um bit no bitmap:
    - 0 → livre
    - 1 → ocupado
- Lista Encadeada
  - Manter uma lista ligada de segmentos de memória livres e alocados.

Algoritmos de **Alocação**

- **Primeira escolha**
  - Percorre a lista até que encontre o primeiro que caiba.
- **Melhor escolha**
  - Busca a lista inteira e toma a menor partição.
- **Pior escolha**
  - Busca a lista inteira para a maior partição.

Parte 9:
Introdução à Memória Virtual e Paginação

Memória **Virtual**

- **É uma técnica que usa a memória secundária como uma “cache” para partes do espaço dos processos.**
- Por que memória virtual?
  - O tamanho do software cada vez maior.
  - Maior grau de multiprogramação.
  - Executa programas maiores que a RAM.
- Um processo usa endereços virtuais e não físicos.
  - Utiliza o MMU para conversão.

Técnicas de Memória Virtual

- **Paginação**
  - Blocos de tamanho fixo (e.g. 4KB).
  - O espaço de endereçamento virtual é dividido em páginas virtuais.
- **Segmentação**
  - Blocos de tamanho arbitrário chamados de segmentos.
  - Contém mesmo tipo de informações (e.g. dados, pilha)
- Mapeamento entre endereços reais e virtuais (MMU).
- Muitos SOs usam uma “mistura” das duas técnicas.

Paginação

- **Páginas –** unidades de tamanho fixo no dispositivo secundário.
- **Frames –** unidades correspondentes na memória física (RAM).
- **Page fault –** é o evento quando uma página que não está na RAM é referenciada.
  - Usa uma trap para carregar ou substituir uma página.
- **Tabela de Páginas –** estrutura para mapear uma página ao frame correspondente.
  - Cada processo tem um.

Falhas de Página

- **Soft miss**
  - Quando a página referenciada não está na TLB, mas na RAM.
  - Basta atualizar a TLB.
- **Hard miss**
  - A página não está na memória física (e nem na TLB).
  - Trazer do disco à RAM (e então à TLB).
  - Muito lento.

Parte 10:
Introdução aos Processos de Entrada e Saída (E/S)

Entrada e Saída no Sistema Operacional

- Além de processos, RAM, etc.
- SO controla todos os dispositivos de E/S (ou I/O).
- Comandos para:
  - Emitir instruções (read, write, etc.).
  - Interceptar interrupções.
  - Tratar erros.
- Interface única
  - Para todos os dispositivos.
- Boa parcela do SO (e.g. drivers, overhead, etc.).

Conexão **Serial**

- Única linha de conexão.
- Vantagens:
  - Mais barata que a paralela.
  - Mais lenta que a paralela.
  - Relativamente confiável.
  - Usada em dispositivos mais baratos e lentos.
  - Impressoras e terminais.

Conexão **Paralela**

- Várias linhas e bits de dados.
- Características principais:
  - Mais complexa que a serial.
  - Mais cara.
  - Mais rápida.
  - Altamente confiável.
  - Usada em dispositivos mais velozes.
    - Exemplo: disco.

Tipo de **transferência de E/S**

- **Dispositivos de bloco (block devices):**
- Blocos de tamanho fixo.
  - Cada um com endereço.
  - Tamanho de 128 a 1024 bytes.
- Transferências com um ou mais blocos.
- Referência de localidade.
- Mais otimizados.
- E/S toma tempo.
- Exemplos: HD, CD-ROM, Drive USB.

Tipo de **transferência de E/S**

- **Dispositivos de caracter (character devices):**
  - Acessam um fluxo de caracteres.
  - Não consideram bloco.
  - Não são endereçáveis.
  - Não possuem acesso aleatório (“seek operation”).
- Ex: impressoras, interfaces de rede, mouses.

Compartilhamento de **conexões**

- **Multiponto**
- Compartilha um conjunto de linhas.
- Entre diversos periféricos.
- Maior escalabilidade que a ponto a ponto.
- Não permite paralelismo.
- Usada para armazenamento.
- Interfaces de rede: crossbar.
- Exemplos: IDE, SCSI, USB, etc.

Princípios de **Hardware**

- As unidades de E/S:
- **Componente mecânico:**
  - O dispositivo em si.
- **Controlador de dispositivo:**
  - Componente eletrônico do dispositivo.
  - Parte programável.
  - Nos PCs, um CI.
  - Há controladores para dispositivos idênticos.
  - Órgãos de padronização: IEEE, ISO, ANSI, etc.

Princípios de **Hardware**

- O SO trata com a controladora.
- Não lida com os dispositivos.
- Comunicação CPU e controladoras.
  - Usa barramentos comuns (interface de alto nível).
- Interface entre controladora e dispositivo: baixo nível.

Atuação da **controladora de disco**

- **Recebe um fluxo de bits com:**
  - Preâmbulo.
  - Os bits do setor.
  - Checksum (Error-Correcting Code – ECC).
- **Do dispositivo:**
  - Monta os bytes em bloco.
  - Coloca-os em um buffer interno.
  - Após verificar a checksum.
  - Copia o bloco na RAM.

Princípios de Hardware: **Controladora**

- Cada controladora possui registradores.
  - **Controle pela CPU.**
  - **Dados** para ler/escrever no dispositivo.
- O SO pode comandar o dispositivo.
- Escrevendo comandos e seus parâmetros.
- Depois de escrever comando.
- CPU executa outra tarefa; interrupção quando termina.
- Usa estes registradores.
- Saber o estado do dispositivo, etc.

Formas de Comunicação: **Controladora X CPU**

- **Mapeada na memória**
- Está dentro do espaço da memória.
- Usa um conjunto de endereços reservados.
- Registradores são tratados como posições de memórias.
- Mapeia todos os registradores no espaço de memória.
- Geralmente o topo do espaço de endereços.

Formas de Comunicação: **Controladora X CPU**

- **Mapeada em E/S**
- Controladora recebe um número de portas de E/S.
- Acessadas via instruções especiais.
- Usadas apenas pelo SO.
  - IN REG, PORT
  - OUT PORT,REG
- Espaços de endereços diferentes.
- Para a memória e E/S.

Formas de Comunicação: **Controladora X CPU**

- **Híbrido**
- Buffers de dados na memória.
- Portas de E/S para controle.
- Exemplo no Pentium:
  - 640 K a 1M – 1 para buffer de dados.
  - Portas de E/S de 0 a 64K – 1 (p/ controle).

Princípios de **Software** para E/S

- Independência de dispositivo:
  - Não deve se preocupar com o dispositivo (CD, HD etc.).
  - Cabe ao SO cuidar das particularidades.
- Nomenclatura uniforme:
  - Nome independente do dispositivo.

Princípios de **Software** para E/S

- Objetivos dos princípios de software.
- Tratamento de erro:
  - Devem ser tratados o mais próximo do HW possível.
  - Sem que o usuário tome conhecimento.
  - Ex: repetindo a operação.

Modos de Operação de E/S

- E/S programada.
- E/S via interrupções.
- E/S via acesso direto à memória.

- O que distingue as três formas?
  - É a participação da CPU.
  - E a utilização das interrupções.

Modos de Operação de E/S: **Programada**

- Forma mais simples de E/S → tudo é feito pela CPU.
- Os dados são trocados entre a CPU e o módulo de E/S.
- A CPU executa um programa que:
  - Verifica o estado do módulo de E/S.
  - Envia o comando de operação.
  - Aguarda o resultado.
  - Efetua a transferência para registrador da CPU.

Modos de Operação de E/S: **Programada**

- **Desvantagens:**
- CPU é ocupada o tempo todo.
- Ela realiza toda a E/S.
- Espera ocupada para saber se imprimiu:
  - também chamada de polling.
- Outro exemplo: leitura.
  - Escrita é semelhante.

Modos de Operação de E/S: **Interrupção**

- Um exemplo de interrupção.
  - CPU requisita uma leitura no disco.
  - Controladora lê os dados enquanto a CPU faz outras tarefas.
  - Controladora envia uma interrupção à CPU.
  - CPU solicita os dados.
  - Controladora envia os dados.

Modos de Operação de E/S: **Interrupção**

- Superar problema da espera da CPU.
  - Ex: impressora que não armazena caracteres.
  - Quando pede mais dados, gera interrupção à CPU.
- A CPU:
  - Envia um comando para E/S.
  - Executar outra tarefa.
  - Envia sinal quando E/S realizada.
  - CPU lê os dados da controladora.

Modos de Operação de E/S: **Interrupção**

- Interrupções são identificadas por números.
- O menor número tem prioridade sobre o maior.
- Ex: mapeamento das interrupções em um sistema compatível com IBM.

Modos de Operação de E/S: **Acesso direto à memória**

- Inconvenientes das técnicas anteriores:
  - Limitam a capacidade de transferência da CPU.
  - CPU fica ocupada no gerenciamento.
  - Desempenho cai para grandes dados.

Modos de Operação de E/S: **Acesso direto à memória**

- Solução?
- Acesso direto à memória.
- Tirando a CPU do gerenciamento.
- **Direct Memory Access (DMA)**.
- Necessita de uma controladora de DMA.
- **DMA faz a E/S programada, em vez da CPU**.

Modos de Operação de E/S: **E/S via DMA**

- Informações DMA?
- Endereço de memória.
- Quantidade de bytes.
- A porta de E/S a ser usada.
- Direção da transferência (do dispositivo ou para o dispositivo).
- Unidade de transferência (byte ou palavra por vez).

Modos de Operação de E/S: **E/S via DMA**

- Desvantagem:
- A CPU mais rápida que a controladora de DMA.
- Arquitetura mais barata.
- Vantagem:
- DMA executa E/S programada.
- Controladora de DMA faz todo o trabalho.
- Libera a CPU.

Princípios de Software para E/S: **Princípio de Camadas**

- Facilita a independência dos dispositivos.
- Modularidade e coesão.
- **Camadas mais baixas:**
  - Detalhes de hardware.
  - Drivers e manipuladores de interrupção.
- **Camadas mais altas:**
  - Interface para o usuário.
  - Aplicações de usuário.
  - Chamadas de sistemas; parte independente de E/S.

Software de E/S: **Independente de Dispositivo**

- Software de E/S:
  - Há partes específicas.
  - Outras partes são independentes de dispositivo.
- Parte independente:
  - Executar E/S comuns a todos os dispositivos.
- Outras funções:
  - Fornecer interface uniforme ao software do usuário.
  - Evitar que a cada novo dispositivo criado, o SO tenha que ser modificado.

Software de E/S: **Independente de Dispositivo**

- Outras funções da parte independente.
- Fazer o escalonamento de E/S.
- Prover buffering:
  - Ajuste de velocidade.
  - Quantidade de dados transferidos.
- Cache de dados:
  - Armazenar um conjunto de dados frequentemente acessados.

Software de E/S: **Independente de Dispositivo**

- Outras funções da parte independente.
- Reportar erros e proteger contra acessos indevidos.
  - Erros de programação:
  - Ex.: leitura de um dispositivo de saída (vídeo).
- Erros de E/S:
  - Ex.: imprimir em uma impressora sem papel.
- Erros de memória:
  - Escrita em endereços inválidos.
- Definir tamanhos de blocos independentes do dispositivo.

Princípios de Software

- Fornecer interface uniforme.
- Cada dispositivo fornece driver com funções.
- Padronizar as funções fornecidas.
- O SO define que funções o driver deve fornecer.
  - sem padrão
  - com padrão

Drivers

- Particularidade dos dispositivos – registradores.
- Código específico para E/S.
- Código específico para controlar:
  - Disco HD com pratos e braço.
  - Disco sem braço como SSD (Solid State Disk).
- Drivers diferentes.
- Driver para cada impressora:
  - Mesmo fabricante.

Drivers

- Escritos pelo fabricante do dispositivo.
- SOs diferentes -> drivers diferentes.
- Fazem parte do kernel do SO.
- Acesso total a todo dispositivo.
- Problema:
  - Podem causar problemas no SO.
- Processo:
  - Compila código do driver.
  - insmod no objeto do driver.
  - rmmod se quiser retirar o driver.

Drivers

- Definir interface padrão para:
  - Drivers de dispositivos de bloco.
  - Drivers de dispositivos de caracteres.
- Requisito do SO para seguir a interface.
  - Aos desenvolvedores dos drivers.
- Cada interface contém os procedimentos para:
  - Resto do SO pode chamar para ler um bloco, escrever um caractere.
- Carregar os drivers dinamicamente, durante a execução.

Drivers: **Funções**

- Aceitar pedidos de leitura/escrita do software.
  - Independente de dispositivos.
  - Pedido vale para qualquer dispositivo.
  - Cuidar que sejam executadas.
- Fazer um check-up no início.
- Inicializar o dispositivo, se necessário.
- Gerenciar as necessidades energéticas do dispositivo.
- Criar um log de eventos.

Parte 11:
Introdução ao Sistema Operacional GNU/Linux

Revisão

- **Sistema Operacional:** é o programa que controla o computador. Serve de Interface entre o usuário e a máquina.
- Faz isso por meio de dois componentes: **Kernel** e o **Shell**.

Revisão

- **Kernel** é o nome dado ao núcleo do SO.
- A comunicação com o hardware é feita pelo kernel.

Revisão

- **Shell** é a "fachada" do SO.
- Essa parte do programa que se comunica com o usuário, recebendo seus comandos e repassando-os ao Kernel.

História do Linux

- 1970
- Computadores de grande porte em
- universidades e empresas
- Criação do Unix para os sistemas de grande porte

História do Linux

- 1980 – 1985
- Utilização dos microcomputadores (XT) com DOS
- Unix torna-se mais popular
- Nasce a FSF (Free Software Foundation) Richard Stallman

História do Linux

- 1985 – 1986
- GNU significar GNU não é Unix..
- Permitiria a criação de um Unix livre, de modo que todos pudessem ter acesso ao SO.
- GPL – General Public Licence
  - Descreve a ideia de software livre.

História do Linux

- 1985 – 1986
- GPL
- Softwares que são distribuídos sob essa licença têm que oferecer, aos usuários os direitos de:
  - Poder utilizá-lo para qualquer finalidade;
  - Copiar e distribuir livremente o programa;
  - Estudar o programa (é necessário ter acesso ao código-fonte);
  - Modificar livremente o programa (é necessário ter acesso ao código-fonte)

História do Linux

- 1986 – 1992
- FSF difunde a ideia do software livre entre usuários e universidades.
- O Shell foi criado nesta época.
- A tentativa de criar um kernel adequado não foi alcançada.
- Diversas ferramentas GNU criadas.

História do Linux

- 1990 – 1992
- Universitários treinavam muito Unix, com o Minix (para PC)
- Linus Torvalds cria o kernel semelhante ao Unix para PCs
- Linux é um Unix-Like (semelhante ao Unix)
- Linus testou seu kernel fazendo várias ferramentas GNUs funcionarem com ele.

História do Linux

- 1992
- Linus dissemina a receita do kernel (código-fontte) pela Internet a diversos programadores
- Comunidades Linux são criadas
- Empresas podem obter os fontes do kernel (www.kernel.org) e fazer seus Linux

Distribuições
Linux

- Conjunto de programas formados pelo kernel e outros softwares (aplicativos, shells, jogos, etc).
- **Você pode criar sua própria distribuição.**
- Desenvolva ou junte seus programas em torno do kernel e tenha uma nova distribuição Linux.

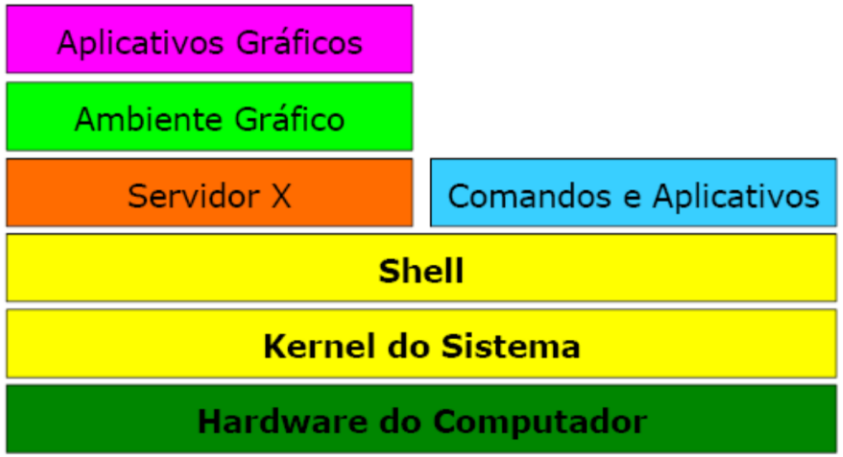
Distribuições Linux

- Distribuições mais conhecidas:
- Red Hat
- Debian
- Suse
- Ubuntu
- Linux Mint
- PopOS
- CentOS
- Slackware
- Arch Linux
- Gentoo
- Kali Linux

Distribuições Linux

- Ambientes gráficos mais conhecidos:
  - KDE
  - Gnome
  - XFCE
  - Cinnamon
  - LXDE
  - Entre outros

Distribuições
Linux



Características do Linux

- **Case-sensitive:** Difere minúsculas de maiúsculas nos nomes dos arquivos
- **Multitarefa:** Vários programas executam ao mesmo tempo
- **Multiusuário:** Vários usuários simultâneos
- **Portável:** Compilado para vários tipos hardware e processadores com diferentes arquiteturas

Características do Linux

- **Contas de Usuários**
- Cada um deve possuir sua conta (login/senha)
- Usuários tem restrições de acesso a diversos arquivos do SO
- Usuários somente podem atuar mediante a permissão de um **super-usuário**, denominado **root**. O root possui poder ilimitado para lidar com o sistema operacional

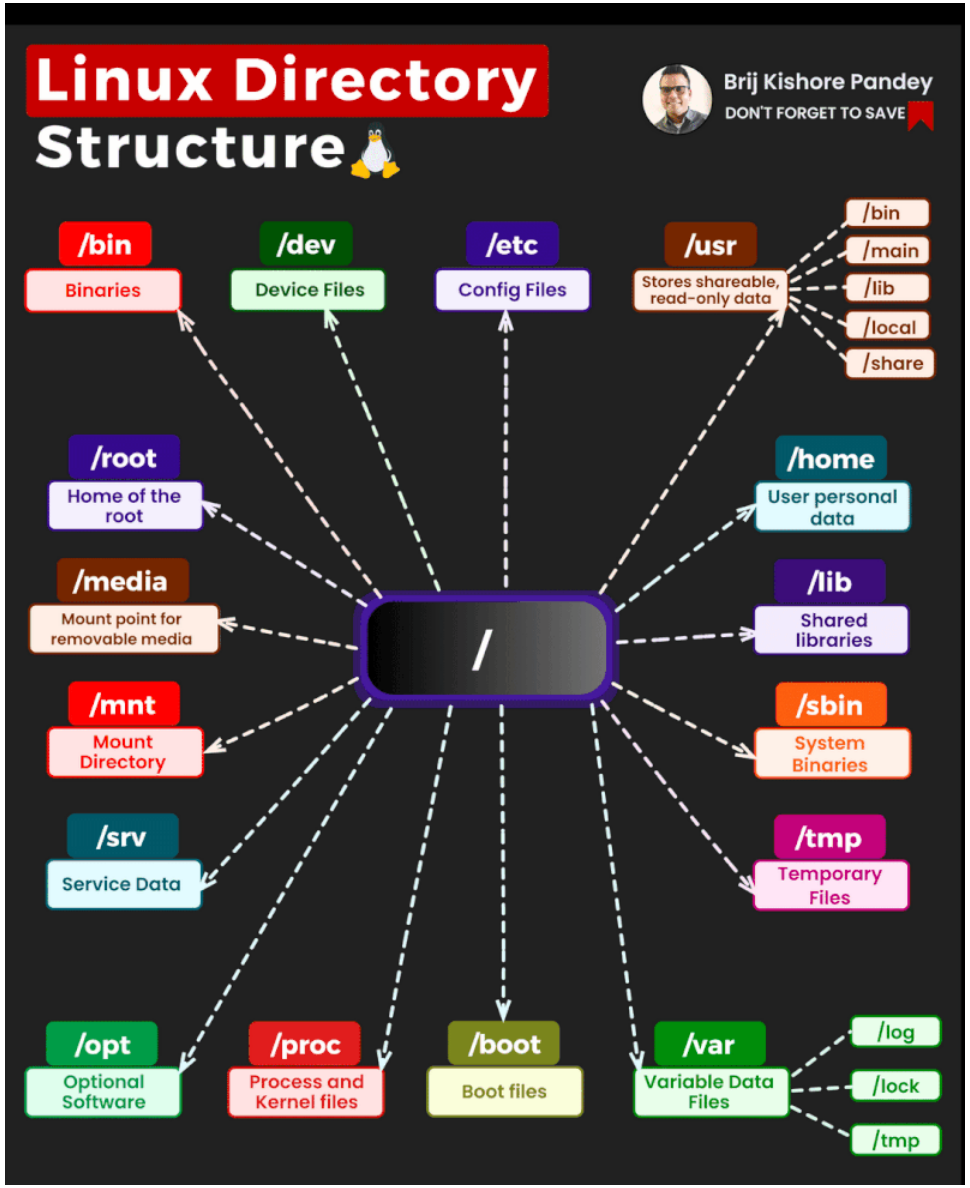
Características do Linux

- **Estrutura de diretórios**
  - **/** → Primeiro a ser lido e todos os outros ficam abaixo dele
  - **/home** → guarda os dados pessoais do usuário
  - **/root** → guarda dados pessoais do super-user do sistema operacional
  - **/bin** → Armazena arquivos executáveis (comandos e programas) → Usuários comuns tem permissão para usar os programas deste diretório
  - **/etc** → guarda os arquivos de configuração do sistema e das aplicações instaladas

Características do Linux

- **Estrutura de diretórios**
  - **/boot →** guarda os arquivos de inicialização do sistema (aqui fica o kernel e o gerenciador de boot (grub)
  - **/dev →** armazena os arquivos de dispositivos (referências ao hardware)
  - **/sbin →** Semelhante ao /bin, porém contem programas que só podem ser usados pelo root
  - **/proc →** trata-se de um diretório virtual (fica na memória) e armazena informações sobre os processos
  - **/temp →** armazena arquivos temporários
  - **/var →** utilizado por pogramas em execução para guardar informações úteis ao seu funcionamento

Estrutura de Diretórios do Linux



Características do Linux

- Permissões em arquivos
- Definem quem pode acessar determinados arquivos ou diretórios, assim mantendo segurança e organização
- Cada arquivo ou pasta tem 3 permissões:
  - (**Usuário Dono**) (**Grupo Dono**) (**outros**)
  - **Usuário dono**: é o proprietário do arquivo;
  - **Grupo Dono**: é um grupo, que pode conter vários usuários;
  - **Outros**: se encaixam os outros usuários em geral.

Características do Linux

- Permissões em arquivos
- Como as permissões são divididas em 3, irá aparecer assim:
  - (_ _ _) (_ _ _) (_ _ _), ou seja, (rwx)(rwx)(rwx)
- Caso não haja todas as permissões, poderá aparecer incompleto:
  - rwxrw_ _ _x, ou seja, neste exemplo:
  - Dono do arquivo tem permissão de Ler, Escrever e executar (rwx);
  - Grupo tem permissão de Ler e Escrever (rw_);
  - Outros tem permissão apenas de executar. (_ _ x);

Características do Linux

- Comandos mais comuns
  - ls
  - uname -a
  - uptime
  - hostname
  - dmesg
  - lsdev
  - cat /proc/devices
  - more
  - free
  - df
  - cd
  - shutdown

Características do Linux

- Comandos mais comuns
  - mkdir
  - who
  - wc
  - find / -name
  - su
  - type
  - date

Características do Linux

- Há diversas outras características do Linux que fogem do escopo desta aula.
- No decorrer da UC aprenderemos diversos outros comandos, os quais são associados também às ferramentas de administração e gerenciamento de redes.

Fim do Módulo