



Olá, Professor!  
Peço que realize a **avaliação do conteúdo** com o intuito de manter o nosso material sempre atualizado.

Avalie este  
conteúdo! 🚀 ✨



<https://bit.ly/451BDqS>



# Sistemas Operacionais



# Conceitos básicos de SO



# — Conceitos



## Hardware

Fornece recursos básicos de computação CPU, memória, dispositivos de E/S.



## Aplicativos

Definem as maneiras como os recursos são usados, para resolver os problemas de computação dos usuários



## Usuários

São as pessoas, máquinas ou outros computadores.



## Sistema operacional

Controla e coordena o uso do hardware entre os vários programas de aplicação, para os diversos usuários.

# — CONCEITOS



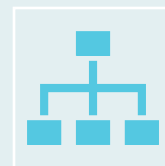
Sistema operacional é um software intermediário entre usuário e hardware, simplificando o uso do computador.



Oferece interfaces de texto ou gráficas e diferencia modo usuário (acesso limitado) de modo kernel (acesso total).



Coordena programas e gerencia dispositivos, abstraindo complexidades de hardware e outros softwares.



Possibilita eficiência ao modularizar e abstrair a visão do usuário, seguindo um modelo de máquina de níveis.



# — Histórico dos sistemas computacionais



## Primeira Geração

1945-1955: Primeira geração de computadores, programação em painéis, hardware com válvulas.



## Segunda Geração

1955-1965: Segunda geração, transistores substituem válvulas, uso de sistemas em lote com cartões e fitas.



## Terceira Geração

1965-1980: Terceira geração estabeleceu o conceito de sistema operacional, trouxe inovações e o UNIX.



## Quarta geração

1980-presente: Quarta geração com avanços como PCs, redes, Intel 4004, Microsoft, internet e dispositivos móveis.

# — Ms windows e unix

## Windows



A história do Windows começa com o MS-DOS em 1981, um sistema de linha de comando.

- O Windows 1.0, lançado em 1985, introduziu uma interface gráfica, mas ainda usava o MS-DOS.
- O Windows NT, lançado em 1993, trouxe um núcleo novo de 32 bits com multitarefa, memória virtual e suporte a múltiplos processadores.
- A evolução continuou com o Windows 2000, XP e versões subsequentes para desktops e servidores.

# — Ms windows e unix

## Unix



O Unix tem origens no desenvolvimento do MULTICS, um sistema de tempo compartilhado.

- Em 1969, Ken Thompson desenvolveu sua versão chamada UNICS, que evoluiu para Unix.
- O Unix foi reescrito em C e portado para o PDP-11 em 1973, depois licenciado para várias universidades.
- O Linux, baseado no Minix, começou em 1991 com a ajuda de programadores voluntários, unificando várias versões do Unix.



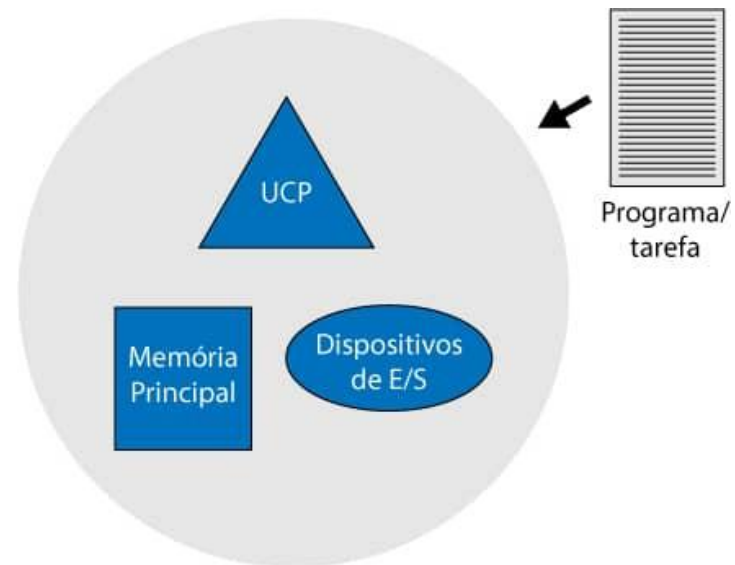
## — Tipos de sistemas operacionais

Os sistemas operacionais podem ser classificados em: Sistemas operacionais monoprogramáveis/monotarefa; sistemas operacionais multiprogramáveis/multitarefa e sistemas com múltiplos processadores.

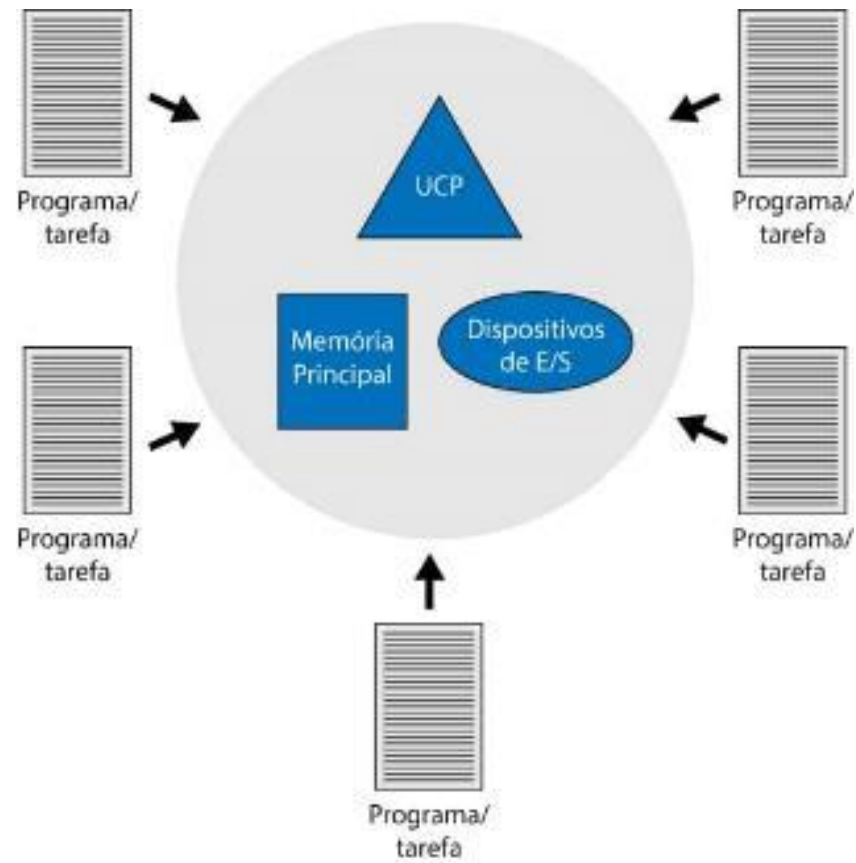


## — Sistemas monoprogramáveis

Alocam recursos exclusivamente para a execução de um único programa, sem multitarefa.

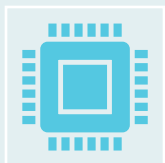


## — Sistemas multiprogramáveis ou multitarefa

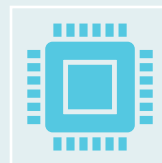


Recursos compartilhados entre usuários e aplicações, possibilitando a execução simultânea de várias tarefas.

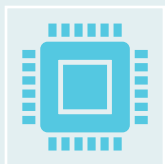
# — Sistemas multiprogramáveis ou multitarefa



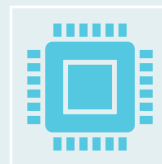
Sistemas multiprogramáveis permitem a execução simultânea de várias tarefas, evitando o desperdício de recursos.



São classificados em sistemas batch, de tempo compartilhado e de tempo real, dependendo das necessidades.



O sistema operacional gerencia vários programas ao mesmo tempo, coordenando o uso do processador e recursos.

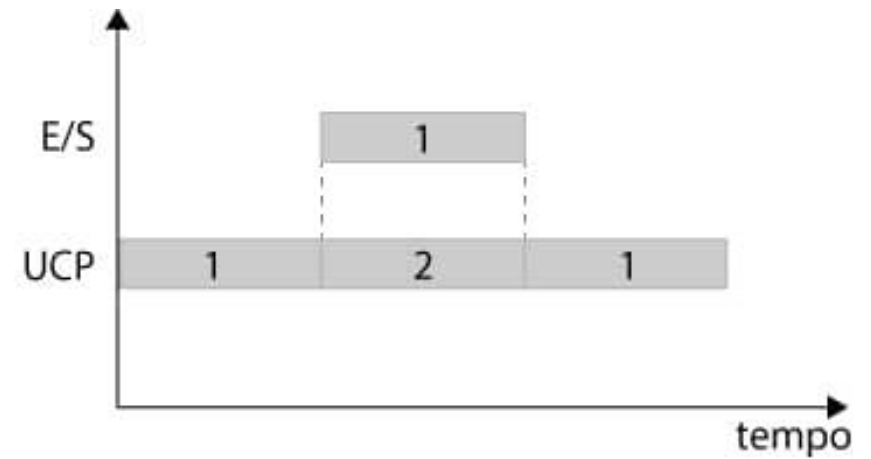


Isso otimiza o uso do processador, evitando que fique ocioso durante operações lentas de E/S.

## — Sistemas multiprogramáveis ou multitarefa



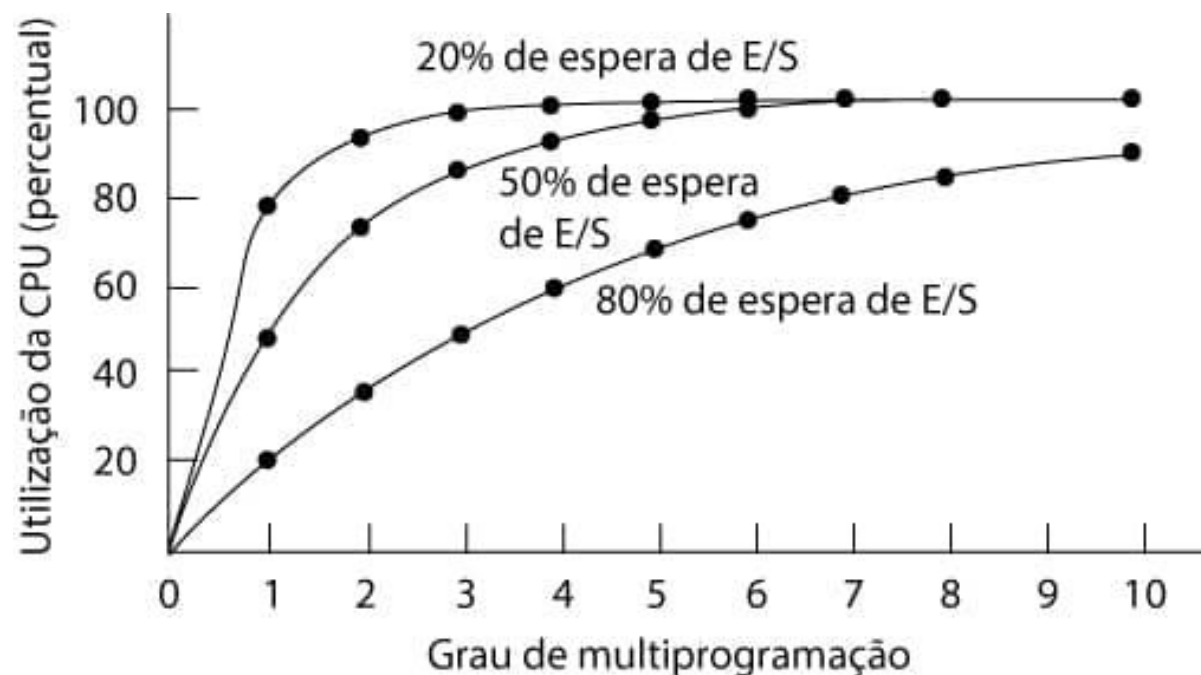
(a) Sistema Monoprogramável



(b) Sistema Multiprogramável

## — Sistemas multiprogramáveis ou multitarefa

Na multiprogramação, vários processos compartilham a CPU. Quanto maior a espera por E/S, mais processos podem ser carregados, minimizando o desperdício de CPU.





## — Sistemas multiprogramáveis ou multitarefa

A utilização da CPU é dada pela fórmula  $U_{cpu} = 1 - p^n$ , onde:

- $p$  = Tempo de espera de (dispositivos de) E/S.
- $n$  = Número de processos carregados na memória.

Por exemplo: Se  $p = 65\%$  e  $n = 3 \rightarrow U_{cpu} = 1 - 0,65^3 = 0,72$  ou 72%.

# — Sistemas com múltiplos processadores

Os **sistemas com múltiplos processadores** possuem dois ou mais processadores atuando juntos, oferecendo vantagens como:

## Escalabilidade

Aumenta a capacidade de processamento.

## Disponibilidade

Se um processador falhar, outro pode assumir a carga de trabalho.

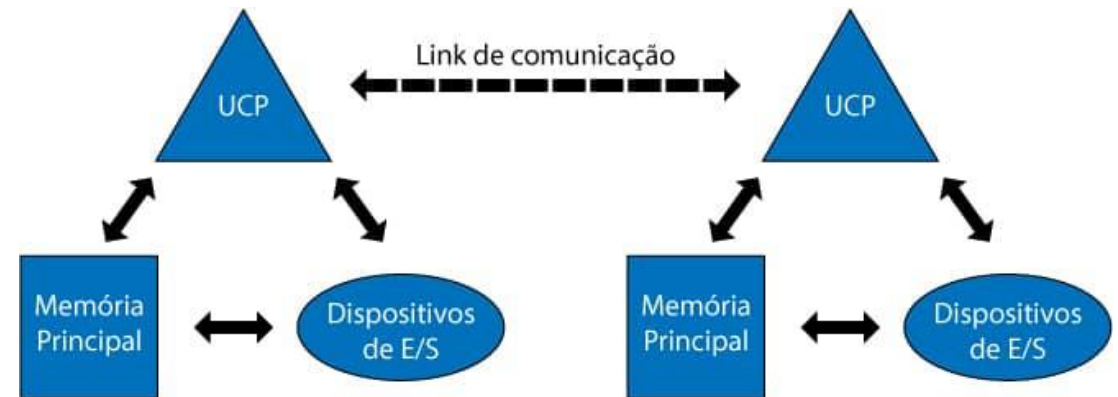
## Balanceamento de carga

Distribuição da carga de trabalho entre os processadores.

# — Sistemas com múltiplos processadores

## Sistemas fortemente acoplados

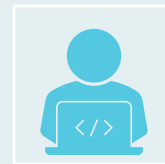
- Em sistemas fortemente acoplados, processadores compartilham memória principal e periféricos.
- Eles podem ser SMP (Symmetric Multiprocessors) ou NUMA (Non-Uniform Memory Access).



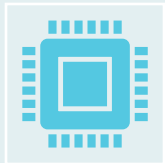
## — Outras classificações dos sistemas operacionais



Sistemas operacionais evoluíram com variedade de serviços ao longo de mais de 50 anos.



Serviços incluem processamento simultâneo, E/S, processamento em lote e tempo compartilhado.



A evolução reflete as necessidades variadas de sistemas de computadores.



Diferentes tipos de sistemas operacionais foram desenvolvidos para atender a essas demandas específicas.

## — Outras classificações dos sistemas operacionais

Quanto aos serviços citados em (3), temos

**Sistema em lote**

**Sistemas de  
processamento de  
transações**

**Sistemas de tempo  
compartilhado**

# — Outras classificações dos sistemas operacionais

Os sistemas de tempo real podem ser dos tipos:



## **Sistema de tempo real crítico**

As ações precisam ocorrer em determinados instantes.  
Exemplo: Processos industriais, aviônica, militares.



## **Sistema de tempo real não crítico**

O descumprimento de um prazo não causa dano permanente. Exemplo: Sistema de áudio digital, multimídia, telefones digitais.



## — Estrutura do so: kernel, system calls, modos de acesso

- O sistema operacional possui um núcleo chamado kernel que oferece serviços aos usuários e aplicativos.
- Estas tarefas ocorrem concorrentemente e assincronamente.



# — Estrutura do so: kernel, system calls, modos de acesso

O núcleo do SO deve atuar considerando essas particularidades, e para isso implementa funções como:

**Tratamento de interrupções e exceções.**

**Criação e eliminação de processos e threads.**

**Sincronização e comunicação entre processos e threads.**

**Gerência de memória.**

**Gerência do sistema de arquivos.**

**Gerência dos dispositivos de E/S.**

**Suporte a redes locais e distribuídas.**

**Contabilização do uso do sistema.**

**Auditoria e segurança do sistema.**

## — System calls

As system calls são usadas para acessar serviços do sistema operacional.

Elas servem como portas de entrada para o núcleo do sistema.

O sistema operacional verifica os privilégios antes de executar a operação.

Cada sistema operacional tem seu conjunto específico de chamadas, como "read" com parâmetros específicos.

## — System calls

contador = read (arq, buffer, nbytes)

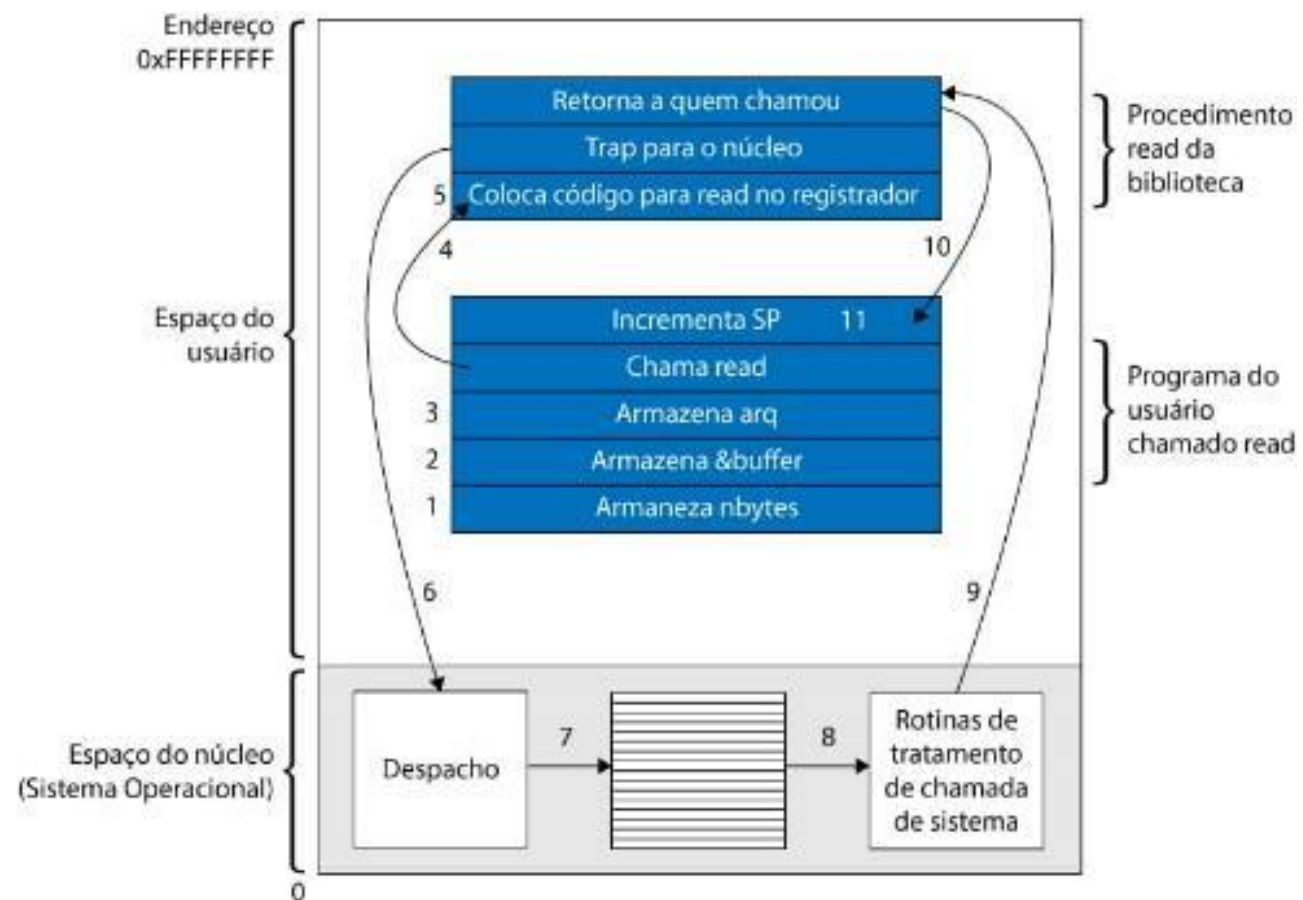
↓  
O 1º especifica  
o arquivo.

↓  
O 2º é um ponteiro  
para o buffer.

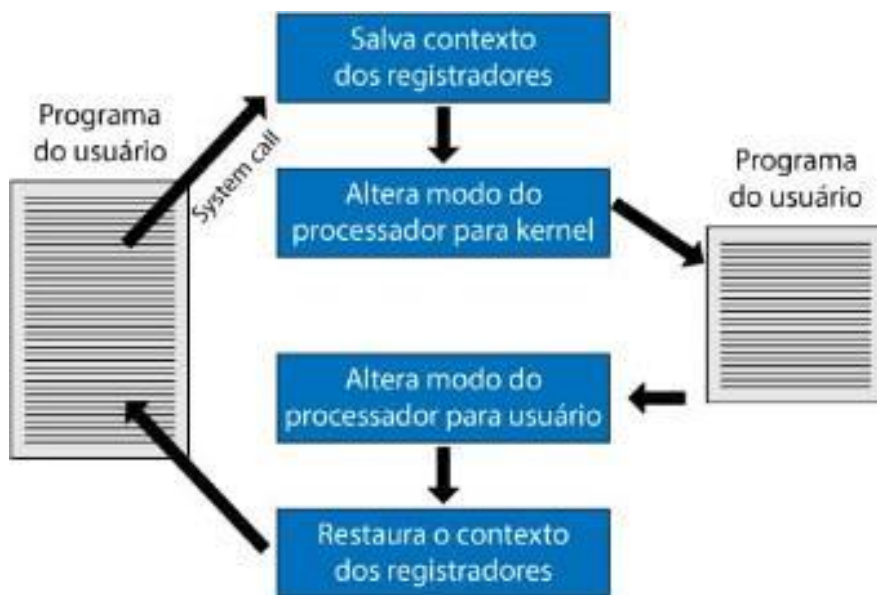
↓  
O 3º dá o número de bytes  
que deverão ser lidos.

# — System calls

Passos para a realização da system call "read"



## — System calls



O funcionamento geral de uma system call envolve uma chamada pela aplicação, processamento pelo kernel e retorno de uma resposta.

- O padrão POSIX buscou unificar as system calls em sistemas Unix.
- A API Win32, usada no Windows, possui um grande número de chamadas para gerenciar a interface gráfica e outros aspectos do sistema.
- A API Win32 difere do POSIX, sendo mais complexa e com maior número de chamadas.



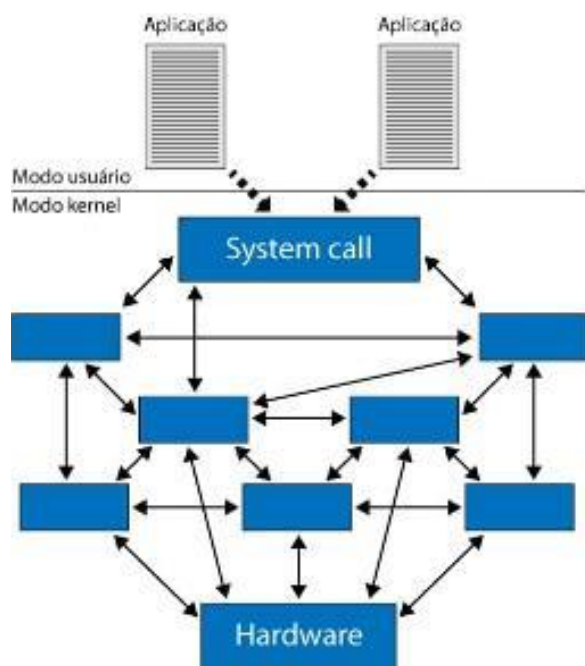
## — Modos de acesso

Os modos de acesso no sistema operacional envolvem:

- **Modo Kernel (ou Supervisor):** Instruções privilegiadas executadas por aplicações com total acesso, incluindo áreas do sistema operacional na memória.
- **Modo de Acesso Usuário:** Instruções não privilegiadas executadas por aplicações que devem chamar rotinas do sistema via System Call.

# — Arquiteturas de kernel

## Arquitetura monolítica (mono-kernel)

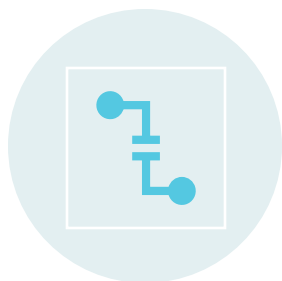


Arquitetura monolítica é uma abordagem onde todas as rotinas do sistema operacional são compiladas e ligadas juntas em um único programa executável.

- Todas as rotinas em uma arquitetura monolítica são visíveis e não há ocultação de informações entre elas.
- Essa arquitetura facilita a interação entre as rotinas, mas pode tornar o sistema menos modular e mais difícil de manter.
- É uma das várias formas de implementar um sistema operacional.

# — Arquiteturas de kernel

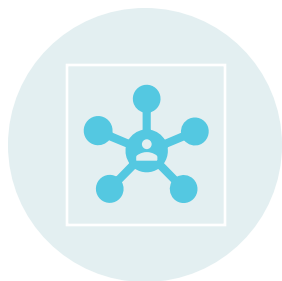
## Arquitetura de camadas



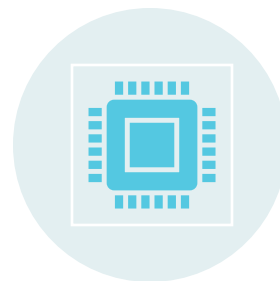
A arquitetura de camadas divide o sistema em várias camadas que oferecem funções isoladas.



Cada camada oferece serviços para a camada imediatamente superior.



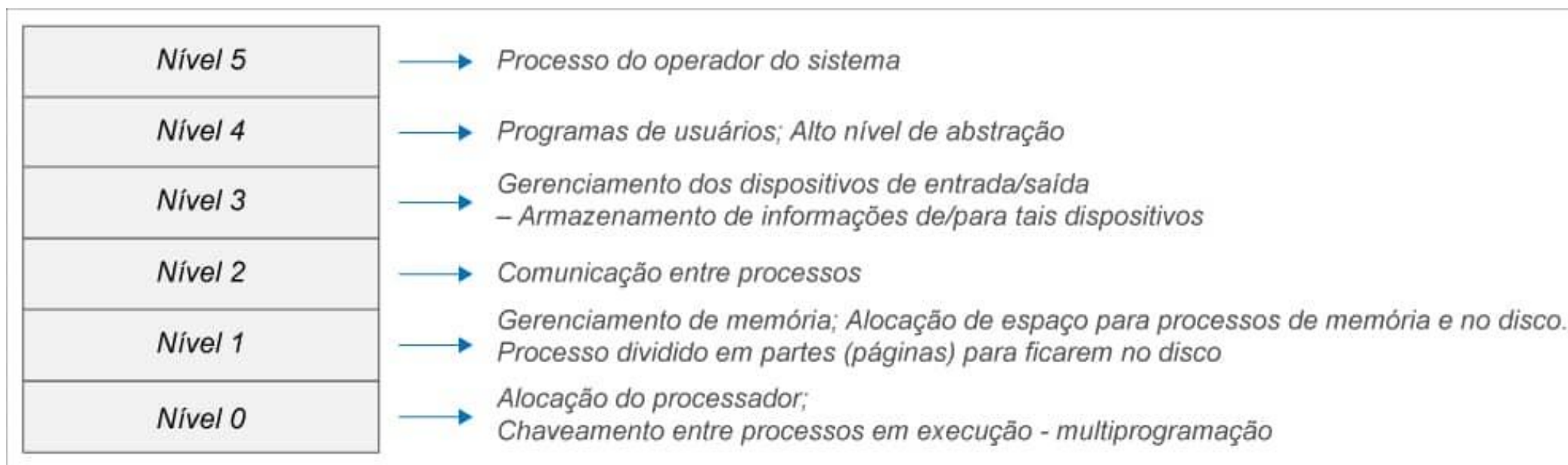
Isola funções do SO, facilita a manutenção, mas pode afetar o desempenho devido à comunicação entre camadas.



Muitos SOs comerciais para workstations usam essa abordagem com duas camadas principais.

# — Arquiteturas de kernel

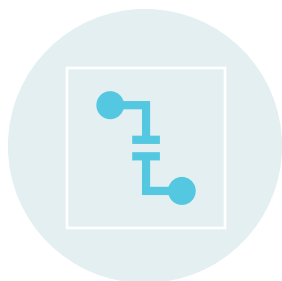
## Arquitetura de camadas



Arquitetura de camadas. Fonte: Adaptado de Tanenbaum, 2010.

# — Arquiteturas de kernel

## Máquina virtual ou virtual machine (VM)



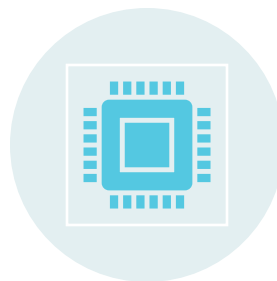
Máquinas virtuais criam um nível intermediário entre hardware e SO, permitindo múltiplos ambientes isolados.



Cada máquina virtual fornece uma cópia virtual do hardware, incluindo modos kernel, usuário e E/S.



Virtualização é usada para economizar recursos e executar vários sistemas em um único hardware.



CPUs virtuais são necessárias para executar software de máquina virtual, evitando instruções privilegiadas no modo usuário.

# — Arquiteturas de kernel

## Arquitetura microkernel



Microkernel: Núcleo mínimo e simples com funções específicas.



Implementação desafiadora; muitas vezes, usada uma abordagem híbrida.



Divisão em módulos para alta confiabilidade; erros locais não afetam o sistema.



Micronúcleo no modo núcleo; outros módulos em processos de usuário.



# — Arquiteturas de kernel

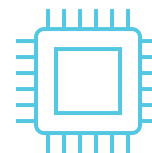
## Exonúcleo



Exonúcleo divide recursos entre máquinas virtuais, poupando camadas de mapeamento.



Alocação de recursos e prevenção de uso por outras máquinas virtuais.



Elimina necessidade de tabelas de remapeamento de endereços de disco.



Mantém registro de recursos atribuídos a cada máquina virtual.

# — Conceitos

- O Linux é um sistema operacional gratuito e colaborativo baseado em padrões Unix.
- É "Unix-like," oferecendo multitarefa e multiusuário.
- Distribuições Linux incluem aplicativos e o kernel.
- Exemplo de distribuição: Ubuntu.



## — Conceitos

Cada versão do Ubuntu recebe uma numeração e um codinome:

Denominação da versão	20.	04	LTS	Focal Fossa
Significado	Ano (2020)	Mês (04 – abril)	Long-Term Support (Suporte de longo prazo - 5 anos de suporte (atualizações))	Codinome

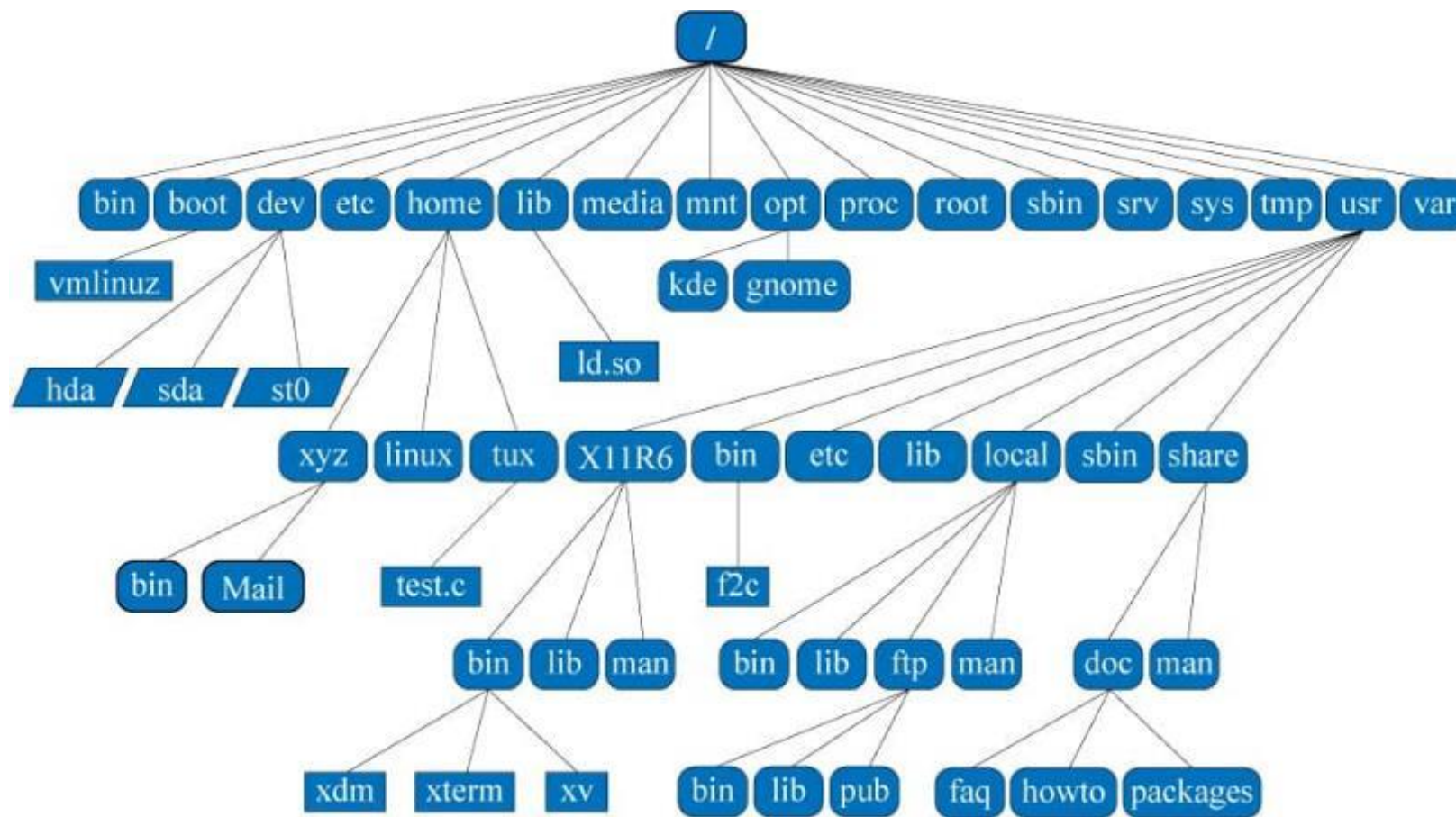
# — Estrutura de diretórios no Linux



O Linux possui uma estrutura única de pastas com uma raiz comum.

- Todos os dispositivos (internos ou externos) são montados abaixo dessa raiz.
- O sistema é case-sensitive, distinguindo maiúsculas de minúsculas.
- Caminhos de diretórios usam "/" como separador. Exemplo: /tmp/Texto.txt.

## — Estrutura de diretórios no Linux



Estrutura de pastas do Linux.

# — Estrutura de diretórios no Linux

As principais pastas e seus significados resumidos:

/	Pasta Raiz	/usr	Programas de Usuário
/bin	Executáveis Binários	/home	Pasta Pessoal
/sbin	Sistema Binário	/boot	Arquivos de Inicialização
/etc	Arquivos de Configuração	/lib	Bibliotecas do Sistema
/dev	Arquivos de Dispositivos	/opt	Aplicações Opcionais
/proc	Informação de Processo	/mnt	Pasta de Montagem
/var	Arquivos Variáveis	/media	Dispositivos Removíveis
/tmp	Arquivos Temporários	/srv	Serviço de Dados

## — Comandos do Linux

O terminal no Ubuntu Desktop é acessado pelo ícone "mostrar aplicativos" e digitando "term" na caixa de pesquisa.

O terminal exibe informações sobre o usuário e a localização na pasta pessoal.

Alguns comandos básicos para usar via terminal estão disponíveis.

Para compilar código-fonte, siga etapas como download, descompactação e uso de comandos específicos.

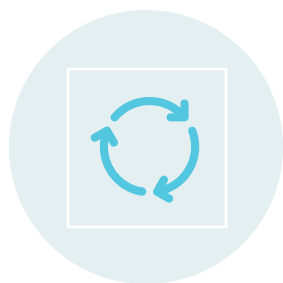
# Processos e gerencia de processador

```
click="on('overlay2')"  
;" src="img/java.png"  
ss="mb-3">Java</h3>  
s="text-muted mb-0">Ja  
ent, class-based, objec  
itation dependencies as  
  
col-lg-3 col-md-6 text  
="service-box mt-5 m  
('overlay3')"
```

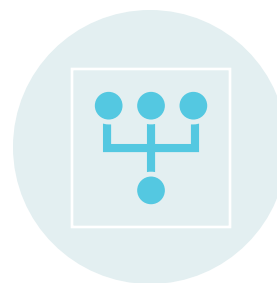


# — Conceitos

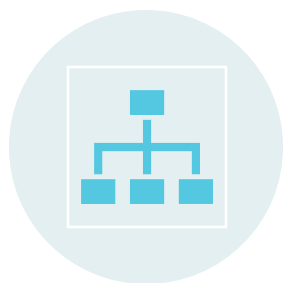
## Modelo de processo



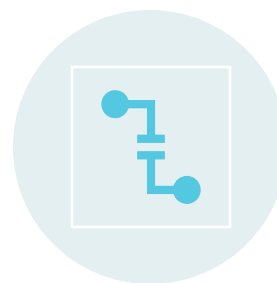
Evolução dos sistemas: Inicialmente, um programa único controlava o sistema; agora, múltiplos programas podem ser executados simultaneamente.



Conceito de processo: Processos são unidades de execução com recursos e espaço de endereçamento próprios, facilitando o paralelismo.



Processos vs. programas: Programas são entidades passivas, enquanto processos são ativos, com contadores de instruções e registradores.



Tempo compartilhado: A alternância rápida entre processos cria a ilusão de execução simultânea em sistemas multiprogramáveis.

# — Criação e encerramento de processos

Sistemas operacionais precisam criar processos durante sua operação. Quatro eventos principais fazem com que processos sejam criados:

Inicialização do sistema.

Execução de uma chamada de sistema de criação de processo por um processo em execução.

Solicitação de um usuário para criar um processo.

Início de uma tarefa em lote.

## — Criação e encerramento de processos

Processos durante inicialização: Sistema operacional inicia vários processos, incluindo daemons (em segundo plano).

Criação de processos: Processos podem criar novos processos usando chamadas de sistema.

Interatividade: Usuários iniciam programas criando novos processos por meio de comandos ou cliques.

Tarefas em lote: Sistemas executam tarefas em lote, criando processos para executar tarefas da fila de entrada.

# — Criação e encerramento de processos

O código a seguir, em Linguagem C, exemplifica a criação de um novo processo com a chamada de sistema **fork()**.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int resultado, pid, ppid;
    resultado = fork();
    if (resultado < 0)
        printf("Algo deu errado!!!\n");
    pid = getpid();
    if (resultado == 0) {
        ppid = getppid();
        printf("Eu sou o processo filho, meu PID é %d e meu pai tem PID=%d.\n", pid, ppid);
    }
    if (resultado > 0) {
        printf("Eu sou o processo pai, meu PID é %d e meu filho tem PID=%d.\n", pid, resultado);
        waitpid(resultado, NULL, 0);
    }
}
```

# — Criação e encerramento de processos

O código ilustra a utilização de **execve()**.

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/wait.h>
4
5  int main(int argc, char **argv, char* envp[]) {
6      int pid, i;
7
8      for (i = 1; i <= 3; i++) {
9          pid = fork();
10         if (pid < 0) {
11             printf("Algo deu errado!!!\n");
12             return 0;
13         }
14         if (pid == 0) { // Processo filho
15             if (i == 1)
16                 execve("/usr/bin/xcalc", argv, envp);
17             if (i == 2)
18                 execve("/usr/bin/gedit", argv, envp);
19             if (i == 3)
20                 execve("/usr/bin/xeyes", argv, envp);
21         }
22         else // Processo pai
23             waitpid(pid, NULL, 0);
24     }
25 }
```

## — Criação e encerramento de processos

Após ter executado sua tarefa, o novo processo terminará devido a uma das seguintes condições:

**Saída normal  
(voluntária)**

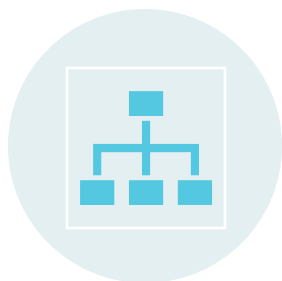
**Erro fatal  
(involuntário)**

**Saída por erro  
(voluntária)**

**Morto por outro  
processo  
(involuntário)**

# — Criação e encerramento de processos

## Hierarquia de processos



Hierarquia de processos: Em alguns sistemas, os processos criados mantêm uma relação hierárquica com o processo pai, formando uma árvore de processos.



Grupos de processos no Linux: No Linux, processos e seus descendentes são agrupados em um grupo de processos.



Processo especial no Linux: O processo systemd (ou init) é o primeiro a ser executado e inicia a execução dos demais processos do sistema operacional.



Árvore de processos no Linux: Todos os processos no Linux fazem parte de uma única árvore, com o systemd (ou init) na raiz.

## — Estados de processos



**Estados de processos:** Processos podem ser novos, prontos, executando, bloqueados ou terminados.



**Transições de estados:** Iniciam como novos, mudam para prontos e, quando a CPU está disponível, para executando.



**Estado bloqueado:** Processos aguardam operações no estado bloqueado, retornando ao estado pronto após a conclusão.



**Estado terminado:** Processos não serão mais executados; o sistema libera seus recursos após a desalocação.



# — Estados de processos

Algumas entradas do BCP são:

**Estado do processo**

**Prioridade do  
processo**

**Número do processo**

**Registradores da UCP**

**Informações relativas  
ao gerenciamento de  
memória**

**Informações de  
contabilidade**

**Informações sobre  
operações de E/S**

# — Estados de processos

## Mudança de contexto

O contexto de um processo pode ser dividido em três elementos básicos:

### Contexto de hardware

O contexto de hardware constitui-se basicamente do conteúdo dos registradores.

### Contexto de software

O contexto de software especifica características do processo que influenciarão na execução de um programa.

### Espaço de endereçamento

O espaço de endereçamento é a área de memória do processo em que o programa será executado e a área de memória onde os dados do processo serão armazenados.

# — Processos no Linux



No Linux, os processos funcionam como processos sequenciais tradicionais em um ambiente multitarefa e multiusuário.



Daemons, como o de impressão, são comuns, permitindo o compartilhamento de recursos entre processos.



A criação de processos é feita via `fork()`, gerando um processo filho com espaço de endereçamento idêntico ao pai.



Cada processo possui um PID (identificação do processo) para identificação única.

## — Subprocesso

Subprocessos: São criados quando um processo pai gera outros processos; podem criar mais subprocessos.



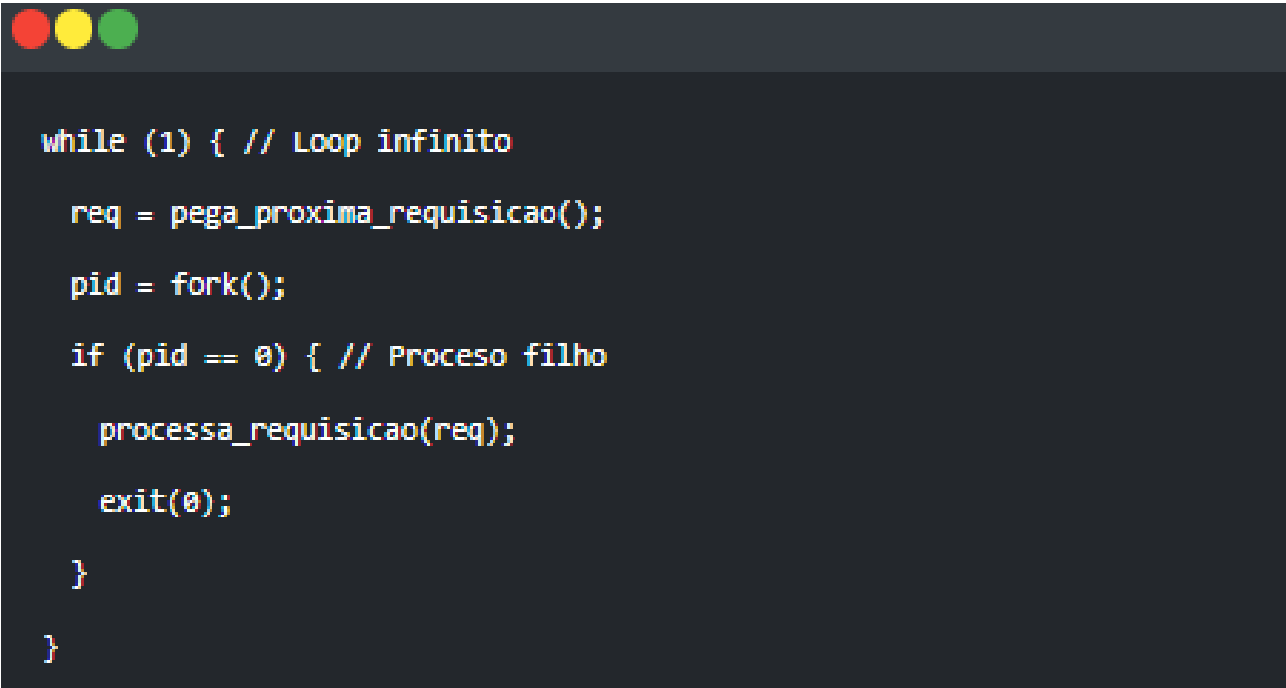
Vantagens da utilização: Dividem tarefas para execução concorrente, aproveitando o multiprocessamento.



Exemplo prático: Em um servidor web, subprocessos podem tratar requisições concorrentemente, melhorando o desempenho e evitando atrasos para tarefas simples.

## — Subprocessso

O trecho de código abaixo em Linguagem C exemplifica a parte de um servidor web simples que cria subprocessos para atendimento das requisições:



```
while (1) { // Loop infinito
    req = pega_proxima_requisicao();
    pid = fork();
    if (pid == 0) { // Processo filho
        processa_requisicao(req);
        exit(0);
    }
}
```

# — Threads



Threads são usadas para executar tarefas concorrentes em um mesmo processo, economizando recursos e tempo.

- Threads compartilham o espaço de endereçamento do processo, enquanto subprocessos têm espaços independentes.
- Mudar entre threads é mais eficiente do que entre processos.
- Úteis em sistemas multiprocessadores para paralelismo real e podem compartilhar recursos, mas cada thread tem sua própria pilha.

## — Threads

No Linux, os threads são criados com a chamada de sistema **clone()**. Sua sintaxe é:

```
int clone(int (*fn)(void *), void *stack, int flags, void *arg)
```

# — Threads

Os flags mais comuns são:

FLAG	COMPORTAMENTO	
	<i>Quando utilizado</i>	<i>Quando não utilizado</i>
CLONE_VM	Cria um thread.	Cria um processo.
CLONE_FS	Compartilha as informações sobre o sistema de arquivos.	Não compartilha informações sobre o sistema de arquivos.
CLONE_FILES	Compartilha os descritores de arquivos.	Copia os descritores de arquivos.
CLONE_SIGHAND	Compartilha a tabela do tratador de sinais.	Copia a tabela do tratador de sinais.
CLONE_PARENT	O novo thread tem o mesmo pai que o chamador.	O chamador é o pai do novo thread.
SIGCHLD	O thread envia o sinal SIGCHLD ao pai quando termina.	O thread não envia o sinal SIGCHLD ao pai quando termina.



# — Threads

## Tipos de processos

Existem basicamente dois tipos de processos relacionados ao tipo de processamento que executam:

### **CPU-bound**

Os processos do tipo CPU-bound passam a maior parte do tempo no estado executando, realizando poucas operações de E/S. Costumam ser encontrados em aplicações científicas.

### **I/O-bound**

Os processos do tipo I/O-bound passam a maior parte do tempo no estado bloqueado, por realizar elevado número de operações de E/S. Costumam ser encontrados em aplicações comerciais.

# — Processos e threads no Linux



```
{
  params = params || {};
  params.method = params.method || get;
  params.callback = params.callback || ();
  params.pre_processing = params.pre_processing;
  params.data = params.data || {};
  params.message = params.message || get;
  params.caching = params.caching || get;
  params.hidden = params.hidden || get;
  params.priority = params.priority || get;
  params.force = params.force || get;
  if (params.force) {
    van...
  }
  return;
}
```

- No Linux, processos e threads são tratados como tarefas (tasks) sem distinção clara.
- As chamadas `fork()` e `clone()` são usadas para criar processos e threads.
- Cada tarefa tem seu próprio contexto de execução e um PID único.
- O Linux compartilha memória entre tarefas sempre que possível, usando cópia na escrita para otimizar o desempenho e economizar memória física.

# — Processos de aplicações concorrentes



Aplicações concorrentes: Sistemas multiprogramáveis permitem a execução simultânea de partes do código, criando aplicações concorrentes.



Compartilhamento de recursos: Processos de aplicações concorrentes frequentemente compartilham recursos do sistema, gerando desafios.

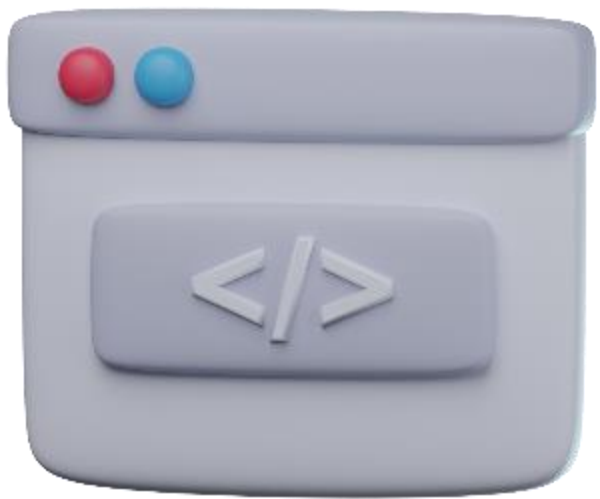


Questões a serem abordadas: Como passar informações entre processos, garantir a não interferência entre eles e sequenciar tarefas dependentes.



Mecanismos de sincronização: Para resolver essas questões, são necessários mecanismos de sincronização, se aplicam tanto a processos quanto a threads.

## — Condição de corrida




**Condição de corrida:** Ocorre quando vários processos acessam dados compartilhados, resultando em resultados incorretos devido à ordem de execução.

- **Exemplo:** Dois caixas de banco atualizam o saldo de uma conta ao mesmo tempo, causando uma inconsistência nos registros.
- **Causa:** A condição de corrida pode ocorrer quando as operações concorrentes não são devidamente sincronizadas, levando a problemas como esse.
- **Solução:** Mecanismos de sincronização, como semáforos e mutexes, são usados para evitar condições de corrida em sistemas concorrentes.

## — Condição de corrida

O trecho do programa que faz a atualização poderia ser:



```
void atualiza_saldo(double valor, int conta) {  
    Registro registro;  
    registro = le_registro(conta);  
    registro.saldo = registro.saldo + valor;  
    grava_registro(registro, conta);  
}
```

# — Condição de corrida

## Região crítica

Para evitar condições de corrida, siga as diretrizes abaixo:

### **Exclusão Mútua:**

Garanta que apenas um processo acesse um recurso compartilhado de cada vez, impedindo acesso simultâneo.

### **Região Crítica:**

Identifique a parte do programa que acessa a memória compartilhada, chamada de região crítica (RC).

### **Entrada e Saída da**

**Região Crítica:** Use mecanismos como semáforos ou mutexes para controlar o acesso à RC, permitindo que apenas um processo acesse por vez.

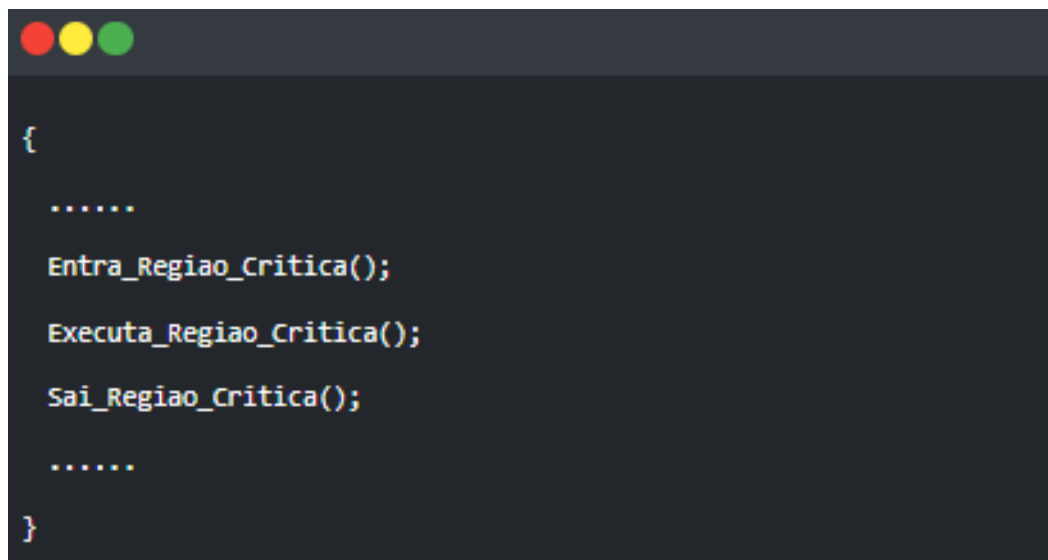
### **Boas Práticas:**

Certifique-se de que nenhum processo bloqueie indefinidamente (starvation), evitando interrupções dentro da RC.

# — Condição de corrida

## Região crítica

A parte do programa que acessa a memória compartilhada é denominada **seção crítica** ou **região crítica (RC)**.

A code snippet in a dark-themed editor window with red, yellow, and green window control buttons at the top left. The code is in C and shows a function body with a critical section. The critical section is enclosed in curly braces and contains three function calls: `Entra_Regiao_Critica();`, `Executa_Regiao_Critica();`, and `Sai_Regiao_Critica();`, separated by ellipses (`.....`) to indicate other code within the section.

```
{  
    .....  
    Entra_Regiao_Critica();  
    Executa_Regiao_Critica();  
    Sai_Regiao_Critica();  
    .....  
}
```

# — Condição de corrida

## Semáforos



**Semáforos:** Variáveis com operações "up" e "down" para controlar o acesso a recursos compartilhados.



**Down:** Decrementa o semáforo e bloqueia se o valor for zero, permitindo acesso à região crítica.



**Up:** Incrementa o semáforo e desbloqueia processos, permitindo que outros acessem o recurso.



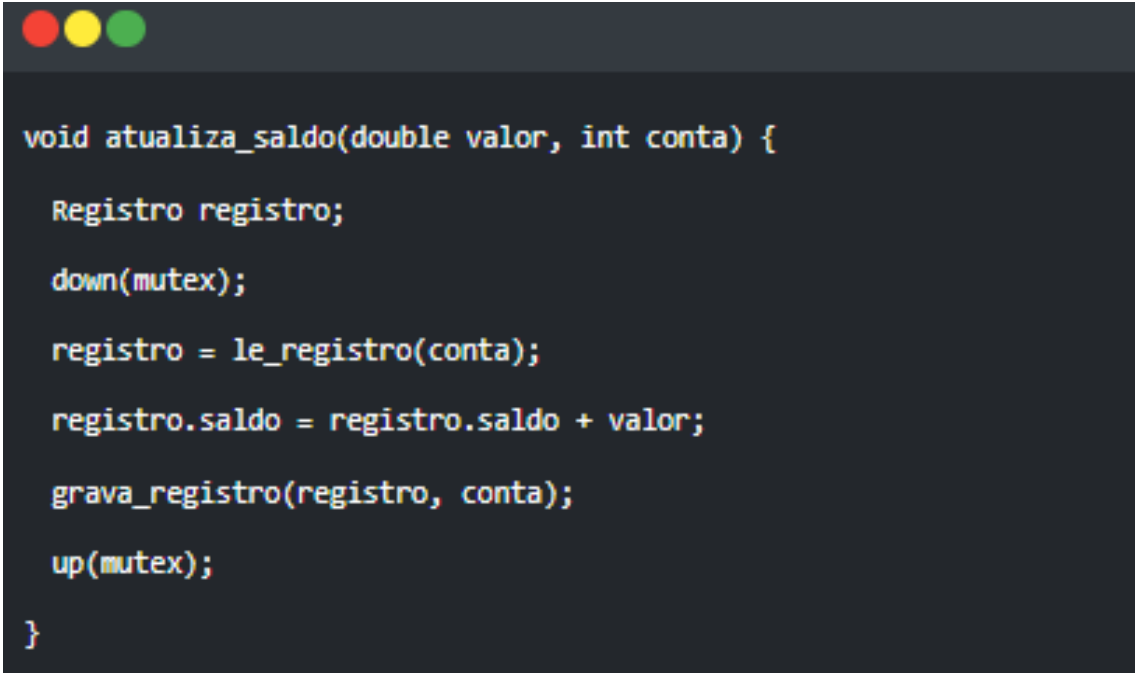
**Exclusão mútua:** Semáforos evitam que vários processos acessem a região crítica simultaneamente, garantindo operações atômicas.



# — Condição de corrida

## Semáforos

A correção para o caso anterior de atualização do saldo de uma conta, com a utilização de semáforos, ficaria:



```
void atualiza_saldo(double valor, int conta) {  
    Registro registro;  
    down(mutex);  
    registro = le_registro(conta);  
    registro.saldo = registro.saldo + valor;  
    grava_registro(registro, conta);  
    up(mutex);  
}
```

# — Monitores

Monitores:  
Mecanismos de sincronização de alto nível que simplificam o desenvolvimento de programas concorrentes.

Características:  
Monitores são coleções de variáveis e procedimentos, permitindo que apenas um processo acesse o monitor de cada vez.

Exclusão Mútua:  
Garantem exclusão mútua nas operações do monitor, tornando o desenvolvimento de programas concorrentes mais fácil.

Exemplo: Em Java, a cláusula "synchronized" é usada para implementar monitores, transformando regiões críticas em procedimentos do monitor.

# — Sincronização no Linux



Sincronização no Linux: O Linux oferece suporte a semáforos para sincronização de tarefas, com as chamadas `sem_wait()` e `sem_post()`.



`sem_trywait()`: Uma alternativa à `sem_wait()` que verifica a disponibilidade do semáforo sem bloqueio.



Comunicação entre processos: O Linux permite a comunicação entre processos usando pipes para troca de mensagens entre eles.



Envio de sinais: Sinais são usados para notificar eventos, como chegada de dados na rede ou término de processos, exigindo rotinas de tratamento.

## — Sincronização no Linux

Os sinais em um processo podem ser tratados das seguintes maneiras:

**Tratamento padrão pelo kernel.**

**Capturados e tratados por funções definidas pelo usuário.**

**Ignorados.**

# — Sincronização no Linux

Sinais podem ser gerados por:

**Exceções de hardware.**

**Condições de software.**

**Comando "kill" no shell.**

**Outro processo usando a chamada de sistema "kill()".**

**Combinação de teclas (por exemplo, Ctrl + C).**

**Controle de processos.**

# — Escalonamento



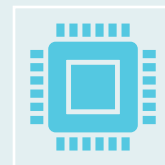
Escalonamento é o processo de seleção de qual processo será executado na UCP em sistemas multiprogramados.



Existem algoritmos de escalonamento preemptivos e não preemptivos, que afetam o tempo de espera dos processos na fila de prontos.



A preempção permite interrupções durante a execução de um processo, enquanto sistemas não preemptivos não permitem.



A troca de contexto causada pela preempção gera overhead, que deve ser gerenciado com cuidado no sistema.

# — Escalonamento

Tipos de escalonamento:

ESCALONAMENTO FIRST IN  
FIRST OUT (FIFO)

ESCALONAMENTO SHORTEST  
JOB FIRST (SJF)

ESCALONAMENTO SHORTEST  
REMAINING TIME NEXT  
(SRTN)

ESCALONAMENTO  
COOPERATIVO

ESCALONAMENTO CIRCULAR  
(ROUND ROBIN)

ESCALONAMENTO POR  
PRIORIDADE

ESCALONAMENTO POR  
MÚLTIPLAS FILAS

ESCALONAMENTO EM  
SISTEMAS DE TEMPO REAL

ESCALONAMENTO DE  
THREADS

## — Escalonamento no Linux

- Linux suporta multitarefa preemptiva com equilíbrio entre justiça e desempenho.
- Classes de tarefas incluem FIFO em tempo real, escalonamento circular e tempo compartilhado.
- FIFO em tempo real usa prioridades sem preempção; escalonamento circular utiliza preempção por quantum.
- Linux tem 140 níveis de prioridade, divididos entre tempo real e tempo compartilhado.



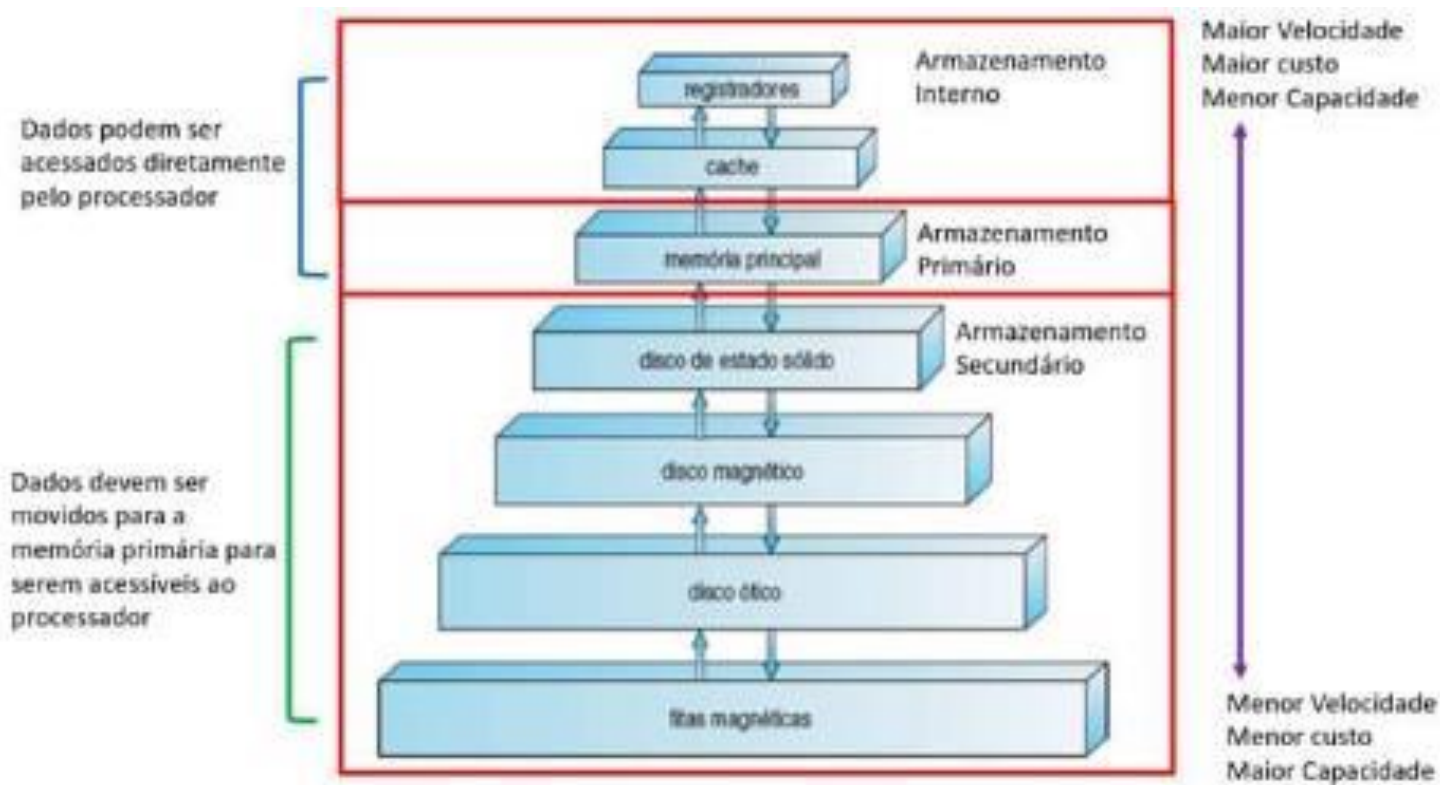


# Memória



# — Gerência de memória

A memória de um computador é composta por registradores e cache; memória principal RAM e armazenamento secundário; discos de estado sólido, magnéticos e óticos; e fitas magnéticas.



# — Gerência de memória

Abaixo os três grandes tipos de armazenamento:

## **Armazenamento Interno**

É constituído pela memória interna do processador e engloba os registradores e o cache.

## **Armazenamento Primário**

É constituído pela memória principal do computador, normalmente referenciada como memória RAM. Armazena o código dos programas e seus dados.

## **Armazenamento Secundário**

Também chamado de Armazenamento de Massa, é a memória que armazena os dados para uso posterior, já que é não volátil.

# — Gerência de memória

O sistema operacional é o responsável por realizar a gerência da memória. Entre suas funções estão:

**Alocar e desalocar os processos na memória.**

**Controlar os espaços disponíveis.**

**Impedir que um processo acesse o conteúdo da memória de outro processo.**

**Transferir dados entre o armazenamento primário e secundário e vice-versa.**

## — Como os processos enxergam a memória

Do ponto de vista processual a memória é vista como um conjunto de endereços lógicos que podem ser acessados diretamente por comandos da linguagem de máquina do processador. Modelo de memória de um processo:

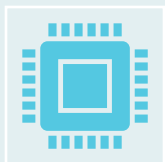
Text	Área da memória que armazena o código do programa.
Data	Área de memória onde residem as variáveis do programa.
Heap	Área de trabalho do programa onde são alocadas variáveis temporárias e variáveis dinâmicas.
Pilha	Área onde ficam os registros de ativação de procedimentos, variáveis locais etc.

## — Como os processos enxergam a memória

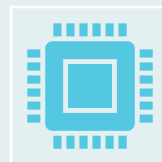
- MMU traduz endereços lógicos em endereços físicos.
- Endereços lógicos são vistos pelo processo, endereços físicos pelo hardware.
- MMU usa tabela de páginas para mapear endereços.
- MMU protege processos de acesso a memória de outros processos.



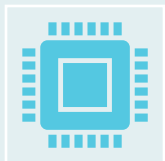
# — Proteção de memória



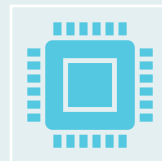
Proteção de memória é essencial para sistemas multiprogramados.



Cada processo deve ter um espaço de memória separado. A proteção é feita usando registradores base e limite.

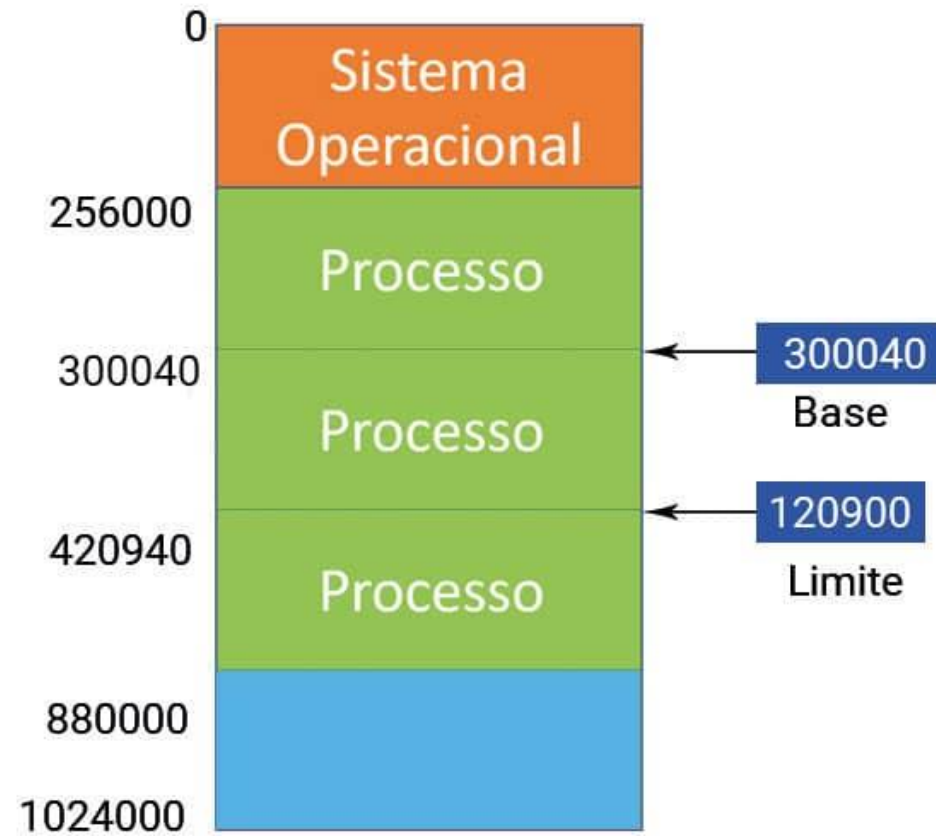


A MMU verifica se o endereço de memória solicitado está dentro do espaço do processo.



O sistema operacional, em modo kernel, pode acessar qualquer endereço de memória.

## — Proteção de memória



Registradores base e limite.



# — Relocação



Relocação é o processo de mapeamento de endereços lógicos em endereços físicos.



A relocação pode ser realizada pelo link-editor ou pelo carregador.



Relocação estática é feita antes do carregamento do módulo.

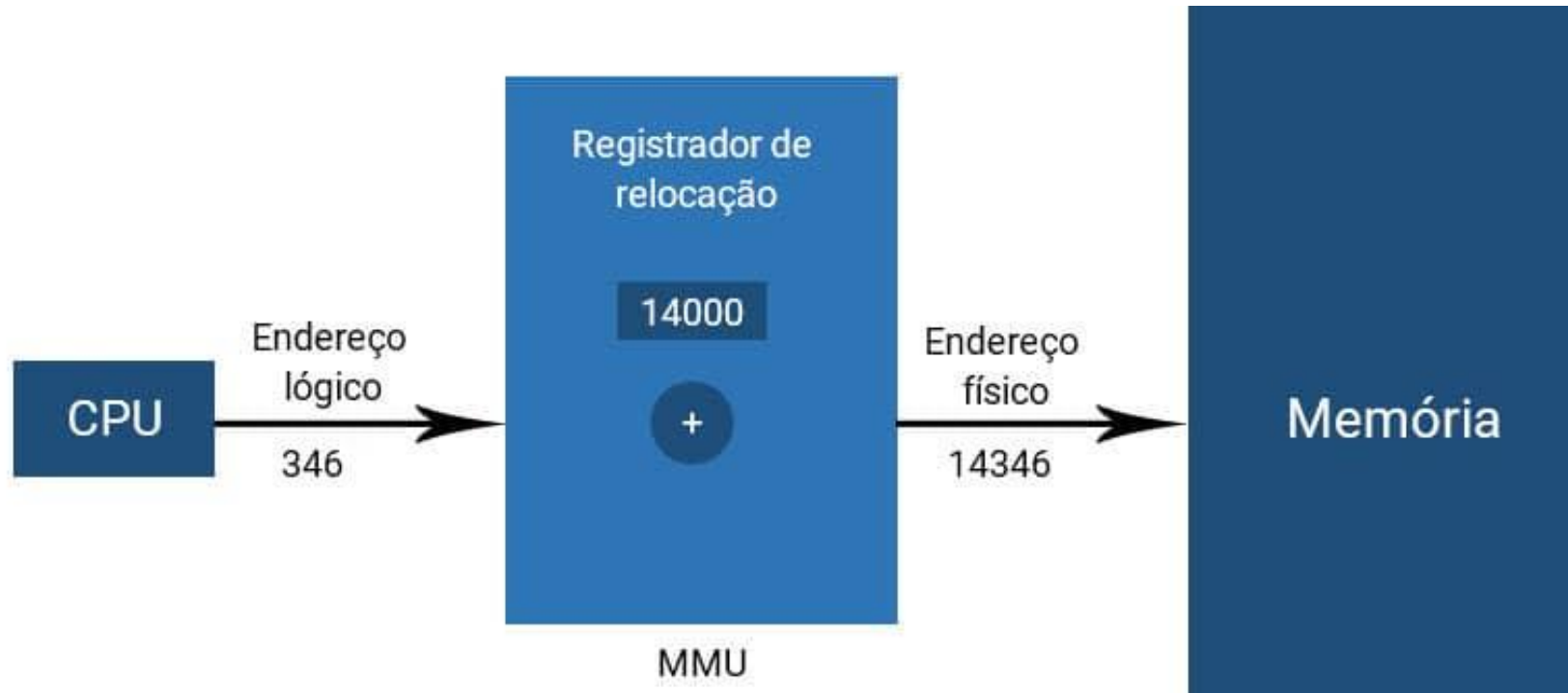


Relocação dinâmica é feita durante a execução do processo.



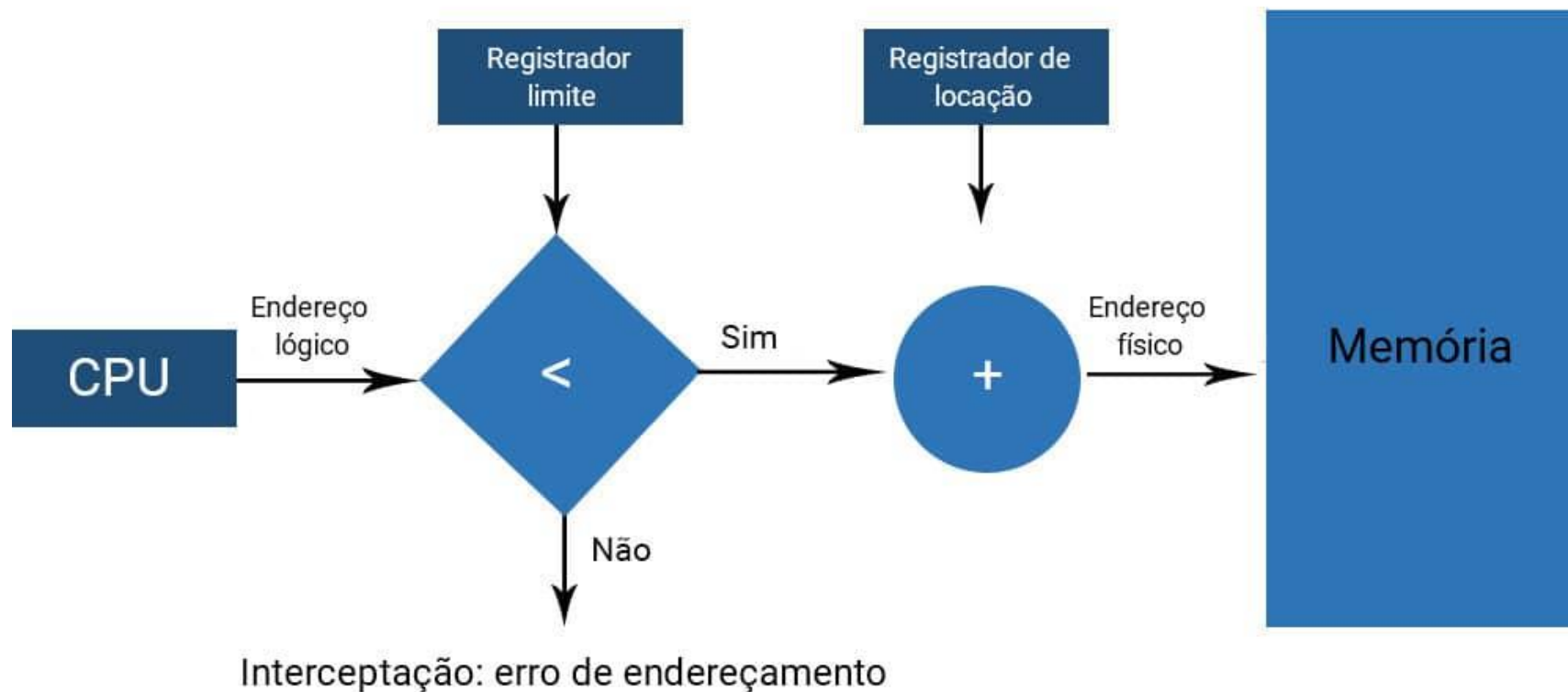
Relocação é necessária para que um processo possa ser alocado em qualquer posição da memória.

## — Relocação



Relocação dinâmica.

## — Relocação



Proteção de memória com relocação dinâmica

# — Políticas de alocação

As políticas de alocação de memória compreendem dois tipos básicos:

Manter os processos na memória principal durante toda a sua execução.

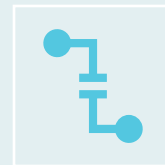
Mover os processos entre a memória principal e a secundária (tipicamente disco), utilizando técnicas de swapping (Permuta) ou de paginação.

# — Gerenciamento de memória sem permuta

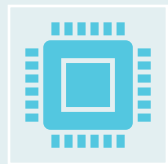
## Alocação contígua



Alocação contígua é um esquema de alocação de memória em que cada processo ocupa um espaço de memória contínuo.



O sistema operacional e os processos de usuário compartilham a memória principal.



A alocação contígua é simples e eficiente, mas limita o tamanho dos processos ao tamanho da memória disponível.



Para superar essa limitação, foi desenvolvida a técnica de overlay.

# — Gerenciamento de memória sem permuta

## Overlay

- Overlay é uma técnica de gerenciamento de memória que permite que processos maiores sejam executados em memórias menores, dividindo o código do processo em módulos independentes.
- O módulo principal é sempre residente na memória, enquanto os módulos secundários são carregados e descarregados conforme necessário.

# — Gerenciamento de memória sem permuta

## Alocação particionada fixa



Alocação particionada fixa é uma técnica de gerenciamento de memória que divide a memória em partições fixas de tamanhos diferentes.



Os processos são alocados em partições que sejam do tamanho adequado.



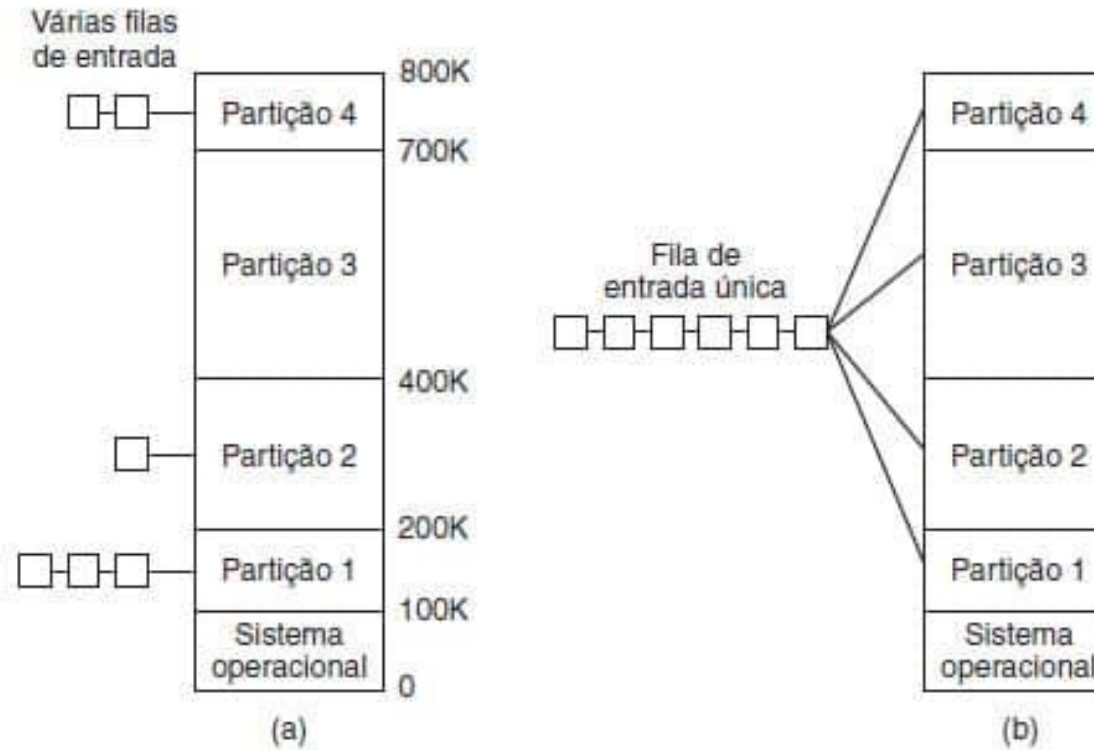
Duas estratégias de alocação podem ser usadas: uma fila por partição ou uma única fila de entrada.



Este método pode gerar desperdício de memória, pois o espaço não utilizado em uma partição não pode ser alocado para outro processo.

# — Gerenciamento de memória sem permuta

## Alocação particionada fixa



Filas de processo e partições fixas

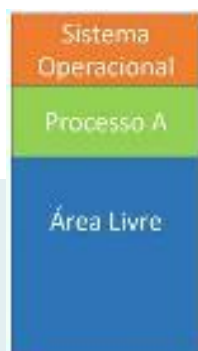


# — Gerenciamento de memória sem permuta

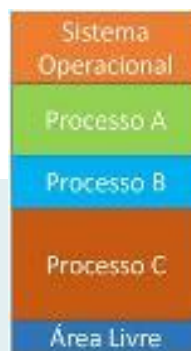
## Alocação com partições variáveis



A memória de usuário é inicialmente considerada uma única partição livre.



Ao iniciar A, ele é carregado na memória e o seu tamanho define o tamanho da sua partição



Os processos B e C, de forma analógica, são carregados no espaço livre ainda disponível.



Ao chegar ao processo d, não existe espaço livre contínuo suficiente para que seja carregado.



Após alocar e desalocar vários processos, a memória pode se fragmentar.

# — Gerenciamento de memória sem permuta

## Alocação com partições variáveis

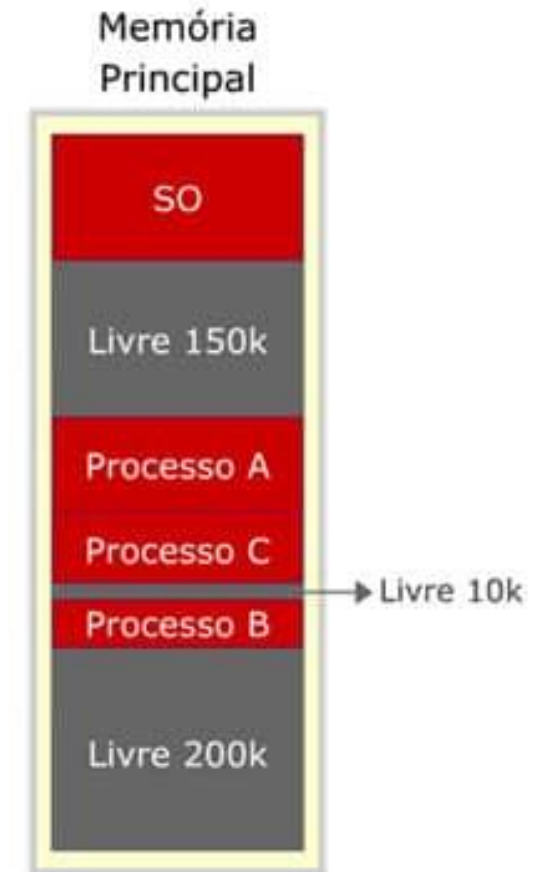
- Para alocar processo em partição fixa, existem diversas políticas.
- **Exemplo:** processo C de 70k em partições de 100, 50, 20 e 10k.
- Políticas: melhor ajuste, primeiro ajuste e pior ajuste.



# — Gerenciamento de memória sem permuta

## Política Best-Fit

- Procura alocar processo na partição disponível de tamanho mais próximo ao do processo.
- Gera fragmentos livres menores, mas é menos eficiente que o primeiro ajuste.



# — Gerenciamento de memória sem permuta

## Política First-Fit

- Procura alocar processo na primeira partição disponível.
- Gera fragmentos livres maiores, mas é mais eficiente que o best-fit.



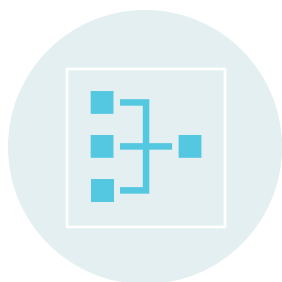
# — Gerenciamento de memória sem permuta

## Política Worst-Fit

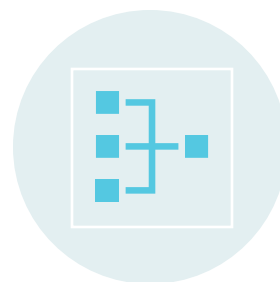
- Procura alocar processo na maior partição disponível.
- Gera fragmentos livres maiores, mas é menos eficiente que o first-fit.



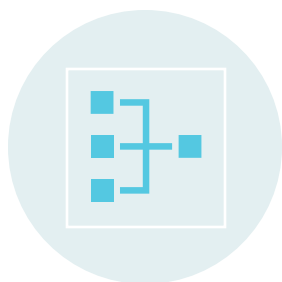
# — Fragmentação externa e interna



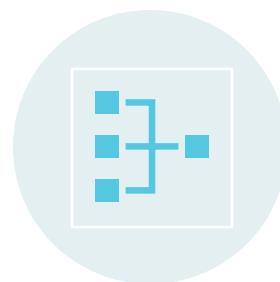
**Fragmentação externa:** ocorre quando existem espaços livres na memória, mas eles não são contíguos.



**Fragmentação interna:** ocorre quando um processo ocupa mais espaço do que realmente precisa.



**Solução para fragmentação externa:** compactação da memória ou alocação de espaço não contíguo.



**Solução para fragmentação interna:** alocação de espaço maior do que o necessário.

# — Gerenciamento de memória com permuta

## Swap de memória



Swap de memória: processo é retirado da memória principal e transferido para a memória secundária, e vice-versa.



Vantagens: maior compartilhamento da memória, menor fragmentação de memória, boa técnica para ambientes com processos pequenos e poucos usuários.



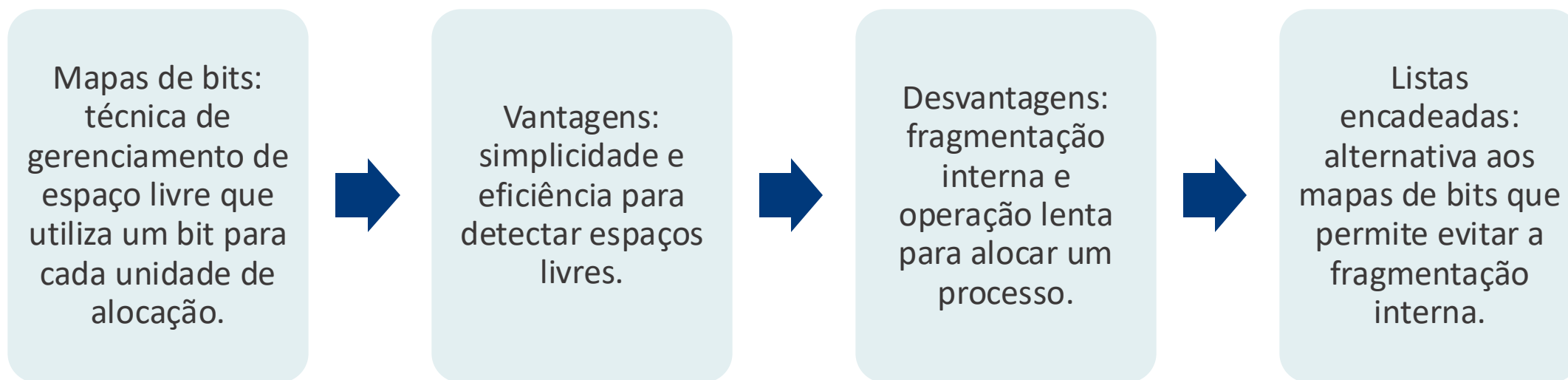
Desvantagem: tempo gasto no swap in e no swap out.



Não é utilizada em sistemas modernos: devido ao tempo gasto no swap in e no swap out, as técnicas de paginação e segmentação são mais eficientes.

## — Gerenciamento de espaço livre

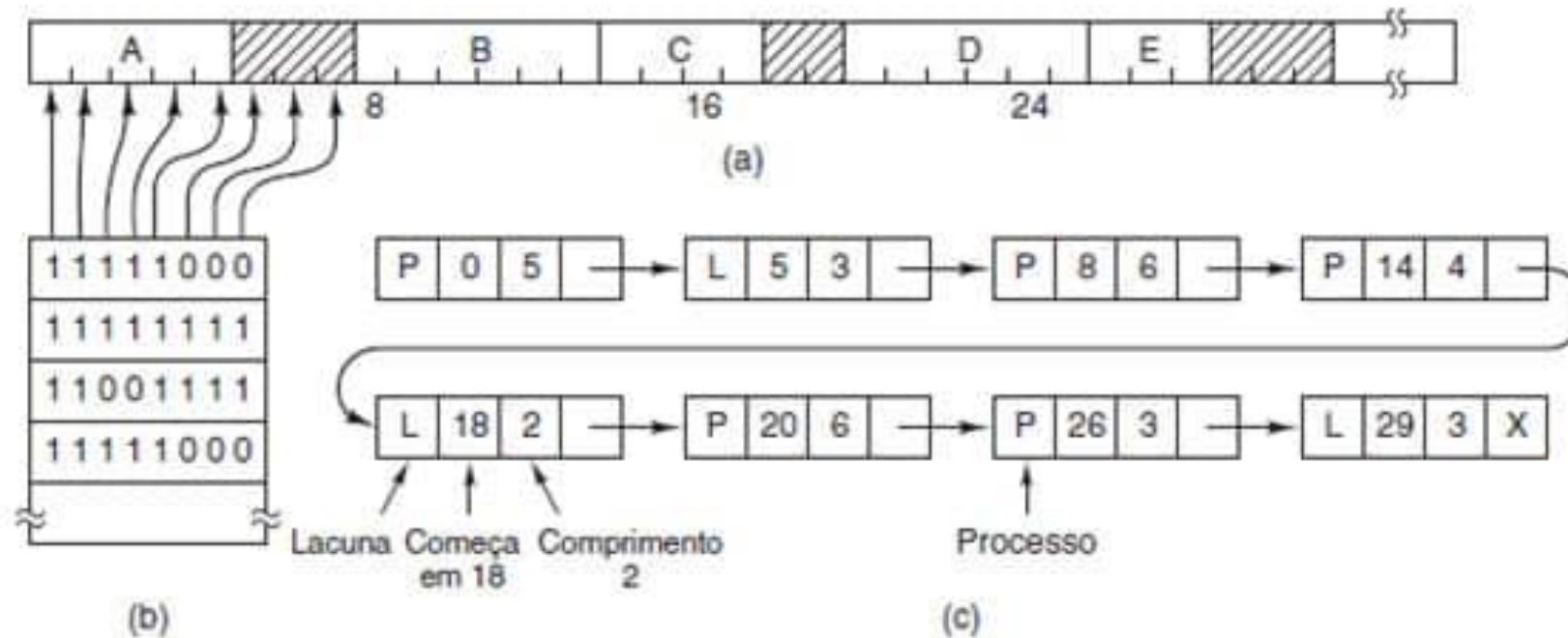
### Gerenciamento de memória com mapas de bits





# — Gerenciamento de espaço livre

## Gerenciamento de memória com mapas de bits

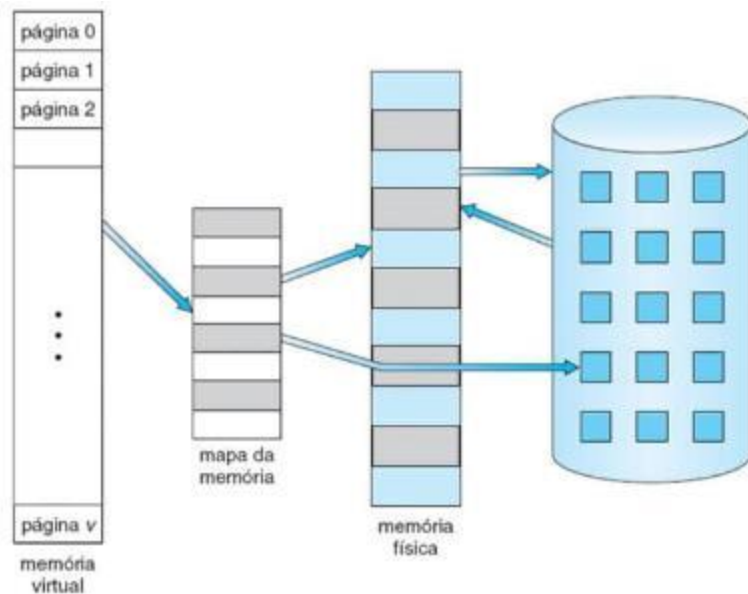


## — Gerenciamento de espaço livre

### Gerenciamento de memória com listas encadeadas

- Listas encadeadas: técnica de gerenciamento de espaço livre que utiliza uma lista de nós, cada um representando um segmento de memória alocado ou livre.
- Vantagens: evita fragmentação interna e operações rápidas para alocar e liberar processos.
- Desvantagens: complexidade de implementação e pode gerar desperdício de memória.

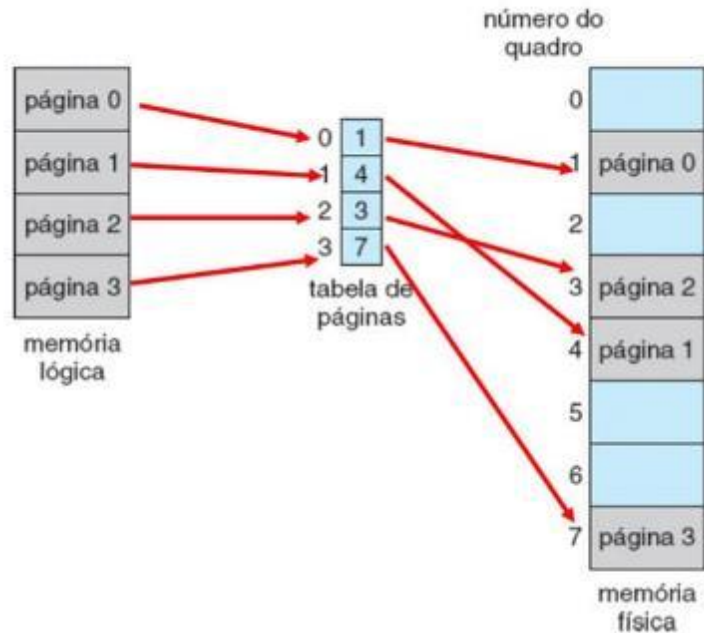
# — Memória virtual



Memória virtual: espaço de endereçamento lógico do processo é maior que a memória física disponível.

- Paginação: técnica de divisão da memória virtual em páginas, que são mapeadas para a memória física.
- Vantagens: permite que processos maiores que a memória física sejam executados e aumenta o grau de multiprogramação.
- Desvantagens: pode gerar fragmentação externa e overhead de paginação.

# — Paginação



Paginação: memória virtual é dividida em páginas de tamanho fixo, que são mapeadas para a memória física.

- **Vantagens:** permite que processos maiores que a memória física sejam executados, aumenta o grau de multiprogramação e reduz a fragmentação externa.
- **Desvantagens:** pode gerar fragmentação interna e overhead de paginação.

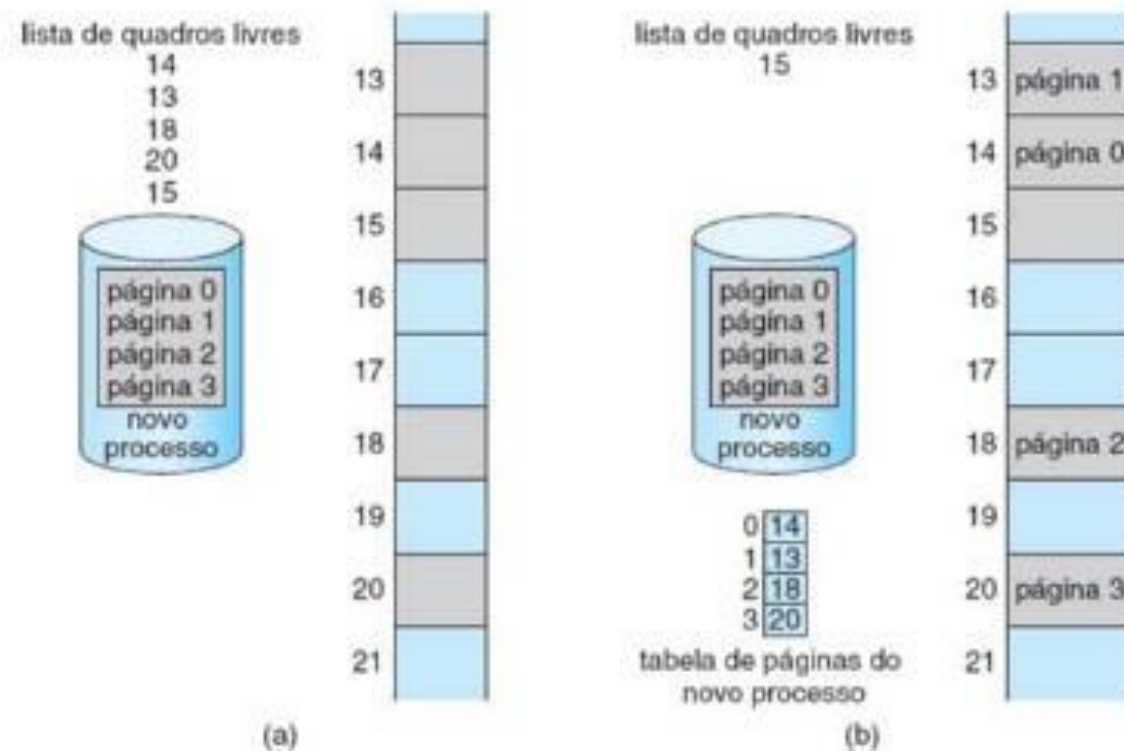
# — Paginação

## Controle de espaço livre

### Alocação de processo:

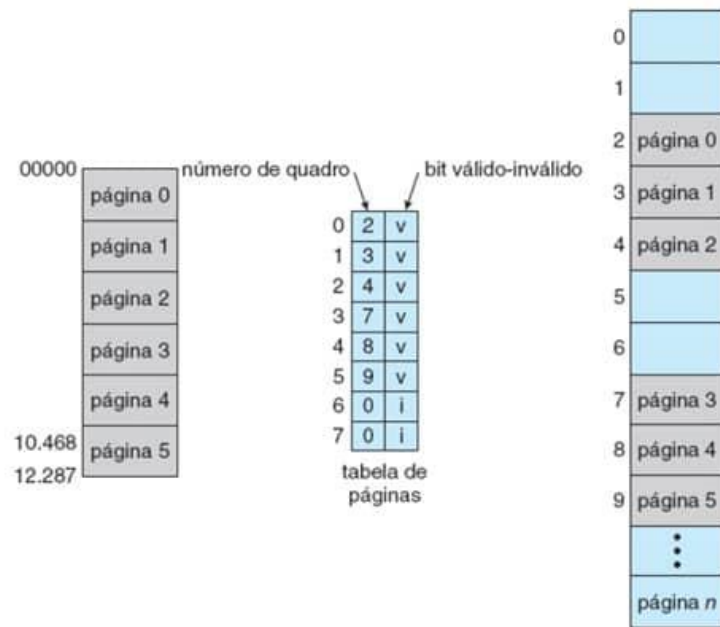
- O SO busca quadros livres suficientes para o processo.
- O processo é alocado em quadros livres não contíguos.
- A tabela de páginas é atualizada.

**Desvantagem:** pode gerar fragmentação interna.



# — Paginação

## Proteção de memória



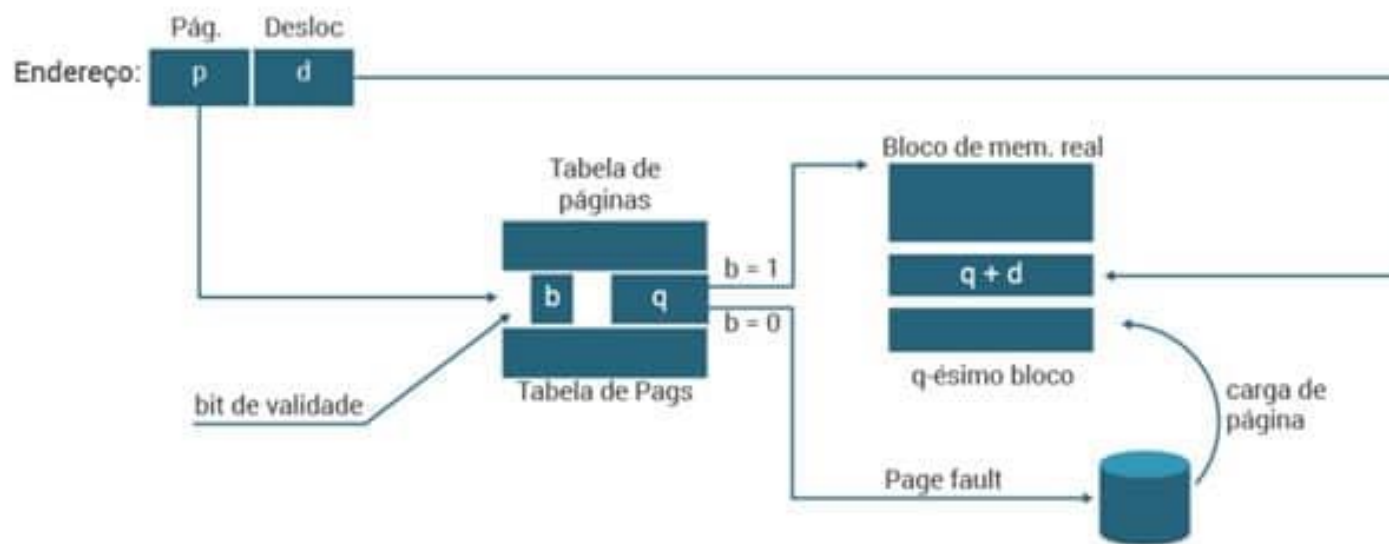
- Bit de permissão: define se a página é apenas de leitura ou se permite também a gravação.
- Bit válido-inválido: define se a página está no espaço de endereçamento lógico do processo.

Proteção de memória: técnica de impedir que um processo acesse memória a que não tem permissão.

# — Paginação

## Paginação sob demanda

Carrega na memória somente páginas referenciadas. Vantagens: evita desperdício e melhora desempenho. Desvantagens: overhead de paginação.



# — Paginação

## Políticas de paginação

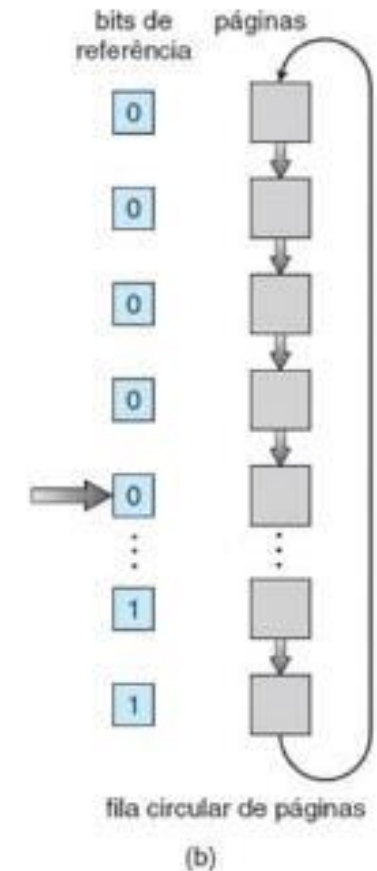
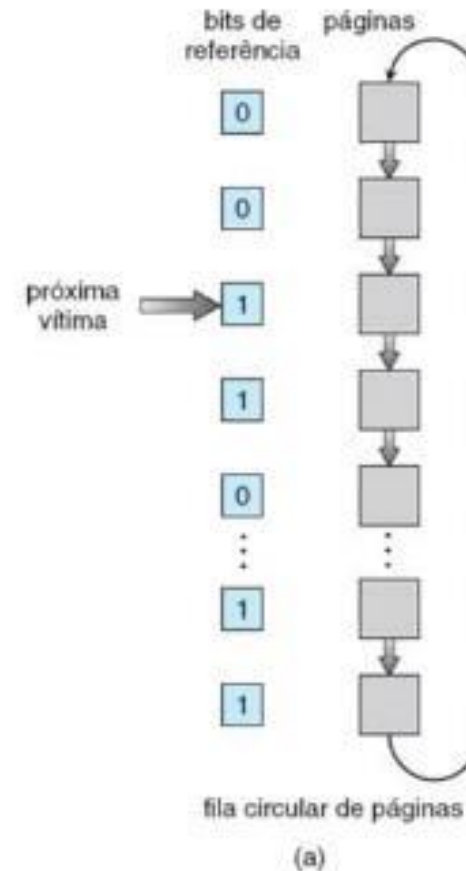
- Aleatória: escolhe-se qualquer página.
- FIFO: retira-se a página carregada há mais tempo.
- LRU: retira-se a página utilizada pela última vez há mais tempo.
- Segunda chance: variante da LRU que evita o overhead de atualização do momento em que a página foi acessada.



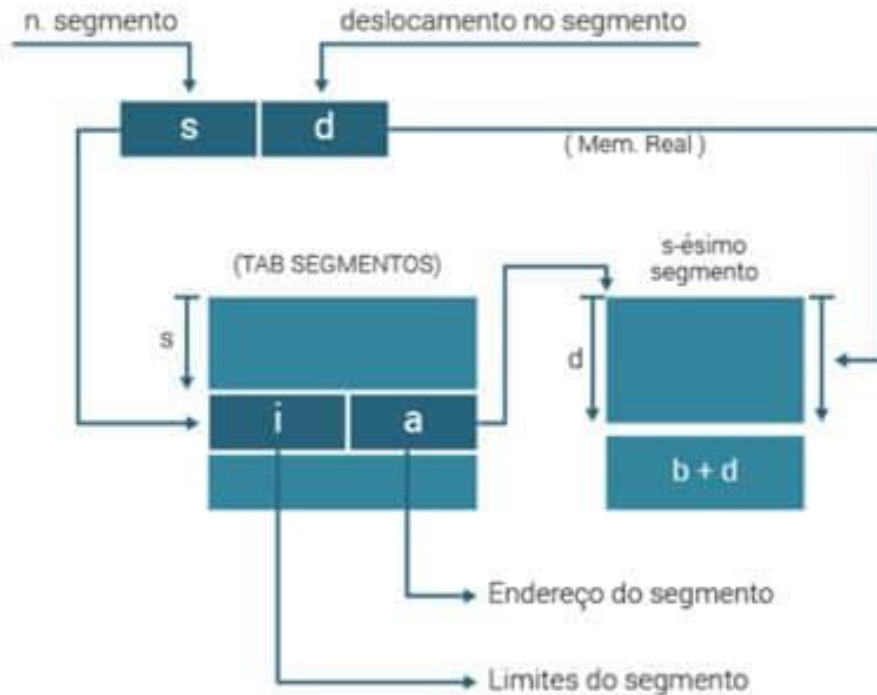
# — Paginação

## Algoritmo de segunda chance

- Política de substituição: FIFO com segunda chance
- Vantagem: evita que páginas sempre referenciadas sejam substituídas
- Desvantagem: na pior das hipóteses, degenera em FIFO



# — Segmentação



- Divisão do espaço de endereçamento em segmentos de tamanho variável.
- Cada segmento agrupa partes do programa que se referenciam mutuamente.
- Mapeamento de endereços realizado por uma tabela de segmentos.
- Diferença da paginação: divisão lógica x divisão física.

---

A gerência de memória no Linux envolve dois aspectos distintos:

- **A alocação e liberação da memória física** – páginas, grupos de páginas e pequenos blocos de RAM.
- **A manipulação da memória virtual** – mapeamento da memória física para o espaço de endereçamento de processos em execução.



# — Gerenciamento da memória física no Linux

Divisão da memória física em quatro zonas:

ZONE\_DMA: para dispositivos ISA

ZONE\_DMA32: para dispositivos que podem acessar os primeiros 4GB da memória em operações de DMA

ZONE\_HIGHMEM: memória alta que não está mapeada para o kernel

ZONE\_NORMAL: todo o resto da memória

# — Memória virtual no Linux

O mecanismo de memória virtual é o responsável por manter o espaço de endereçamento disponível para cada processo. Para fazer o gerenciamento são mantidos, para cada processo, duas visões diferentes:

## **Visão lógica**

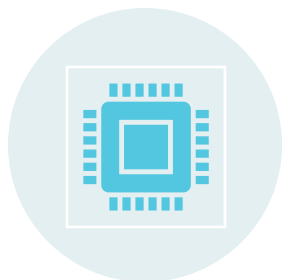
Descreve as instruções do sistema de memória virtual relacionadas com o layout do espaço de endereçamento.

## **Visão física**

Corresponde a entradas nas tabelas de páginas de hardware do processo.

# — Memória virtual no Linux

## Regiões de memória virtual



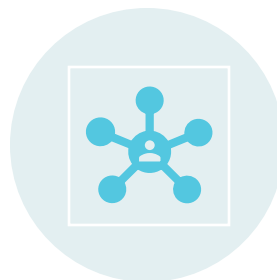
O Linux usa várias regiões de memória virtual.



Cada região tem um arquivo associado, funcionando como porta de acesso a uma parte do arquivo de paginação.



As referências a páginas preenchem a tabela de páginas com endereços do cache de páginas do kernel.



Regiões podem ser mapeadas de forma compartilhada ou privada em relação às gravações.

## — Memória virtual no Linux

### Tempo de vida de um espaço de endereçamento virtual

Duas situações geram a criação de um novo espaço de endereçamento virtual:

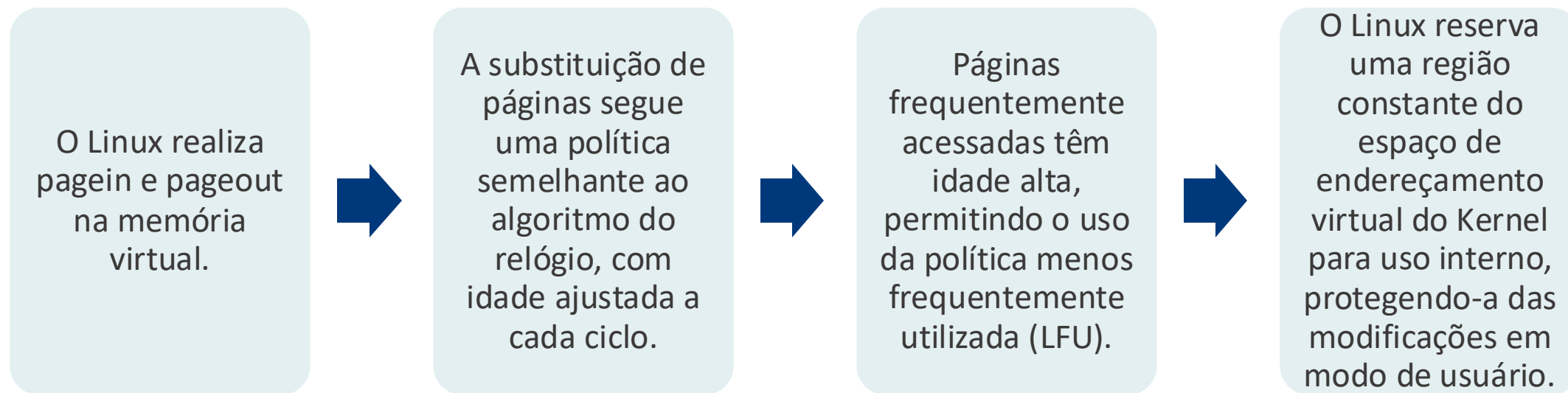
Quando um processo executa  
um novo programa com a  
chamada de sistema `exec( )`



Quando um novo processo é  
criado pela chamada de  
sistema `fork( )`

# — Memória virtual no Linux

## Permuta e paginação





# — Execução e carga de programas de usuário

## Mapeamento de programas para a memória



A execução de programas de usuário no Linux é feita com a chamada de sistema `exec()`, substituindo o contexto atual.



O carregador binário mapeia páginas do arquivo binário na memória virtual conforme necessário.



O carregador define o mapeamento inicial da memória, incluindo pilha, texto, dados e heap.



O processo é inicializado com o ponto de entrada do programa ELF e pode ser incluído no escalonador.

# — Execução e carga de programas de usuário

## Vinculação estática e dinâmica



Vinculação estática envolve a incorporação direta de código de bibliotecas no executável, permitindo a execução imediata.



Vinculação dinâmica carrega bibliotecas do sistema na memória apenas uma vez, economizando espaço.



Programas vinculados dinamicamente têm uma função estática que mapeia bibliotecas e executa o código.



Bibliotecas dinâmicas resolvem referências a símbolos e podem ser mapeadas em qualquer lugar da memória.

# — Utilitários e comandos para gerenciar a memória do sistema Linux

## Obtendo informações pela linha de comando

O comando `free` fornece informações detalhadas sobre o uso da memória, incluindo a quantidade de RAM e swap disponíveis.

O comando `top` mostra informações sobre os processos em execução, incluindo o uso de memória.

O comando `vmstat` exibe a situação da memória virtual do sistema.

`getconf  
PAGESIZE`  
mostra o tamanho da página no sistema de memória virtual (geralmente 4KB).

`swapon` é utilizado para ativar dispositivos de swap no sistema.

# — Utilitários e comandos para gerenciar a memória do sistema Linux

## Utilitários para acesso à informação e Htop

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
671	ventury	20	0	366M	176M	87260	S	8.9	4.5	5:01.12	/usr/lib/
988	ventury	20	0	3735M	478M	154M	S	6.2	12.2	5:07.04	/usr/bin/
52002	ventury	20	0	766M	129M	99M	S	2.7	3.3	0:13.99	ksysguard
20304	ventury	20	0	809M	64812	47384	S	1.4	1.6	1:07.22	gnome-sys
50861	ventury	20	0	947M	56864	41124	S	1.4	1.4	0:05.10	/usr/libe
51979	ventury	20	0	10596	4000	3336	R	0.0	0.1	0:02.35	htop
761	ventury	20	0	366M	176M	87260	S	0.0	4.5	0:06.98	/usr/lib/
52008	ventury	20	0	4368	2496	2272	S	0.0	0.1	0:00.35	/usr/bin/
1375	ventury	20	0	486M	36092	26288	S	0.0	0.9	0:00.19	update-no
926	ventury	20	0	159M	6384	5736	S	0.0	0.2	0:00.18	/usr/libe
891	ventury	20	0	380M	9228	7076	S	0.0	0.2	0:02.13	/usr/bin/
898	ventury	20	0	372M	60856	39696	S	0.0	1.5	0:01.27	/usr/libe
1004	ventury	20	0	3735M	478M	154M	S	0.0	12.2	0:00.53	/usr/bin/

O utilitário **htop** é uma evolução do comando **top** e exibe informações do sistema de forma interativa.

- Ele fornece informações detalhadas sobre processos, CPU, memória e outros recursos.
- O htop é uma ferramenta de linha de comando que permite monitorar o sistema de maneira eficiente.
- Suas informações são apresentadas de forma clara e colorida para facilitar a análise.

# — Utilitários e comandos para gerenciar a memória do sistema Linux

## Utilitários para acesso à informação e Htop

### CPU

Verde: Threads rodando com prioridade normal.

Azul: Threads rodando com baixa prioridade.

Vermelho: Threads rodando em favor do kernel.

### MEMÓRIA

Verde: Memória em uso pelas aplicações.

Azul: Buffers em utilização.

Amarelo / Laranja: Cache.

### SWAP

Vermelha: Representa a quantidade de memória swap utilizada

# — Utilitários e comandos para gerenciar a memória do sistema Linux

## Utilitários para acesso à informação e Htop

Essas informações podem ser obtidas ainda apertando F1



```
htop 2.2.0 - (C) 2004-2019 Hisham Muhammad
Released under the GNU GPL. See 'man' page for more info.

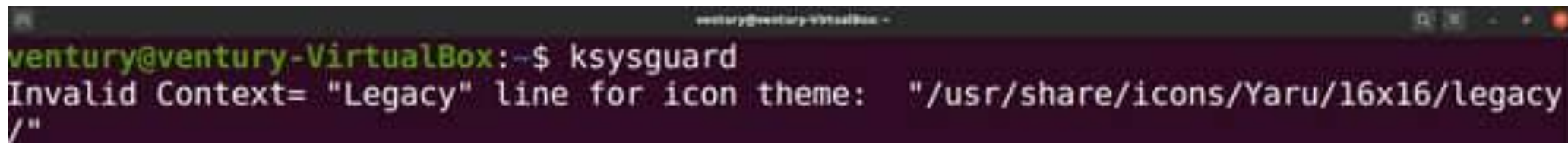
CPU usage bar: [low-priority/normal/kernel/virtualiz      used%]
Memory bar:    [used/buffers/cache                        used/total]
Swap bar:      [used                                      used/total]
Type and layout of header meters are configurable in the setup screen.

Status: R: running; S: sleeping; T: traced/stopped; Z: zombie; D: disk sl
Arrows: scroll process list           Space: tag process
Digits: incremental PID search       c: tag process and its child
F3 /: incremental name search        U: untag all processes
F4 \: incremental name filtering     F9 k: kill process/tagged proce
F5 t: tree view                     F7 ]: higher priority (root onl
p: toggle program path              F8 [: lower priority (+ nice)
u: show processes of a single user  a: set CPU affinity
H: hide/show user process threads  e: show process environment
K: hide/show kernel threads        i: set IO priority
```

# — Utilitários e comandos para gerenciar a memória do sistema Linux

## Ksysguard

O utilitário ksysguard é outro que pode ser chamado na linha de comando e que não vem instalado por padrão.

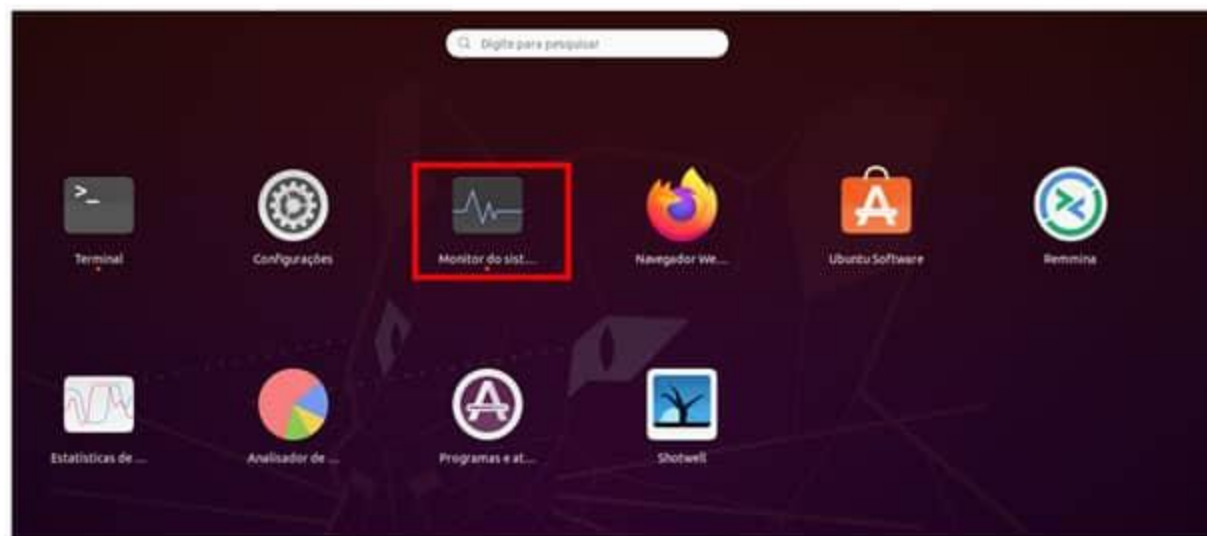
A screenshot of a terminal window with a dark purple background. The window title is 'ventury@ventury-VirtualBox -'. The prompt is 'ventury@ventury-VirtualBox:~\$'. The command 'ksysguard' has been entered. The output is an error message: 'Invalid Context= "Legacy" line for icon theme: "/usr/share/icons/Yaru/16x16/legacy /"'.

```
ventury@ventury-VirtualBox:~$ ksysguard
Invalid Context= "Legacy" line for icon theme:  "/usr/share/icons/Yaru/16x16/legacy
/"
```

# — Utilitários e comandos para gerenciar a memória do sistema Linux

## Monitor do Sistema

Este aplicativo vem instalado por padrão e pode ser acessado na aba Ferramentas.





# Sistema de Arquivos



# — Conceitos

As principais exigências para armazenamento de informações são:



Deve ser possível armazenar uma grande quantidade de informações.



A informação deve sobreviver à finalização do processo que a utiliza (deve ser persistente).



Múltiplos processos devem ser capazes de acessar as informações concorrentemente.

## — Conceitos

O sistema operacional utiliza arquivos em discos e mídias para organizar informações.

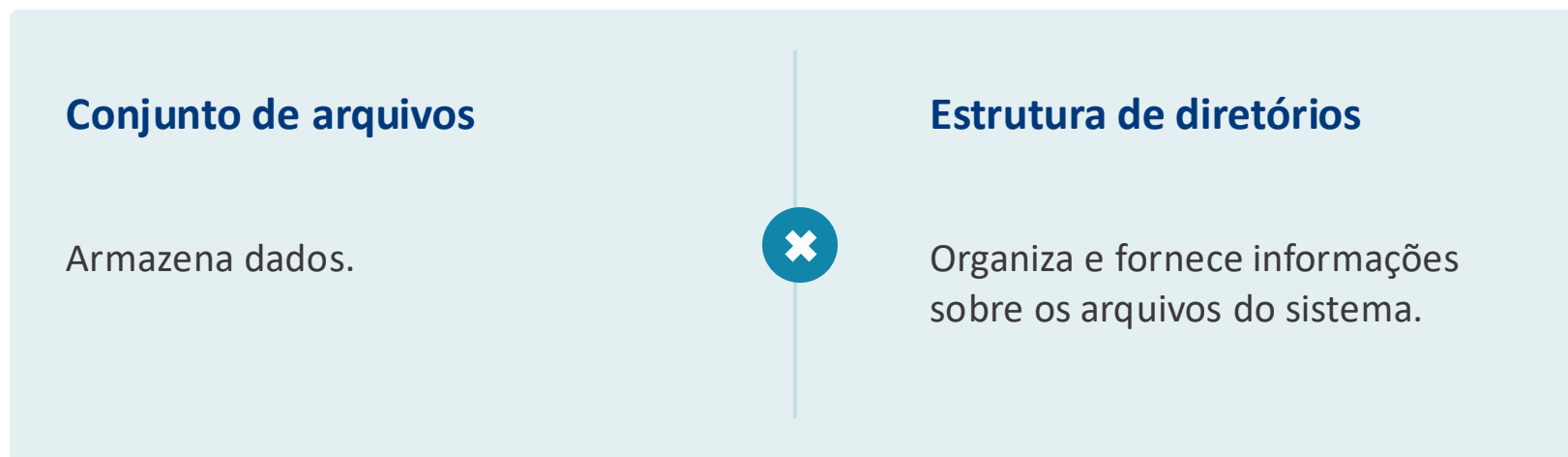
O sistema de arquivos gerencia a estrutura e manipulação de arquivos de forma uniforme.

Em sistemas complexos, é necessário lidar com questões como localização, permissões e espaço livre.

O sistema de arquivos consiste em conjuntos de arquivos que armazenam dados e uma estrutura de diretórios para organizá-los.

## — Conceitos

O sistema de arquivos é constituído de duas partes distintas:



# — Arquivos



Um sistema operacional deve oferecer uma visão uniforme de dispositivos de armazenamento para facilitar o acesso a dados.



Arquivos contêm informações e podem representar programas ou dados.



Um arquivo é identificado por um nome, e as regras para nomes variam de sistema para sistema.



Algumas regras incluem a quantidade máxima de caracteres, diferenciação entre maiúsculas e minúsculas e uso de caracteres especiais. As extensões podem ser usadas para identificar o conteúdo.

# — Arquivos

## Estrutura de Arquivos

No momento da criação de um arquivo, é possível definir qual organização será adotada. Esta organização pode ser uma estrutura suportada pelo sistema operacional ou definida pela própria aplicação.

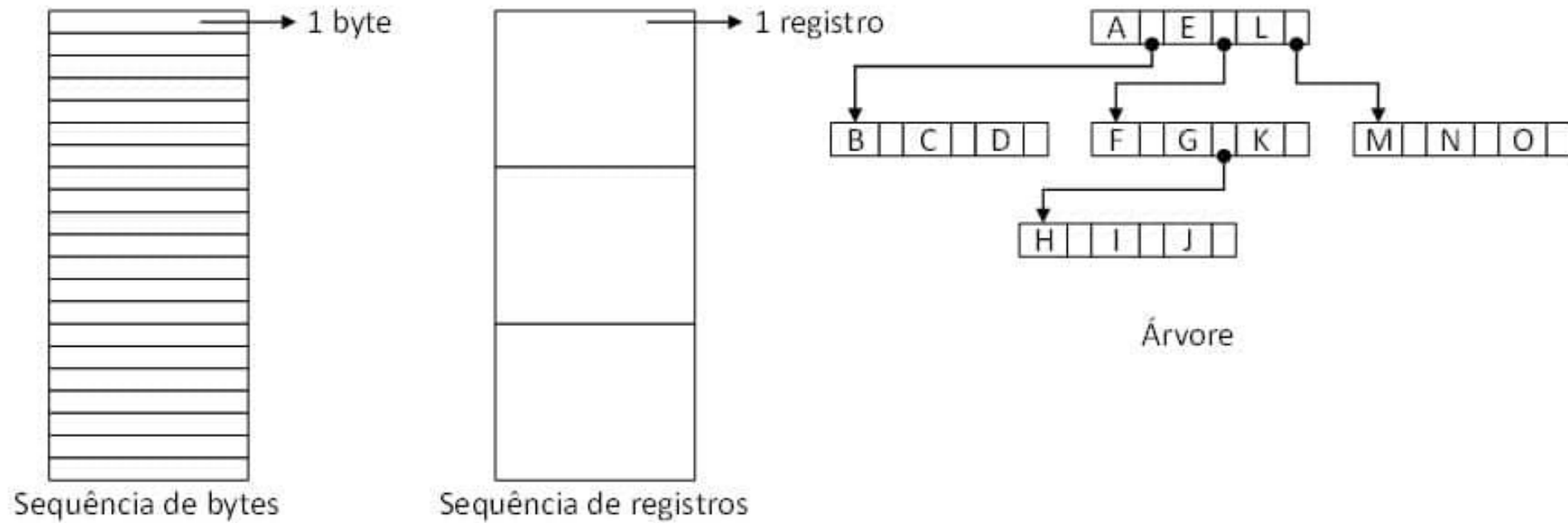
**Sequência desestruturada  
de bytes**

**Sequência de registros de  
tamanho fixo**

**Árvore de registros**

# — Arquivos

## Estrutura de Arquivos



# — Arquivos

## Métodos de Acesso

- **Acesso sequencial:** leitura/gravação na ordem em que os registros foram gravados.
- **Acesso aleatório:** leitura/gravação em qualquer posição do arquivo.
- **Acesso indexado:** leitura/gravação usando um índice para localizar a posição desejada.



# — Arquivos

## Tipos de Arquivos

**Arquivos regulares**

**Diretórios**

**Arquivos especiais de  
caractere**

**Arquivos especiais de  
bloco**

# — Arquivos

## Tipos de Arquivos

Em geral, arquivos regulares são classificados:

### Arquivo texto

Um arquivo texto (ou arquivo ASCII) é constituído por linhas de texto que podem ter tamanhos diferentes e terminam por caracteres especiais para indicar o fim da linha.

≠

### Arquivo binário

Arquivos binários não são arquivos texto. Sua listagem gera um conjunto de caracteres incompreensíveis.

# — Diretórios



Diretórios organizam arquivos em discos, com informações como nome e localização.



Ao abrir um arquivo, o sistema operacional busca suas informações no diretório.



Estruturas de diretórios incluem nível único, dois níveis e hierárquico em árvore.



Caminhos de arquivo podem ser absolutos (desde o raiz) ou relativos (baseados no diretório atual).

## — Implementação do sistema de arquivos



Arquivos em discos são divididos em blocos, compostos por setores de tamanhos variáveis (geralmente 512 bytes).



Discos podem ter múltiplas partições com sistemas de arquivos independentes, controladas por MBR e tabelas de partição.



Cada partição inicia com um bloco de inicialização e contém estruturas como superbloco, i-nodes e diretório raiz.

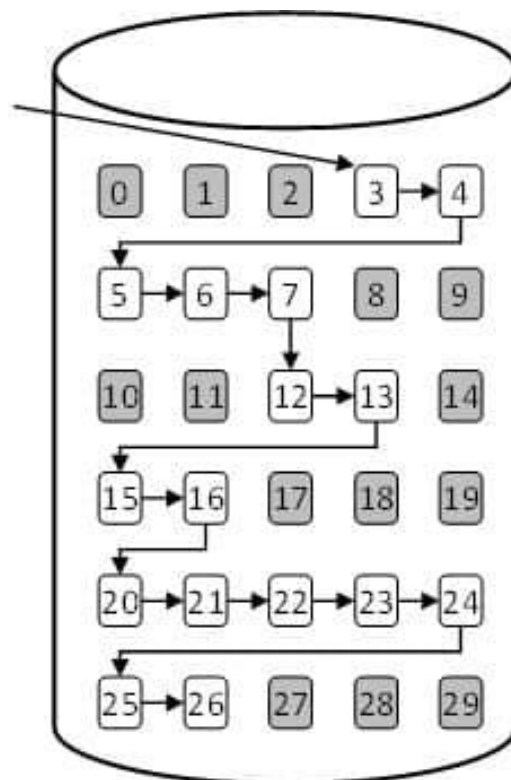


Para gerenciar blocos livres, os sistemas de arquivos usam mapas de bits, listas encadeadas ou tabelas de blocos livres segmentados.

## — Implementação do sistema de arquivos

1 1 1 0 0
0 0 0 1 1
1 1 0 0 1
0 0 1 1 1
0 0 0 0 0
0 0 1 1 1

Mapa de bits



Lista encadeada

Bloco	Contador
03	05
12	02
15	02
20	07

Tabela de blocos livres

# — Implementação do sistema de arquivos

## Alocação Contígua



Alocação contígua armazena arquivos em blocos adjacentes, facilitando a leitura e é usada em cd-roms.



Desvantagem: complexa alocação para novos arquivos pode levar à fragmentação do disco.



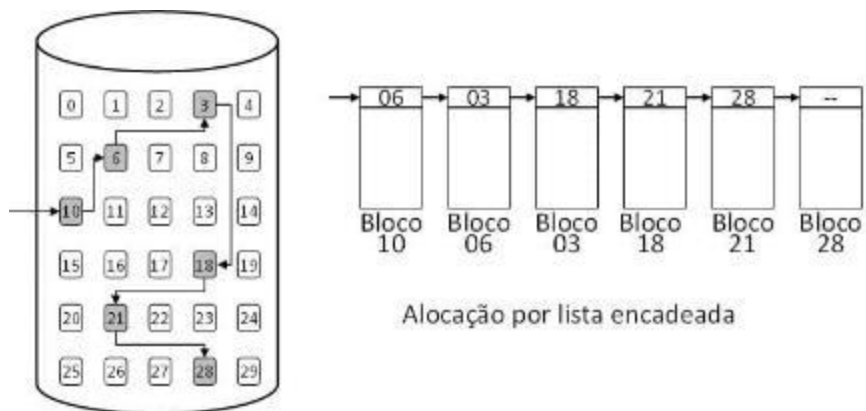
Estratégias de alocação ajudam a escolher blocos livres para novos arquivos.



Fragmentação excessiva exige desfragmentação periódica do disco.

# — Implementação do sistema de arquivos

## Alocação por Lista Encadeada

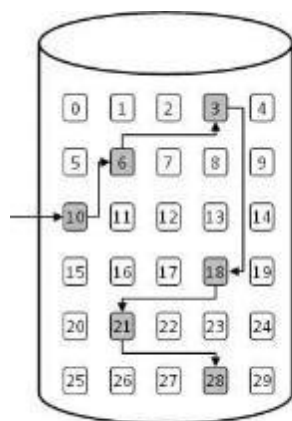


Na alocação por lista encadeada, um arquivo é composto por blocos logicamente ligados, com cada bloco apontando para o próximo.

- A entrada de diretório só precisa armazenar o endereço do primeiro bloco.
- Não há fragmentação de disco, mas pode ocorrer fragmentação de arquivos.
- A alocação por lista encadeada permite apenas acesso sequencial aos blocos e é vulnerável a falhas de confiabilidade.

# — Implementação do sistema de arquivos

## Alocação por Lista Encadeada Utilizando Índice



..	..	..	18	..
..	03	..	..	..
06	..	..	21	..
..	28	..	..	..
..	..	..	-1	..

Alocação por lista encadeada utilizando índice

- Alocação por lista encadeada com índice: usa uma tabela (FAT) para manter o encadeamento de blocos em vez de ponteiros.
- A tabela economiza espaço, mas pode se tornar grande em discos maiores.



# — Implementação do sistema de arquivos

## I-nodes (Alocação Indexada)

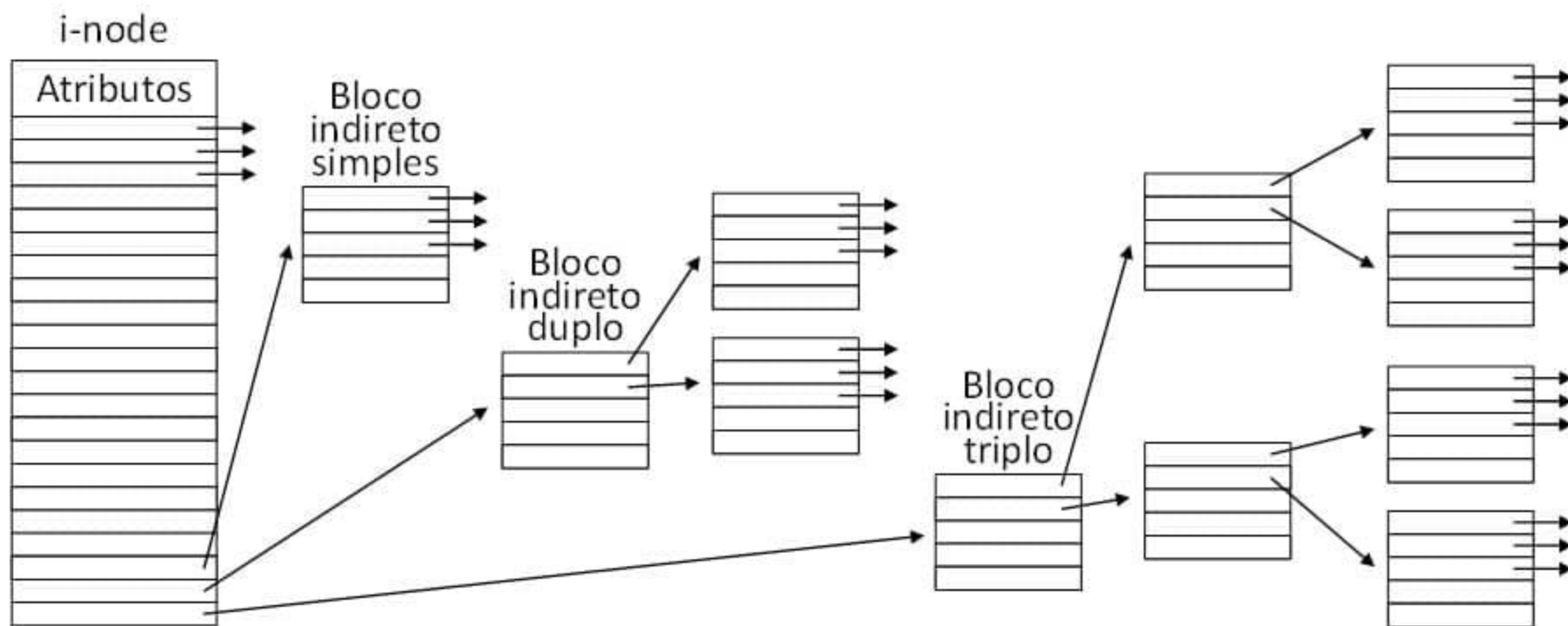
Alocação Indexada usa i-nodes, uma tabela associada a cada arquivo que lista os atributos e endereços dos blocos.

Vantagem: i-nodes precisam estar na memória apenas quando o arquivo está aberto, economizando espaço.

i-nodes contêm informações de pequenos arquivos. Grandes arquivos usam blocos indiretos simples, duplos e triplos para endereçamento.

# — Implementação do sistema de arquivos

## I-nodes (Alocação Indexada)



## — Implementação de cache



Acesso a disco é lento; sistemas usam cache na memória para acelerar operações.



Cache armazena dados frequentemente usados, reduzindo a necessidade de acessos ao disco.



Políticas de substituição de blocos na cache são implementadas, levando em consideração a segurança.



Duas abordagens comuns para atualização de cache são write-through (seguro) e write-back (menos E/S).

## — O sistema de arquivos do Linux



O Linux usa um sistema de arquivos com i-nodes, superblocos e dentry.

- Suporta diversos sistemas de arquivos e carrega módulos dinamicamente.
- Não impõe extensões de arquivos, deixando essa interpretação para programas.
- Possui uma estrutura de diretórios comuns, incluindo /bin, /boot, /etc, /home, entre outros.

# — O sistema de arquivos do Linux

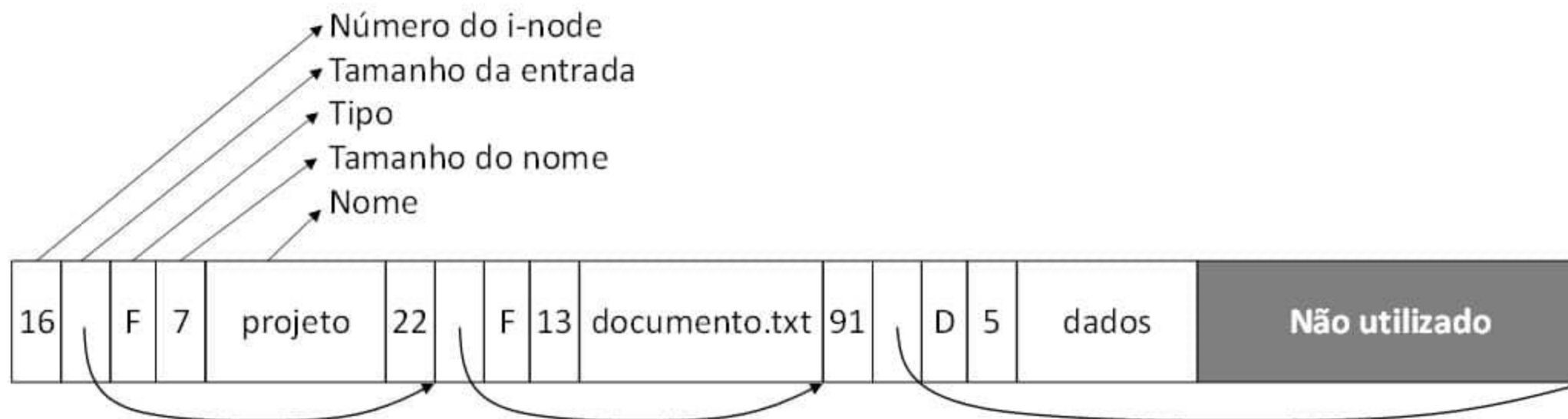
## *ext2*

- Estrutura inclui superbloco, descrição do grupo e i-nodes.
- Dois mapas de bits controlam blocos e i-nodes livres.
- Diretórios contêm entradas para nomes de arquivos e diretórios.
- Organização pode resultar em entradas fora de ordem e espaços não utilizados em blocos de disco.



# — O sistema de arquivos do Linux

*ext2*



# — O sistema de arquivos do Linux

## *ext2*

Cada entrada de diretório consiste em 4 campos de comprimento fixo e 1 campo de comprimento variável.

O primeiro campo é o número do i-node (16 para projeto, 22 para documento.txt e 91 para dados).

O segundo campo contém o tamanho da entrada, incluindo espaços não utilizados no seu final.

O terceiro campo identifica o tipo (arquivo, diretório, link etc.).

O quarto campo é o tamanho do nome do arquivo.

O quinto campo possui tamanho variável e armazena o nome do arquivo.

# — O sistema de arquivos do Linux

## *ext2*

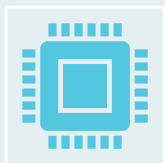
Cada i-node do *ext2* possui o formato ilustrado acima, em que:

- Permissões → Permissões de acesso ao arquivo;
- Contador de links → Quantidade de **hardlinks** que o arquivo possui;
- UID → Dono do arquivo;
- GID → Grupo do dono do arquivo

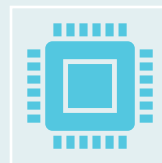
Permissões
Contador de links
UID
GID
Tamanho do arquivo
Horários
Endereços dos 12 primeiros blocos do disco
Bloco indireto simples
Bloco indireto duplo
Bloco indireto triplo



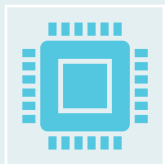
# — Journaling



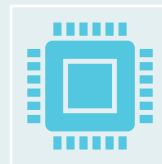
Evita perda de dados após falhas, como falta de energia.



Registra alterações sequencialmente em um diário.

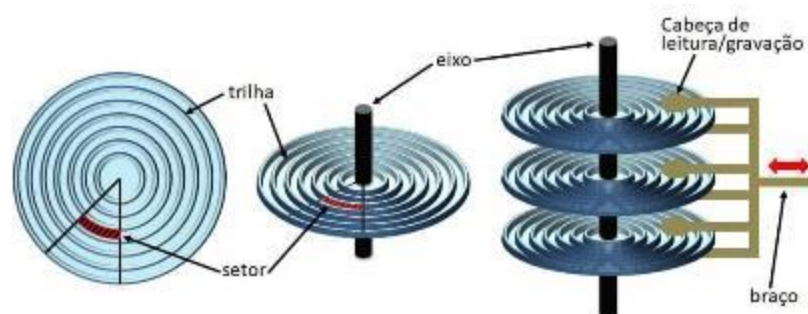


As transações são reexecutadas nas estruturas reais do sistema de arquivos.



Implementado no ext3 do Linux para melhorar a confiabilidade e a recuperação após falhas.

# — Disco rígido



Disco rígido é composto por discos sobrepostos, trilhas concêntricas e setores.

- Tempo de acesso ao disco inclui busca, latência e transferência.
- O tempo de busca é para mover a cabeça até o cilindro desejado.
- O tempo de latência é a espera até o setor desejado passar sob a cabeça.

# — Disco rígido

## Partições e Formatações



Particionamento divide o espaço de disco em seções independentes.



Partições são identificadas por arquivos especiais no diretório /dev.



Formatação cria a estrutura do sistema de arquivos em uma partição.



Processo de formatação envolve a criação de grupos de blocos e reserva de espaço para dados.

# — Montagem do sistema de arquivos

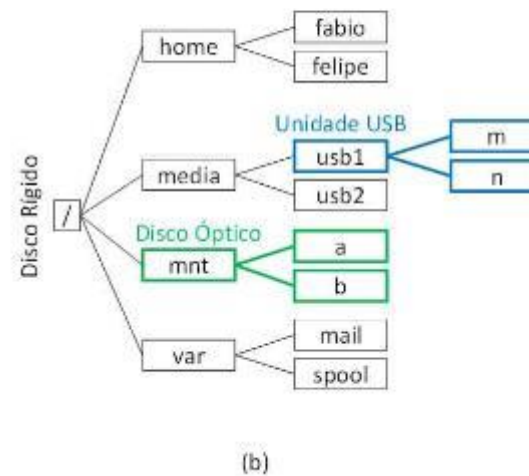
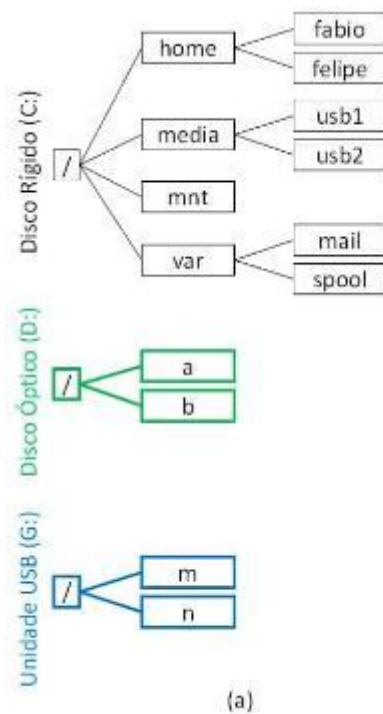


Em sistemas com múltiplos discos ou partições, é necessário encontrar uma forma de acessar os sistemas de arquivos.

- No Windows, os dispositivos são identificados por letras e dois pontos (C:, D:, etc.).
- No Linux, os sistemas de arquivos são montados na árvore de diretórios do sistema.
- Por exemplo, um dispositivo USB pode ser montado em /media/usb1.

# — Montagem do sistema de arquivos

Estrutura do sistema de arquivos descrito (a) no Microsoft Windows e (b) no Linux.



## — Outros comandos para gerenciamento de partições

### **fsck**

Utilitário utilizado para verificar e, se necessário, corrigir um sistema de arquivos. Se não for informado o tipo do sistema de arquivos, o fsck procurará por essa informação no arquivo `/etc/fstab`.

### **df**

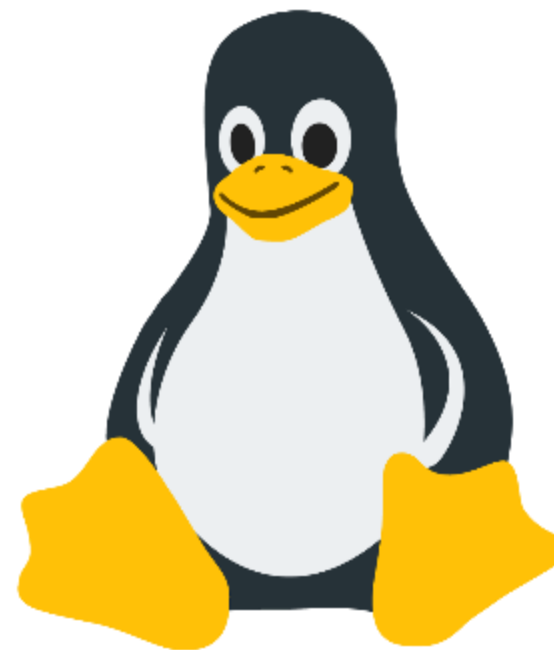
Utilitário para exibição do espaço livre/ocupado por cada sistema de arquivos montados.

### **lsblk**

Utilitário que lista informações sobre os dispositivos de bloco do sistema, excetuando-se os discos de RAM

## — Conceitos

- Linux é um núcleo de sistema operacional.
- Distribuições Linux combinam o Linux com softwares adicionais.
- Existem várias interfaces gráficas disponíveis para o Linux.
- Arquivos ocultos no Linux começam com "." e não são exibidos a menos que solicitados.



## — Comandos para manipulação de diretórios



ls



pwd



cd



mkdir



rmdir



## — Comandos para manipulação de arquivos



**rm**



**cp**



**mv**



**cat**



**find**

## — Links simbólicos e hardlinks

Links simbólicos apontam para entradas de diretório, funcionam como atalhos e são transparentes para o usuário.



Hardlinks permitem que várias entradas de diretório apontem para o mesmo i-node, economizando espaço.



O comando `'ln'` é usado para criar links simbólicos e hardlinks.



Alterações em um arquivo acessado por qualquer link refletem em todos, pois compartilham o mesmo conteúdo.

## — Editor de arquivos x processador de textos



Arquivos de texto puro contêm caracteres ASCII, enquanto documentos de processadores de texto têm formatação, fontes e elementos binários.



No Linux, os arquivos de texto puro são comuns para configuração de sistemas.



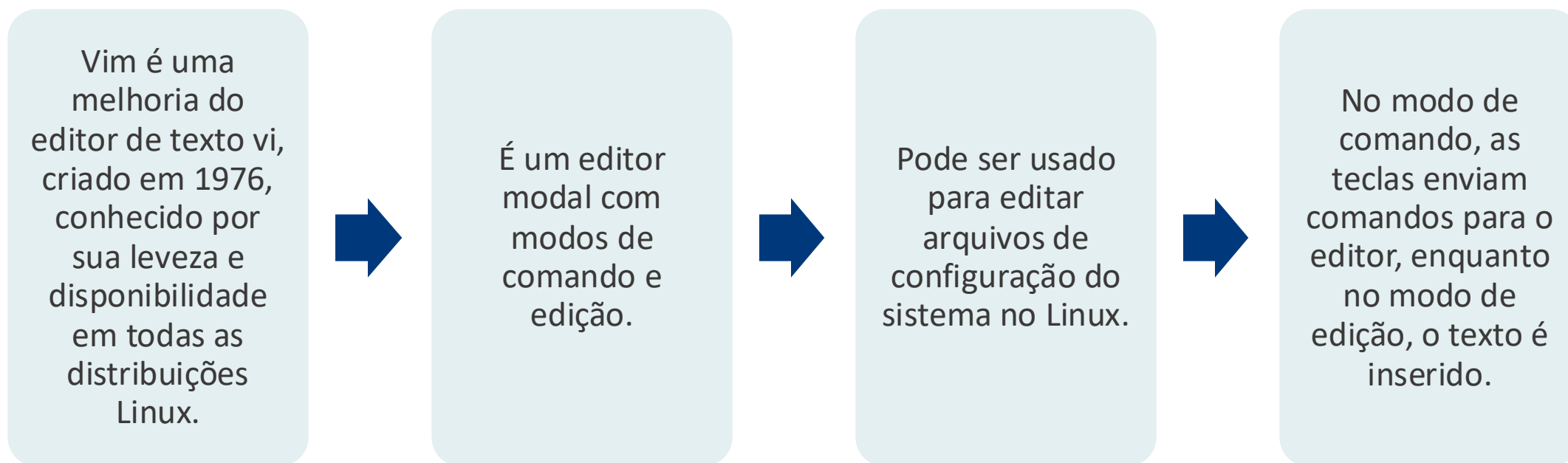
O Writer do LibreOffice é um processador de texto popular no Linux.



Administradores de sistemas usam editores de texto puro para configurar sistemas Linux.

# — Editor de arquivos x processador de textos

## Vim



# — Editor de arquivos x processador de textos

## Principais comandos do vim

**Entrar em modo de edição**

**Sair do modo de edição**

**Salvar e sair**

**Salvar e sair**

**Movimentação**

**Apagando**

**Desfazendo alterações**

**Copiando**

**Colando**

**Repetição de comandos**

**Buscar**

# — Editor de arquivos x processador de textos

## Nano

Nano é um editor de texto para linha de comando presente em distribuições Linux.

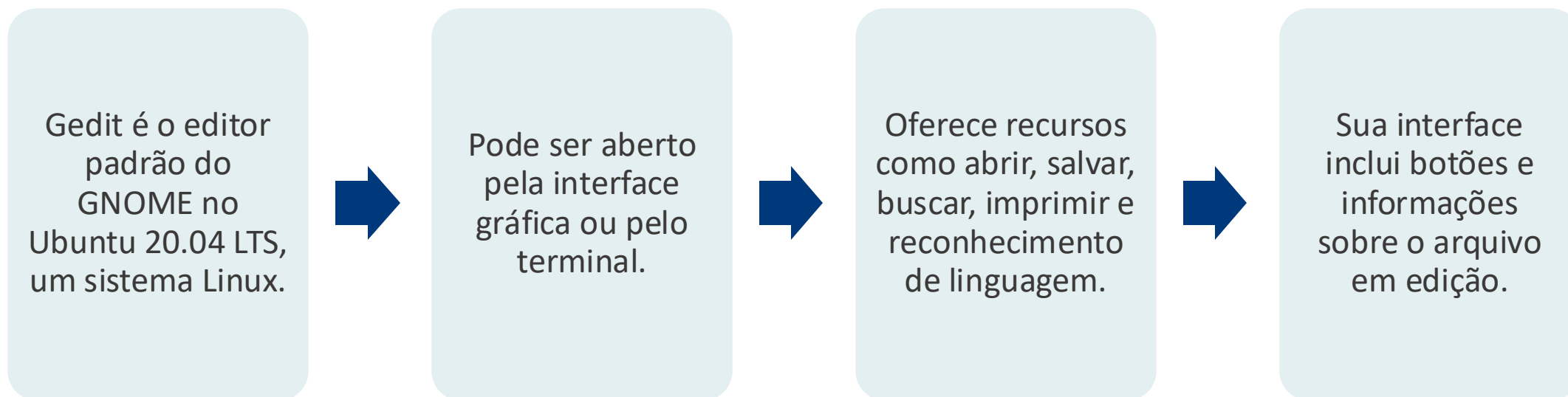
Diferente do Vim, o Nano é mais intuitivo e não tem modos de operação complexos.

Os comandos no Nano são exibidos na parte inferior da tela para facilitar o uso.

Pode ser iniciado com o comando `"nano nome_do_arquivo"` e possui comandos como salvar, sair e procurar no texto.

## — Editor de arquivos x processador de textos

### Gedit



# Automatizando tarefas no Linux





## — Ferramenta multiusuário

O CRON é uma ferramenta multiusuário que permite a configuração de tarefas específicas para cada usuário no sistema.

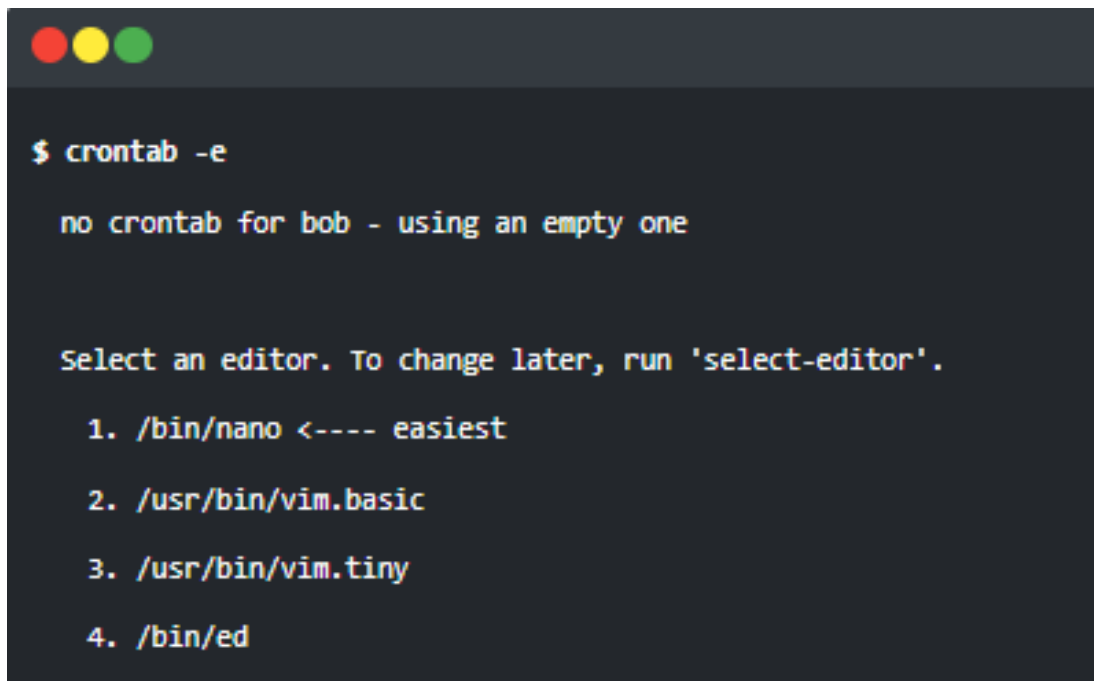
Exemplo: o usuário 'root' pode configurar uma tarefa diária às 23h, enquanto o usuário 'bob' pode ter uma tarefa semanal aos domingos às 9h.

O CRON executará automaticamente essas tarefas de acordo com as configurações de cada usuário.

Certifique-se de considerar as permissões necessárias para que cada comando seja executado com sucesso.

## — Como configurar o cron de usuário

Para configurar o CRON de um usuário, utilize o comando "crontab -e" e escolha um editor de texto. Certifique-se de salvar as alterações após a edição do CRONTAB.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays the command '\$ crontab -e' and its output: 'no crontab for bob - using an empty one'. It then prompts the user to 'Select an editor. To change later, run \'select-editor\'.' and lists four options: 1. /bin/nano <---- easiest, 2. /usr/bin/vim.basic, 3. /usr/bin/vim.tiny, and 4. /bin/ed.

```
$ crontab -e

no crontab for bob - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/nano <---- easiest
 2. /usr/bin/vim.basic
 3. /usr/bin/vim.tiny
 4. /bin/ed
```

## — Regras e formatos do crontab



As configurações CRON são definidas em um arquivo de texto com regras específicas.



Cada tarefa possui 5 campos de tempo (minuto, hora, dia do mês, mês, dia da semana).



Os campos podem conter números, abreviações ou asteriscos para representar qualquer valor.



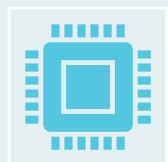
Vírgulas, traços e barras são usados para criar agendamentos sofisticados.

## — Exemplo de uma tarefa automática de backup

Para automatizar um backup diário às 20h, adicione a seguinte linha ao seu CRONTAB:

```
0 20 * * * tar -czf /tmp/backup.tar.gz /home/bob
```

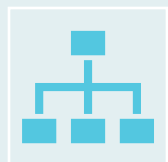
## — A configuração de cron para o sistema



O CRON permite configurações específicas para tarefas de usuários e do sistema no Linux.



O arquivo `/etc/crontab` é usado para configurar tarefas do sistema, com um campo extra para o nome de usuário.



Tarefas do sistema são frequentemente organizadas em diretórios como `/etc/cron.daily` e `/etc/cron.hourly`.



Essas configurações garantem a execução automática de comandos em horários agendados.

## — Conceitos

- Os scripts são sequências de comandos que automatizam tarefas no terminal do Linux, sem necessidade de compilação.
- O Shell permite a criação de variáveis e estruturas de repetição, tornando possível automatizar tarefas complexas.



## — O interpretador de shell



- Existem diversos interpretadores disponíveis para o Linux, e as sintaxes para SCRIPTS variam entre eles.
- Atualmente, as principais distribuições adotam o interpretador BASH como o padrão, portanto vamos utilizá-lo como referência nesse curso.

## — A palavra mágica que identifica um script

- No Linux, os scripts são executados como comandos ou programas.
- Para identificar um arquivo como script, use a "palavra mágica" '#!' seguida do interpretador de SHELL.
- Exemplo: '#!/bin/bash' define o arquivo como um script Bash.
- A "palavra mágica" é crucial para o Linux reconhecer e executar o script corretamente.





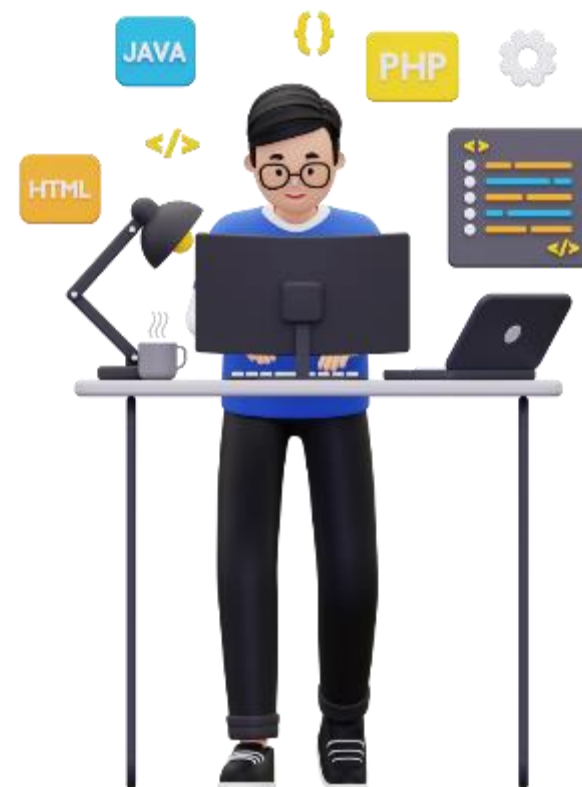
## — Linhas em branco e comentários



- Linhas vazias e iniciadas com '#' são ignoradas pelo interpretador.
- Linhas com '#' são usadas para comentários, observações e instruções.
- A "palavra mágica" na primeira linha (!) é diferente e essencial, não deve ser confundida com comentários.

## — Incluindo textos na resposta do script

- Para incluir texto na saída de um script, use o comando 'echo'.
- Modifique o comando 'date' para formatar a hora desejada (por exemplo, 'date +%H:%M').
- Use 'echo' para exibir o texto e a hora no mesmo comando (por exemplo, 'echo "Hora certa \$(date +%H:%M)"').
- Evolua o script para incluir data e hora juntas (por exemplo, 'echo "Data: \$(date +%d/%m/%Y) Hora: \$(date +%H:%M)"').



## — Inserindo caracteres especiais no comando 'echo'

- O comando 'echo -e' permite o uso de caracteres especiais, como '\n' para inserir uma quebra de linha.
- No exemplo anterior, o script exibe a data e hora em linhas separadas com quebras de linha.
- Isso torna a saída mais legível e organizada.



## — Criando pausas na execução de um script



- O comando 'sleep' permite criar pausas na execução do script, com o tempo em segundos como parâmetro.
- Substituindo 'sleep' por 'read', o script aguardará a interação do usuário pressionando ENTER.
- Para evitar as linhas em branco após o 'read', use 'read -s' como alternativa.

## — Apagando todo o conteúdo do terminal

- Para apagar o conteúdo do terminal, use o comando 'clear' no início ou durante a execução do script.
- Isso limpará a tela e tornará as respostas do script mais visíveis e organizadas.



## — Terminando um script com valor de retorno

O comando 'exit' permite encerrar um script e retornar um valor numérico ao sistema operacional.

Um valor de retorno 0 geralmente indica execução bem-sucedida, enquanto outros valores (de 1 a 255) podem indicar erros ou resultados específicos.

O valor de retorno pode ser definido explicitamente (por exemplo, 'exit 1') ou será o resultado do último comando executado no script.

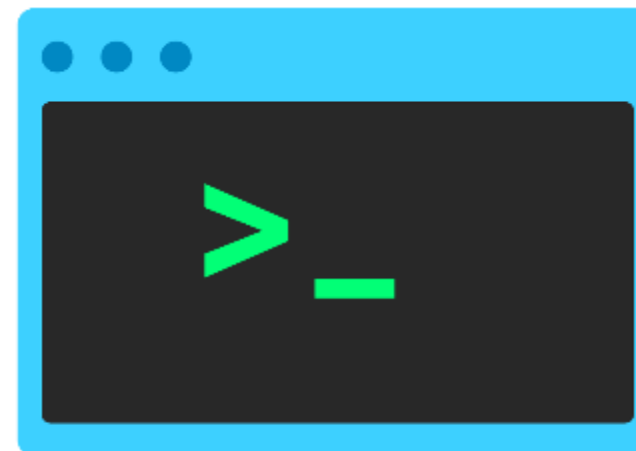
## — Atribuição de valores a uma variável

- Para atribuir valores a variáveis no shell, use a sintaxe: **VAR=1**.
- Não deve haver espaços em branco em torno do sinal de igual.
- Use aspas para atribuir valores de texto com espaços, por exemplo:  
**'VAR="Terminal do Linux".'**
- Lembre-se de que os nomes de variáveis são sensíveis a maiúsculas e minúsculas.



## — Referenciando variáveis

- Para referenciar o conteúdo de uma variável, use o símbolo '\$' à esquerda do nome da variável, por exemplo: \$VAR.
- Isso permite que você use o valor da variável em comandos, como no exemplo: echo "\$VAR", que exibirá o conteúdo da variável "VAR".





## — Atribuição da saída de um comando a uma variável

- Você pode atribuir a saída de um comando a uma variável usando a sintaxe: `VAR=$(comando)`.
- Nesse exemplo, a hora atual é armazenada na variável "INICIO" usando `INICIO=$(date +%T)`.
- Isso permite comparar o início e o término da execução do script e calcular a duração.



## — Passando parâmetros para o script



É possível passar parâmetros para um script ao executá-lo, separados por espaços.



Dentro do script, você pode acessar esses parâmetros usando as variáveis \$1, \$2, \$3, e assim por diante.



O nome do próprio script pode ser acessado usando \$0.



O exemplo de 'script4' mostra como exibir os parâmetros passados durante a execução do script.

## — O operador lógico if

- O comando 'if' no Bash permite a execução condicional com base em uma condição.
- A sintaxe básica é: `if [ CONDIÇÃO ]; then ... fi`, onde o código entre `then` e `fi` é executado se a condição for verdadeira.
- O 'if' também pode ser estendido com 'else' para executar código quando a condição for falsa.

## — Comparadores numéricos

No Bash, você pode usar comparadores numéricos para avaliar condições.



Os comparadores numéricos incluem: -eq (igual), -ne (diferente), -gt (maior que), -ge (maior ou igual), -lt (menor que) e -le (menor ou igual).



Exemplo: `if [[ "$A" -eq "$B" ]]` verifica se a variável \$A é igual à variável \$B.



O script5 exemplifica como usar esses comparadores para realizar comparações numéricas entre variáveis.

## — Comparadores de cadeias de caracteres (strings)

No Bash, você pode usar comparadores de cadeias de caracteres para avaliar condições.

Os comparadores de cadeias de caracteres incluem: = (igual), != (diferente), -z (cadeia nula ou vazia), e -n (cadeia não nula).

Exemplo: `if [[ "$A" = "$B" ]]` verifica se a variável \$A é igual à variável \$B em termos de texto.

O exemplo de 'script5' demonstra como usar esses comparadores de cadeias de caracteres, incluindo a validação de parâmetros para evitar erros quando parâmetros estão ausentes.

## — Executando o script sem parâmetros

Scripts interativos permitem que o usuário forneça valores durante a execução.

O comando 'read' é usado para receber a entrada do usuário e atribuí-la a variáveis.

No exemplo interativo, o usuário fornece os valores de A e B, que são lidos com 'read'.

O restante do script continua a partir da validação desses valores, como no exemplo anterior.

## — Realizando operações aritméticas em script

- Scripts podem realizar operações aritméticas com variáveis.
- No exemplo do 'script6', o usuário fornece dois valores, e o script calcula a soma deles.
- O cálculo pode ser armazenado em uma variável (por exemplo, C) ou calculado diretamente dentro do comando 'echo'.


## — Incrementando variáveis

- Para incrementar o valor de uma variável em uma unidade, use a expressão: `(( X++ ))`.
- Isso aumentará o valor da variável X em 1.
- Por exemplo, se X for igual a 1 e usarmos `(( X++ ))`, o valor de X será incrementado para 2.




## — Operações com ponto flutuante

O shell não lida bem com operações de ponto flutuante, mas você pode usar o comando 'bc' para executar cálculos com precisão de ponto flutuante.



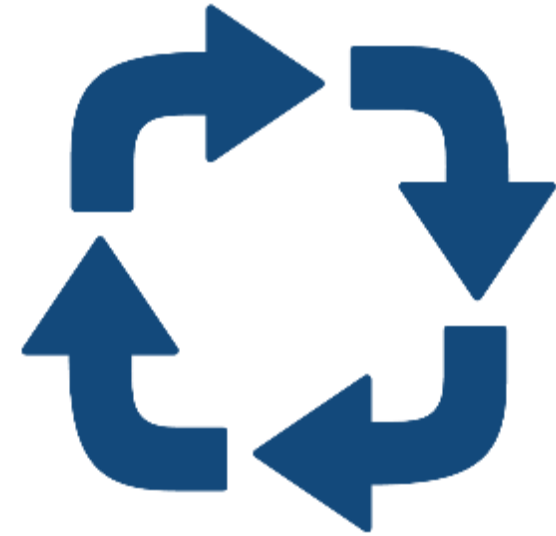
No exemplo, o comando 'bc' é usado para calcular a média de três valores (A, B e C) e preservar as casas decimais no resultado.



A parte "scale=2" define o número de casas decimais no resultado.

## — Conceitos

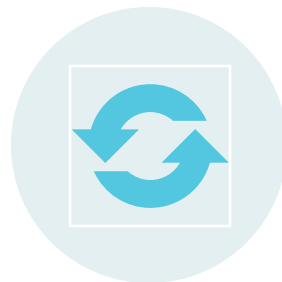
- Estruturas de repetição, também chamadas de LOOPS, permitem a execução repetitiva de código até que um objetivo seja alcançado.
- Para evitar loops infinitos, é necessário definir uma condição de saída que determina quando o loop deve ser encerrado.



## — A estrutura de repetição while



O 'while' é uma estrutura de repetição que executa um bloco de comandos enquanto uma condição é verdadeira.



Cada iteração do loop verifica a condição e executa os comandos dentro do loop se a condição for verdadeira.



A condição deve ser definida antes do início do loop e pode ser baseada em variáveis ou expressões.



O loop 'while' é uma ferramenta poderosa para automatizar tarefas repetitivas em scripts.

## — Interferindo na execução de um loop

**break:** Comando que força a interrupção imediata de um loop, independentemente da condição de continuação.

**continue:** Comando que força o início de uma nova iteração em um loop, pulando o restante do código da iteração atual.

São úteis para controlar a execução de loops com base em condições específicas.

Permitem lidar com casos especiais e personalizar o comportamento do loop.

## — A estrutura de repetição for

A estrutura de repetição for é utilizada para iterar sobre uma lista de elementos.

No exemplo, o for itera sobre uma lista de números, executando o código dentro do loop para cada número na lista.

Também pode ser usado para iterar sobre arquivos no sistema, onde \* é uma máscara que representa todos os arquivos no diretório atual.

É uma ferramenta poderosa para automatizar tarefas em lote, processar dados em massa e executar ações em vários elementos.

## — Obtendo uma variável a partir de um arquivo texto



Separar a lista de diretórios do script principal é uma prática recomendada para facilitar a manutenção e evitar alterações acidentais no código.



Crie um arquivo separado que contenha a lista de diretórios, com cada diretório em uma linha.



No script principal, leia a lista de diretórios a partir desse arquivo e atribua-a a uma variável.



Utilize a variável contendo a lista de diretórios para iterar sobre os diretórios no seu script principal, tornando-o mais flexível e independente da lista em si.

## — Considerações ao executar um script a partir do cron

- Ao executar scripts com o CRON, evite interações com o usuário, pois não haverá um terminal ativo durante a execução.
- Para a entrada de dados, utilize arquivos ou parâmetros de execução para passar comandos e informações variáveis.
- Para a saída de dados, salve mensagens em arquivos de log e use redirecionadores no CRON para direcionar a saída para esses arquivos.
- Expressões regulares (regex) são úteis para validar dados, como garantir que números tenham formatos específicos.