Por: Alice Dantas

Processos

Um processo é simplesmente um programa em execução. Toda vez que você executa um comando ou abre um programa no Linux, ele se torna um processo. Exemplos práticos:

- O comando firefox → abre o Firefox como um processo.
- O comando nano arquivo.txt → executa o editor de texto nano como um processo.

- Por que o sistema precisa de processos?

O computador não consegue executar diretamente arquivos no disco. O sistema operacional precisa:

- Carregar o programa na memória RAM.
- Alocar recursos (CPU, memória, arquivos, dispositivos).
- Controlar o tempo de execução, comunicação e acesso a hardware.

Cada processo é tratado como uma **entidade independente**, com seus próprios recursos e seu próprio ciclo de vida.

- Processo Pai e Processo Filho:

Quando um processo cria outro, ele vira um **processo pai**, e o novo é o **processo filho**. Isso é comum, por exemplo, quando você abre um terminal (bash) e dentro dele roda o comando ls:
O bash é o pai e o ls é o filho, que existe apenas enquanto está rodando.

No Linux, todos os processos são descendentes do processo PID 1, que é o init ou systemd, responsável por inicializar todo o sistema.

- Multiprocessamento e Concorrência

O Linux (e outros SO modernos) são **multitarefa**, ou seja:

- Conseguem manter vários processos ativos ao mesmo tempo.
- Isso é feito graças à concorrência e à alternância rápida da CPU entre processos.

→ Parece que tudo roda ao mesmo tempo, mas na verdade, a CPU troca rapidamente de um processo para outro, milhares de vezes por segundo.

Por: Alice Dantas

- Isolamento e Comunicação

Cada processo tem seu próprio espaço de memória, não podendo acessar diretamente a memória de outro processo. Isso garante segurança e estabilidade. Se um processo falha, não derruba os outros. Contudo, embora isolados, os processos podem se comunicar por:

Pipes (|): Serve para enviar a saída de um processo diretamente como entrada de outro. É temporário e funciona só entre processos que estão rodando no mesmo terminal, em sequência. Ex.: `ls | grep .txt` →lista arquivos e filtra só os .txt.

Arquivos temporários: Um processo escreve dados em um arquivo e outro processo lê desse arquivo. Ex.: `echo "Alice" > temp.txt` `cat temp.txt` → Um comando escreve, outro lê.

Sockets: Permite que dois processos conversem entre si, seja: dentro do mesmo computador ou entre computadores diferentes via rede (internet ou intranet). É uma conexão bidirecional, ou seja, vai e volta. Exs.:

- Quando abre o navegador → Ele cria um socket para se conectar ao servidor do site (Google, YouTube, etc.).

- Quando usa WhatsApp Web, Instagram, Netflix... → São sockets trocando dados o tempo todo entre seu navegador e os servidores.

Shared Memory (memória compartilhada): É uma área da memória RAM que pode ser acessada por dois ou mais processos ao mesmo tempo. É o método mais veloz de comunicação, porque usa direto a memória RAM, sem gravar no disco ou ficar trocando mensagens. Exs.:

- Jogos Online e Sistemas em Tempo Real → Processos sincronizam informações como posição dos jogadores, pontuação e estado dos objetos no mapa, garantindo atualizações em tempo real com baixíssima latência.

- Sistemas de Bolsa de Valores e Trading → Processos compartilham dados de mercado (preços, ordens, volume) e atualizações em tempo real, com desempenho que nenhuma solução baseada em arquivos ou rede conseguiria alcançar.

Semáforos: São mecanismos de controle usados para sincronizar o acesso de processos a recursos compartilhados. Funcionam como um sinal de trânsito, que impede que dois processos acessem o mesmo dado ou recurso ao mesmo tempo. Exs.:

- Robótica, IoT, Automação Industrial → Controlam o acesso de múltiplos processos a informações de sensores, motores e dispositivos, evitando que comandos sejam executados ao mesmo tempo de forma desordenada.

- E-commerce, Sites e APIs → Controlam operações críticas, como atualização de estoque, vendas simultâneas e processamento de pedidos, garantindo que não ocorram erros como vender um produto que já acabou.

Por: Alice Dantas

- Conceitos importantes sobre Processos

Conceito	Significado
PID	**Process ID** → Identificador único de cada processo.
PPID	**Parent Process ID** → PID do processo pai (quem criou aquele processo).
Estado	O que o processo está fazendo (executando, esperando, etc.).
Usuário	Quem é o dono do processo.
Prioridade	Nível de prioridade na CPU.

- Visualizando Processos

1. Usando ps

Exibe processos ativos.

`ps` → mostra processos no seu terminal
`ps aux` → mostra todos os processos do sistema
`ps aux | grep firefox` → filtrar por nome

2. Usando top

`top` → mostra os processos em tempo real através de uma interface interativa.
Para sair: **q**

3. Usando htop (versão mais bonita do top)

`htop`
Se não tiver, instale: `sudo apt install htop`

Por: Alice Dantas

- Entendendo a saída do comando *ps aux*

Coluna	Significado
USER	Usuário dono do processo
PID	Process ID (identificador do processo)
%CPU	Porcentagem de uso da CPU
%MEM	Porcentagem de uso da memória RAM
VSZ	Memória virtual usada (em KB)
RSS	Memória física usada (em KB)
TTY	Terminal associado
STAT	Estado do processo
START	Quando iniciou
TIME	Tempo de CPU consumido
COMMAND	O comando que gerou o processo

- Estados dos Processos (Coluna STAT)

Letra	Estado
R	Executando (Running)
S	Em espera (Sleeping)
D	Esperando I/O (Disk)
T	Parado (Stopped)
Z	Zumbi (Zombie)

Por: Alice Dantas

- Comandos para finalizar processos

- Finalizar por PID:

`kill [PID]` → Envia sinal SIGTERM (educado): **"Por favor, termine."**

- Forçar finalização (quando não responde):

`kill -9 [PID]` → Sinal SIGKILL: **"Morre, agora!"**

- Finalizar por nome:

`pkill nome_do_processo`

- Verificar se finalizou:

`ps aux | grep nome_do_processo`