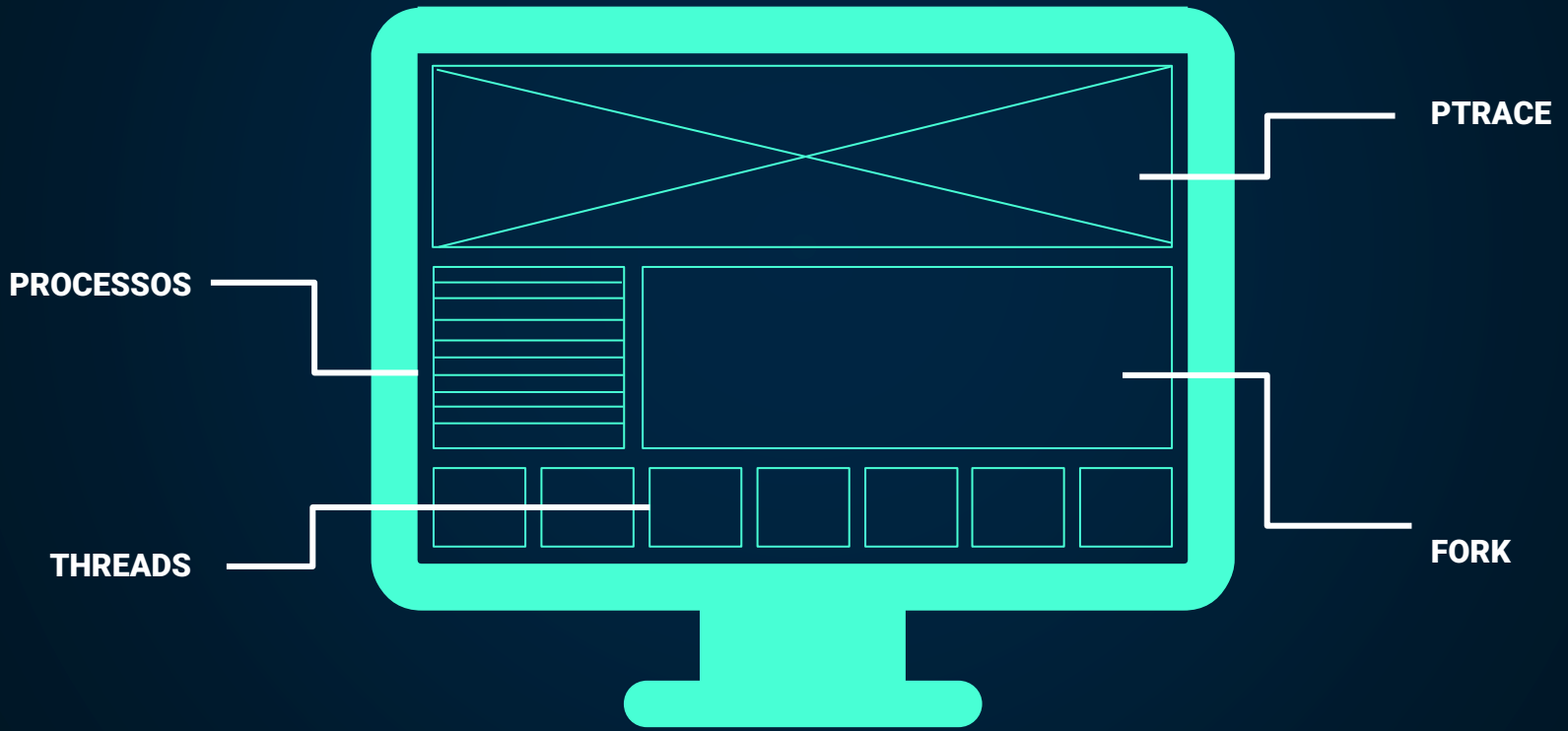
Sistemas Operacionais

LINUX



CARACTERISTICAS DO LINUX

- **Criado por Linus Torvalds em 1991;**
- **Foi criado, por falta de várias características do Minix, as quais o autor recusou a colocar por querer deixar o sistema pequeno para que estudantes pudessem entende-lo em um semestre.**
- **Kernel de código-fonte aberto que foi e continua sendo desenvolvido ao longo do tempo;**

PRINCIPAIS DIFERENÇAS ENTRE WINDOWS E LINUX

- Código fonte aberto

- Mais segurança e privacidade

- Mais liberdade no controle do sistema

- Gratuito

POR QUE LINUX É MAIS UTILIZADO EM SERVIDORES?

- **Preço**
- **Mais fácil automatizar a instalação e configuração**
- **Atualizações de segurança**
- **Integração**
- **É altamente estável**
- **É open source**

PROCESSOS

- **Forma de representar um programa em execução**
- **Utiliza recursos do computador (processador, memoria, etc.)**

"Um processo é um tipo de atividade. Ele tem um programa, entrada, saída e um estado. Um único processador pode ser compartilhado entre vários processos, com algum algoritmo de agendamento sendo utilizado para determinar quando parar de trabalhar em um processo e servir a um diferente"

Características

- **Proprietário do processo**
- **Estado do processo (em espera, em execução, etc.)**
- **Prioridade de execução**
- **Recursos da memoria**
- **PID (Process Identifer)**
- **PPID (Parent Process Identifier)**

CRIAÇÃO DE PROCESSOS (fork)

- Através da chamada ao sistema (ou seja, invoca o SO para fazer alguma coisa que o usuário não pode) **fork**, que cria uma cópia exata do processo original
  - Mesmas variáveis, registros, descritores de arquivos, etc

- O processo criador é chamado de processo pai, o novo de processo filho

- Cada um tem sua própria imagem da memória

- Depois de criado, o processo filho segue seu caminho

A função fork é chamada uma única vez (no pai) e retorna duas vezes (uma no processo pai e uma no processo filho)

O processo filho herda as informações do processo pai:

- User ID
- GID
- Prioridade
- Diretório raiz
- Variáveis de ambiente
- Descritores de arquivos
- Mascara de criação de arquivos

O processo filho possui as seguintes informações diferentes diferente do processo pai:

- PID
- PPID
- Um conjunto de sinais pendentes para o processo é inicializado como estando vazio
- Locks de processo, códigos e dados
- Sinais de tempo são desligados

EXEMPLO DE CRIACAO DE UM PROCESSO

```
pid_t  PID
PID = fork ();

if(PID < 0) {
                O fork falhou; }
else if (PID>0){
                Faz o código do pai aqui;
}
else {
                Faz o código do filho aqui
}
```

PERMISSÕES NOS PROCESSOS

"Cada processo precisa de um proprietário e cada usuário precisa pertencer a um ou mais grupos"

- UID (User Identifier) → Identificador de usuário
- GID (Group Identifier) → Identificador de grupos

Variam de 0 a 65536
(dependendo do sistema o valor limite pode ser maior)

Usuários root tem o UID e GID = 0

ÁRVORE DE PROCESSOS

Cada processo possui um pai, com exceção do processo init, que é o processo raiz da árvore de processos do SO

Para ver a árvore de processos usa-se o comando pstree, por exemplo:

```
$ pstree -a
init
  ├─accounts-daemon
  │    └─{accounts-daemon}
  ├─apache2 -k start
  │    ├─apache2 -k start
  │    ├─apache2 -k start
  │    ├─apache2 -k start
  │    ├─apache2 -k start
  │    └─apache2 -k start
  ├─atd
  ├─automount
  │    └─2*[{automount}]
  ├─cron
  │    └─cron
  │         └─bash /usr/local/bin/gbzando/test.sh
```

Entre as opções, tem-se:

-u: mostra o proprietário do processo;

-p: exibe o PID após o nome do processo;

-c: mostra a relação de processos ativos;

-G: usa determinados caracteres para exibir o resultado em um formato gráfico.

Se o PID não for informado na hora de usar o pstree, todos os processos serão listados

SINAIS DE PROCESSOS

Forma de comunicação entre processos e para que o sistema possa interferir no seu funcionamento

Pode ser definido com o prefixo “SIG”, sem o prefixo “SIG” e com numeração

Categorias de sinais:

Convencional (Standard) – Numeração de 1 a 31, apenas um dos sinais do mesmo tipo é enfileirado para ser processado, o resto é ignorado

Tempo real (Realtime) – Numeração acima de 31, todos os sinais são enfileirados e processados na ordem de chegada

NUMERAÇÃO DOS SINAIS DE PROCESSOS

Podem variar nos sistemas Unix

1) SIGHUP	2) SIGINT
3) SIGQUIT	4) SIGILL
5) SIGTRAP	6) SIGABRT
7) SIGBUS	8) SIGFPE
9) SIGKILL	10) SIGUSR1
11) SIGSEGV	12) SIGUSR2
13) SIGPIPE	14) SIGALRM
15) SIGTERM	16) SIGSTKFLT
17) SIGCHLD	18) SIGCONT
19) SIGSTOP	20) SIGTSTP
21) SIGTTIN	22) SIGTTOU
23) SIGURG	24) SIGXCPU
25) SIGXFSZ	26) SIGVTALRM
27) SIGPROF	28) SIGWINCH
29) SIGIO	30) SIGPWR
31) SIGSYS	34) SIGRTMIN
35) SIGRTMIN+1	36) SIGRTMIN+2
37) SIGRTMIN+3	38) SIGRTMIN+4
39) SIGRTMIN+5	40) SIGRTMIN+6
41) SIGRTMIN+7	42) SIGRTMIN+8
43) SIGRTMIN+9	44) SIGRTMIN+10
45) SIGRTMIN+11	46) SIGRTMIN+12
47) SIGRTMIN+13	48) SIGRTMAX+14
49) SIGRTMAX+15	50) SIGRTMAX-14
51) SIGRTMAX-13	52) SIGRTMAX-12
53) SIGRTMAX-11	54) SIGRTMAX-10
55) SIGRTMAX-9	56) SIGRTMAX-8
57) SIGRTMAX-7	58) SIGRTMAX-6
59) SIGRTMAX-5	60) SIGRTMAX-4
61) SIGRTMAX-3	62) SIGRTMAX-2
63) SIGRTMAX-1	63) SIGRTMAX

EXEMPLOS DE SINAIS DE PROCESSOS

STOP – Pausa forçadamente o processo até receber outro sinal (reativar ou terminar)

CONT – A ação padrão é continuar a executar o processo (retorna um processo pausado)

SEGV – Informa erros de endereço de memória

TERM – Termina forçadamente a execução do processo

ILL – Informa erros de instrução (tipo uma divisão por 0)

KILL – Mata um proceso
O KILL também é um comando usado para enviar qualquer sinal , porém se for usado de maneira isolada (sem um parâmetro de um sinal) por padrão executa o TERM

SINTAXE PARA UTILIZAÇÃO DO COMANDO KILL

KILL –SINAL PID

EXEMPLOS:

kill –STOP 4420 → **Pausa o processo com PID 4420**

kill –CONT 4420 → **Processo com PID 4420 volta a ser executado**

kill –STOP -1 → **Todos os processos são pausados**

Quando um sinal precisa ser enviado para todos processos

ESTADO DO PROCESSOS

Running: Processo ativo, rodando normalmente no sistema
O processo pode não estar rodando no sistema, naquele instante. Ele pode estar apenas na ***fila de processos***, podendo ser escalonado a qualquer momento

Sleeping: Processo fica dormindo por tempo finito. Este processo fica em fila diferente, não consome CPU e voltara para a fila de processos quando seu tempo de ***sleep*** acabar

Waiting: O processo fica esperando alguma coisa (recurso ou evento do sistema) acontecer
Este estado é comum quando se espera por alguma I/O

Suspended: Processo está congelado, quando o usuário pausa um processo, assim, ele só volta pra ***fila de processos*** quando o usuário der um ***resume*** (a partir do exato ponto (instrução) em que havia parado

Zombie: Processo que finalizou a execução (portanto não ocupada memória), mas que ainda tem uma entrada na tabela de processos, porque o processo pai ainda não sabe que o processo filho terminou.

Em situações normais, o pai ao ser notificado que o filho terminou, desaloca recursos e executa outras ações necessárias e depois o sistema retira o processo filho da tabela de processos.
Se o pai não realizar nenhuma ação, o filho permanece na tabela de processos até que o processo pai termine sua execução.

Comando NICE

Atributo que permite ao administrador influenciar a prioridade do processo, ajustando o tempo disponível de CPU para este (para mais ou menos prioridade)

- **Varia de -20 (maior prioridade, menos gentil) até +20 (menor prioridade, mais gentil)**
- **Quanto menor o valor, maior a prioridade**
- **Usuários normais podem alterar apenas de 0 até 20, superusuários podem qualquer altera-la para qualquer valor**

PRIORIDADE DE PROCESSOS

Quem executa primeiro

- É possível alterar (o que é muito útil para sistemas multiusuários)
- Ótima forma de colocar todo processamento possível em uma aplicação

Varia de 0 (maior prioridade) a 39 (menor prioridade)

Exemplo:

Um banco de dados precisa melhorar seu desempenho, porem tem restrições de memoria, HD, trafego, o que limitam seu funcionamento num todo, e agora?

Nessa hora uma mudança na politica de acessos aos dados de processamento pode ajudar muito. 😁

Principalmente se tiver muitas reclamações sobre o desempenho

SINTAXE PARA UTILIZAÇÃO DO COMANDO NICE

$ NICE [–N AJUSTE DE PRIORIDADE] [COMANDO]

Varia de -20 até 20

Comando ao qual a prioridade será aplicada

EXEMPLOS:

nice –n -5 ntpd → **Prioridade alta**

nice updatedb & → **Como não passou nenhum argumento o comando nice ajusta a prioridade para +10 como padrão**

O nice é usado apenas para programas que ainda não estão rodando

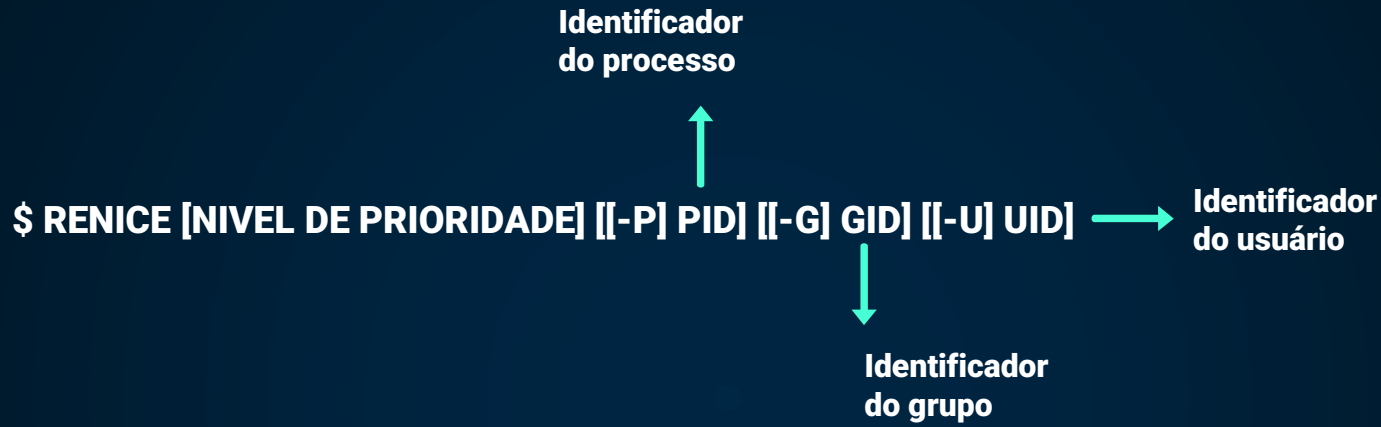
Comando RENICE → Ajusta a prioridade de execução de processos que já estão rodando

Por padrão recebe como parâmetro o PID de um determinado processo

As opções mais usuais no renice são:

- -u: Recebe o nome de um usuário para alterar a prioridade de todos os processos pertencentes a esse usuário
- -g: Recebe um nome de um grupo para alterar a prioridade de todos os processos pertencentes a esse grupo
- -p: Recebe um PID para alterar sua prioridade

SINTAXE PARA UTILIZAÇÃO DO COMANDO RENICE



Exemplos:

renice -1 1234	→	Processo com PID “1234” fica com alta prioridade (-1)
renice -18 –g aulunos_do_renatao	→	Grupo de usuários fica com altíssima prioridade (quase máxima)
renice -2 –u renatao	→	Usuário fica com prioridade alta (-2)

O renice recebe como parâmetro o PID de um processo, por padrão

OBSERVAÇÕES

- Apenas o administrador do sistema (root) pode alterar prioridades dos processos de outros usuários.
- Os usuários comuns podem somente incrementar o valor das prioridade dos seus processos, ou seja, diminuir a prioridade de execução.
- No Linux, os processos em tempo real possuem prioridade negativa. O usuário não pode alterar as prioridades deste tipo de processo.

VERIFICANDO PROCESSOS

PS: Da pra saber quais processos estão em execução atualmente, quais UIDs e PIDs correspondentes, entre outras informações

As opções de combinação com o ps são:

-a: Mostra todos processos existentes

-e: Mostra as variáveis de ambiente relacionadas aos processos

-f: Exibe a arvore de execução dos processos

-l: Exibe mais campos no resultado

-m: Exibe a quantidade de memoria ocupada por cada processo

-u: Mostra o nome do usuário que iniciou determinado processo e a hora que isso ocorreu

-x: Mostra os processos que não estão associados a terminais

-w: Se o resultado do processo não couber em uma linha, essa opção faz com que o resto seja exibido na próxima linha

Exemplo: ps aux
ps lax

DESCRIÇÃO DE ALGUMAS INFORMAÇÕES QUE PODEM APARECER QUANDO OS PROCESSOS FOREM VERIFICADOS

USER - nome do usuário dono do processo;

UID - número de identificação do usuário dono do processo;

PID - número de identificação do processo;

PPID - número de identificação do processo pai;

%CPU - porcentagem do processamento usado;

%MEM - porcentagem da memória usada;

VSZ - indica o tamanho virtual do processo;

RSS - sigla de Resident Set Size, indica a quantidade de memória usada (em KB);

TTY - indica o identificador do terminal do processo;

START - hora em que o processo foi iniciado;

TIME - tempo de processamento já consumido pelo processo;

COMMAND - nome do comando que executa aquele processo;

PRI - valor da prioridade do processo; PRI = 20+NI
NI - Numero setado pelo “nice
WCHAN - mostra a função do kernel onde o processo se encontra em modo suspenso;
STAT - indica o estado atual do processo, sendo representado por uma letra: **R** - executável; **R+** - executando em primeiro plano; **D** - em espera no disco; **S** - Suspenso; **T** - interrompido; **Z** - Zumbi. Essas letras podem ser combinadas e ainda acrescidas de: **W** - processo paginado em disco; **<** - processo com prioridade maior que o convencional; **N** - processo com prioridade menor que o convencional; **L** - processo com alguns recursos bloqueados no kernel.

TOP: Coleta informações dos processos, porem atualiza regularmente (geralmente a cada 10 segundos)

As opções de combinação com o top são:

-d: atualiza o top após um determinado período de tempo (em segundos)

-c: exibe a linha de comando ao invés do nome do processo

-i: faz o top ignorar processos em estado de zumbi

-s: executa o top em modo seguro

Da pra manipular o TOP através das teclas do teclado também, como:

Pressionando a tecla espaço atualiza imediatamente o TOP, se pressionar a tecla “q”, o TOP é finalizado, se pressionar a tecla “h” da para ver a lista completa de opções e teclas de atalho

Para ter ainda mais controle sobre processos no Linux da pra usar alguns outros comandos, como:

JOBS: Visualizar processos que estão parados ou executando em segundo plano

Uma dica pra saber se o processo esta em background é verificar a existência do caractere & no final da linha, se o processo tiver parado a palavra “stopped” aparece na linha, do contrario aparece “running”

Execução feita pelo kernel sem que esteja vinculada a um terminal

As opções disponíveis são:

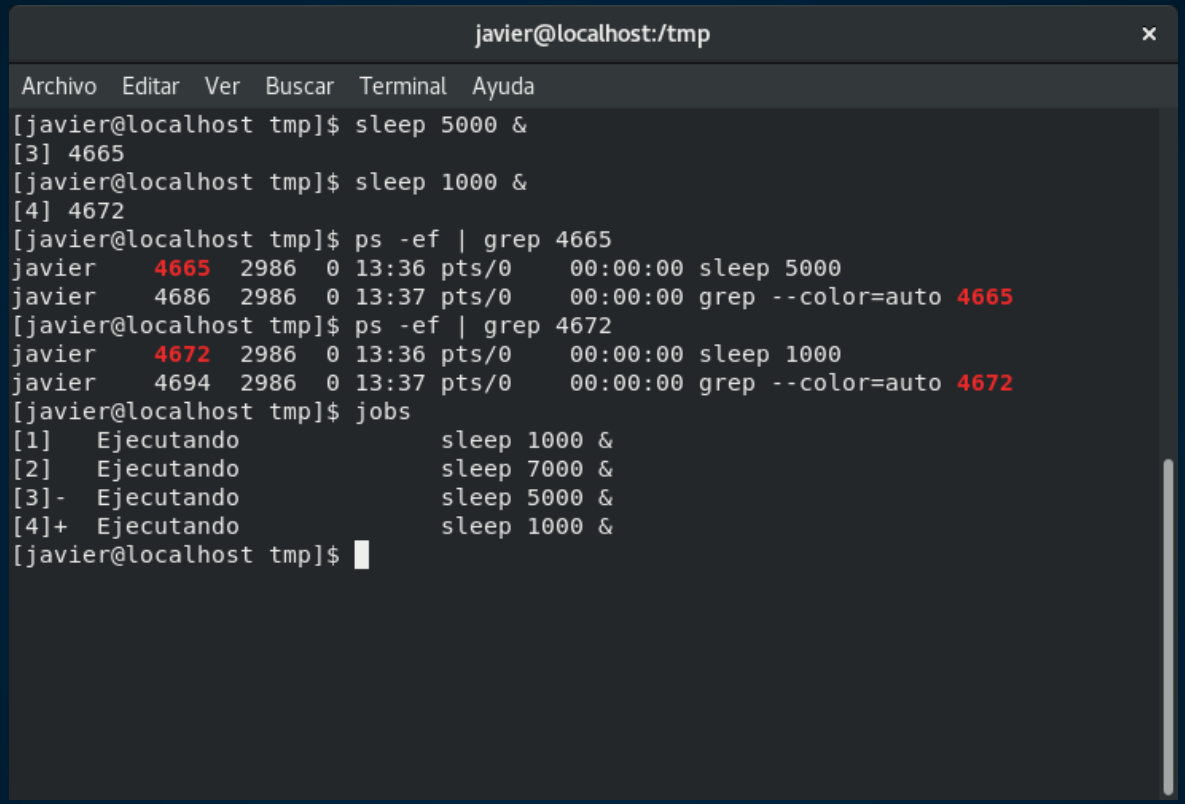
-l: lista os processos através do PID

-r: lista apenas os processos em execução

-s: lista apenas os processos parados

Se na linha de um processo aparecer o sinal positivo (+), significa que este é o processo mais recente a ser paralisado ou a estar em segundo plano. Se o sinal for negativo (-), o processo foi o penúltimo.

EXEMPLO



Note também que no início da linha um número é mostrado entre colchetes. Muitos confundem esse valor com o PID do processo, mas, na verdade, trata-se do número de ordem usado pelo jobs.

FG: Permite que um processo em segundo plano (background) ou parado passe para o primeiro plano (foreground). O processo deve ser identificado pelo seu PID ou pela ordem de entrada do processo no background (a gente ve usando o JOBS)

Sintaxe:

fg [PID]

Ou

Fg [%ordem]

Caso o comando seja digitado sem nenhum argumento, o sistema assume o ultimo processo a entrar em background será o primeiro a sair

BG: Move um processo que esta no primeiro plano para o segundo (background)

Para usar o bg, o processo em questão é paralisado, depois disso é só usar o JOBS para pegar o valor de ordem

Caso o numero do processo não seja fornecido, o sistema assume que o processo a ser colocado em background é o ultimo processo que foi suspenso pelo usuário

Sintaxe:

bg [%ordem]

A grande vantagem em se colocar um processo em *background* é deixar o terminal livre para a entrada de outros comandos, ou seja, o processo está sendo executado, mas ele não está alocado a um terminal.

FUSER: Mostra qual processo faz uso de determinado arquivo ou diretório

Entre as opções de combinações tem-se:

-k: mata(finaliza) o processo que utiliza o arquivo/diretório em questão;
-i: pede confirmação antes de matar um processo;
-u: mostra o proprietário do processo;
-v: informa cada processo listado, usuário que esta acessando , o PID do processo acessado, como o processo acessa o arquivo/diretório e o comando usado para acessar

A grande vantagem em se colocar um processo em *background* é deixar o terminal livre para a entrada de outros comandos, ou seja, o processo está sendo executado, mas ele não está alocado a um terminal.

TIPOS DE ACESSO

-c: diretório atual (a partir do qual o processo foi inicializado);

-e: arquivo sendo executado pelo processo;

-f: arquivo aberto;

-F: arquivo aberto para escrita;

-r: diretório raiz do sistema;

-m: arquivo mmap (mapeamento de memória) ou **biblioteca** compartilhada

└ Tem o conteúdo de um arquivo de memoria virtual, que, permite que um aplicativo modifique o arquivo ao ler e gravar diretamente na memoria

Exemplo:

Para exibir os processos (com o UID) que estão usando o shell *bash*, o editor de texto vim e o diretório */home/renatao*

```
fuser -u /bin/bash /usr/bin/vim /home/renatao
```

NOHUP: Possibilita um processo ficar ativo mesmo quando seu proprietário faz logout, ignorando os sinais de interrupção durante a execução do comando.

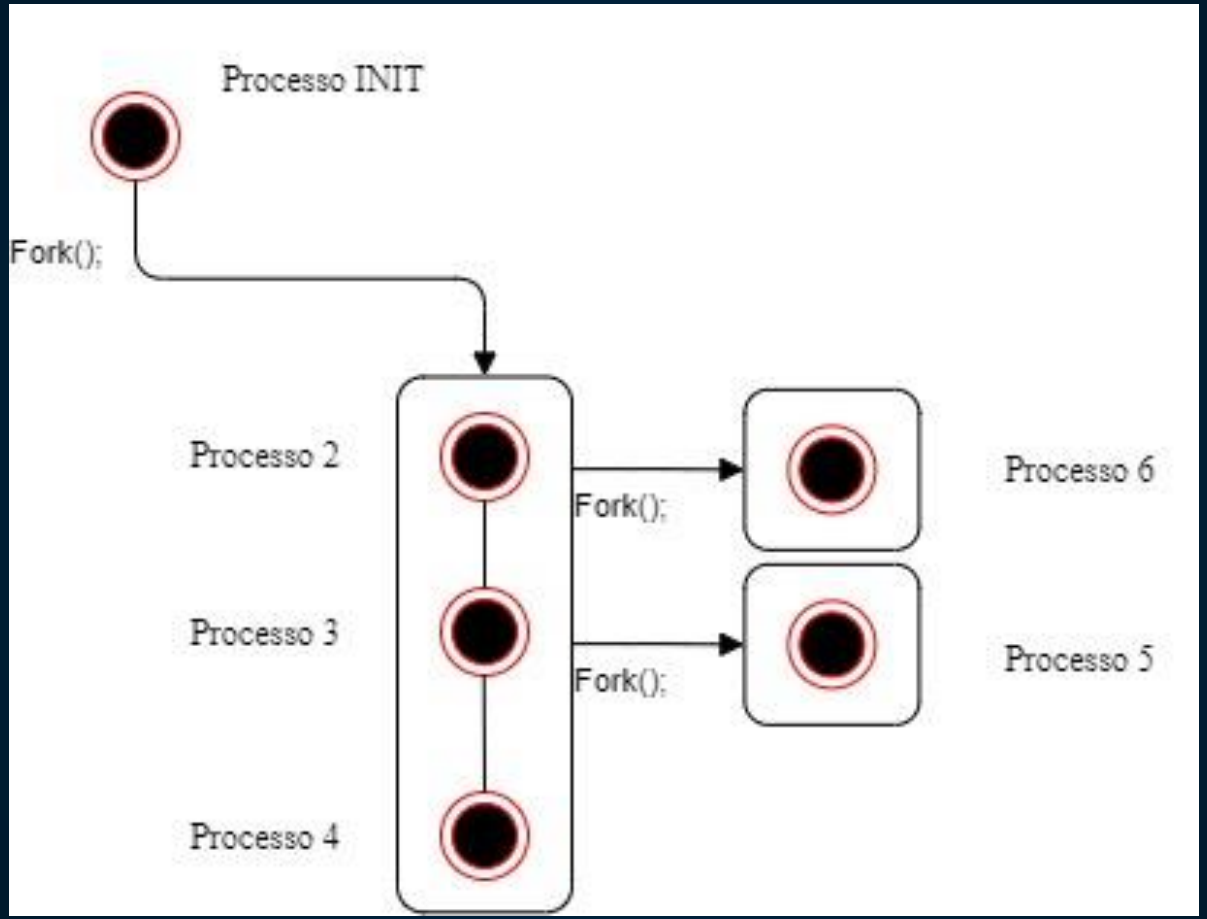
Sintaxe:

Nohup [opções] [comando]

Exemplo:

nohup find / -name gcc & → executa o comando find em background (&) sem permitir interrupções

Se usar o NOHUP e terminar o processo pai dele, o INIT assume a paternidade do processo


Processo INIT
Fork();
Processo 2
Processo 3
Processo 4
Fork();
Fork();
Processo 6
Processo 5

CONTROLE DE PROCESSOS EM LINUX

O que é um processo?
Processos em primeiro e segundo plano
Daemons

CONTROLE DE PROCESSOS EM LINUX

Daemons - é um tipo especial de processo cuja principal característica é ser carregado durante a inicialização do Linux ou rodar como um serviço. Existem daemons que podem ser encerrados e não voltam, existem os que podem ser encerrados e voltam automaticamente e os que não podem ser encerrados.

Exemplo: init(PID 1), que inicializa o sistema e é carregado pelo kernel e não pode ser encerrado.

COMANDOS IMPORTANTES PARA O GERENCIAMENTO DE PROCESSOS NO LINUX	
CHAMADA	**FUNÇÃO**
ps	**Mostra os processos**
pidof	**Mostra o id do processo**
pstree	**Mostra a árvore de processos (dependentes, filhos, etc.)**
top	**Monitora em tempo real os processo**
Ctrl-Z	**Envia o TSTP sinal para o processo. Isso interrompe a execução.**
bg	**Faz com que o processo que está sendo executado em primeiro plano, passe a ser executado em segundo plano**
Fg	**Faz com que o processo que está sendo executado em segundo plano, passe a ser executado em primeiro plano.**
Who/w	**Imprime informações sobre todos os usuários que estão conectados no momento.**
Kill	**Encerra o processo**
Killall	**Encerra todos os processos relacionados a um programa ou comando**
renice	**Altera o nível de prioridade de processo**

PTRACE

- **ptrace** (Process trace) é uma chamada de sistema que permite a um processo controlar a execução de outro.

- Um processo pode iniciar um rastreamento chamando fork e fazendo com que o filho resultante faça um **PTRACE_TRACEME** , seguido (normalmente) por um execl. Alternativamente, um processo pode começar a rastrear outro processo usando **PTRACE_ATTACH** ou **PTRACE_SEIZE** .

- Quando isso ocorre, o processo entra no estado interrompido e informa o processo de rastreamento por uma chamada **wait()**.

- Após o rastreamento, o processo rastreador pode matar o rastreado ou sair para continuar com sua execução.

PROTÓTIPO DO PTRACE():

```
#include <sys/ptrace.h>
long int ptrace(enum_ptrace_request request, pid_t_pid, void * addr, void * data)
```

São os quatro argumentos:

- O valor do pedido decide o que fazer;
- PID é o ID do processo que vai ser rastreado;
- ADDR é o deslocamento no espaço do usuário do processo rastreado para onde os dados são gravados quando instruído a fazê-lo. É o deslocamento no espaço do usuário do processo rastreado de onde uma palavra é lida e retornada como resultado da chamada.

CHAMADAS DO PTRACE():

PTRACE_TRACEME: Chamado quando o filho deve ser rastreado pelo pai.

PTRACE_ATTACH: Um processo deseja controlar outro.

PTRACE_DETACH: Para de rastrear um processo.

PTRACE_SYSCALL, PTRACE_CONT: Ambos ativam o processo interrompido.
PTRACE_SYSCALL faz com que o filho pare após a próxima chamada do sistema.
PTRACE_CONT apenas permite que a criança continue.

PTRACE_SINGLESTEP: Faz o mesmo que PTRACE_SYSCALL, exceto que o filho é parado após cada instrução.

VALORES DE RETORNO DO PTRACE()

0: ptrace bem sucedido
-1: pode ser um dos possíveis erros:

1 - EPERM: O processo solicitado não pôde ser rastreado. Permissão negada.
2 - ESRCH: O processo solicitado não existe ou está sendo rastreado.
3 - EIO: a solicitação era inválida ou leitura / gravação foi feita de / para área inválida da memória.
4 - EFAULT: A leitura / gravação foi feita de / para a memória que não foi realmente mapeada.

SINCRONIZAÇÃO DE PROCESSOS

INTRODUÇÃO

- **É normal que os processos de uma aplicação concorrente compartilhem recursos do sistema, como arquivos, dispositivos de entrada e saída, e áreas de memória.**

- **O compartilhamento de recursos entre processos pode gerar situações indesejáveis.**

- **Para evitar esse tipo de situação, os processos concorrentes devem ter suas ações sincronizadas, a partir de mecanismos oferecidos pelo S.O, para garantir o processamento correto dos programas.**

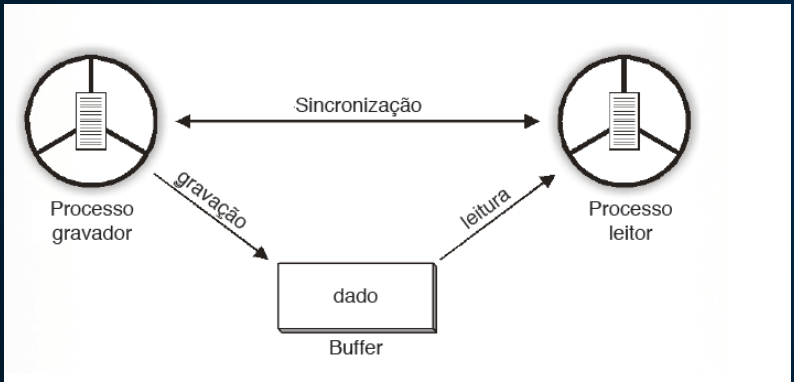
APLICAÇÕES CONCORRENTES

- **Na maioria das vezes, é necessário que os processos se comuniquem;**

- **Essa comunicação deve ser implementada por meio de diversos mecanismos, como variáveis compartilhadas na memória principal ou trocas de mensagens;**

- **Assim, é necessário que os processos concorrente tenha sua execução sincronizada, através de mecanismos do sistema operacional.**

- **Exemplo:**



Os mecanismos que garantem a comunicação entre os processos concorrentes e o acesso a recursos compartilhados são chamados de mecanismos de sincronização.

Exemplos de mecanismos de sincronização:

Semáforos (Dijkstra)
Monitores (Brinch-Hansen)
Passagem de mensagens

TERMINO DE UM PROCESSO

Um processo pode ser terminado pelas seguintes condições:

- **Saída normal (voluntário)**
- **Saída por erro (voluntário)**
- **Erro fatal (involuntário)**
- **Cancelamento por um outro processo (involuntário)**

THREADS

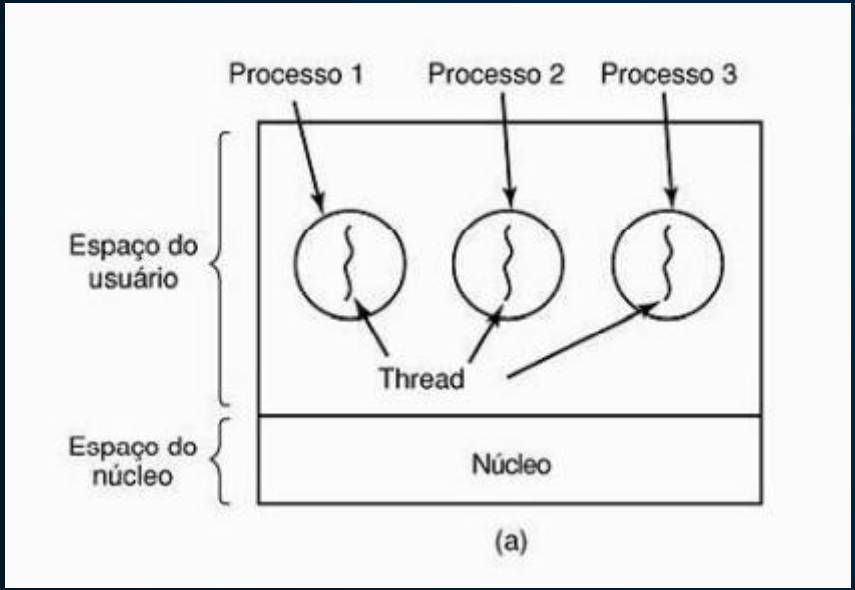
- **Conceito/Uso de Threads**
- **Múltiplas atividades**
- **Espaço de Endereçamento**
- **Dinâmico e rápido**
- **Desempenho**

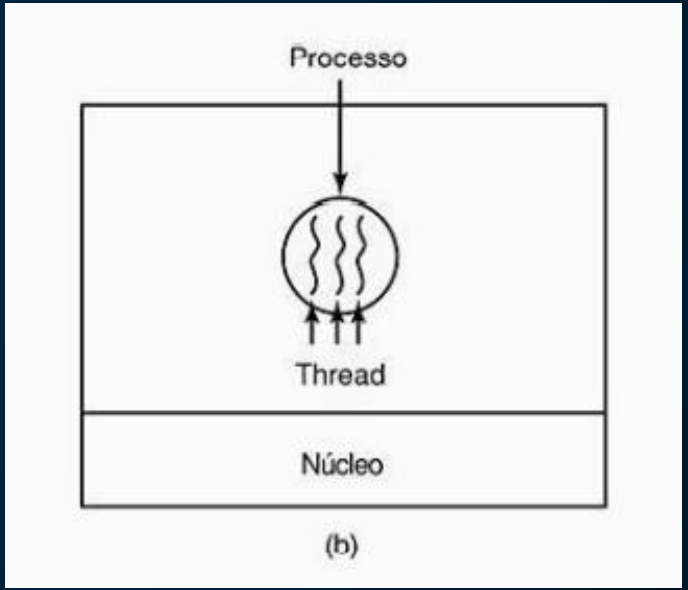
THREADS

Itens por processo	Itens por thread
Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	
Sinais e manipuladores de sinais	
Informação de contabilidade	

THREADS

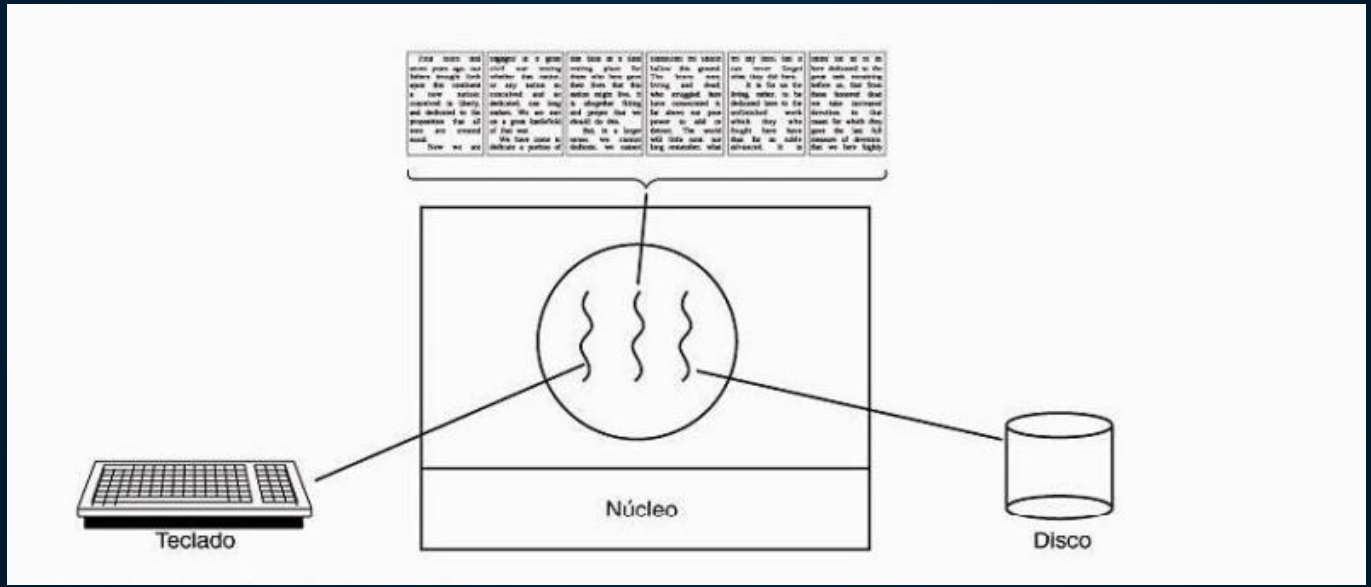
-Desempenho (single-core vs multi-core)





THREADS

-Exemplos: Editor de texto



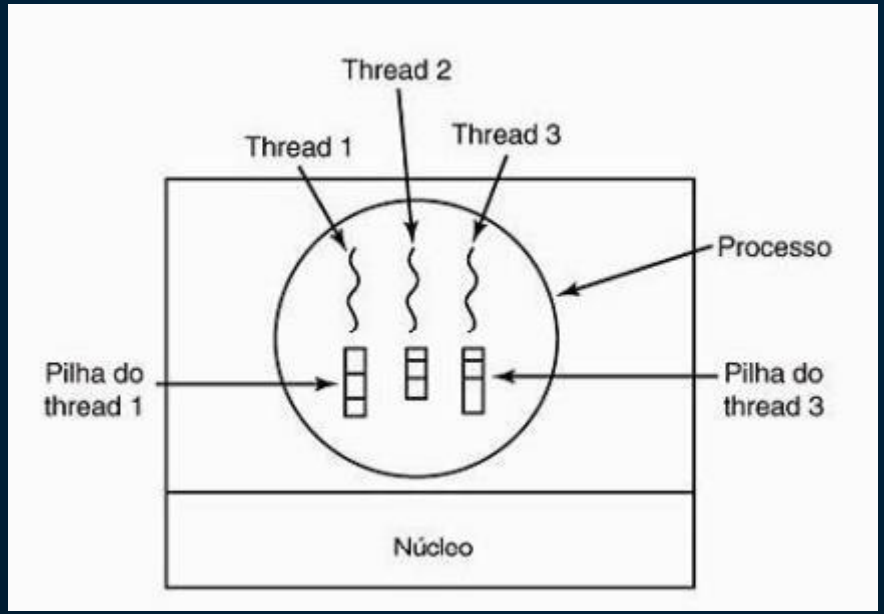
THREADS

-Exemplos: Editor de texto

Modelo	Características
Threads	Paralelismo, chamadas de sistema bloqueante
Processo monothread	Não paralelismo, chamadas de sistema bloqueantes
Máquina de estados finitos	Paralelismo, chamadas não-bloqueantes, interrupções

THREADS

-Modelo Clássico de Threads



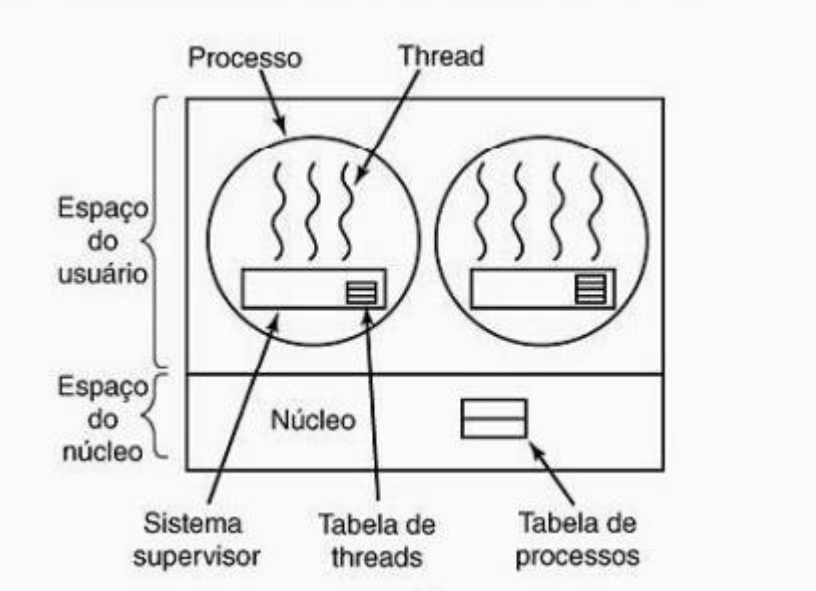
THREADS

- **Threads Posix**

Chamada de thread	Descrição
pthread_create	Cria um novo thread
pthread_exit	Conclui a chamada de thread
pthread_join	Espera que um thread específico seja abandonado
pthread_yield	Libera a CPU para que outro thread seja executado
pthread_attr_init	Cria e inicializa uma estrutura de atributos do thread
pthread_attr_destroy	Remove uma estrutura de atributos do thread

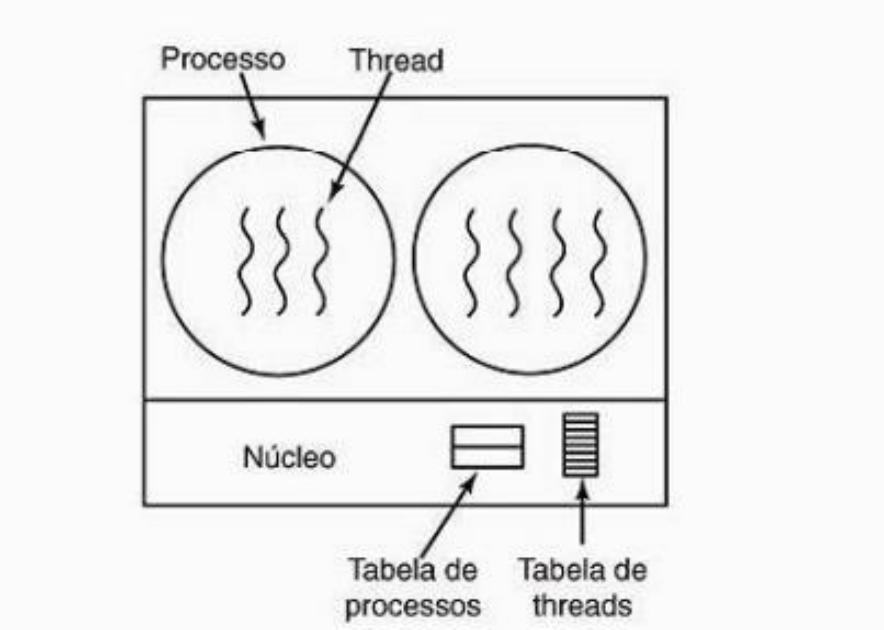
THREADS

-Implementação no espaço do usuário



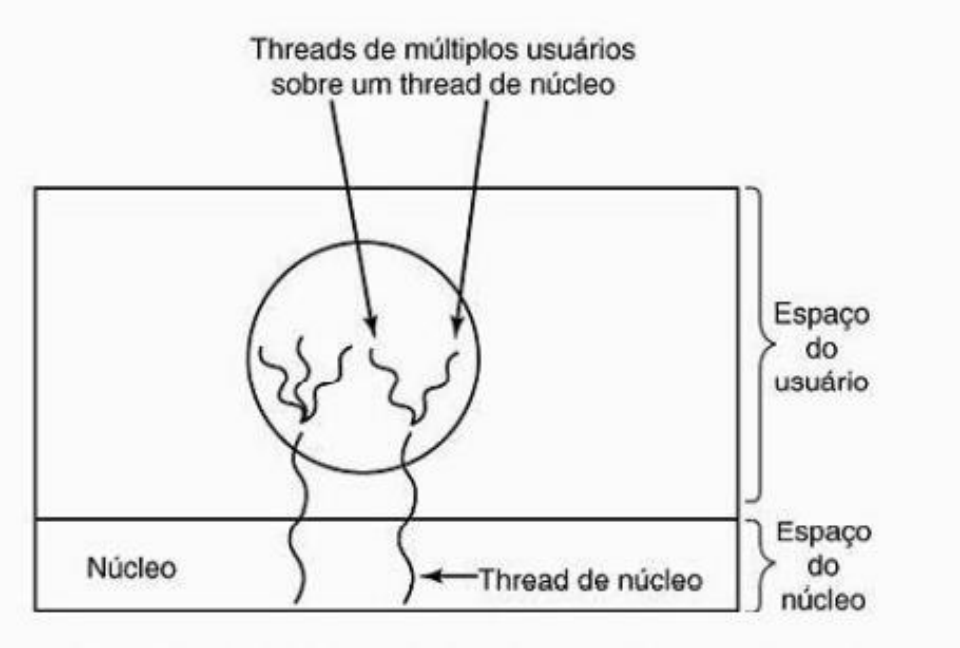
THREADS

-Implementação no espaço do núcleo



THREADS

-Implementações híbridas



THREADS

-Programa de Threads

THREADS

-Programa de Threads