



# Processos e Threads

Alberto Felipe Friderichs Barros  
Bruno Costa de Bem

## Agenda

- Estrutura de Sistemas Operacionais;
- Chamadas de Sistemas e Interrupções;
- Processos e Threads.



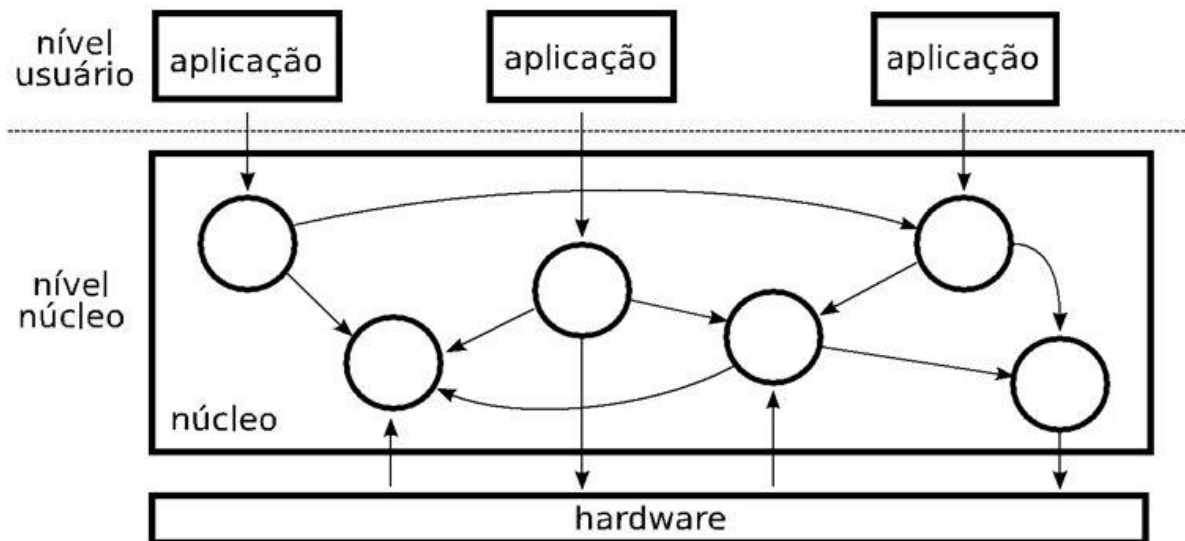
## Estrutura do S.O

SOs são normalmente grandes e complexas coleções de rotinas de software. Projetistas devem dar grande ênfase à sua organização interna e estrutura.

- 1) Monolítico
- 2) Microkernel
- 3) Kernel Híbrido
- 4) Camadas
- 5) Máquina Virtual

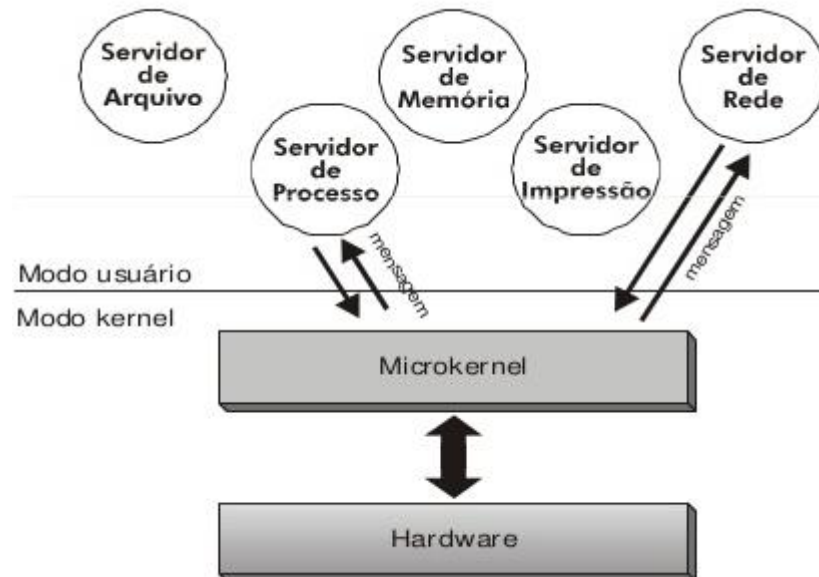
## Monolítico

S.O é um único módulo, consiste de um conjunto de programas que executam sobre o hardware, como se fosse um único programa. Os programas dos usuários invocam rotinas do S.O. Ex: **Linux, Android, Windows, Unix, MS-DOS**.



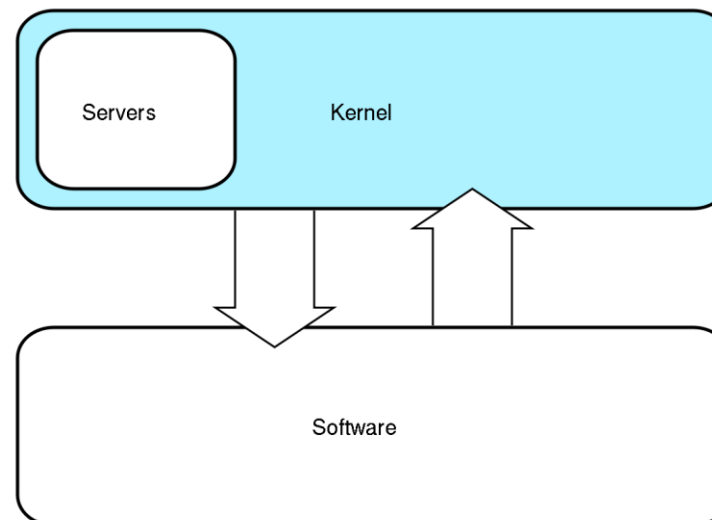
## Microkernel

Busca tornar o núcleo do S.O o menor possível, sendo a principal função do núcleo, gerenciar a comunicação entre esses processos. Apenas o Kernel responsável pela comunicação entre clientes e servidores, executa no modo kernel. Exemplos: **Hurd**, **Minix**, **Symbian**.



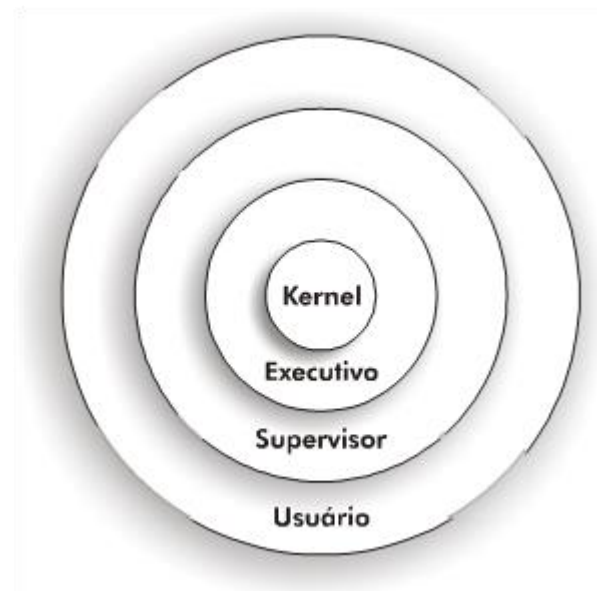
## Kernel Híbrido

Kernel híbrido são um acordo entre o desenvolvimento de microkernel e kernel monolíticos. Isto implica executar alguns serviços (como a pilha de rede ou o sistema de arquivos por exemplo) no espaço do núcleo para reduzir o impacto na performance. Exemplos: **BeOS, MacOS X. Windows NT, Windows Server, etc.**



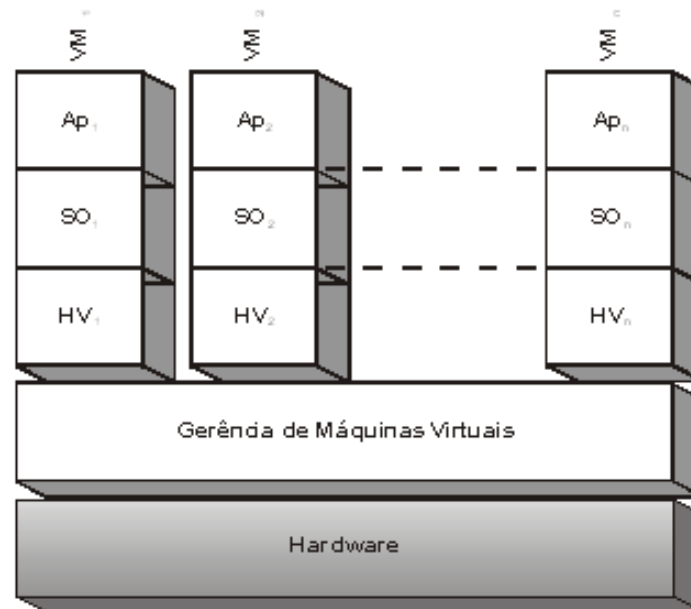
## Camadas

A ideia é criar um sistema modular e hierárquico. Na estrutura de camadas o sistema é dividido em níveis sobrepostos, cada camada oferece um conjunto de funções que podem ser utilizadas pelas camadas superiores. Exemplo: **OpenVMS e Multics**.



## Arquitetura de Máquina Virtual

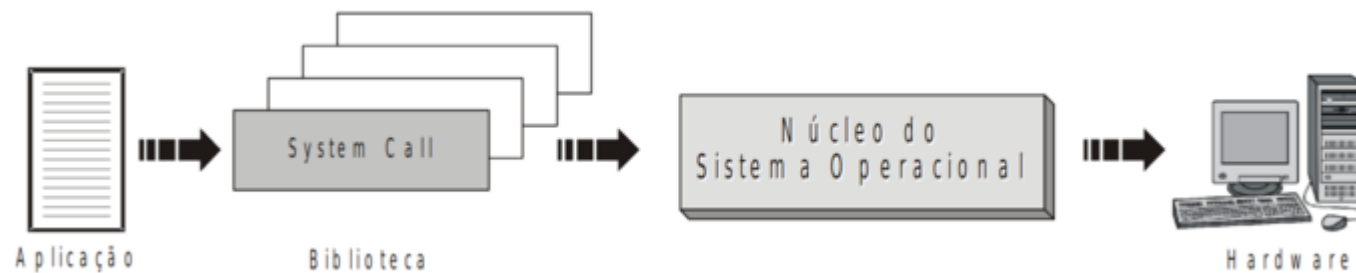
O Modelo de Máquina Virtual (VM) cria um nível intermediário entre o hardware e o sistema operacional, denominado gerenciador de máquinas virtuais. Ex: **IBM360/CMS**.



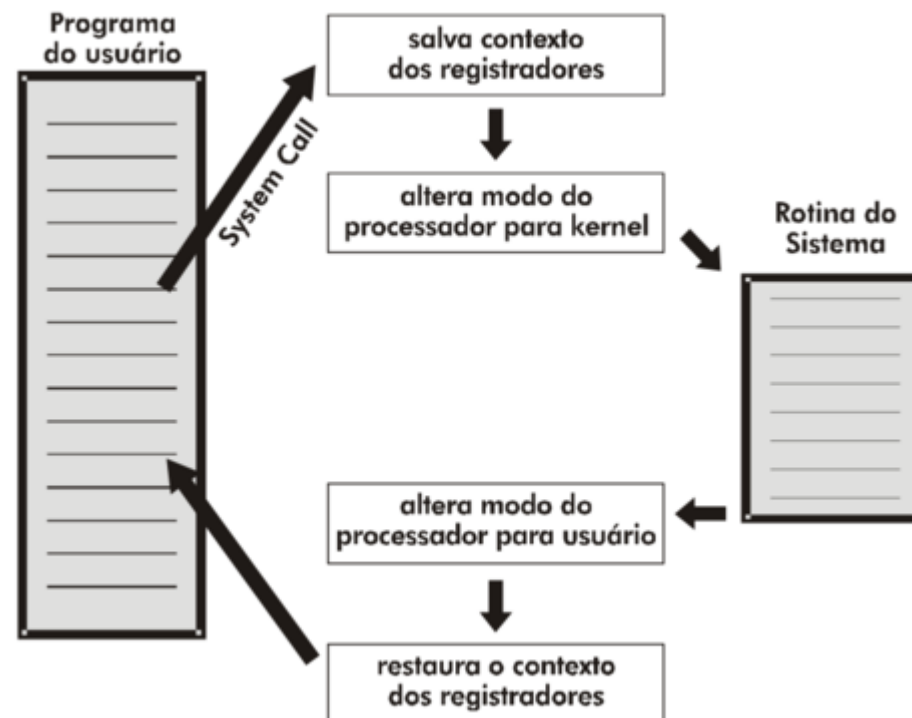


## Chamadas de Sistemas

Se uma instrução precisa realizar alguma instrução privilegiada (**imprimir um arquivo**), ela realiza uma chamada de sistema, que altera do modo usuário para o modo kernel. Ex: ler um arquivo (open), imprimir (printf), criar um processo (fork), etc.

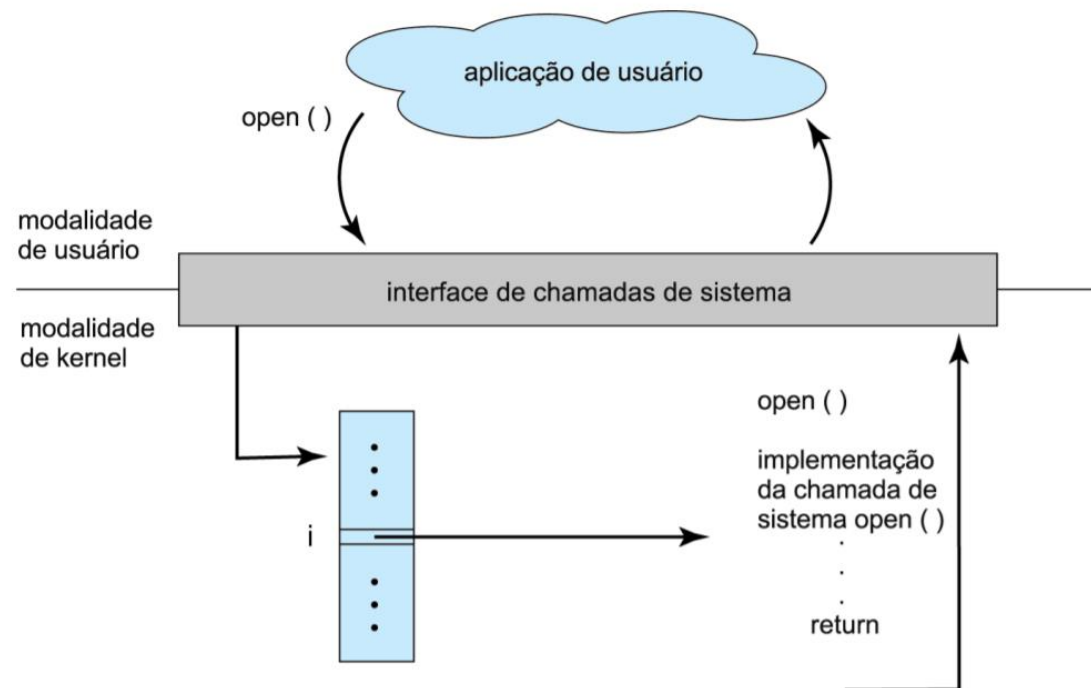


## Passos para chamada de Sistema



## Exemplo com chamada open ( )

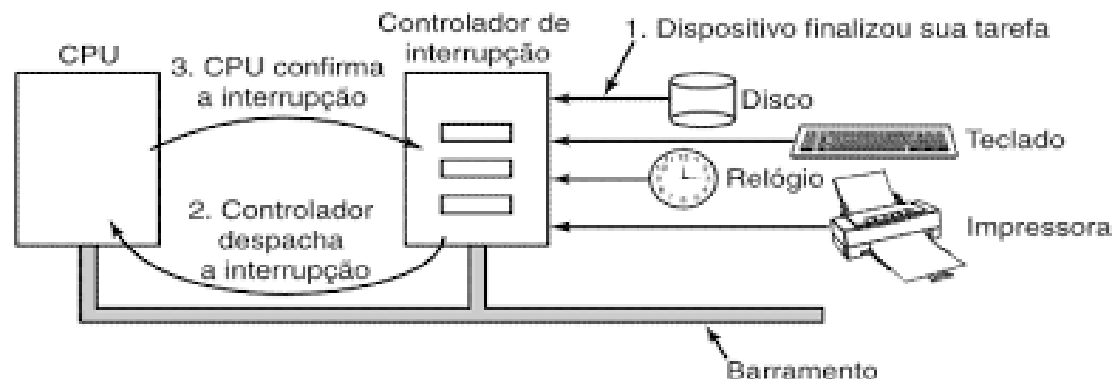
Normalmente as aplicações utilizam uma *Application Program Interface (API)*. Interface para esconder a complexidade das syscalls. Geralmente escrita em C, C++, ou java.



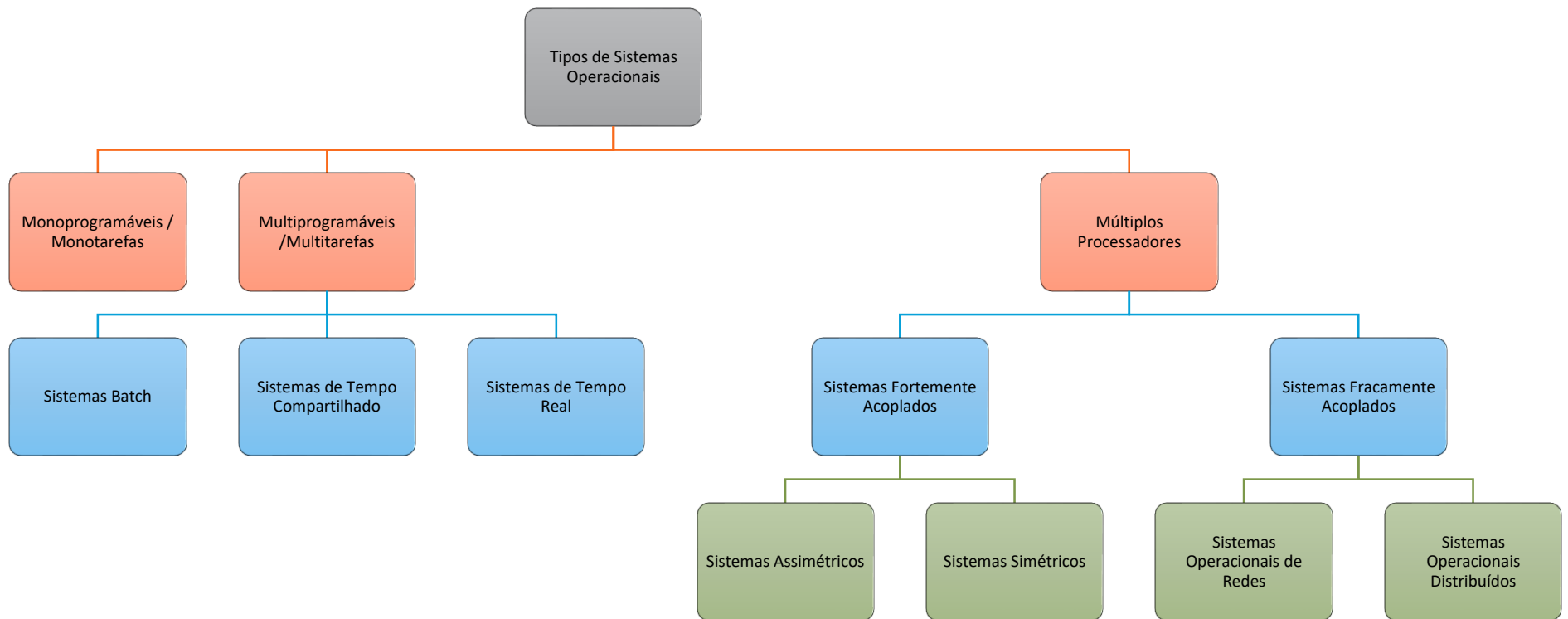
## Interrupções

Evento externo ao processador, gerado por dispositivos que precisam da atenção do S.O, é uma chamada gerada pelo hardware podendo ser:

- 1) Interrupções de hardware (evento externo);
- 2) Um sinal elétrico no hardware; e
- 3) Dispositivos de E/S ou o clock.



## Classificação dos Sistemas



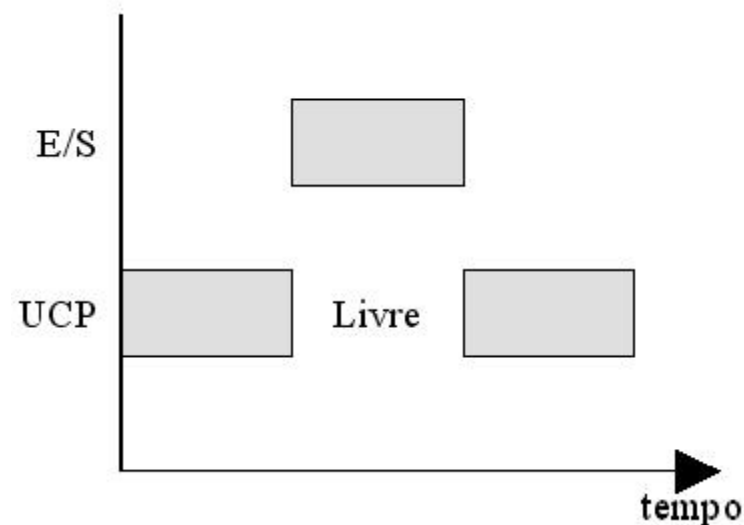
## S.O Multiprogramáveis ou Multitarefa

Os primeiros computadores permitiam que apenas um programa fosse executado de cada vez. Esse programa tinha controle total sobre o sistema e acesso a todos os seus recursos. Por outro lado, os sistemas de computação contemporâneos permitem que vários programas sejam carregados na memória e executados concorrentemente.



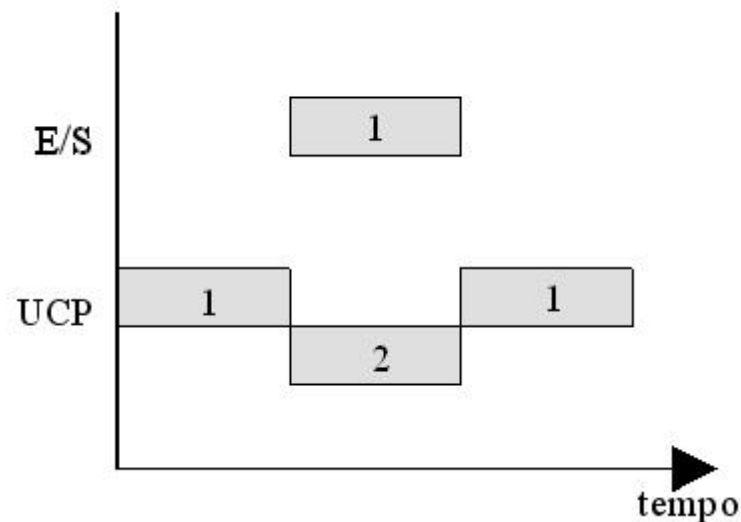
## S.Os Monoprogramáveis ou Monotarefas

- 1) Caracterizam-se por permitir que o processador a memória e os periféricos permaneçam **exclusivamente dedicados à execução de um único programa.**
- 2) Recursos são **mal** utilizados, entretanto, são implementados com facilidade.  
**Ex: MS-DOS.**



## S.O Multiprogramáveis ou Multitarefas

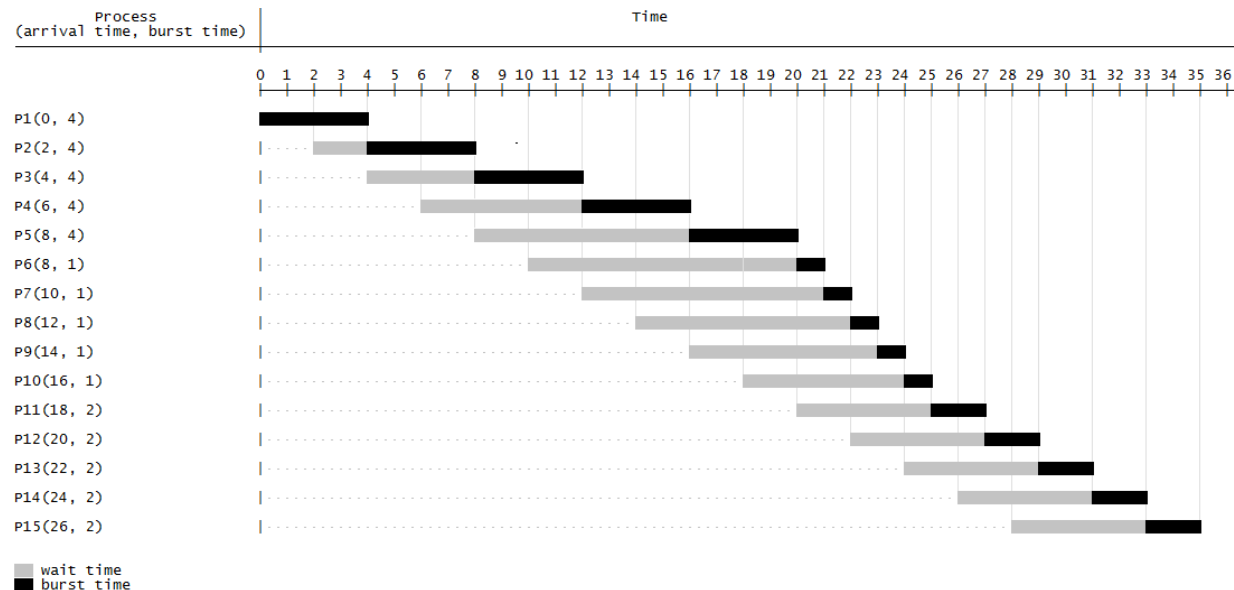
- 1) A ideia é manter **vários programas** na memória ao mesmo tempo.
- 2) Várias tarefas simultâneas em um único processador: **enquanto uma espera a outra roda**, demandando mecanismos de **trocas rápidas entre os processos**. **Ex:** Windows, MacOS...





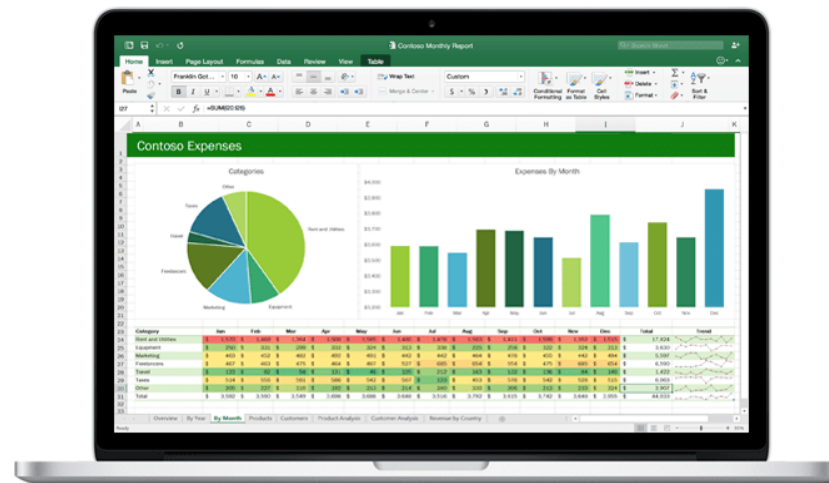
## Pseudo-Paralelismo / Concorrência

Para possibilitar essa "experiência multitarefa", o que ocorre é que a CPU é compartilhada entre os processos **que estão na memória aguardando pela CPU**. Funciona da seguinte forma: **A CPU alterna na execução desses processos** fazendo parecer que os vários processos executam juntos, mas o que houve foi um compartilhamento do tempo de CPU. Essa técnica é conhecida como **time-sharing**.



## Processo

Essas necessidades resultaram na noção de **processo, que é um programa em execução**. Um processo é a unidade de trabalho em um sistema moderno de tempo compartilhado (time share). Um processo é um programa, que contém várias instruções, em execução. Podendo ser entendido também como uma instância de um programa e possui dados de entrada e saída e um estado (**executando, bloqueado e pronto**).



## Processo

**Um programa torna-se um processo quando um arquivo executável é carregado na memória.** Duas técnicas comuns para a carga de arquivos executáveis são clicar duas vezes em um ícone representando o arquivo executável ou dar entrada no nome do arquivo executável via interface de linha de comando (CLI).



→  
Execução de programa



## Processo de Primeiro Plano

- 1) Interage com o usuário
- 2) Exemplos: Ler um arquivo, iniciar um programa. (linha de comando ou duplo clique do mouse)



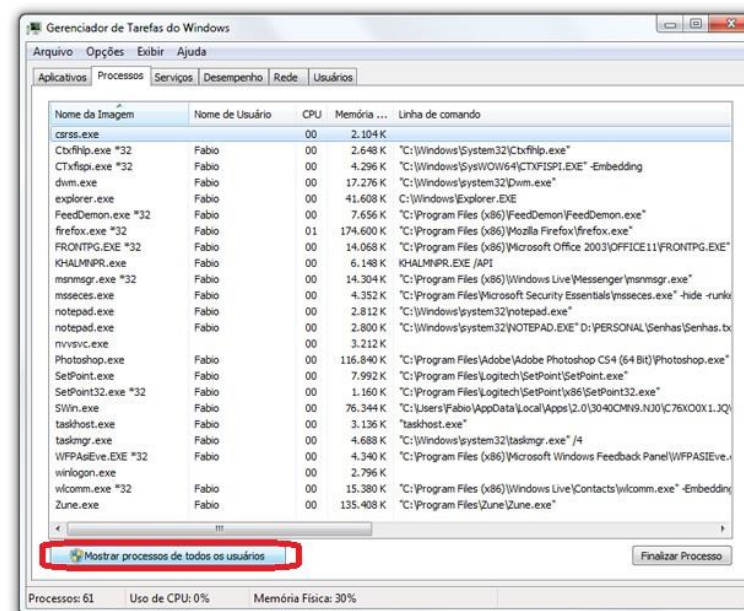
## Processo de Segundo Plano (Background)

- 1) Processos com funções específicas que independem do usuário (daemons);
- 2) Exemplos: serviços de recebimento de e-mails, serviços de impressão, etc.



## Processo

Portanto, um sistema é composto por um conjunto de processos: **processos do sistema operacional executando código de sistema (Daemons) e processos de usuário executando código de usuário (Programas)**. Potencialmente, todos esses processos podem ser executados concorrentemente, com a CPU ou CPUs alternando entre eles.



## Processo x Programa

**Enfatizamos que um programa por si só não é um processo.**

- Um programa é uma entidade passiva, como um arquivo contendo uma lista de instruções armazenadas em disco, geralmente chamado de arquivo executável;
- Algoritmo codificado;
- Forma como o programador vê a tarefa a ser executada.
- Um processo é único;
- Código acompanhado de dados e estado;
- Forma pela qual um S.O vê um programa e possibilita sua execução.

## Cada Processo Possui

- 1) Conjunto de Instruções;
- 2) **Espaço de Endereçamento** (espaço reservado para que o processo possa ler e escrever);
- 3) Contexto de Hardware (registradores, program counter, ponteiros...); e
- 4) Contexto de Software (atributos em geral como variáveis, lista de arquivos abertos, etc)



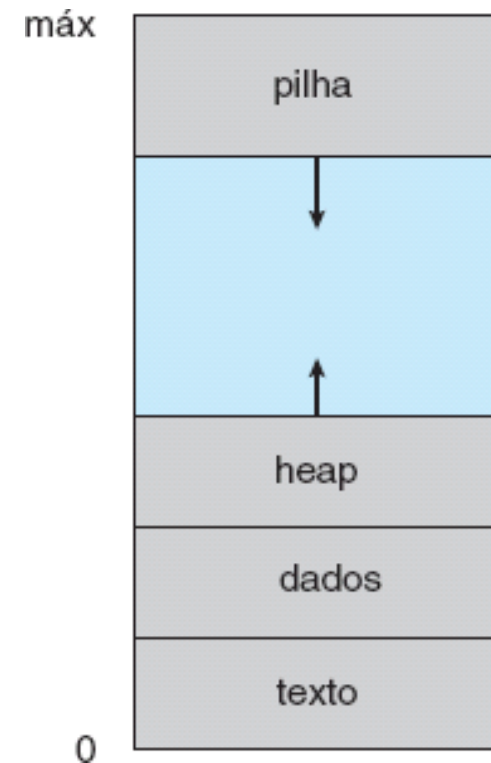
## Espaço de Endereçamento

Um processo é mais do que o código executável do programa, que também é conhecido como **texto**.

Geralmente, um processo também inclui a **pilha** que contém dados temporários como parâmetros de funções, endereços de retorno e variáveis locais.

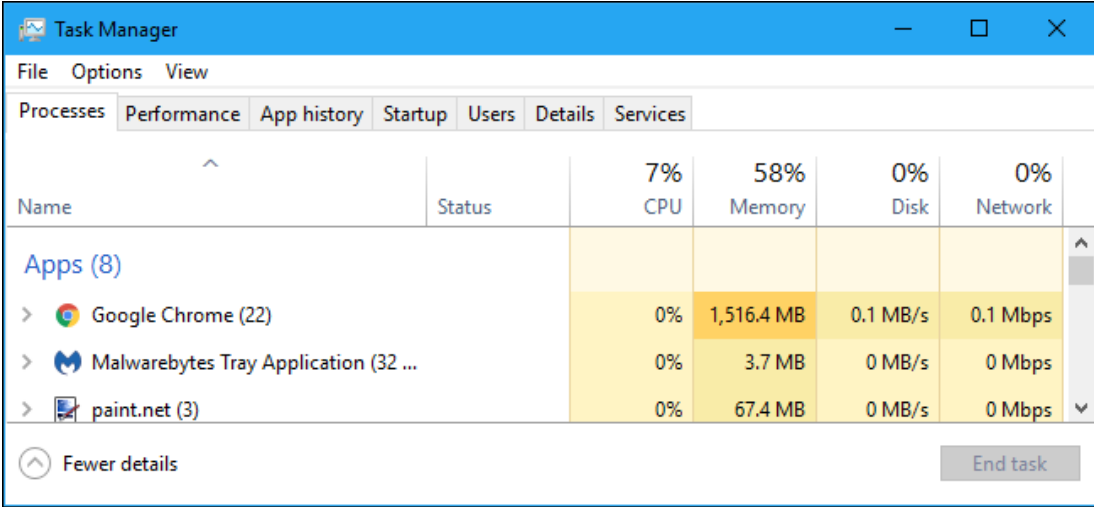
Há também uma seção de **dados**, que contém as variáveis globais.

Um processo também pode incluir um **heap**, que é a memória dinamicamente alocada durante o tempo de execução do processo.



## Multiprogramação

As instruções que executam na CPU devem passar obrigatoriamente pela memória RAM, sendo assim, **um processo para ser executado precisa estar na RAM**. Em um sistema multiprogramado, a memória poderá ser ocupada por vários processos.



The screenshot shows the Windows Task Manager window with the 'Performance' tab selected. The window title is 'Task Manager'. The menu bar includes 'File', 'Options', and 'View'. The tabs at the top are 'Processes', 'Performance', 'App history', 'Startup', 'Users', 'Details', and 'Services'. The 'Performance' tab displays a table with columns: 'Name', 'Status', '7% CPU', '58% Memory', '0% Disk', and '0% Network'. Below the table, there is a section for 'Apps (8)' with a list of applications: 'Google Chrome (22)', 'Malwarebytes Tray Application (32 ...)', and 'paint.net (3)'. Each application row shows its CPU, Memory, Disk, and Network usage. At the bottom left, there is a 'Fewer details' button, and at the bottom right, there is an 'End task' button.

Name	Status	7% CPU	58% Memory	0% Disk	0% Network
Apps (8)					
> Google Chrome (22)		0%	1,516.4 MB	0.1 MB/s	0.1 Mbps
> Malwarebytes Tray Application (32 ...)		0%	3.7 MB	0 MB/s	0 Mbps
> paint.net (3)		0%	67.4 MB	0 MB/s	0 Mbps

## Estados de um Processo

Quando um processo é executado, ele muda de estado. O estado de um processo é definido, em parte, pela atividade corrente do processo. Um processo pode estar em um dos seguintes estados:

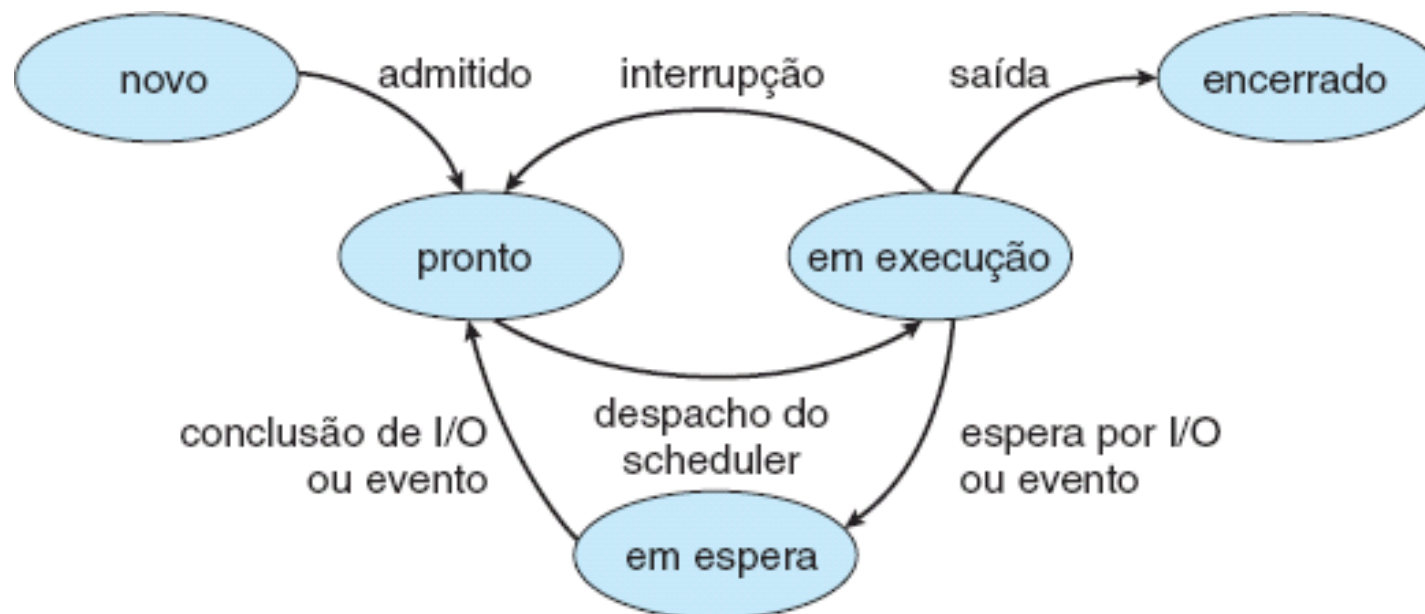
**Executando** - Instruções estão sendo executadas na CPU.

**Bloqueado** - O processo está esperando que algum evento ocorra.

**Pronto** - O processo está esperando que seja atribuído a um processador.



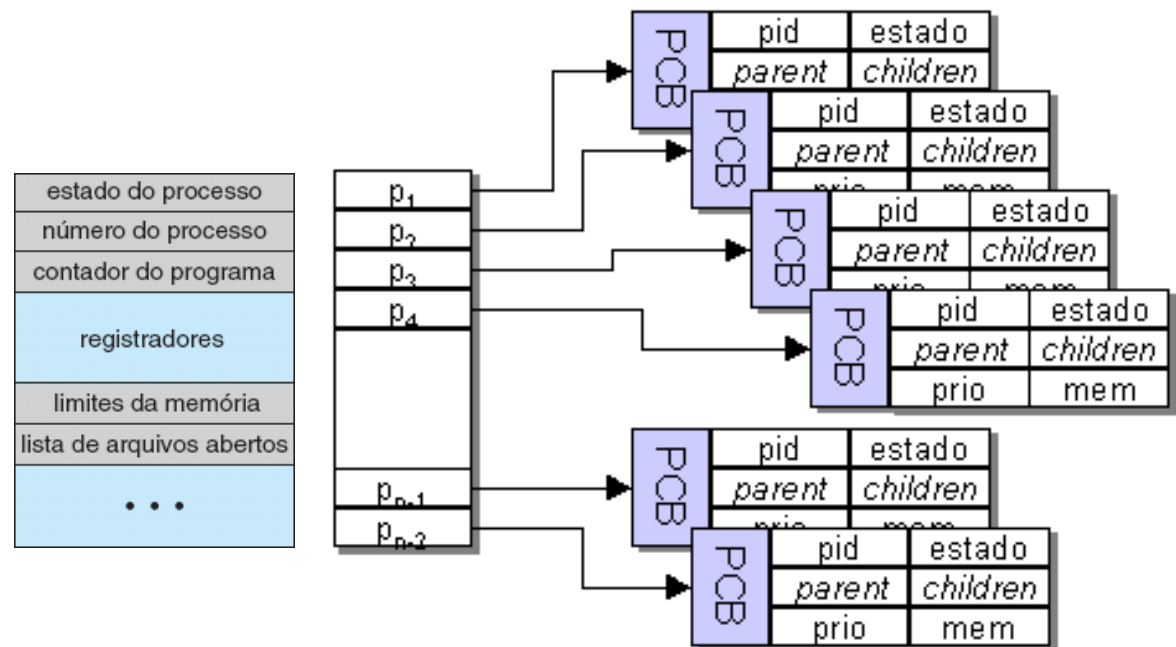
## Diagrama de Estados



## Tabela de Processos

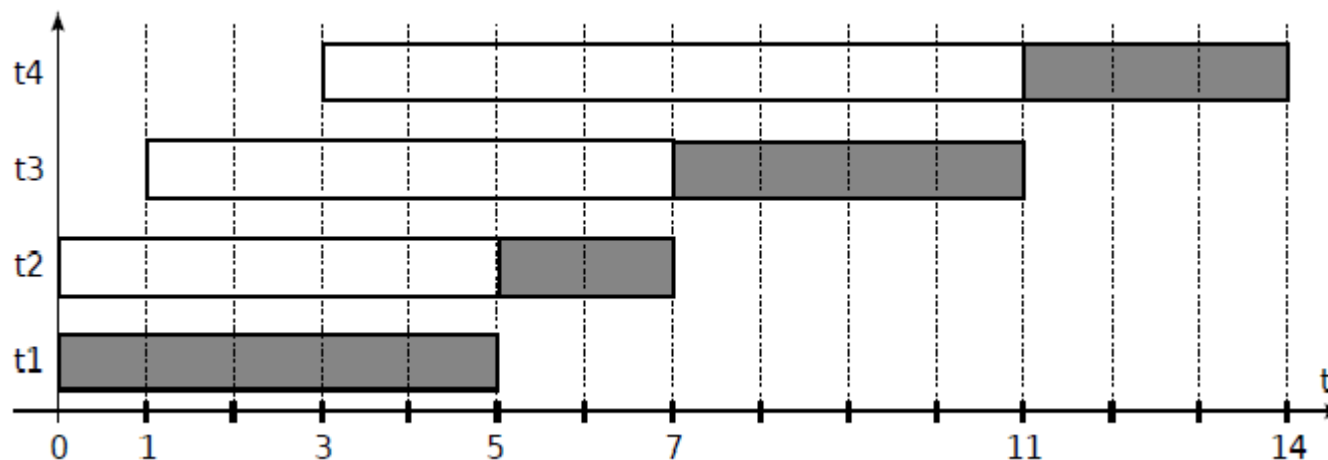
Cada processo é representado, no sistema operacional, por um **bloco de controle de processo (PCB)** Que contém muitas informações associadas a um processo específico, incluindo :

- Estado do processo.
- Contador do programa.
- Registradores da CPU.
- Informações de scheduling
- Informações de memória.
- Informações de contabilização.
- Informações de status de I/O.



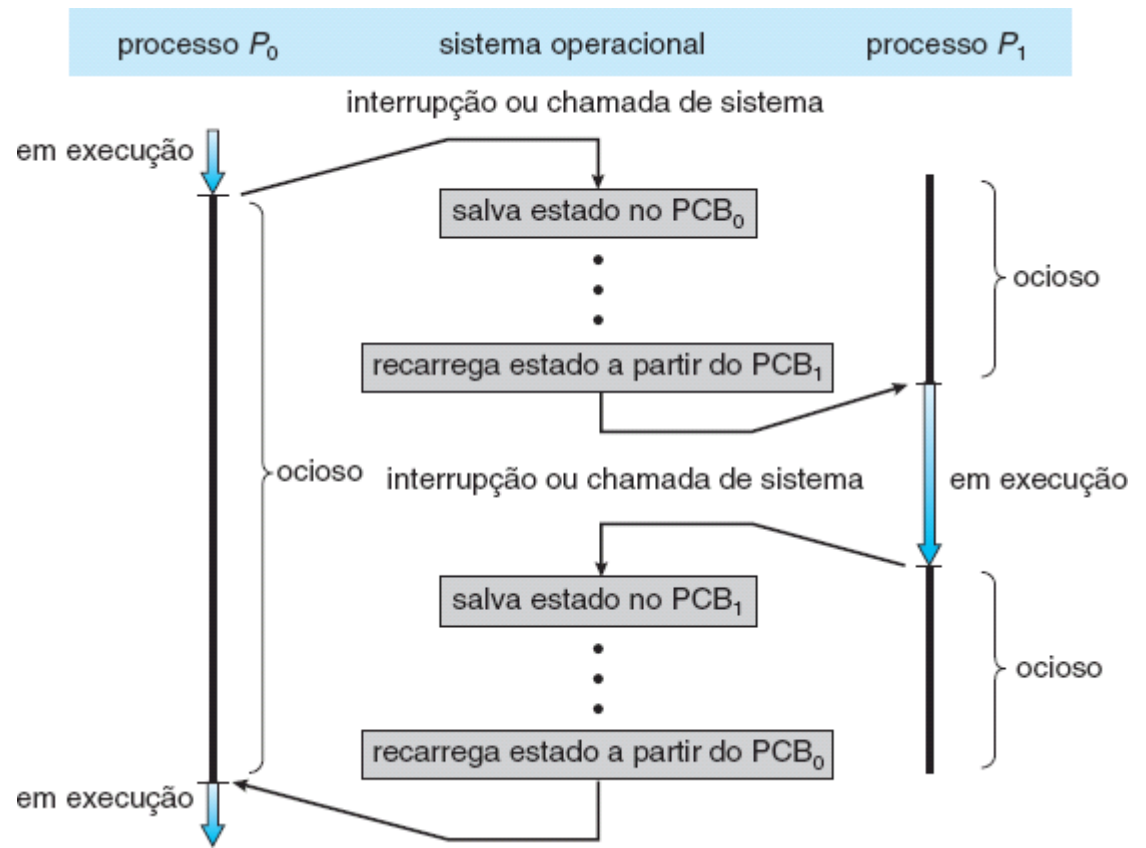
## Escalonamento de Processos

O objetivo da multiprogramação é haver sempre algum processo em execução para maximizar a utilização da CPU. O objetivo do compartilhamento de tempo é alternar a CPU entre os processos, com tanta frequência, que os usuários possam interagir com cada programa enquanto ele está sendo executado. Para alcançar esses objetivos, o **escalonador de processos** seleciona um processo disponível (possivelmente em um conjunto de vários processos disponíveis) para execução na CPU.



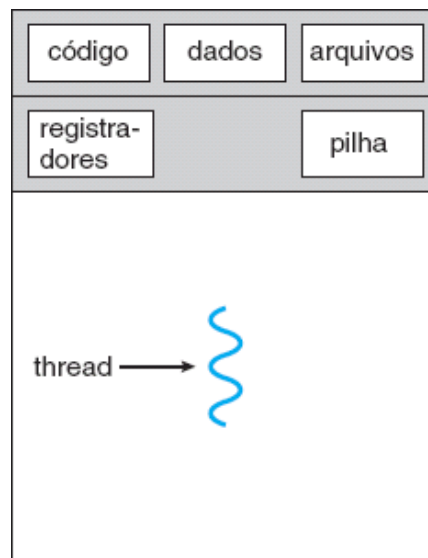
## Escalonamento de Processos

Quando os processos entram no sistema, eles são inseridos em uma **fila de jobs** que é composta por todos os processos no sistema. Os processos que estão residindo na memória principal e estão prontos e esperando execução são mantidos em uma lista chamada **fila de prontos**. Essa fila é, em geral, armazenada como uma lista encadeada. O cabeçalho de uma fila de prontos contém ponteiros para o primeiro e o último PCBs da lista. Cada um dos PCBs inclui um campo de ponteiro para o próximo PCB da fila de prontos.

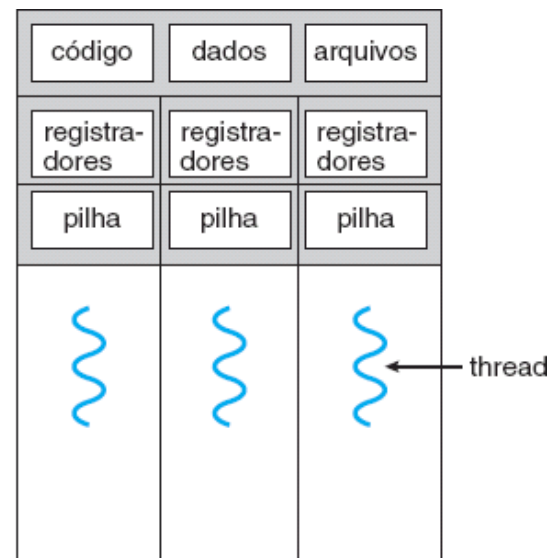


## Thread

O modelo de processo supõe que um processo é um programa em execução com **um único thread de controle**. Praticamente todos os sistemas operacionais modernos, no entanto, fornecem recursos que habilitam um processo a conter múltiplos threads de controle. **Se um processo tem múltiplos threads, ele pode executar mais de uma tarefa ao mesmo tempo.**



processo com um único thread

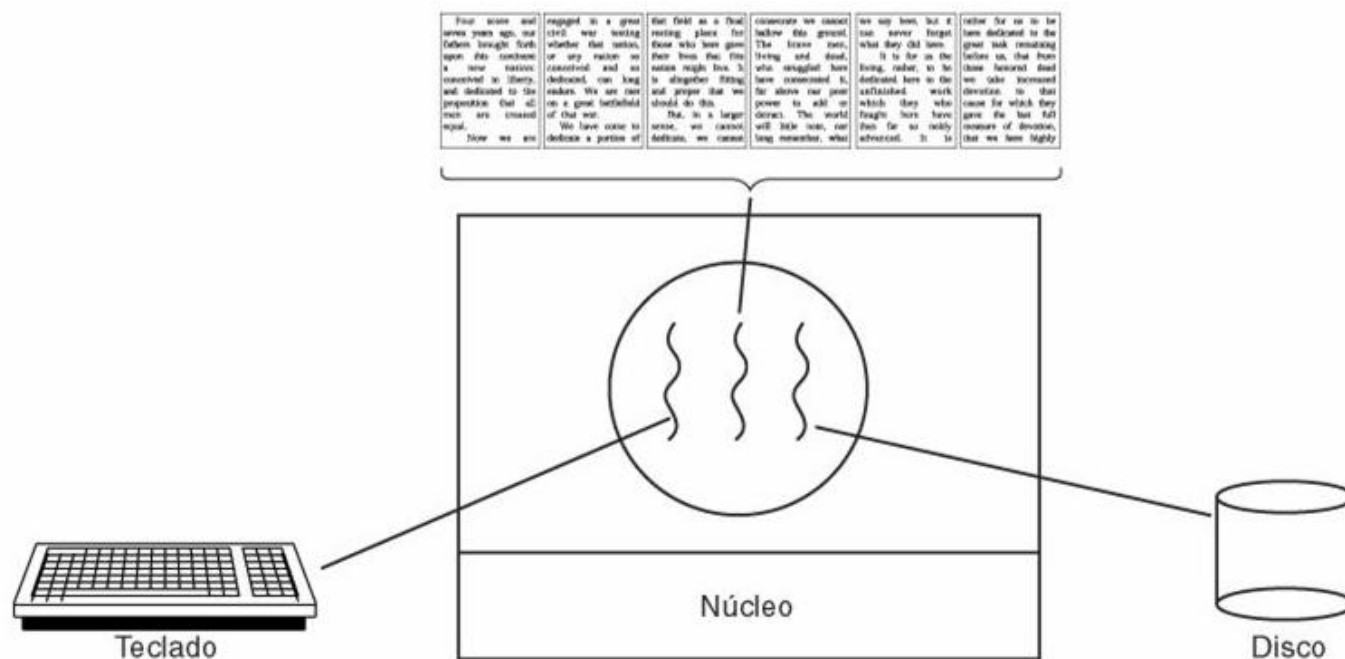


processo com múltiplos threads



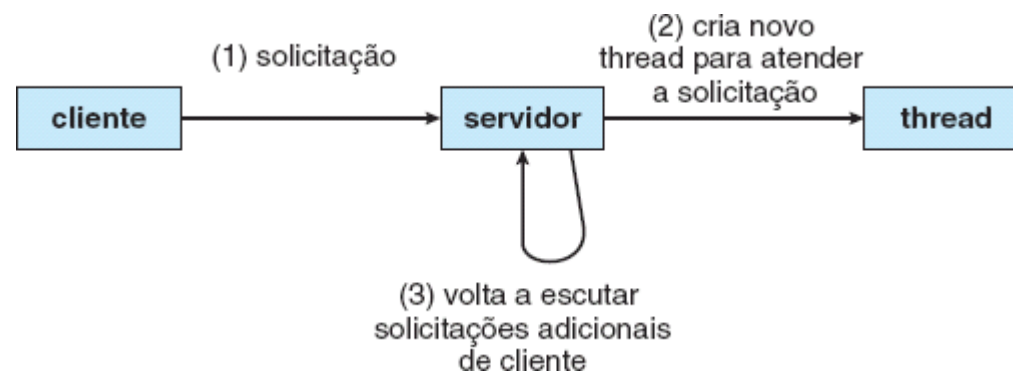
## Exemplo 1

Editores de texto, pode possuir uma thread para recebimento do sinal do teclado, outra para o salvamento periódico em disco, formatação, ortografia, etc..

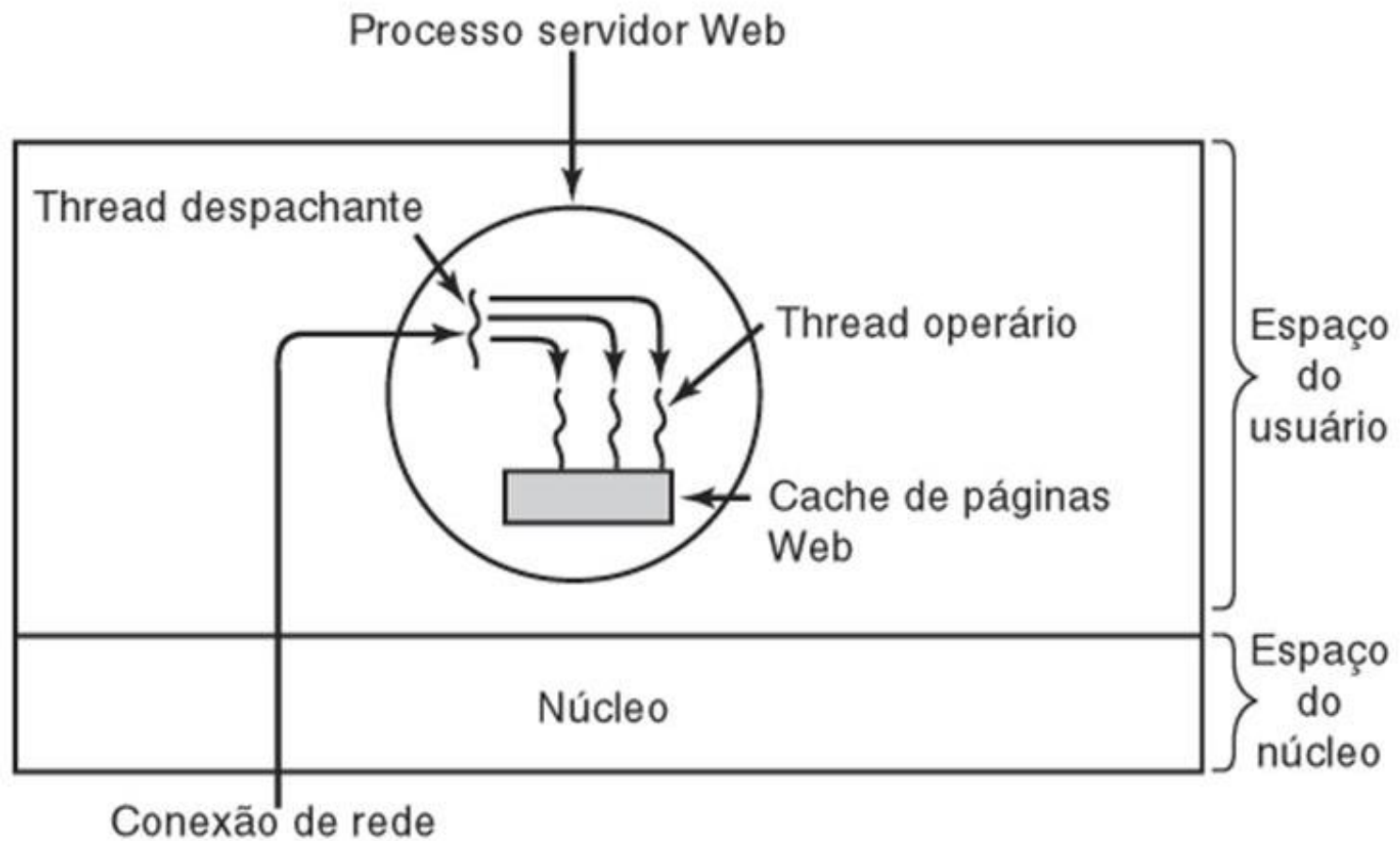


## Thread

A criação de processos é demorada e usa muitos recursos, no entanto. Se o novo processo tiver que executar as mesmas tarefas do processo existente, isso gera overhead. Portanto é mais eficiente usar um processo contendo múltiplos threads. Se o processo do servidor web for multithreaded, o servidor criará um thread separado para escutar as solicitações de clientes. **Quando uma solicitação for feita, em vez de criar outro processo, o servidor criará um novo thread para atender a solicitação e voltará a esperar por solicitações adicionais.**



## Exemplo 2

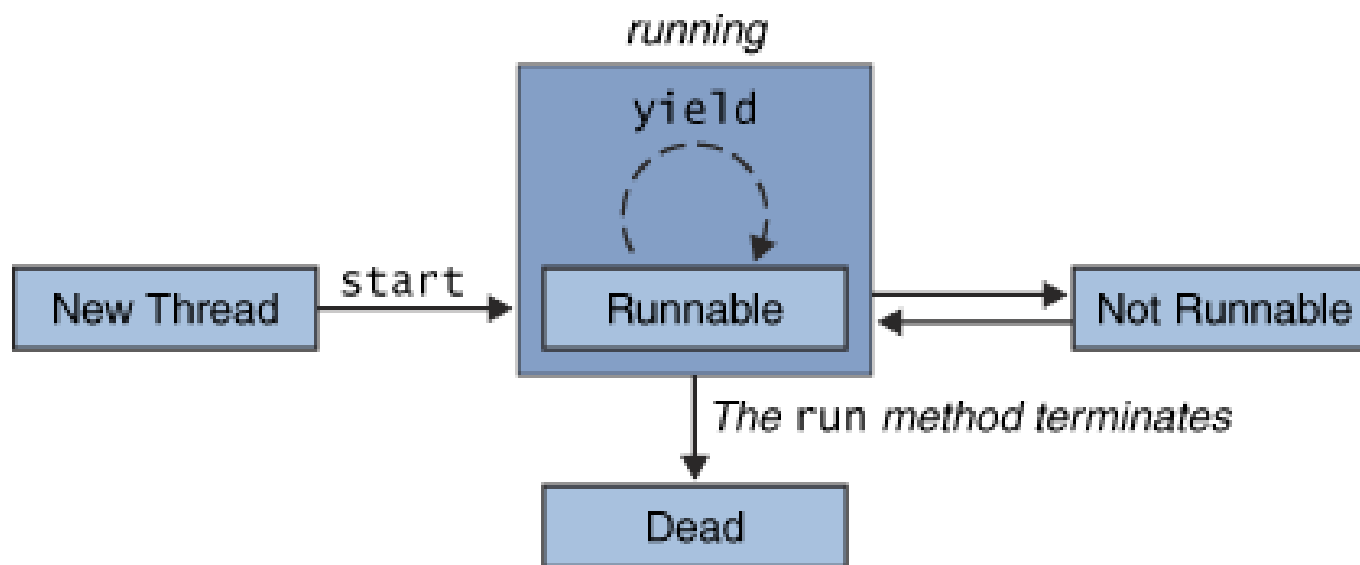


## Thread

**Um thread não é um programa, mas é executado em um programa.** Cada Thread tem sua própria pilha de execução (pois chamam rotinas diferentes) registradores e estado, embora compartilhe os dados e o espaço de endereços.

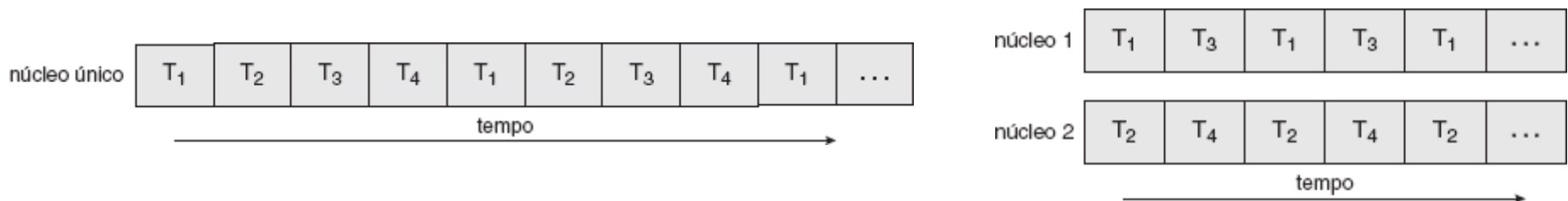
Thread:	Processo:
Program Counter	Program Counter
Stack (pilha)	Stack
Conjunto de registradores	Conjunto de registradores
Registrador de “status”	Registrador de “status”
Threads filhos	-
-	Espaço próprio de endereço
-	Variáveis globais
-	Arquivos abertos
-	Semáforos
-	Informações de contabilização
-	Processos filhos

## Ciclo de Vida de uma Thread



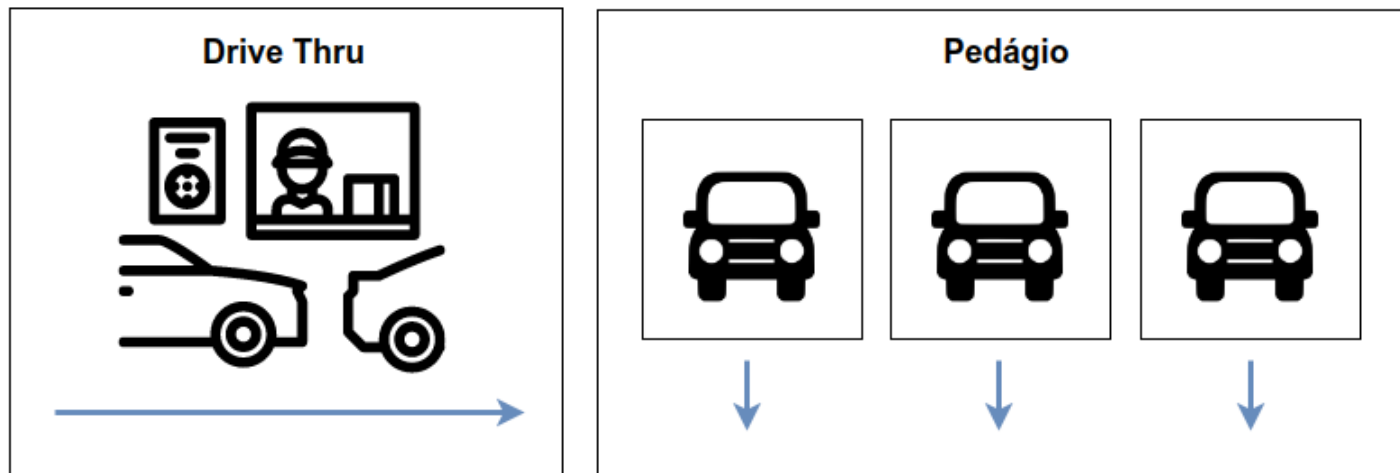
## Programação Multicore

Sistemas com uma única CPU evoluíram para sistemas com várias CPUs. Uma tendência semelhante e mais recente no projeto de sistemas é a inserção de múltiplos núcleos de computação no mesmo chip. Cada núcleo aparece como um processador separado para o sistema operacional, chamamos esses sistemas de **multicore** ou **multiprocessadores**. A programação com múltiplos threads fornece um mecanismo para o uso mais eficiente desses múltiplos núcleos de computação e o aumento da concorrência. Considere uma aplicação com quatro threads. Em um sistema com um único núcleo de computação, concorrência significa simplesmente que a execução dos threads será intercalada por meio do tempo, já que o núcleo de processamento é capaz de executar apenas um thread de cada vez. Em um sistema com múltiplos núcleos, no entanto, concorrência significa que os threads podem ser executados em paralelo, já que o sistema pode atribuir um thread separado a cada núcleo.



## Paralelismo e Concorrência

Observe a diferença entre **paralelismo** e **concorrência** nessa discussão. Um sistema é paralelo quando pode executar mais de uma tarefa simultaneamente. Por outro lado, um sistema concorrente dá suporte a mais de uma tarefa, permitindo que todas elas progridam. Assim, é possível haver concorrência sem paralelismo.



## Benefícios

- 1. Capacidade de resposta.** Tornar uma aplicação interativa multithreaded pode permitir que um programa continue a ser executado, mesmo que parte dele esteja bloqueada, aguardando um processamento.
- 2. Compartilhamento de recursos.** Os threads compartilham a memória e os recursos do processo ao qual pertencem. Isso permite que uma aplicação tenha múltiplas atividades diferentes dentro do mesmo espaço de endereçamento.
- 3. Economia.** A alocação de memória e recursos para a criação de processos é dispendiosa. Já que os threads compartilham os recursos do processo ao qual pertencem, é mais econômico criar threads e permutar seus contextos. Em algumas ocasiões, até 100 vezes mais rápida.
- 4. Escalabilidade.** Os benefícios da criação de múltiplos threads podem ser ainda maiores em uma arquitetura multiprocessadora, em que os threads possam ser executados em paralelo em diferentes núcleos de processamento (Paralelismo real).



## Finalizando processos

- 1) **Término Normal (voluntário)** - A tarefa a ser executada é finalizada. Ao terminar, o processo executa uma chamada comunicando ao SO que terminou: `exit` (Unix) `exitprocess` (Windows)
- 2) **Término por erro (voluntário)** - O processo sendo executado não pode ser finalizado. Ex: `gcc filename.c`; o arquivo `filename.c` não existe
- 3) **Término com erro fatal (involuntário)** – Causado por algum erro no programa (bug). Ex: divisão por 0; referência a memória inexistente; execução de uma instrução ilegal.
- 4) **Término (involuntário)** – via chamada, `Kill` (Unix) ou `Terminateprocess` (Windows).

## Gerenciamento de Processos Linux

Os dois comandos mais usados para visualizar processos são **top** e **ps**. A diferença entre os dois é que o top é mais usado interativamente e ps é mais usado em scripts, combinado com outros comandos.

```
Terminal
top - 16:54:02 up 1 day, 1:31, 1 user, load average: 0.31, 0.26, 0.29
Tasks: 243 total, 2 running, 239 sleeping, 2 stopped, 0 zombie
%Cpu(s): 2.8 us, 2.3 sy, 0.0 ni, 91.6 id, 3.3 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 8010248 total, 2563860 free, 2711448 used, 2734940 buff/cache
KiB Swap: 8219644 total, 8211328 free, 8316 used. 4214880 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
15121 root        20   0   517420  115160 104260 S   8.6   1.4   2:24.23 Xorg
15718 tautvyd+   20   0  1173920   98776   58136 S   6.0   1.2   2:30.48 compiz
19505 tautvyd+   20   0   617384   32004   25148 S   3.0   0.4   0:00.52 gnome-scre+
15913 tautvyd+   20   0  1254200  247236  109348 S   1.0   3.1   5:07.88 chrome
15984 tautvyd+   20   0  1277680  428300  119400 S   0.7   5.3   2:41.40 chrome
16868 tautvyd+   20   0  1041548  263828   73944 S   0.7   3.3   1:13.54 chrome
    7 root        20   0         0         0         0 S   0.3   0.0   2:27.84 rcu_sched
15593 tautvyd+   20   0   633824   35324   25340 S   0.3   0.4   0:03.80 unity-pane+
16047 tautvyd+   20   0   819588   95604   58088 S   0.3   1.2   0:12.78 chrome
16698 tautvyd+   20   0  1554504  134512   80180 S   0.3   1.7   0:42.05 slack
16815 tautvyd+   20   0  1057504  251084   96444 S   0.3   3.1   1:33.90 chrome
19497 tautvyd+   20   0    52200    4132    3416 R   0.3   0.1   0:00.11 top
    1 root        20   0  185424     6072    4000 S   0.0   0.1   0:08.87 systemd
    2 root        20   0         0         0         0 S   0.0   0.0   0:00.19 kthreadd
    3 root        20   0         0         0         0 S   0.0   0.0   0:01.02 ksoftirqd/0
    8 root        20   0         0         0         0 S   0.0   0.0   0:00.00 rcu_bh
    9 root        rt    0         0         0         0 S   0.0   0.0   0:00.04 migration/0
```

## Gerenciamento de Processos Linux

Para matar um processo você pode utilizar o comando kill. Por exemplo: kill pid, se o processo é teimoso e não quer ser morto facilmente, você pode usar: kill -9 pid. Ainda pode ser utilizado o comando killall e o nome do processo.

```
Mem: 509248k total, 494536k used, 14712k free, 20484k buffers
Swap: 916472k total, 2640k used, 913832k free, 142056k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
864	root	20	0	63144	35m	9396	S	14.6	7.2	0:17.59	Xorg
1923	ihaveapc	20	0	140m	19m	14m	S	4.6	3.9	0:01.44	nautilus
1922	ihaveapc	20	0	115m	15m	11m	S	1.7	3.1	0:00.97	gnome-panel
2148	ihaveapc	20	0	118m	13m	9m	S	1.0	2.6	0:00.44	gnome-terminal
2177	ihaveapc	20	0	2544	1224	928	R	0.7	0.2	0:00.20	top
1896	ihaveapc	20	0	3056	1284	636	S	0.3	0.3	0:00.26	dbus-daemon
1899	ihaveapc	20	0	7644	4376	2232	S	0.3	0.9	0:00.30	gconfd-2
1917	ihaveapc	20	0	35528	11m	8344	S	0.3	2.3	0:01.39	metacity
1939	ihaveapc	20	0	81640	30m	16m	S	0.3	6.1	0:03.96	docky
2031	ihaveapc	20	0	85072	4560	3684	S	0.3	0.9	0:00.04	indicator-sound
2124	ihaveapc	20	0	251m	45m	23m	S	0.3	9.2	0:02.20	firefox-bin
1	root	20	0	2804	1564	1224	S	0.0	0.3	0:01.07	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
6	root	20	0	0	0	0	S	0.0	0.0	0:00.05	events/0

```
ihaveapc@ihaveapc-desktop ~ $ sudo kill 2124
[sudo] password for ihaveapc:
ihaveapc@ihaveapc-desktop ~ $
```



Obrigado!