

Fundamentos sobre Processos

Prof. Dr. Márcio Castro
marcio.castro@ufsc.br



1

Ciclo de vida de um processo

Ciclo de vida de um processo

- **Processos são criados e destruídos**

- Momento e a forma pela qual são criados e destruídos depende do SO
- Normalmente são criados/destruídos através de **chamadas de sistema**
- Após a criação, o processo é então executado pelo processador (**ciclo de processador**), podendo deixá-lo para realizar operações de E/S (**ciclo de E/S**)

- **Dois tipos de processo**

- Processos de sistema (*daemons*)
- Processos de usuário

Ciclo de vida de um processo

■ Perfis de processos

- **CPU-bound:** tempo de execução do processo é definido principalmente pelo tempo dos seus **ciclos de processador**
- **Memory-bound:** tempo de execução do processo é definido principalmente pelo **tempo de acesso à memória**
- **I/O-bound:** tempo de execução é definido principalmente pela duração das **operações de E/S**

Ciclo de vida de um processo

- **Ciclo de vida de um processo**

- Processos alternam entre diferentes estados durante seus ciclos de vida

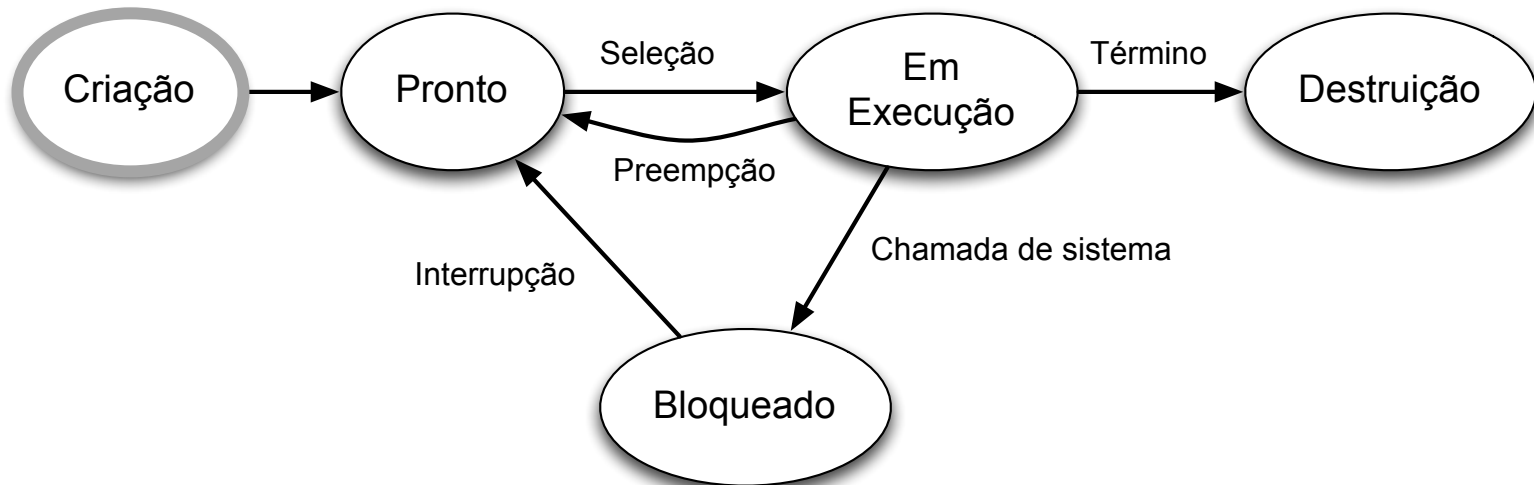
- **Estados de um processo**

- Criação
 - Pronto
 - Em execução
 - Bloqueado
 - Término

Ciclo de vida de um processo

▪ Criação

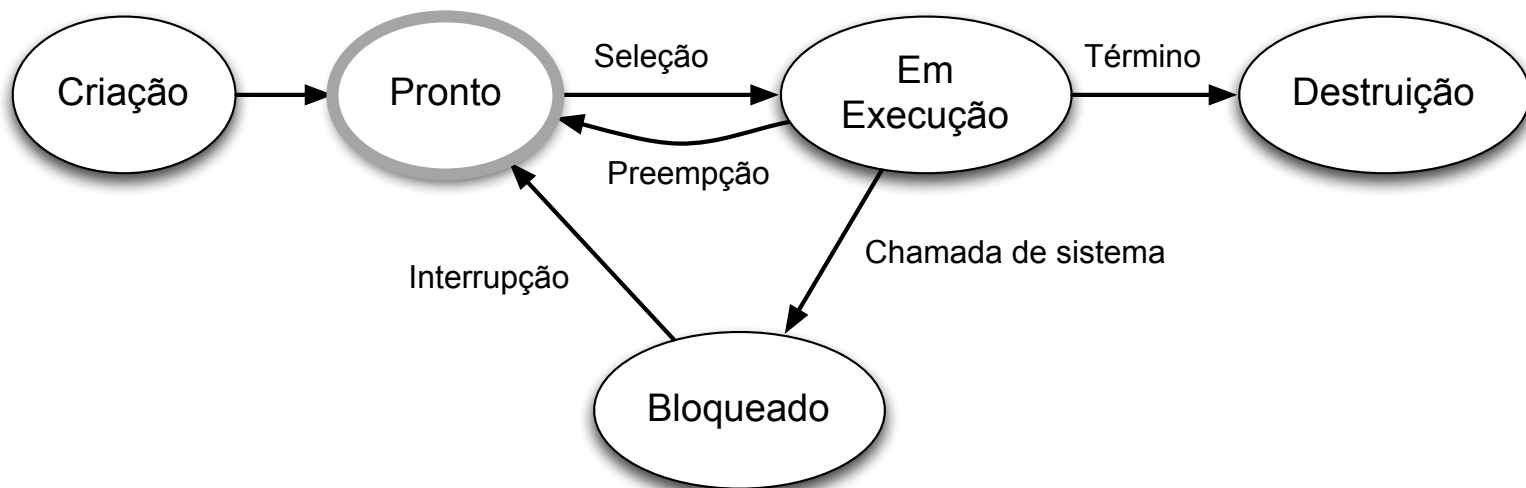
- Quando um usuário executa um programa, o SO cria um processo
- Criação é feita através de **chamadas de sistema**
 - **Exemplos:** fork, spawn, ...
- Processo recebe uma **identificação única**
 - *Process Identification (PID)*



Ciclo de vida de um processo

▪ Pronto

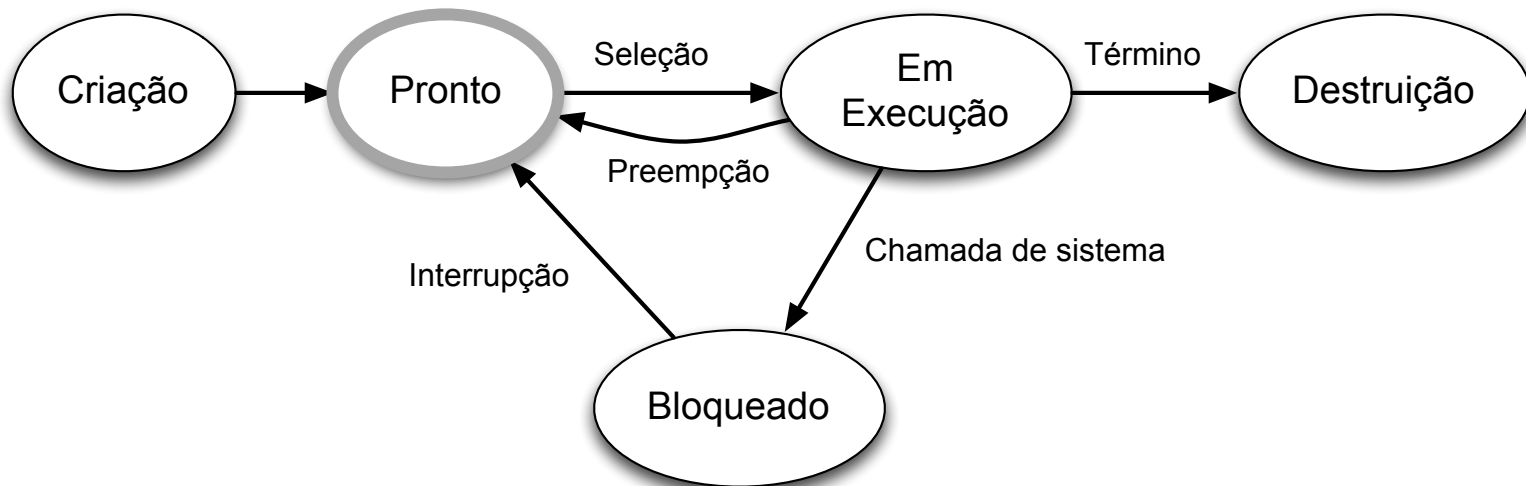
- Após ser criado, o processo passa para o estado **pronto** e está apto a ser executado pela CPU
- Como podem existir diversos processos no estado pronto (**fila de processos prontos**), o SO deve selecionar um para executar
- **Escalonador**: responsável por selecionar um processo pronto para executar na CPU



Ciclo de vida de um processo

- **Escalonador: qual processo escolher?**

- Depende do **algoritmo de escalonamento** utilizado
- Cada algoritmo terá critérios diferentes



Ciclo de vida de um processo

■ Em execução

- Ao ser selecionado, o processo é executado pela CPU
- O processo permanece em execução na CPU até que termine ou que ocorra uma **preempção**
- Ao ocorrer uma preempção o processo **volta para o estado pronto** e um **outro processo é selecionado**



Ciclo de vida de um processo

▪ Preempção: quando ocorre?

- Depende do algoritmo de escalonamento utilizado
- Cada algoritmo terá critérios diferentes



Ciclo de vida de um processo

▪ Preempção: exemplos

1. Quando o tempo limite em que um processo pode executar na CPU é atingido:

- O processo executa um certo número de instruções
- Quando o limite de tempo é atingido, ele libera a CPU temporariamente para dar chance a outros processos

2. Quando um processo de maior prioridade (mais importante) é criado



Ciclo de vida de um processo

▪ Bloqueado

- O processo passa para o estado **bloqueado** ao realizar uma **chamada de sistema**
 - Os processos bloqueados são armazenados em filas
 - Cada dispositivo de E/S pode possuir uma fila diferente
- **Chamada de sistema:** quando um processo requisita algum serviço do SO



Ciclo de vida de um processo

■ Bloqueado

- Ao final da execução do serviço requisitado é gerada uma **interrupção**, informando o SO do término da operação



Ciclo de vida de um processo

▪ Destruição

- Quando um processo termina ele passa para o estado **destruição**
- Nesse momento, a memória utilizada pelo processo é liberada
- **Outros motivos que levam a destruição:**
 - Erro, intervenção de outro processo (*kill*), ...



Implementação de processos

- O SO mantém uma **tabela de processos**
 - Uma entrada na tabela é chamada de ***Process Control Block (PCB)*** ou **descritor de processo**
 - Contém informações sobre **contexto do processo**

Gerência de processos	Gerência de memória
Registradores	Ponteiro para segmento de texto
Contador de programa (PC)	Ponteiro para segmento de dados
Ponteiro de pilha (SP)	Ponteiro para pilha
Estado do processo	Gerência de arquivos
PID	Diretório raiz
PID do processo pai	Diretório corrente
Estatísticas de uso do processador	Descritores de arquivos

2 Processos no Linux

Chamadas de sistema

Exemplos de chamadas de sistema

Grupo	Windows	Linux
Controle de processos	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
Manipulação de arquivos	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Utilitários	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()

Processos no Linux

- Criados através da chamada de sistema:
 - *fork()*
- Noção de processo pai e filho
 - Um processo é criado quando o programa é executado (**processo pai**)
 - Um processo pai pode criar **processos filhos**
 - **O processo filho é uma cópia do processo pai (pai e filho executarão o código que estiver após o fork)**

Processos no Linux

- A chamada de sistema ***wait(NULL)*** permite que o processo aguarde o término de um dos seus filhos
 - **Retorno de *wait(NULL)*:**
 - -1: quando o último filho do processo terminou;
 - > 0: quando um processo filho terminou mas ainda existem outros processos filhos em execução o PID do filho que terminou é retornado
 - Para que cada processo aguarde o **término de todos os seus processos filhos** é necessário que o mesmo execute:
 - `while(wait(NULL) > 0);`

Processos no Linux

■ Processo pai ou processo filho?

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
    pid_t pid;
    pid = fork();

    if(pid >= 0) { // se pid é positivo, criou o processo
        if (pid == 0)
            ProcessoFilho();
        else {
            ProcessoPai();
            wait(NULL); // aguarda o termino do filho
        }
        return 0;
    }
    else // se pid é negativo, não pode criar o processo
        printf("\n Não pode criar o processo.\n");
    return 1;
}
```

Processos no Linux

■ Exemplo: hierarquia de processos

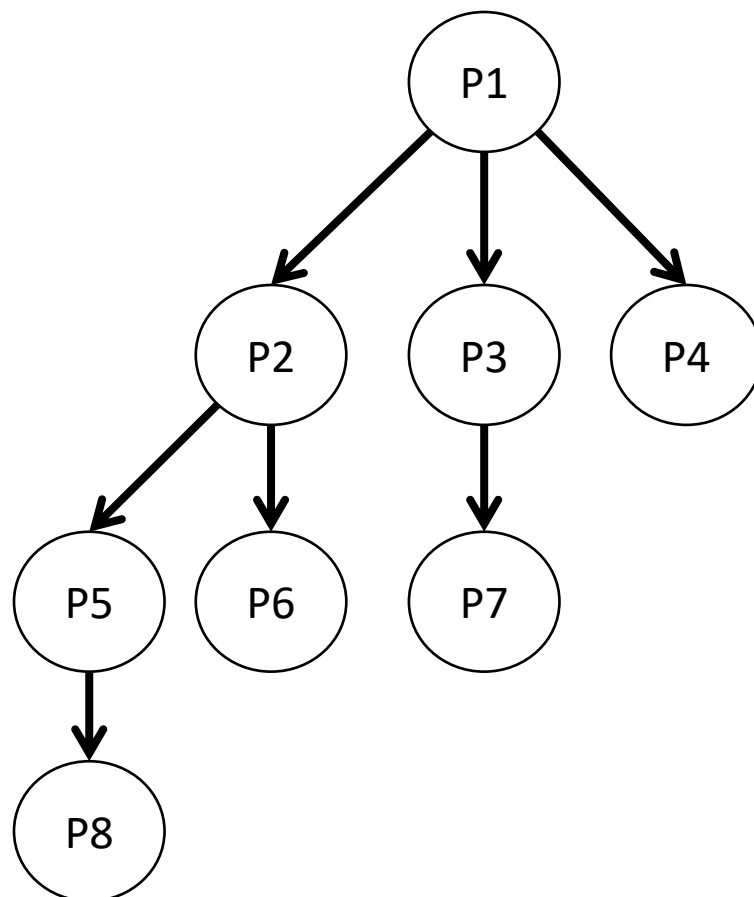
```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
    fork();
    fork();
    fork();

    printf("Novo processo!\n");
    return 0;
}
```

```
$ ./meu_programa_com_fork
```

```
Novo processo!
Novo processo!
Novo processo!
Novo processo!
Novo processo!
Novo processo!
Novo processo!
Novo processo!
```



Processos no Linux

▪ **Compartilhamento de dados**

- Processos **não compartilham** o mesmo espaço de endereçamento
- Logo, processos pai e filho terão cópias idênticas dos dados após um *fork()*
- **Não há compartilhamento de variáveis globais** entre processos pai/filho criados a partir de uma chamada *fork()*: cada processo terá uma **cópia das variáveis globais** no seu próprio espaço de endereçamento

Processos no Linux


```
#include <stdio.h>
#include <unistd.h>

int var_global = 0;

int main(int argc, char **argv) {
    int var_local = 0;

    pid_t pid;
    pid = fork();

    if (pid == 0) { // filho
        var_global = 1;
        var_local = 2;
    }
    else { // pai
        var_global = 50;
        var_local = 100;
    }
    return 0;
}
```



The diagram illustrates the state of the parent and child processes after the `fork()` call. An arrow points from the `fork()` call in the code to the first table, which shows the initial state where both parent and child have `var_global = 0` and `var_local = 0`. Another arrow points from the `if (pid == 0)` block in the code to the second table, which shows the state after the child process has modified the global variable to 1 and the local variable to 2, while the parent process has modified the global variable to 50 and the local variable to 100.

Processo pai	Processo filho
var_global = 0	var_global = 0
var_local = 0	var_local = 0

Processo pai	Processo filho
var_global = 50	var_global = 1
var_local = 100	var_local = 2

Administração de processos no Linux

- Dois modos de execução de processos
 - **Foreground:** enquanto o processo especificado na linha de comando não termina, o terminal permanece bloqueado, impedindo o disparo de novos processos

```
$ ./meu_programa
```

- **Background:** o terminal é liberado imediatamente, permitindo o disparo de novos processos. O PID do processo é retornado

```
$ ./meu_programa1 &  
[1] 8229  
$ ./meu_programa2 &  
[2] 8230  
  
[1]+  Done                  ./meu_programa1  
[2]+  Done                  ./meu_programa2
```


Administração de processos no Linux

- **top**: lista os processos em execução, a memória ocupada e o uso dos processadores

```
top - 15:03:30 up 6 days, 5:02, 1 user, load average: 0.76, 0.33, 0.17
Tasks: 332 total, 5 running, 327 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.3 us, 1.9 sy, 0.0 ni, 95.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 66096356 total, 9651792 used, 56444564 free, 2067116 buffers
KiB Swap: 11718610+total, 0 used, 11718610+free. 6389404 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
67914	bastosca	20	0	16416	4756	1764	R	74.4	0.0	0:41.32	bash
67913	bastosca	20	0	74412	2264	924	R	40.2	0.0	0:29.17	screen
67699	root	20	0	0	0	0	S	9.0	0.0	0:00.79	kworker/20:0
1806	root	20	0	0	0	0	S	8.0	0.0	0:07.12	kworker/8:0
207	root	20	0	0	0	0	R	4.0	0.0	0:46.43	kworker/28:1
67804	root	20	0	0	0	0	S	3.3	0.0	0:00.78	kworker/16:1
54786	root	20	0	0	0	0	S	1.3	0.0	0:20.00	kworker/4:2
157	root	20	0	0	0	0	S	0.3	0.0	0:00.98	ksoftirqd/29
17097	snmp	20	0	100980	8588	3432	S	0.3	0.0	3:26.67	snmpd
67819	bastosca	20	0	134540	3124	1024	S	0.3	0.0	0:01.36	sshd
68017	bastosca	20	0	17928	1752	1088	R	0.3	0.0	0:00.11	top
1	root	20	0	15444	768	628	S	0.0	0.0	0:04.04	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.03	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:08.76	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
6	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kworker/u160:0
7	root	20	0	0	0	0	S	0.0	0.0	0:23.82	kworker/u161:0
8	root	20	0	0	0	0	S	0.0	0.0	2:03.56	rcu_sched
9	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
10	root	rt	0	0	0	0	S	0.0	0.0	0:00.19	migration/0

Administração de processos no Linux

- **htop**: simular ao top porém visualmente mais completo, além de ser interativo

```
1 [ 0.0% 17 [ 0.5%
2 [ 0.0% 18 [ 0.0%
3 [ 0.0% 19 [ 0.0%
4 [ 0.0% 20 [ 0.0%
5 [ 0.0% 21 [ 31.0%
6 [ 0.0% 22 [ 0.0%
7 [ 0.0% 23 [ 0.0%
8 [ 0.0% 24 [ 0.0%
9 [ 0.0% 25 [ 0.0%
10 [ 0.0% 26 [ 0.0%
11 [ 0.0% 27 [ 0.0%
12 [ 0.0% 28 [ 0.0%
13 [ 22.7% 29 [ 77.3%
14 [ 0.0% 30 [ 0.0%
15 [ 0.0% 31 [ 0.0%
16 [ 0.0% 32 [ 0.0%
Mem[ 1167/64547MB Tasks: 49, 86 thr, 283 kthr; 2 running
Swp[ 0/114439MB Load average: 0.97 0.41 0.20
Uptime: 6 days, 05:02:50

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
67914 bastosca 20 0 16416 4756 1764 R 75.0 0.0 0:57.93 /bin/bash
67913 bastosca 20 0 74412 2264 924 R 40.8 0.0 0:38.20 SCREEN
67699 root 20 0 0 0 0 R 14.2 0.0 0:01.57 kworker/20:0
207 root 20 0 0 0 0 S 9.5 0.0 0:49.30 kworker/28:1
1806 root 20 0 0 0 0 S 1.9 0.0 0:08.37 kworker/8:0
19815 root 20 0 99104 11068 8992 S 0.5 0.0 5:33.87 /usr/Atempo/tina/Bin/tina_daemon -td_op 1 -comm_by_pipe YES
64617 root 20 0 0 0 0 S 0.0 0.0 0:09.74 kworker/12:0
68019 bastosca 20 0 17052 2536 1324 R 0.0 0.0 0:00.07 htop
19817 root 20 0 96584 10448 8456 S 0.0 0.0 16:10.78 /usr/Atempo/tina/Bin/tina_daemon -td_op 20 -td_shm_id 0 -comm_by_pipe YES
54786 root 20 0 0 0 0 S 0.0 0.0 0:20.19 kworker/4:2
1459 root 39 19 0 0 0 S 0.0 0.0 10:46.18 kipmi0
12595 root 20 0 0 0 0 S 0.0 0.0 0:05.06 kworker/9:1
67804 root 20 0 0 0 0 S 0.0 0.0 0:01.02 kworker/16:1
20735 vtunesag 20 0 44448 3264 1524 S 0.0 0.0 0:05.92 ntd -fdwrite 4
59632 root 20 0 0 0 0 S 0.0 0.0 0:18.80 kworker/24:2
8 root 20 0 0 0 0 S 0.0 0.0 2:03.57 rcu_sched
1 root 20 0 15444 768 628 S 0.0 0.0 0:04.04 init [2]
2 root 20 0 0 0 0 S 0.0 0.0 0:00.03 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:08.75 ksoftirqd/0
5 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 kworker/0:0H
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice +F9Kill F10Quit
```

Administração de processos no Linux

- **ps:** lista os processos ativos

```
marcio@idrouille:/$ ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	15444	768	?	Ss	Aug11	0:04	init [2]
root	16192	0.0	0.0	35216	1948	?	Ss	Aug11	0:00	udev
root	16514	0.0	0.0	0	0	?	S	Aug12	0:19	[kworker/11:0]
root	18226	0.0	0.0	23276	876	?	Ss	Aug11	0:00	/sbin/rpcbind -w
root	18364	0.0	0.0	0	0	?	S<	Aug11	0:00	[rpciod]
root	18366	0.0	0.0	0	0	?	S<	Aug11	0:00	[nfsiod]
root	18777	0.0	0.0	0	0	?	S<	Aug11	0:00	[iprt]
root	67811	0.0	0.0	133836	6064	?	SLs	14:57	0:00	sshd: bastosca [priv]
bastosca	67819	0.1	0.0	134540	3124	?	R	14:57	0:02	sshd: bastosca@pts/0
bastosca	67820	0.0	0.0	16484	4684	pts/0	Ss	14:57	0:00	-bash
bastosca	68509	0.0	0.0	11148	1156	pts/0	R+	15:17	0:00	ps aux

Administração de processos no Linux

- **kill:** mata um processo

```
bastosca@idrouille:/$ ps gaux | grep SCREEN
bastosca 68377  0.0  0.0  74412  2224 ?        Ss   15:12   0:00 SCREEN
bastosca 68465  0.0  0.0  12944  956 pts/0    S+   15:13   0:00 grep SCREEN

bastosca@idrouille:/$ kill 68377

bastosca@idrouille:/$ ps gaux | grep SCREEN
bastosca 68467  0.0  0.0  12944  952 pts/0    S+   15:14   0:00 grep SCREEN
```

Administração de processos no Linux

- **pstree**: permite visualizar a hierarquia de processos

Programa “EXEMPLO” composto por 7 processos

```
$ pstree marcio -c -l -p

... varios outros processos omitidos ...

init(1)-+-bash(3373)---EXEMPLO(3818)-+-EXEMPLO(3819)-+-EXEMPLO(3822)
                                     |               ^-EXEMPLO(3824)
                                     |               ^-EXEMPLO(3820)-+-EXEMPLO(3821)
                                     |               ^-EXEMPLO(3823)

... varios outros processos omitidos ...
```

! Obrigado pela atenção!



Dúvidas? Entre em contato:

- marcio.castro@ufsc.br
- www.marciocastro.com



UNIVERSIDADE FEDERAL
DE SANTA CATARINA



Distributed Systems Research Lab
www.lapesd.inf.ufsc.br