

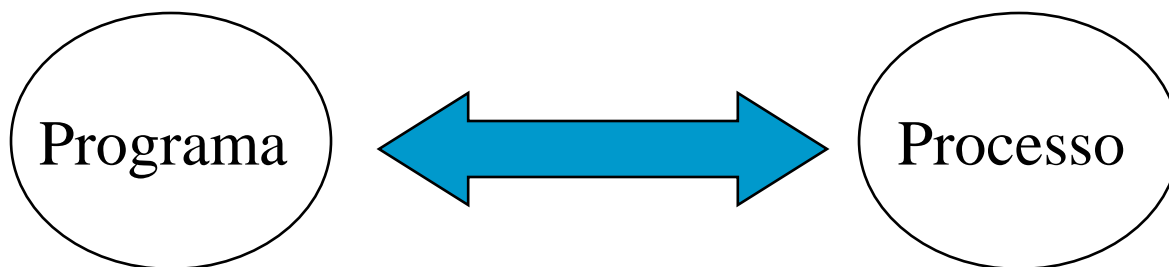
Processos

Adão de Melo Neto

Processos

■ Introdução

- Para se poder controlar o uso concorrente (ao mesmo tempo) do processador, da memória e dos dispositivos de E/S, um programa deve sempre estar sempre associado a um processo.



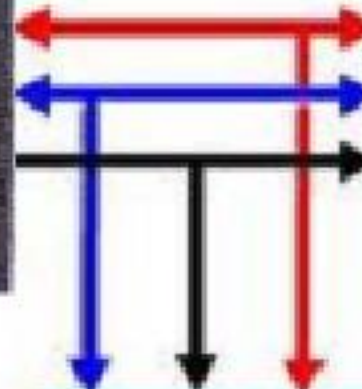
TODO PROGRAMA AO SER CRIADO JÁ VEM ASSOCIADO A UM PROCESSO

Relembrando

Executar programas armazenados na memória

Armazena programas a serem executados pelo processador

PROCESSADOR



RI (registrador de instrução)

Armazena a instrução da Memória que está sendo executada

PC (contador de instrução)

Armazena o endereço da próxima instrução da Memória a ser executada

Dispositivos de entrada /saída



Relembrando

MEMÓRIA PRINCIPAL

PC (contador de instrução)
Armazena o endereço da próxima instrução a ser executada

**PC (Program counter) =
0000H**

RI (registrador de instrução)
Armazena a instrução que está sendo executada

**RI(register instruction) =
Instrução 02H**

Endereço de Memória (Hexa)	Conteúdo de Memória (Hexa)	Linguagem Assembly
0000	02	LJMP 0100H
0001	01	
0002	00	
0100	60	JZ 010AH
0101	08	
0102	F5	MOV P1, A
0103	90	
0104	12	LCALL 1_SEC_DELAY
0105	28	
0106	55	
0107	14	DEC A
0108	70	JNZ 0102H
0109	F8	
010A	Aqui é onde o resto do programa continua

SALTAR PARA O ENDEREÇO 0100H

Os registradores mantêm informações sobre o programa em execução por isso Suas informações precisam ser guardadas na mudança de contexto

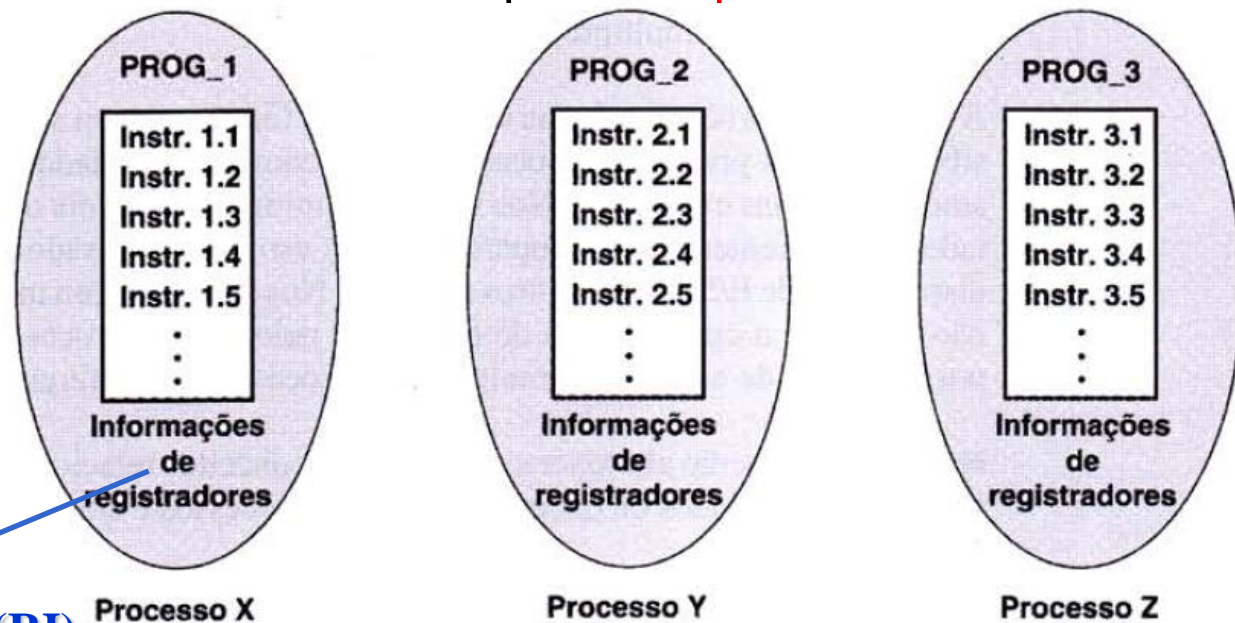
Processo e Concorrência

■ Processo

- **Conjunto de informações necessárias** para que o sistema operacional implemente a concorrência entre programas pelo uso dos recursos do sistema (processador, memória e dispositivos de E/S)

■ Concorrência

- Três **programas** associados aos respectivos **processos**.



Exemplo de registradores:

Registrador de Instruções (RI)

Registrador Program Counter (PC)

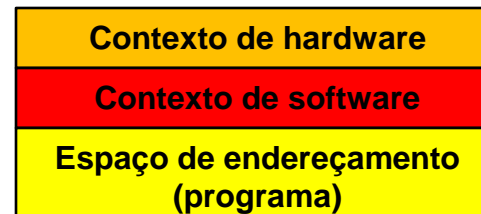
Processo e Concorrência



Registrador de Instruções (RI)
Registrador Program Counter (PC)



Os registradores mantêm informações sobre o programa em execução por isso suas informações precisam ser guardadas na mudança de contexto



MEMÓRIA PRINCIPAL

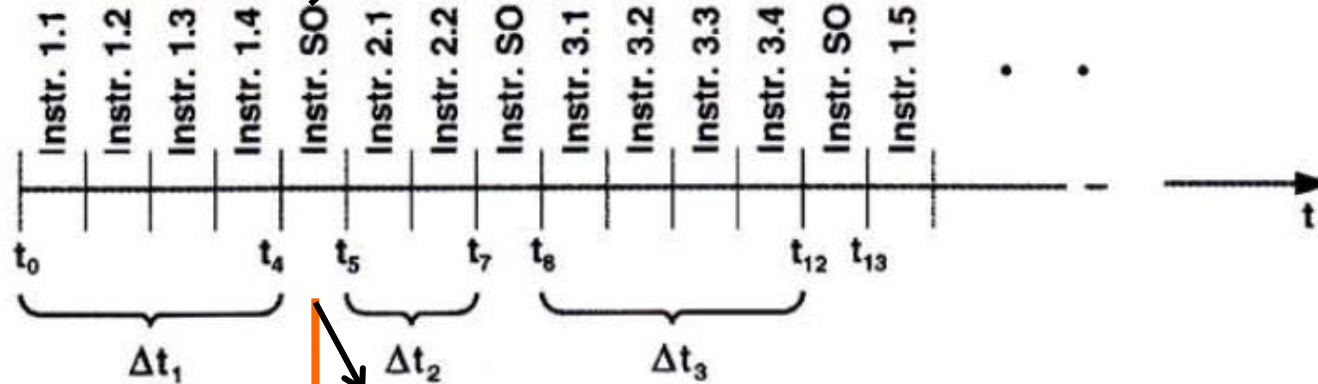
PROCESSO X

PROCESSO Y

PROCESSO Z

O programa 2 é iniciado e executado ao longo do intervalo Δt_2

Mudança de contexto: troca de um processo por outro no processador gerenciado pelo SO



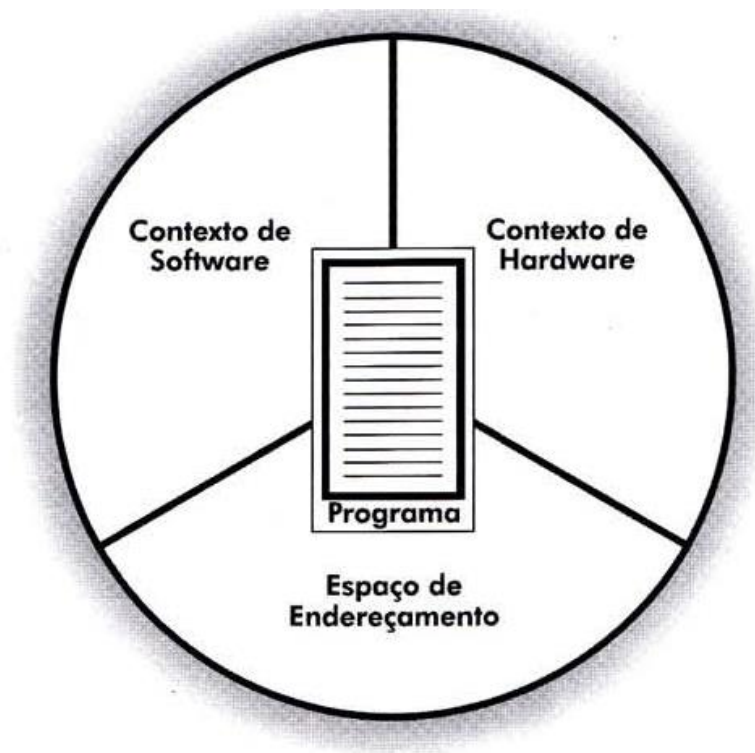
O SO decide interromper temporariamente a execução do programa 2 e salva o conteúdo dos registradores armazenando-os no processo Y

O SO decide interromper temporariamente a execução do programa 1 e salva o conteúdo dos registradores armazenando-os no processo X.



Processo

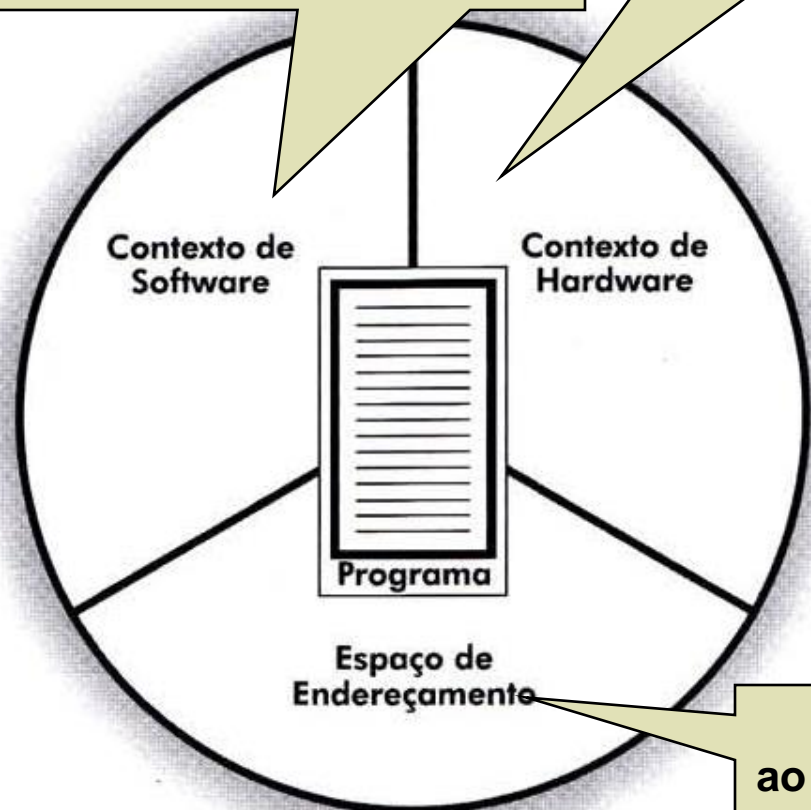
- É formado por três partes (**contexto de hardware**, **de software** e **espaço de endereçamento**) que juntas mantêm informações necessárias a execução de um programa em um sistema em que exista concorrência (multiprogramação).



Processo

São especificados os limites e características dos recursos que podem ser alocados pelo processo

Armazena o conteúdo os registradores gerais, além dos de uso específico, como o program counter (PC) e o instrutor register (RI)



Contexto de hardware

Contexto de software

Espaço de endereçamento (programas)

MEMÓRIA PRINCIPAL

PROCESSO X

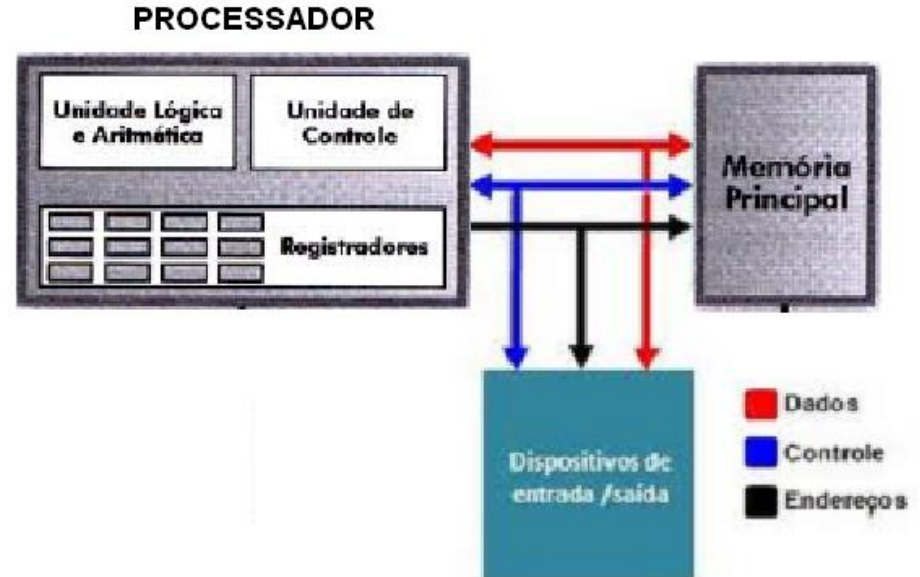
PROCESSO Y

PROCESSO Z

É a área de memória pertencente ao processo onde instruções e dados do programa são armazenados para execução

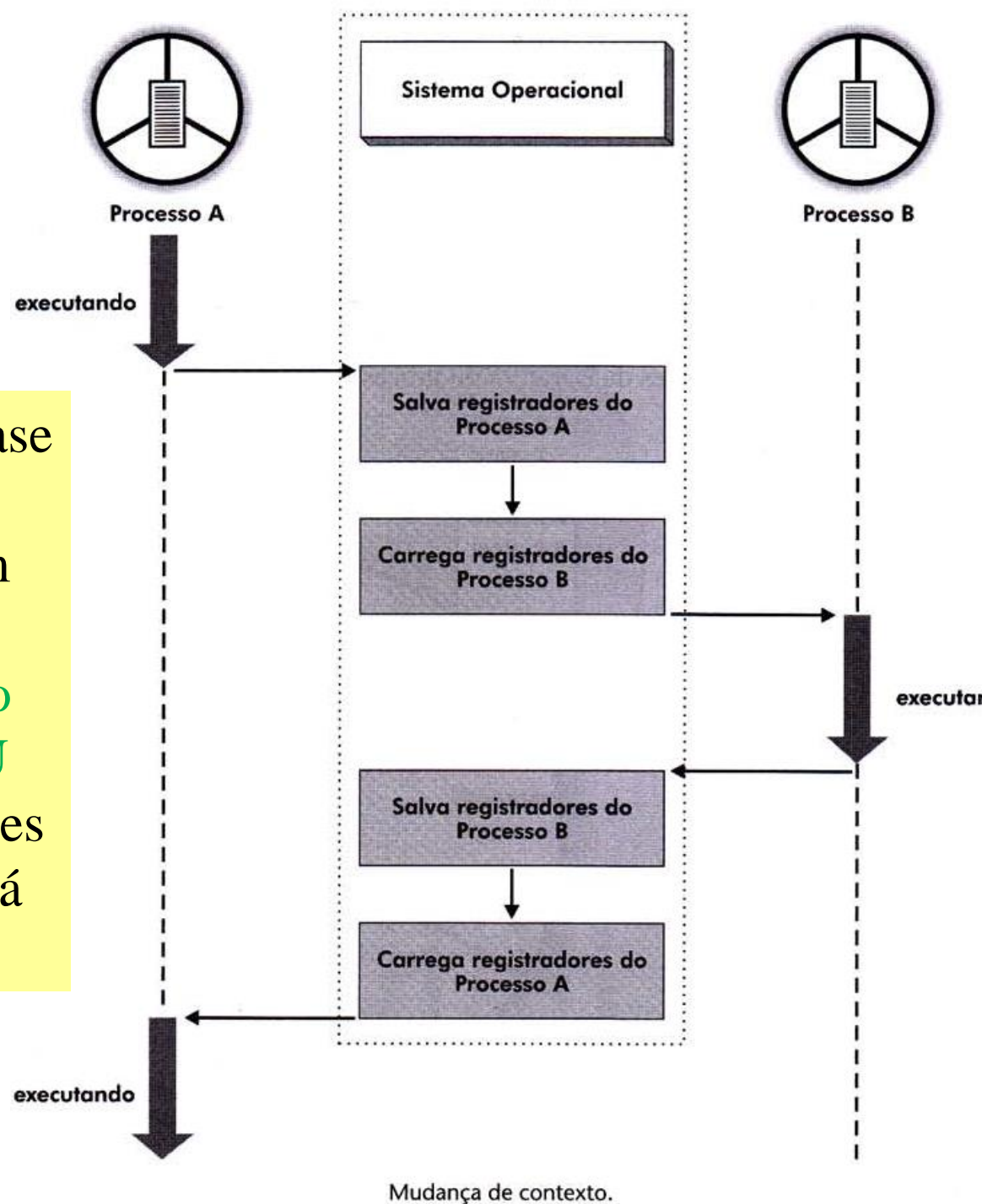
Contexto de Hardware

O contexto de hardware armazena o conteúdo dos registradores gerais, além dos de uso específico, como o program counter (PC) e o instrutor register (RI)



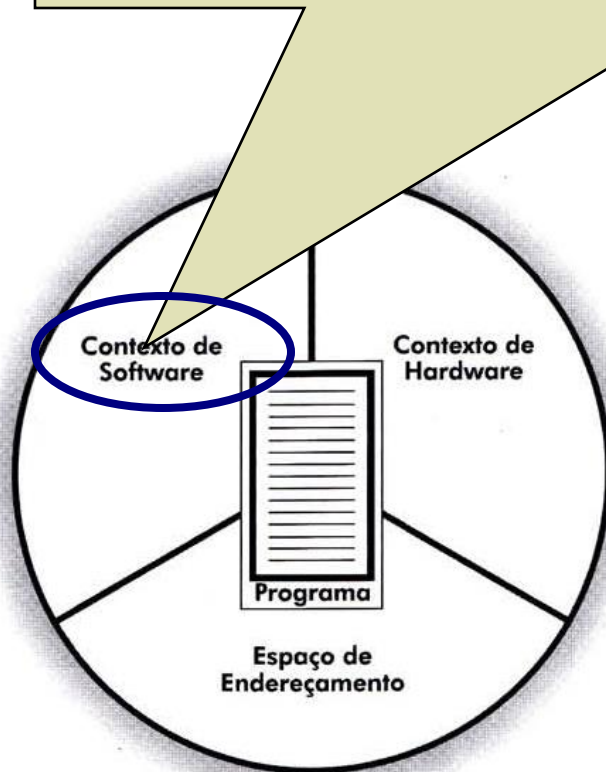
Contexto de Hardware

A mudança de contexto, base para a implementação da concorrência consiste em **salvar o conteúdo dos registradores do processo que está deixando a CPU** e carregá-los com os valores do novo processo que será executado.



Contexto de Software

No contexto de software são especificados os limites e características dos recursos que podem ser alocados pelo processo



Identificação

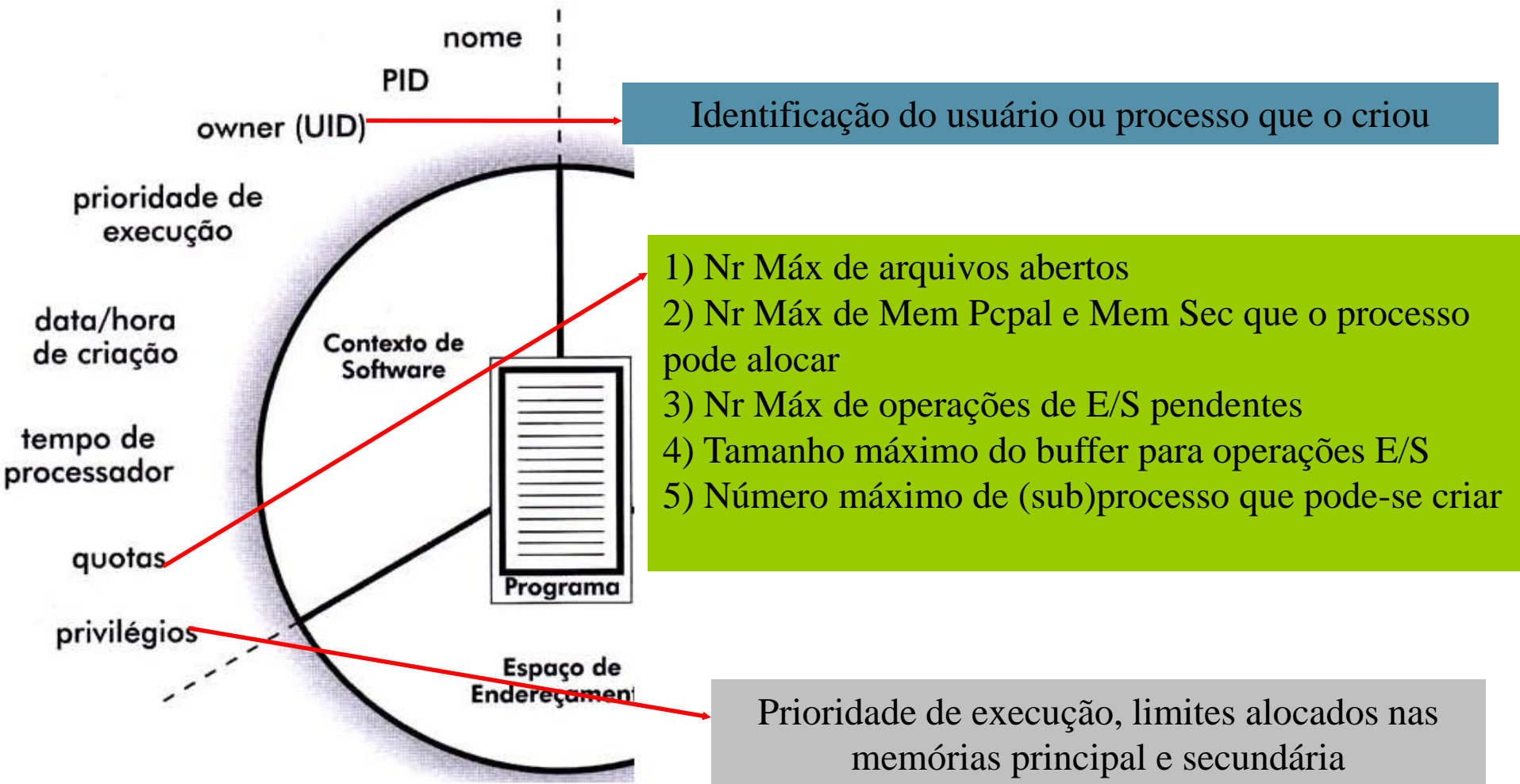
- PID (process identification)
- UID (user identification)

Quotas: são os limites dos recursos do sistema que o processo pode alocar.

Privilégios: são as ações que um processo pode fazer em relação a ele mesmo, aos demais processos e ao sistema operacional

- Afetam o próprio processo
- Afetam outros processos.

Contexto de Software



Listagem de alguns processos (estação com sistema linux)

IDT USUÁRIO

IDT PROCESSO

Tempo de utilização do processador

#	ps	-l	-A	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
4	S			0	1	0	0	75	0	-	378	schedu	?	00:00:04	init
1	S			0	2	1	0	75	0	-	0	contex	?	00:00:00	keventd
1	S			0	3	1	0	94	19	-	0	ksofti	?	00:00:00	ksoftirqd/0
1	S			0	6	1	0	85	0	-	0	bdfus	?	00:00:00	bdfush
1	S			0	4	1	0	75	0	-	0	schedu	?	00:05:35	kswapd
1	S			0	5	1	0	75	0	-	0	schedu	?	00:03:45	kscand
1	S			0	7	1	0	75	0	-	0	schedu	?	00:00:00	kupdated
1	S			0	8	1	0	85	0	-	0	md_thr	?	00:00:00	mdrecoveryd
1	S			0	21	1	0	75	0	-	0	end	?	00:05:40	kjournald
1	S			0	253	1	0	75	0	-	0	end	?	00:00:00	kjournald
1	S			0	254	1	0	75	0	-	0	end	?	00:00:00	kjournald
1	S			0	255	1	0	75	0	-	0	end	?	00:55:28	kjournald
1	S			0	579	1	0	75	0	-	399	schedu	?	00:02:00	syslogd
5	S			0	583	1	0	75	0	-	383	do_sys	?	00:00:00	klogd
5	S	32		600	1	0	75	0	0	-	414	schedu	?	00:00:00	portmap
5	S	29		619	1	0	85	0	0	-	416	schedu	?	00:00:00	rpc.statd
1	S			0	631	1	0	75	0	-	393	schedu	?	00:00:00	mdadm
5	S			0	702	1	0	75	0	-	917	schedu	?	00:00:30	sshd
5	S			0	716	1	0	75	0	-	539	schedu	?	00:00:00	xinetd
5	S			0	745	1	0	75	0	-	398	schedu	?	00:00:00	gpm
5	S			0	765	1	0	75	0	-	607	schedu	?	00:00:16	crond

Listagem de alguns processos (prática)



Listagem de alguns processos (prática)

Ubuntu - VMware Player (Non-commercial use only)

Player

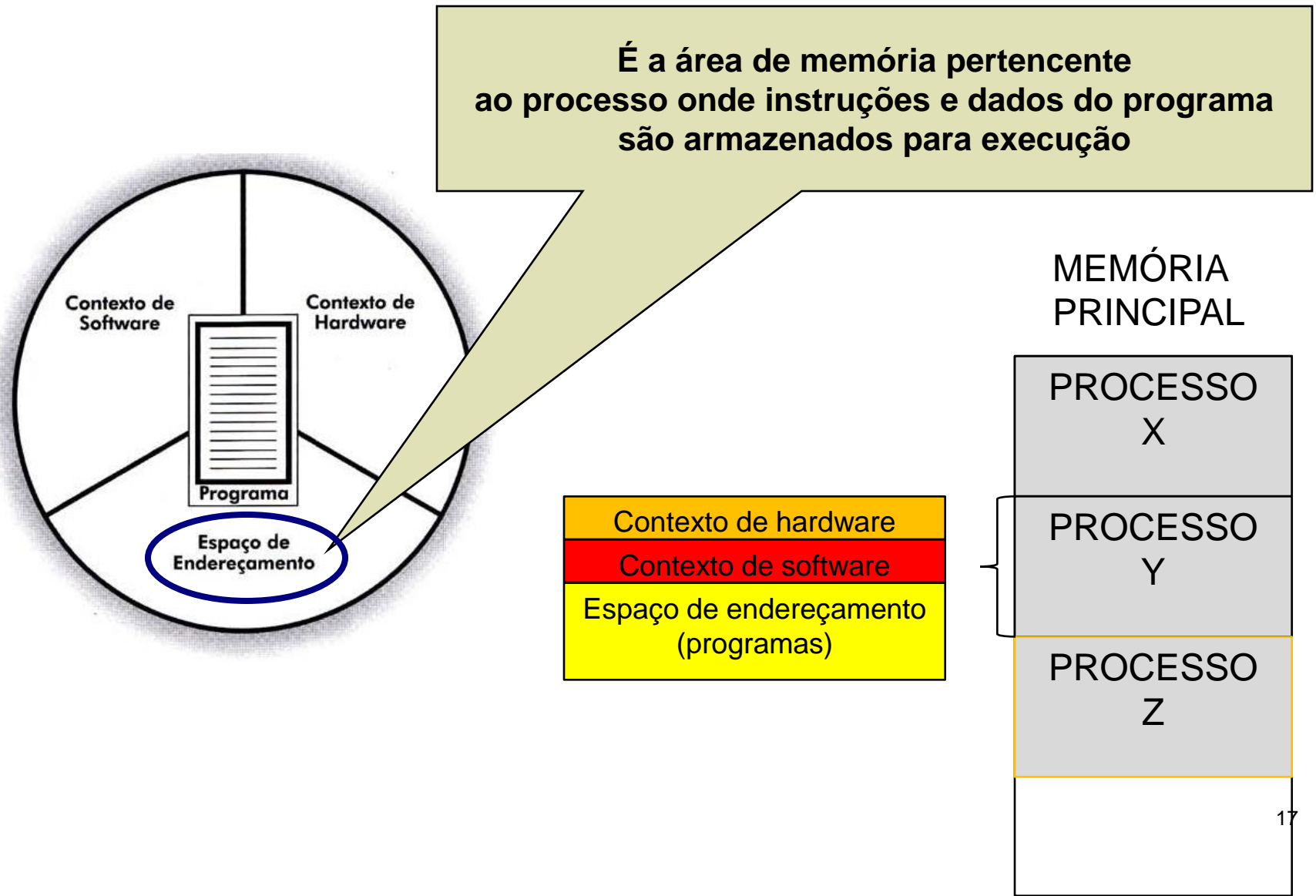
aluno@ubuntu: ~

3;J

aluno@ubuntu:~\$ ps -l -A

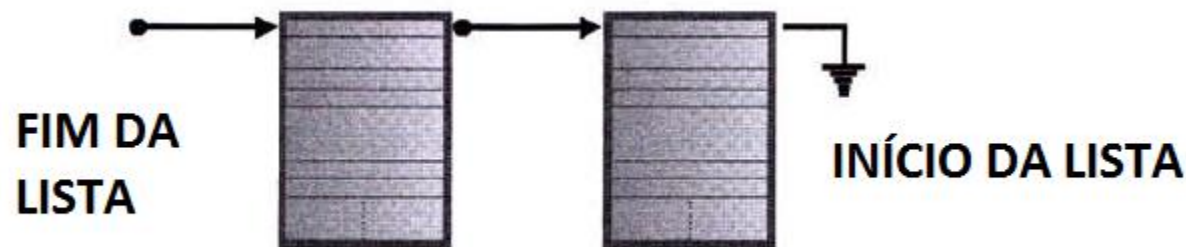
F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
4	S	0	1	0	1	80	0	-	1062	poll_s	?	00:00:06	init
1	S	0	2	0	0	80	0	-	0	kthrea	?	00:00:00	kthreadd
1	S	0	3	2	0	80	0	-	0	smpboo	?	00:00:01	ksoftirqd/0
1	S	0	5	2	0	60	-20	-	0	worker	?	00:00:00	kworker/0:0H
1	S	0	6	2	0	80	0	-	0	worker	?	00:00:00	kworker/u16:0
1	S	0	7	2	0	80	0	-	0	rcu_gp	?	00:00:01	rcu_sched
1	S	0	8	2	0	80	0	-	0	rcu_gp	?	00:00:00	rcu_bh
1	S	0	9	2	0	-40	-	-	0	smpboo	?	00:00:01	migration/0
5	S	0	10	2	0	-40	-	-	0	smpboo	?	00:00:00	watchdog/0
5	S	0	11	2	0	-40	-	-	0	smpboo	?	00:00:00	watchdog/1
1	S	0	12	2	0	-40	-	-	0	smpboo	?	00:00:00	migration/1
1	S	0	13	2	0	80	0	-	0	smpboo	?	00:00:00	ksoftirqd/1
1	S	0	15	2	0	60	-20	-	0	worker	?	00:00:00	kworker/1:0H
1	S	0	16	2	0	60	-20	-	0	rescue	?	00:00:00	khelper
5	S	0	17	2	0	80	0	-	0	devtmp	?	00:00:00	kdevtmpfs

Espaço de Endereçamento

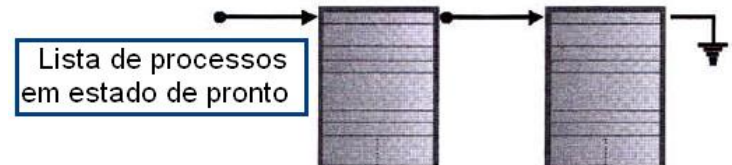
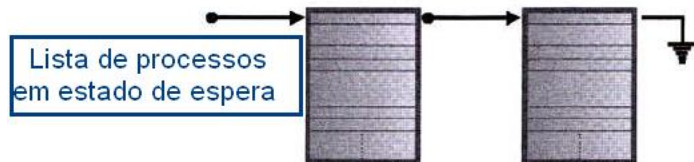
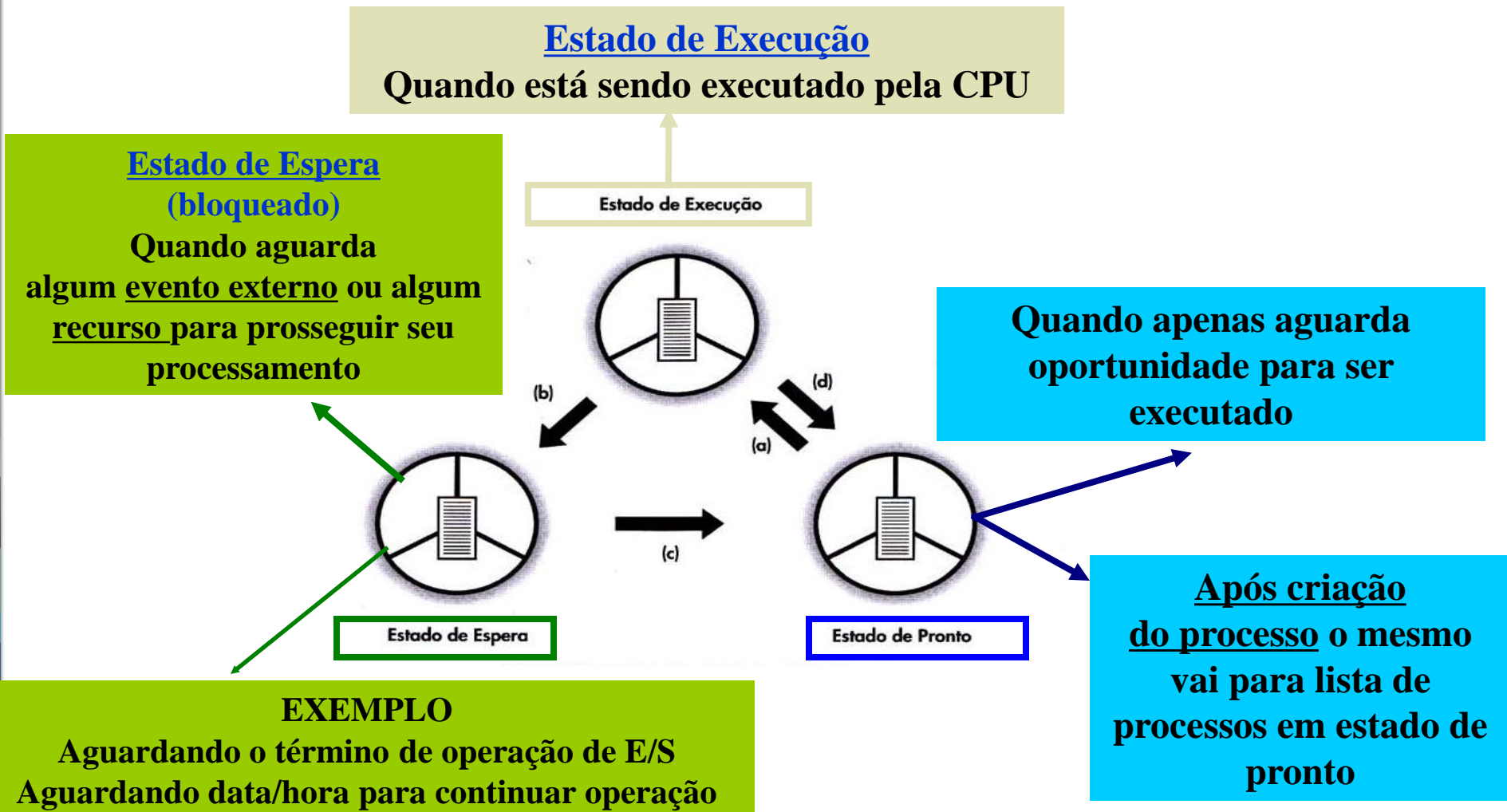


LISTA

- É uma estrutura de armazenamento de dados
- Os processos são organizados em listas



Estados de um processo

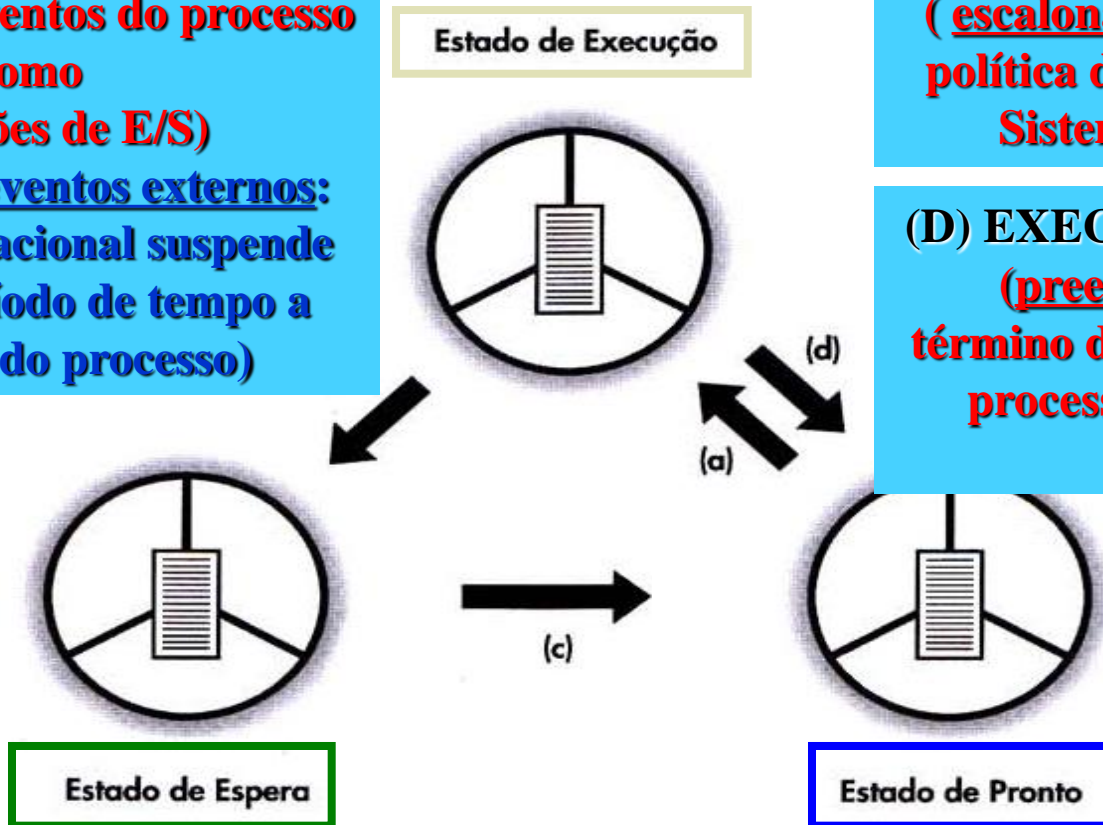


Mudança de Estados de um Processo

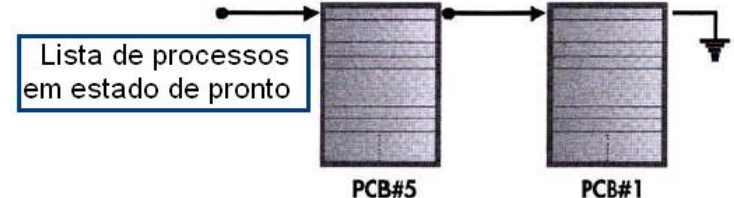
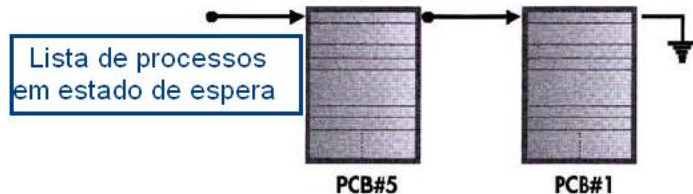
(B) EXECUÇÃO → ESPERA
(gerada por eventos do processo como operações de E/S)
(gerada por eventos externos: Sistema operacional suspende por um período de tempo a execução do processo)

(A) PRONTO → EXECUÇÃO
(escalonamento: depende da política de escalonamento do Sistema Operacional)

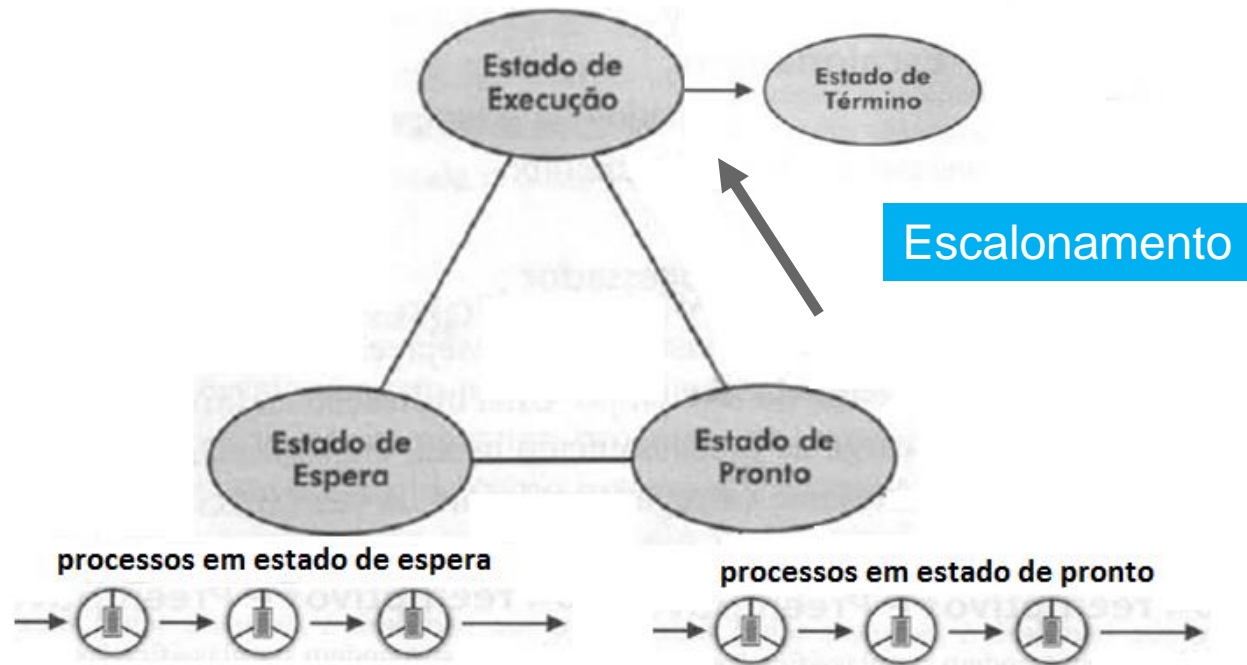
(D) EXECUÇÃO → PRONTO
(preempção: exemplo término da fatia de tempo que processo possui para sua execução)



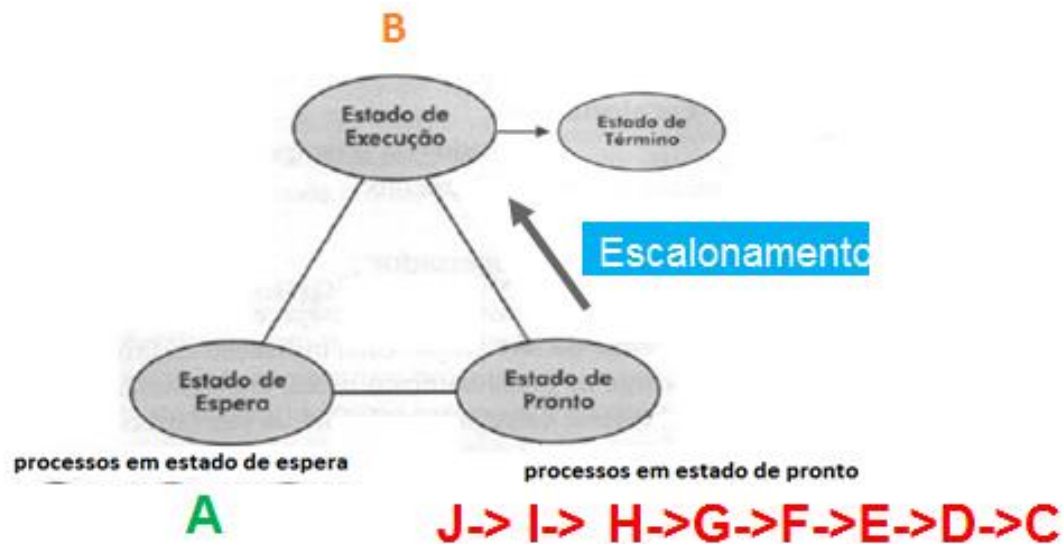
(C) ESPERA → PRONTO
(operação solicitada é atendida ou recurso esperado é concedido)



Exercício sobre Mudança de Estados de um Processo

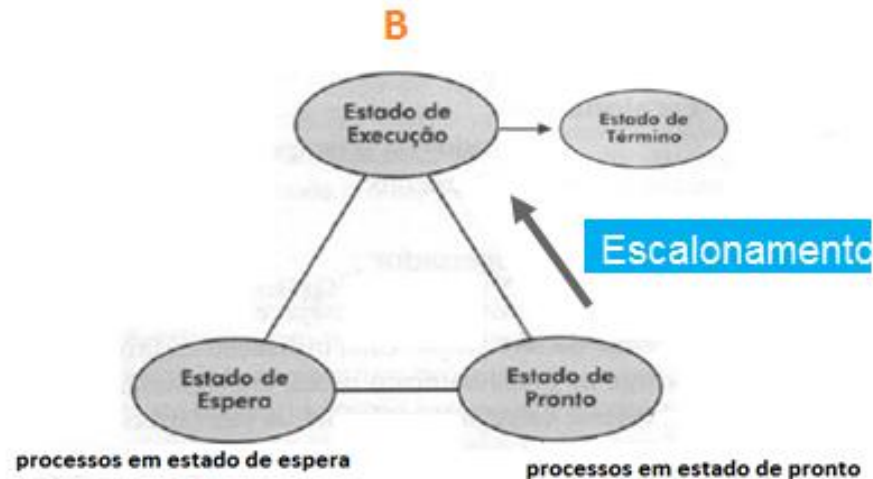


- Vamos supor que temos a seguinte situação:
 - Processos na fila estado de pronto:
 - **J-> I-> H->G->F->E->D->C**
 - Processo **B** em execução
 - Processos na fila do estado de espera:
 - **A**
- Pergunta: Como ficarão as filas e o processo em execução de acordo com determinados eventos.





Evento	Fila de pronto	Execução	Fila de Espera
-	J-> I-> H->G->F->E->D->C	B	A
Fim de B Escalonamento de C	J-> I-> H->G->F->E->D	C	A
Fim de C Escalonamento de D	J-> I-> H->G->F->E	D	A
Fim de D Escalonamento de E	J-> I-> H->G->F	E	A
A obtêm recurso que aguardava	A ->J-> I-> H->G->F	E	
Fim de E Escalonamento de F	A ->J-> I-> H->G	F	

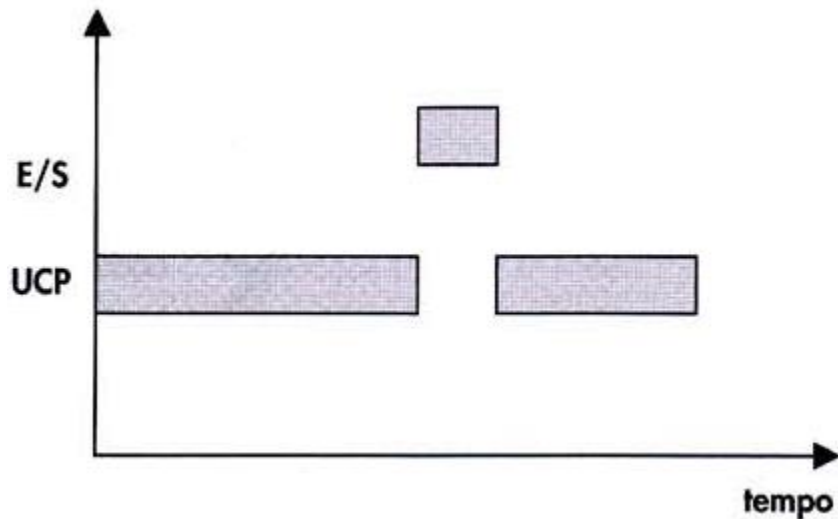


A

J-> I-> H->G->F->E->D->C

Processo F aguardando recurso Escalonamento de G	A ->J-> I-> H	G	F
Processo G aguardando recurso Escalonamento de H	A ->J-> I	H	G->F
Fim de H Escalonamento de I	A ->J	I	G->F
F obtêm recurso que aguardava	F->A ->J	I	G
G obtêm recurso que aguardava	G->F->A ->J	I	
Fim de I Escalonamento de J	G->F->A	J	
Fim de J			

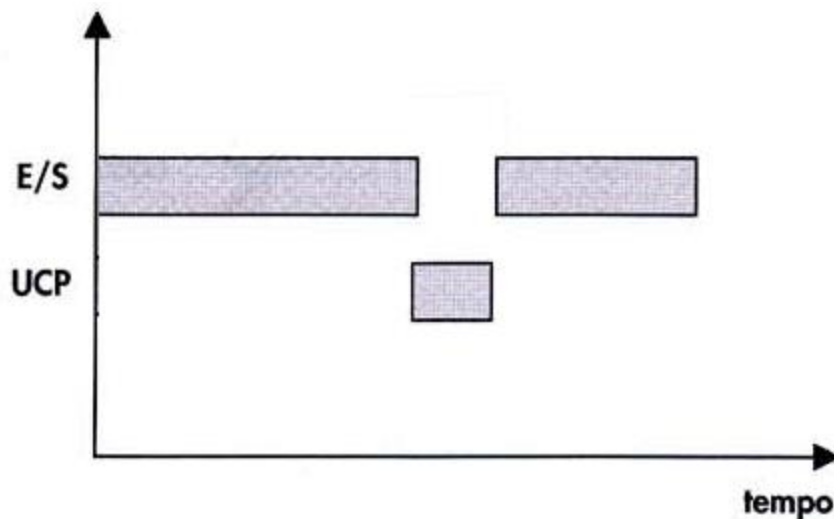
Processos CPU-bound (ligado à UCP)



Quando passa a maior parte do tempo no estado de execução, utilizando o processador, ou em estado de pronto.

Aplicações científicas que realizam muitos cálculos

Processos I/O-bound (ligado à E/S)



Quando passa a maior parte do tempo no estado de espera, pois realiza um elevado número de operações de entrada e saída.

Aplicações comerciais que se baseiam em leitura, processamento e gravação

Processo Foreground

Permite a comunicação direta do usuário com o processo durante o processamento (processamento iterativo)

(a) Processo Foreground



Processo Background

Não existe a comunicação com o usuário durante o processamento

(b) Processo Background



Forma de Criação de um processo

- Logon Interativo
- Linguagem de comandos
- Usando rotinas do Sistema Operacional

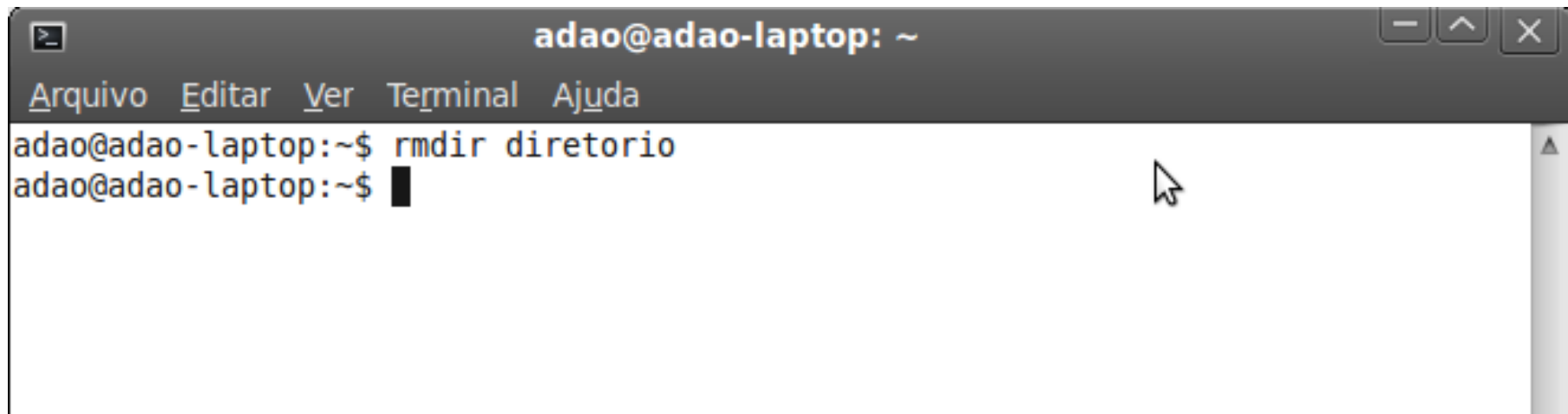
Formas de Criação de Processo (logon Interativo)

```
root@mrtg:~  
login as: root  
root@10.3.80.31's password:  
Last login: Thu Aug 17 18:13:15 2006 from 10.3.80.231  
[root@mrtg root]#
```

O usuário fornece ao sistema um nome (*username*) e uma senha (*password*) e o sistema faz a autenticação

Quando se faz o logon, um processo é criado

Formas de Criação de Processo (Via Linguagem de Comandos)



A terminal window titled 'adao@adao-laptop: ~' with standard window controls (minimize, maximize, close) in the top right. The menu bar includes 'Arquivo', 'Editar', 'Ver', 'Terminal', and 'Ajuda'. The terminal shows the command 'rmdir diretorio' being executed, with the prompt changing from 'adao@adao-laptop:~\$' to 'adao@adao-laptop:~\$' after the command. A mouse cursor is visible on the right side of the terminal area.


```
adao@adao-laptop:~$ rmdir diretorio
adao@adao-laptop:~$
```

Um processo é criado para atender ao comando de eliminação do diretório

Formas de Criação de Processo (Usando rotina do Sistema Operacional)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
    int i,j;
    pid_t filho;
    filho = fork();
    if (filho == 0)
    {
        //O código aqui dentro será executado no processo filho
        printf("sou o processo filho.\n");
        for (i=0;i<50;i++){
            printf("%d: ",i); printf("sou o processo filho\n");
            sleep(1);
        }
    }
    else
    {
        //O código neste trecho será executado no processo pai
        printf("sou o processo pai\n");
        for (j=51;j<100;j++){
            printf("%d: ",j); printf("sou o processo pai\n");
            sleep(1);
        }
    }
    printf("\n Esta regioao sera executada por ambos processos\n\n");
    exit(0);
}
```

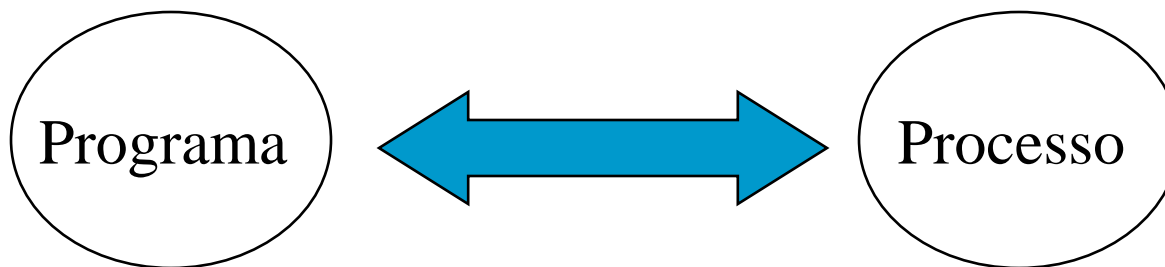
**Rotina de criação de um subprocesso
filho**

- 
- Processo
 - Subprocesso
 - Threads

PROCESSO

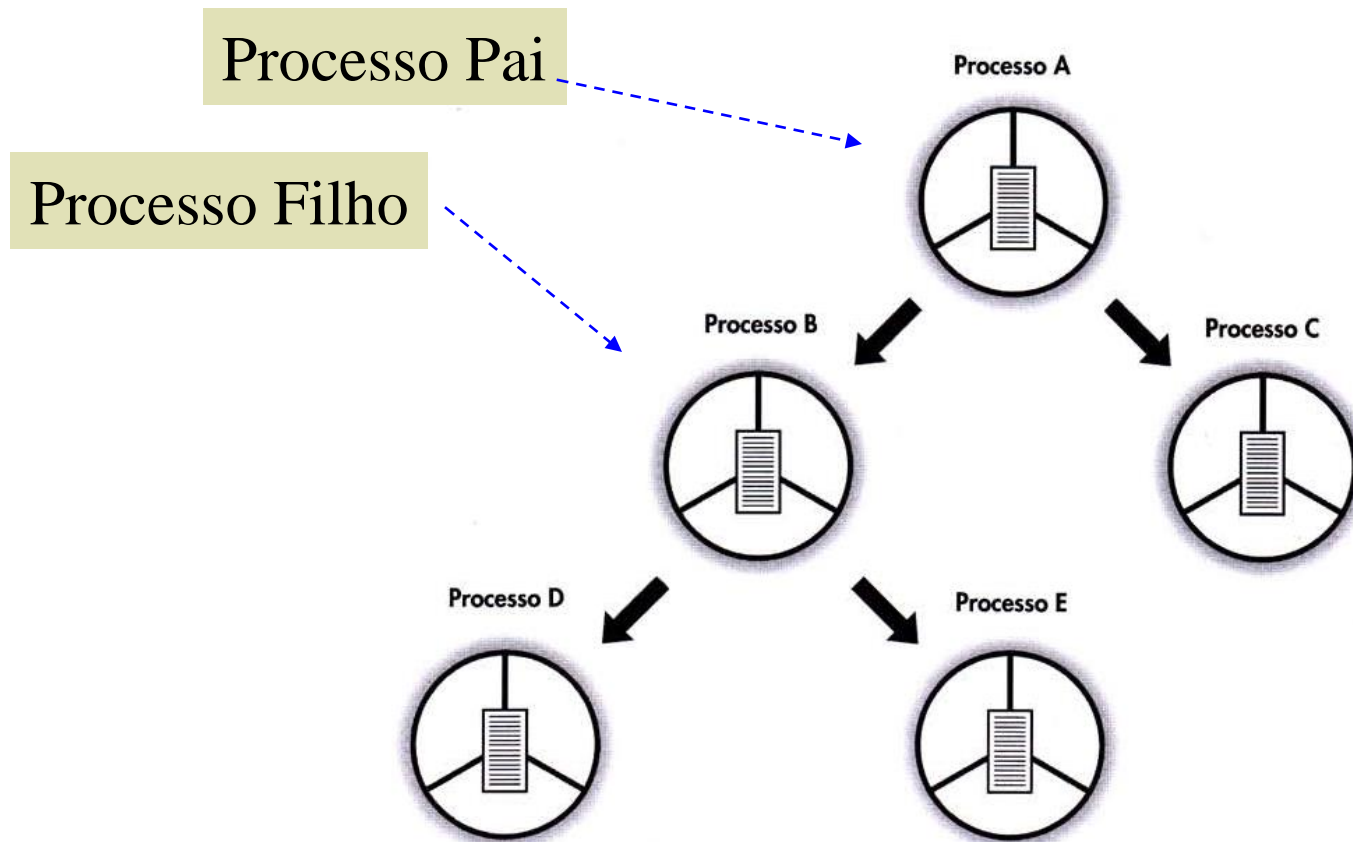
Forma de implementar a concorrência entre programas pelo uso dos recursos do sistema.

Cada programa ao ser criado já está associado a um processo



SUBPROCESSOS

Dependência existencial entre processo pai e processo filho
Cada um possui seu próprio contexto de hardware, contexto de software e espaço de endereçamento



Como criar um subprocesso

Criar um subprocesso filho

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
    int i,j;
    pid_t filho;
    filho = fork();
    if (filho == 0)
    {
        //O código aqui dentro será executado no processo filho
        printf("sou o processo filho.\n");
        for (i=0;i<50;i++){
            printf("%d: ",i); printf("sou o processo filho\n");
            sleep(1);
        }
    }
    else
    {
        //O código neste trecho será executado no processo pai
        printf("sou o processo pai\n");
        for (j=51;j<100;j++){
            printf("%d: ",j); printf("sou o processo pai\n");
            sleep(1);
        }
    }
    printf("\n Esta regioao sera executada por ambos processos\n\n");
    exit(0);
}
```

- A rotina `fork()` cria um novo processo, que executará o mesmo código do programa

- Retorna

- o **PID** do processo criado para o pai
- **0** para o filho

- O processo filho imprime de 0 a 49

- O processo pai de 51 a 99

Como criar um subprocesso

aluno@ubuntu: ~/Downloads

```
aluno@ubuntu:~$ ls
```

```
Desktop      Downloads      Music          Public         Videos
```

```
Documents    examples.desktop Pictures        Templates
```

```
aluno@ubuntu:~$ cd Downloads/
```

```
aluno@ubuntu:~/Downloads$ ls
```

```
fork01.c  fork02.C  threads.c
```

```
aluno@ubuntu:~/Downloads$ gcc fork01.c -o fork01
```

```
aluno@ubuntu:~/Downloads$ ./fork01
```

execução

compilação

Como criar um subprocesso

```
aluno@ubuntu:~/Downloads$ ./fork01
```



execução

```
sou o processo pai
```

```
51: sou o processo pai
```

```
sou o processo filho.
```

```
0: sou o processo filho
```

```
52: sou o processo pai
```

```
1: sou o processo filho
```

```
53: sou o processo pai
```

```
2: sou o processo filho
```

```
54: sou o processo pai
```

```
3: sou o processo filho
```

```
55: sou o processo pai
```

```
4: sou o processo filho
```

```
56: sou o processo pai
```

```
5: sou o processo filho
```

```
57: sou o processo pai
```

```
6: sou o processo filho
```

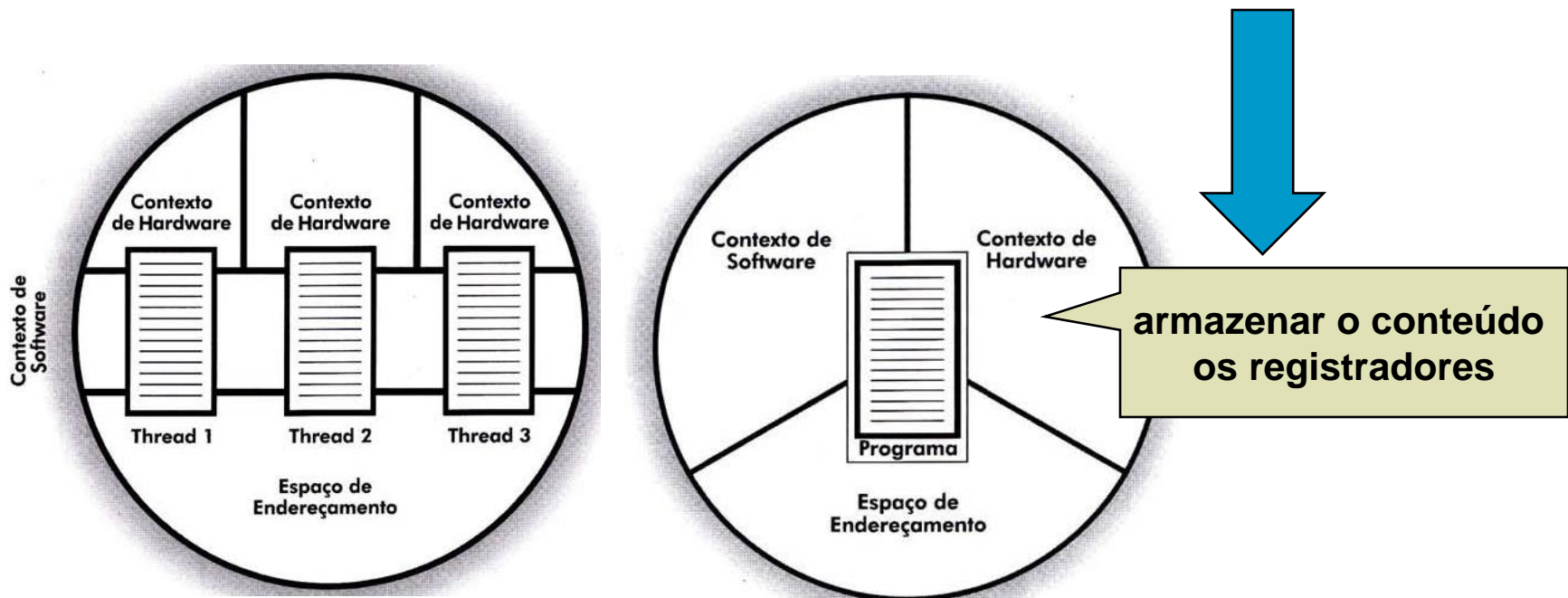

THREADS

UM PROCESSO PODE ARMAZENAR VÁRIAS THREADS

THREADS (objetivos)

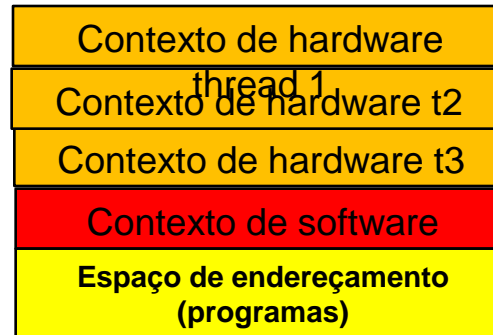
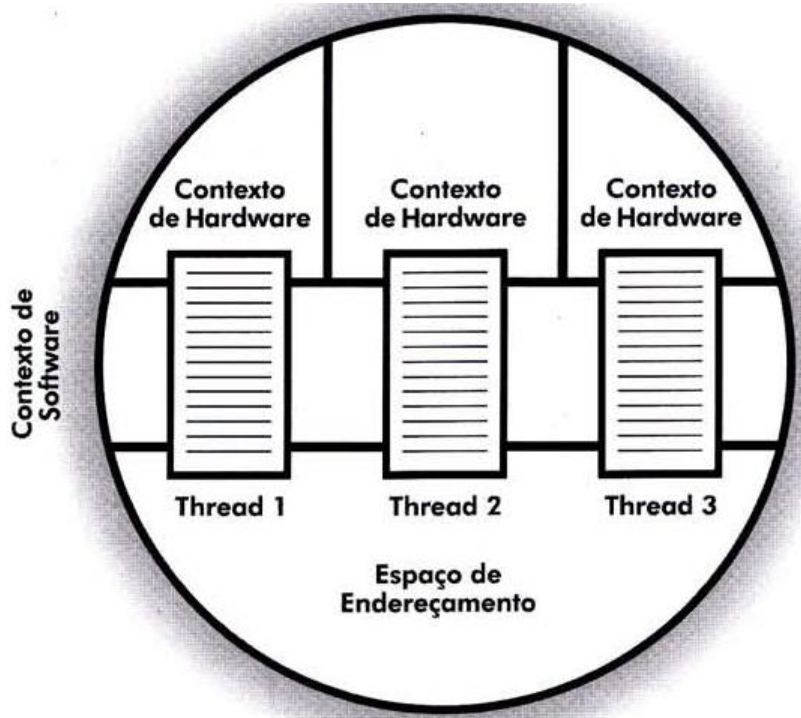
- Reduzir o tempo gasto na criação/eliminação de processos
- Reduzir o tempo gasto na troca de contexto em processos
- Economizar recursos do sistema como um todo

Compartilham o mesmo contexto de software e espaço de endereçamento, mas possuem contexto de hardware distintos

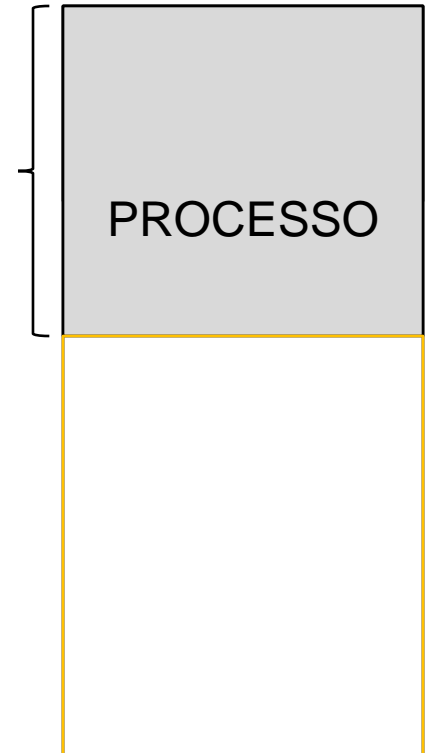


THREADS

UM PROCESSO PODE ARMAZENAR VÁRIAS THREADS



MEMÓRIA
PRINCIPAL



Programação Multithreads

```
//gcc threads.c -lpthread
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
struct valor{
    int tempo;
    int id;
};
void *espera(void *tmp){
    struct valor *v = (struct valor *) tmp;
    sleep(v->tempo);
    printf("Oi eu sou a thread %d esperei %d segundos antes de executar\n",v->id,v->tempo);
}
int main(){

    pthread_t linhas[10];
    int execute,i;
    struct valor *v;
    srand(time(NULL));
    for (i=0;i<10;i++){
        v = (struct valor *) malloc(sizeof(struct valor *));
        v->tempo = (rand()%10)+2;
        v->id = i;
        printf("Criei a thread <%d> com tempo <%d>\n",i,v->tempo);
        execute = pthread_create(&linhas[i],NULL,espera,(void *)v);
    }
    pthread_exit(NULL);
}
```

•O programa como um todo é está associado a um processo e dentro deste processo são criadas 10 threads.

•Rotina de criação das threads. São criadas 10 threads que executam cada uma a rotina espera.

Programação Multithreads

aluno@ubuntu: ~/Downloads

```
aluno@ubuntu:~$ cd Downloads/
```

```
aluno@ubuntu:~/Downloads$ ls
```

```
divbyzero divbyzero.c fork01 fork01.c fork02.C threads.c
```

```
aluno@ubuntu:~/Downloads$ gcc -pthread threads.c -o threads
```

```
aluno@ubuntu:~/Downloads$
```



Opção de compilação



compilação

Programação Multithreads

```
aluno@ubuntu:~$ cd Downloads/  
aluno@ubuntu:~/Downloads$ ls  
divbyzero divbyzero.c fork01 fork01.c fork02.C threads.c  
aluno@ubuntu:~/Downloads$ gcc -pthread threads.c -o threads  
aluno@ubuntu:~/Downloads$ ./threads → execução  
Criei a thread <0> com tempo <11>  
Criei a thread <1> com tempo <7>  
Criei a thread <2> com tempo <10>  
Criei a thread <3> com tempo <7>  
Criei a thread <4> com tempo <6>  
Criei a thread <5> com tempo <9>  
Criei a thread <6> com tempo <4>  
Criei a thread <7> com tempo <10>  
Criei a thread <8> com tempo <9>  
Criei a thread <9> com tempo <8>  
Oi eu sou a thread 6 esperei 4 segundos antes de executar  
Oi eu sou a thread 4 esperei 6 segundos antes de executar  
Oi eu sou a thread 3 esperei 7 segundos antes de executar  
Oi eu sou a thread 1 esperei 7 segundos antes de executar  
Oi eu sou a thread 9 esperei 8 segundos antes de executar  
Oi eu sou a thread 8 esperei 9 segundos antes de executar  
Oi eu sou a thread 5 esperei 9 segundos antes de executar
```

QUAL DIFERENÇA DE UMA THREAD PARA UM PROCESSO ?

- Para se fazer a mesma coisa uma thread é mais eficiente pois: reduz o tempo gasto na criação/eliminação de processos, **Reduz o tempo gasto na troca de contexto em processos e economizar recursos do sistema como um todo.**
- Criei 1 programa (que naturalmente já está associado a um processo – o processo pai) para imprimir de 51 a 99 e ele criou um processo filho (**usando o comando fork()**) que imprimiu de 1 a 49.
- Poderia ter feito isso com um programa (que naturalmente já está associado a um processo) e duas (2) threads

