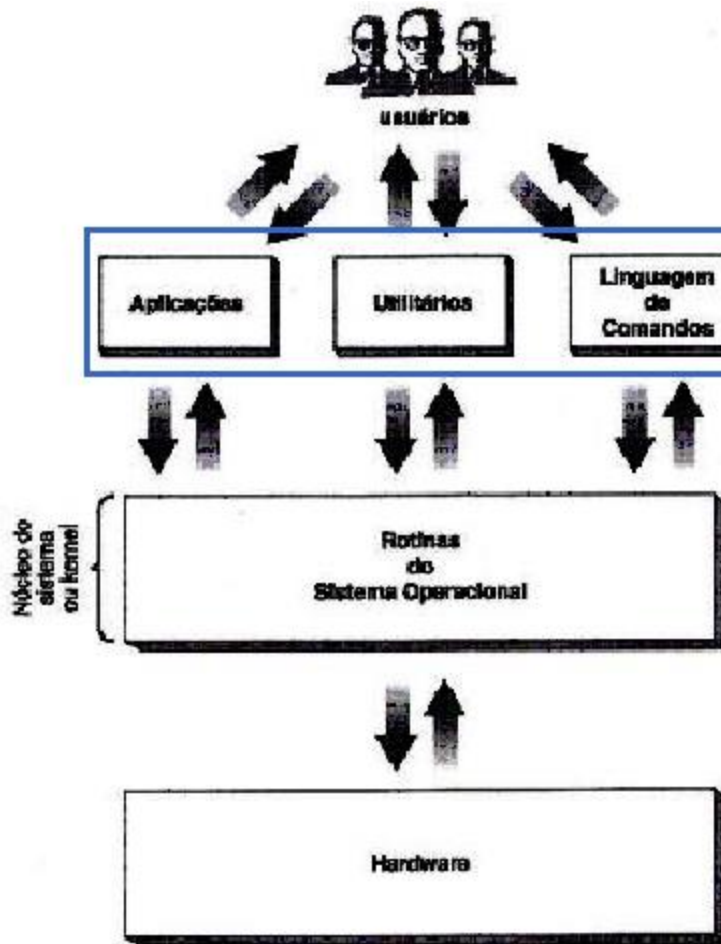


Estrutura dos Sistemas Operacionais



Sérgio Portari Júnior - 2016


Sistema Operacional



- Formas de acessar o KERNEL do SISTEMA OPERACIONAL (SO)
- A linguagem de comandos faz parte do SO

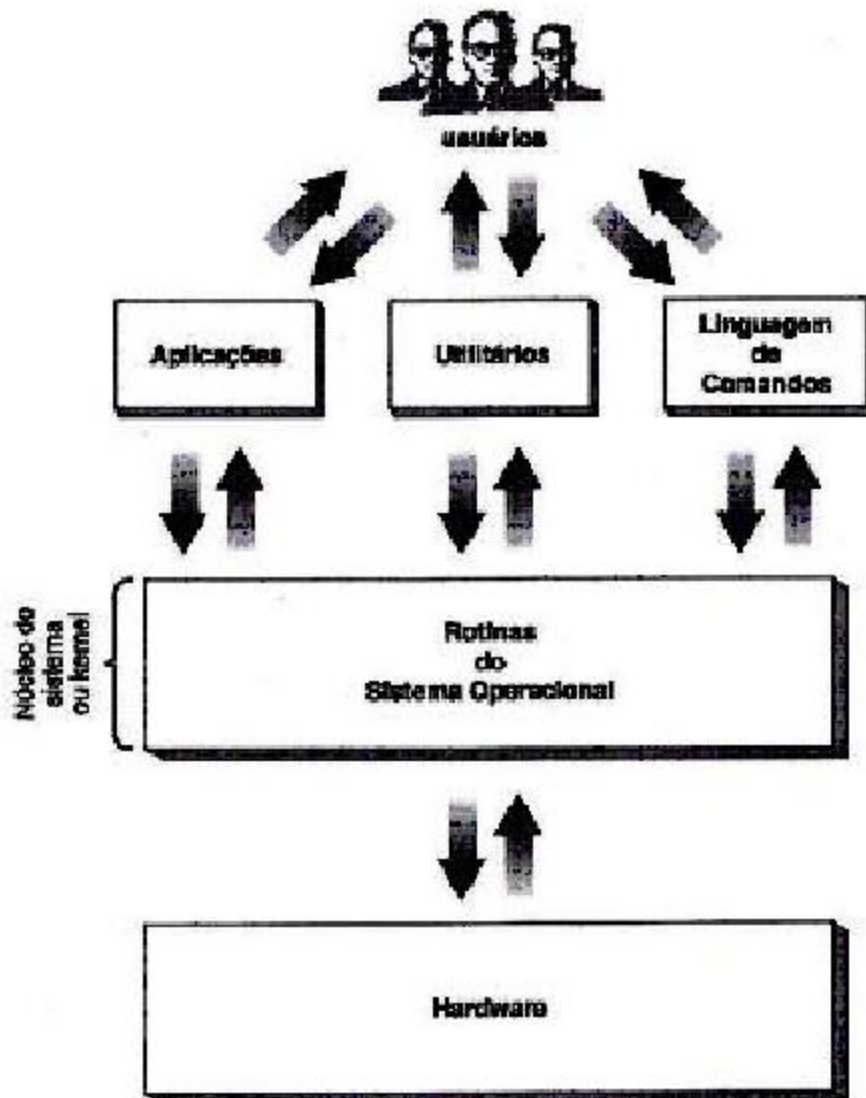
- O Sistema Operacional é formado por um Conjunto de rotinas (denominado de núcleo do sistema ou kernel) que oferece serviços aos usuários e suas aplicações

Linguagem de Comandos (revisão)

A terminal window titled "adao@adao-laptop: ~" with standard window controls (minimize, maximize, close) in the top right. The menu bar contains "Arquivo", "Editar", "Ver", "Terminal", and "Ajuda". The terminal content shows the command "rmdir diretorio" being executed, followed by a new prompt line "adao@adao-laptop:~\$" with a black cursor. A mouse cursor is visible on the right side of the terminal area.

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

Funções do Kernel



As rotinas do sistema são executadas **concorrentemente** (ao mesmo tempo) sem uma ordem pré-definida, com base em eventos dissociados do tempo (**assíncronos**)

Listagem de alguns processos (prática)



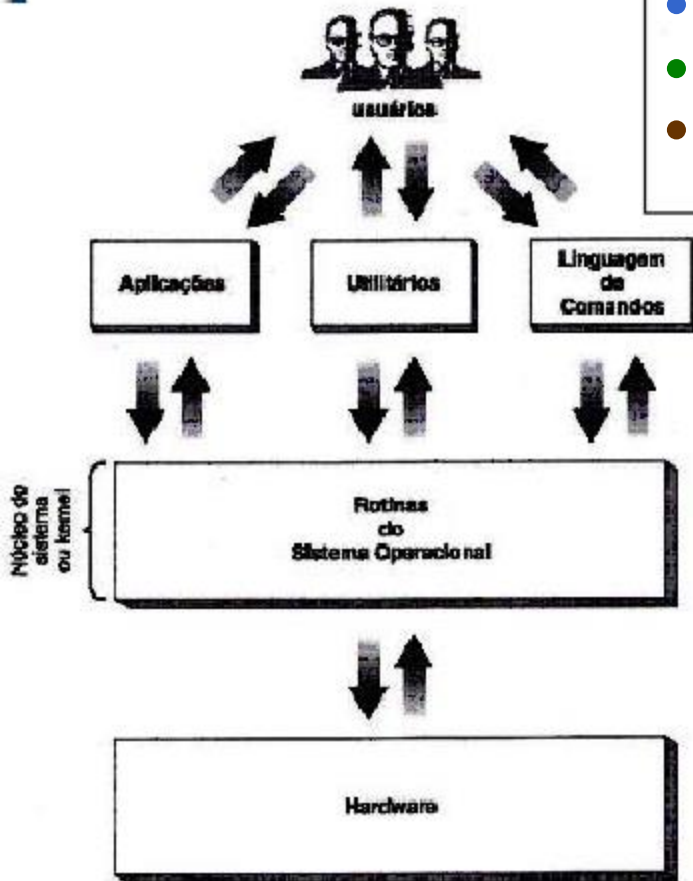
Listagem de alguns processos (rotinas) (prática)

```
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

Funções do Kernel

- Tratamento de interrupções e exceções;
- Criação, eliminação, sincronização, escalonamento e controle de processos
- Gerência da memória
- Gerência do sistema de arquivos.
- Gerências das operações de entrada e saída;
- Suporte a redes locais e distribuídas
- contabilização, auditoria e segurança do sistema



Exemplo de função do Kernel (gerência do sistema de arquivos)

The image shows a Windows XP desktop environment. In the background, there is a presentation slide titled "Aritmética Binária e Complemento a Base" from the "LPRM Laboratório de Pesquisa em" series. In the foreground, a Windows Explorer window is open to the folder "2 BIMESTRE" at the path "C:\Faculdade Dom Bosco\slides_OC\2 BIMESTRE". The window displays a list of files with columns for Name, Size, and Type. An error dialog box titled "Erro ao excluir arquivo ou pasta" is overlaid on the Explorer window, displaying a red 'X' icon and the message: "Não é possível excluir 1-ARITMETICA BINÁRIA. Ele está sendo usado por outra pessoa ou programa. Feche os programas que possam estar usando o arquivo e tente novamente." with an "OK" button.

Erro ao excluir arquivo ou pasta

Não é possível excluir 1-ARITMETICA BINÁRIA. Ele está sendo usado por outra pessoa ou programa. Feche os programas que possam estar usando o arquivo e tente novamente.

OK

2 BIMESTRE

Arquivo Editar Exibir Favoritos Ferramentas Ajuda

Endereço C:\Faculdade Dom Bosco\slides_OC\2 BIMESTRE

Nome	Tamanho	Tipo
1-ARITMETICA BINÁRIA	469 KB	Adobe Acrobat Document
2-ctIssequencias	569 KB	Adobe Acrobat Document
CIRCUITOS INTEGRADOS	421 KB	Apresentação do Microsoft Office
EMENTA DE OC	53 KB	Documento do Microsoft Office
FlipFlops	220 KB	Apresentação do Microsoft Office
MICROCOMPUTADORES	87.980 KB	Rich Text Format
Organização de Computadores	53 KB	Documento do Microsoft Office
von newman	106 KB	Apresentação do Microsoft Office

Tarefas de arquivo e pasta

- Renomear este arquivo
- Mover este arquivo
- Copiar este arquivo
- Publicar este arquivo na Web
- Enviar este arquivo por e-mail
- Imprimir este arquivo
- Excluir este arquivo

www.inf.uf... Discador OI 2 BIMESTRE 2 AVG Use... Microsoft P... Imagem - Paint 13.Exerceo... 1-ARI

Funções do Kernel

Como diversos usuários compartilham os mesmos recursos (memória, processador e dispositivos de E/S), o SO deve garantir a confiabilidade na execução concorrente de todos os programas e **NOS DADOS DOS USUÁRIOS,** além da garantia da integridade do sistema operacional.

Modos de Acesso

- Os sistemas operacionais restringem as operações executadas pelas aplicações*, por razões de segurança e estabilidade:
- Exemplo de restrição:
 - Acesso a dispositivos de hardware (disco, memória, etc ...)
 - *NOTA: *aplicações, ou um utilitário, ou um comando de linguagem de comandos*

Modos de Acesso

- Muitas implementações de segurança do núcleo de um SO e de acesso aos seus serviços utilizam o modo de acesso dos processadores.
- Modos de acesso dos processadores:
 - Mecanismo presente no hardware dos processadores
 - MODO USUÁRIO:
 - uma aplicação* só pode executar instruções não privilegiadas
(instruções que não oferecem riscos ao sistema)
 - MODO KERNEL:
 - uma aplicação* pode executar instruções não privilegiadas e privilegiadas, ou seja:
(instruções que oferecem risco ao sistema)
(exemplo: instruções que acessam dados no disco)

Modos de Acesso (exemplo de uso)

Para que **uma aplicação*** possa escrever em uma **área de memória onde encontra-se o sistema operacional**, a **aplicação*** deve estar sendo executado com o processador no modo kernel.



*NOTA: *aplicações, ou um utilitário, ou um comando de linguagem de comandos*

System Calls (Chamadas de Sistema)



Como as rotinas do sistema **possuem em seu código instruções privilegiadas**, então o processador deve estar em modo kernel para executá-las.

As System Calls são como portas de entrada para se ter acesso as rotinas do SO (ao **KERNEL** do SO).

*NOTA: *aplicações, ou um utilitário, ou um comando de linguagem de comandos*

System Calls (Chamadas de Sistema)

Uma aplicação* sempre deve executar com o processador no modo usuário.

Se uma aplicação* desejar chamar uma rotina do sistema operacional (que possui instruções privilegiadas): o mecanismo de system call verificará se a aplicação* possui os privilégios necessários.

- Em caso negativo o SO impedirá o desvio para a rotina do sistema sinalizando a aplicação* chamadora que a operação não é possível
- Em caso positivo (figura seguinte)

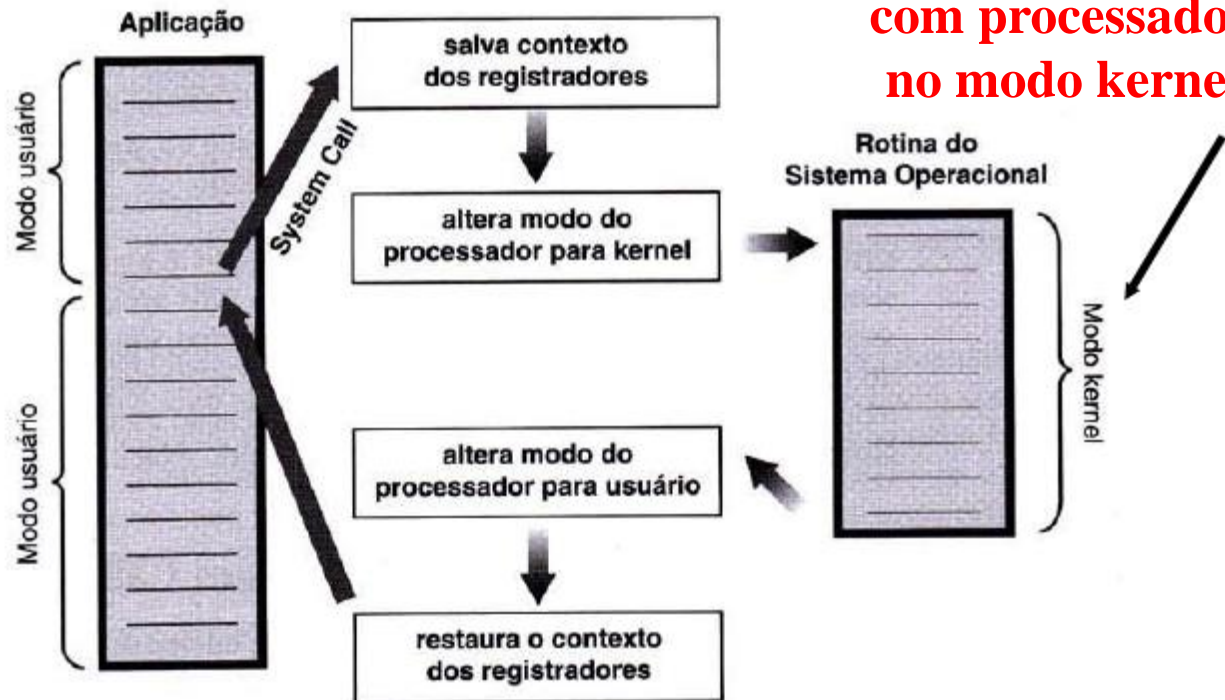


System Calls são portas de entrada para o acesso as rotinas do sistema que Possuem instruções privilegiadas (executadas no modo kernel)

Em caso positivo ...

System Calls (Chamadas de Sistema)

Aplicação*
executada
como processador
no modo usuário



*NOTA: *aplicações, ou um utilitário, ou um comando de linguagem de comandos*

Em caso negativo ...

LPRM

Laboratório de Pesquisa em

Aritmética Binária e Complemento a Base

Erro ao excluir arquivo ou pasta



Não é possível excluir 1-ARITMETICA BINÁRIA. Ele está sendo usado por outra pessoa ou programa.

Feche os programas que possam estar usando o arquivo e tente novamente.

OK

2 BIMESTRE

Arquivo Editar Exibir Favoritos Ferramentas Ajuda

Endereço C:\Faculdade Dom Bosco\slides_OC\2 BIMESTRE

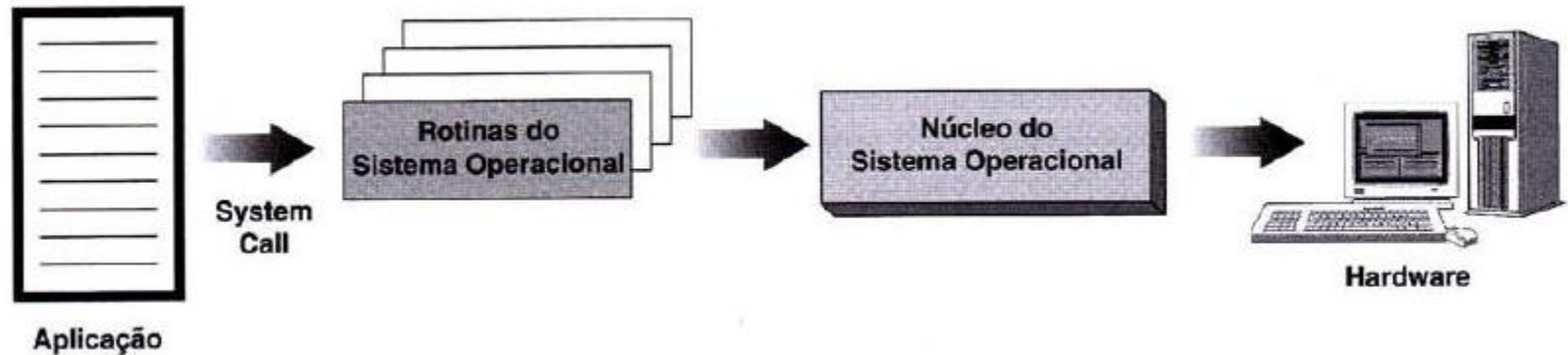
Tarefas de arquivo e pasta

- Renomear este arquivo
- Mover este arquivo
- Copiar este arquivo
- Publicar este arquivo na Web
- Enviar este arquivo por e-mail
- Imprimir este arquivo
- Excluir este arquivo

Nome	Tamanho	Tipo
1-ARITMETICA BINÁRIA	469 KB	Adobe Acrobat Document
2-cktssequencias	569 KB	Adobe Acrobat Document
CIRCUITOS INTEGRADOS	421 KB	Apresentação do Microsoft Office
EMENTA DE OC	53 KB	Documento do Microsoft Office
FlipFlops	220 KB	Apresentação do Microsoft Office
MICROCOMPUTADORES	87.980 KB	Rich Text Format
Organização de Computadores	53 KB	Documento do Microsoft Office
von newman	106 KB	Apresentação do Microsoft Office

www.inf.uf... Discador OI 2 BIMESTRE 2 AVG Use... Microsoft P... Imagem - Paint 13.Exerce... 1-ARI

System Calls (Nomenclaturas)



- Unix: system Call
- OpenVMS: system Services
- MS Windows: Application Program Interface (API)

Arquiteturas do Kernel

- O projeto de um sistema operacional depende muito do hardware a ser utilizado e do tipo de SO que se deseja construir (tempo compartilhado, tempo real, etc)
- Primeiros sistemas operacionais
 - Foram desenvolvidos em linguagem assembly (IBM OS/360)
- Nos sistemas operacionais atuais
 - Grande parte escrito em linguagem C/C++ (MS Windows).
- Linguagem de alto nível
 - Vantagem:
 - O SO pode ser facilmente alterado em outra arquitetura de hardware (portabilidade do código)
 - Desvantagem:
 - Perca do desempenho

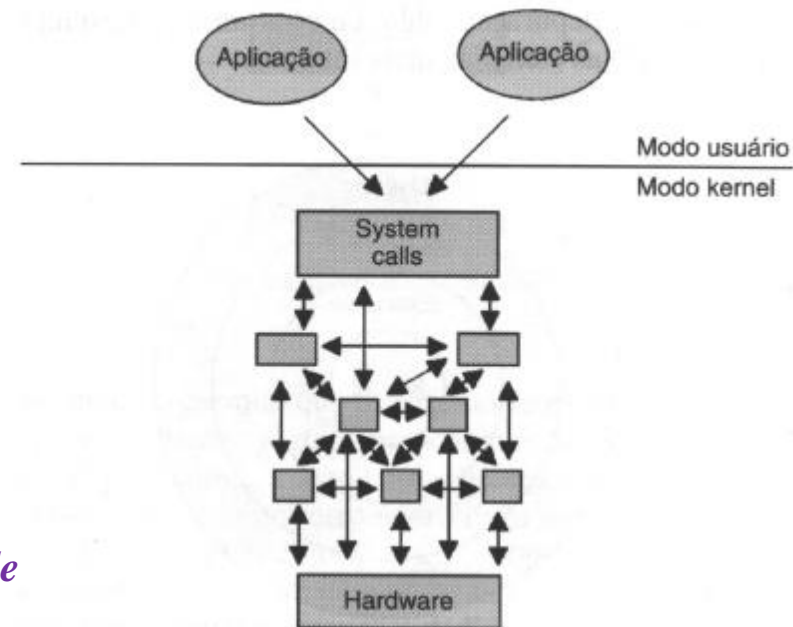
Arquiteturas do Kernel

- A maneira como o código do sistema é organizado e o inter-relacionamento entre os seus diversos componentes pode variar conforme a concepção do projeto.
- As principais arquiteturas dos SO são:
 - Arquitetura monolítica
 - Arquitetura em camadas
 - Gerência de Máquinas virtuais
 - Arquitetura microkernel

Arquitetura Monolítica

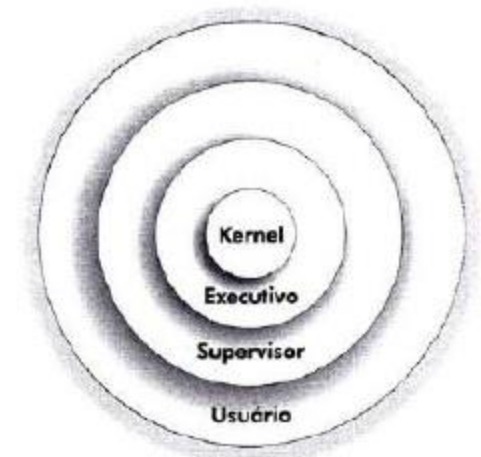
- Compara-se a uma aplicação formada por vários módulos que são **compilados*** separadamente e depois **linkados** formando um único programa executável onde os módulos podem interagir livremente.
- **Desvantagem:**
 - desenvolvimento e manutenção bastante difíceis
- **Vantagem:**
 - simplicidade e bom desempenho
- **MS-DOS e primeiros sistemas UNIX**

Compilação: transformação do código em linguagem de alto nível (C por exemplo) em código de máquina



Sistema em Camadas

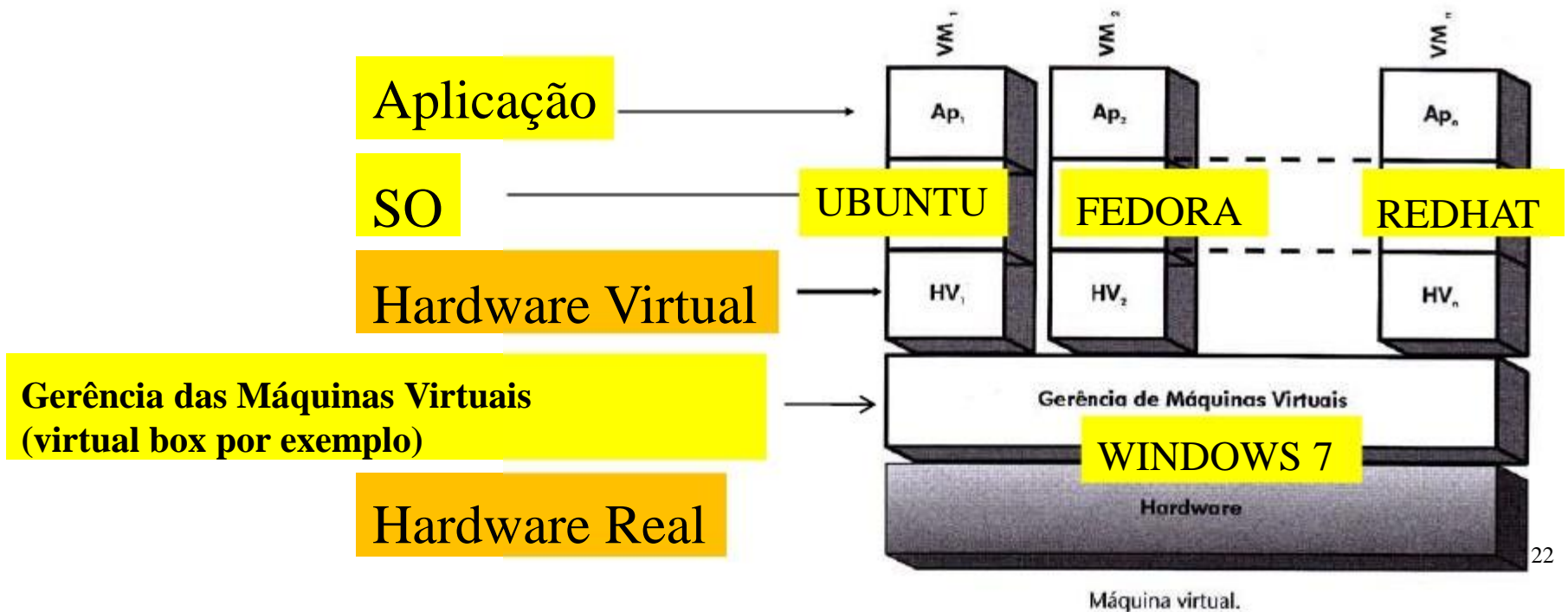
- O sistema é dividido em níveis sobrepostos
- Cada camada oferece um conjunto de funções que podem ser utilizadas apenas pelas camadas superiores.
- As camadas mais internas são mais privilegiadas que as externas.
- Vantagem:
 - Facilita a manutenção e depuração
 - Cria uma hierarquia de níveis de modos de acesso
- Desvantagem:
 - Desempenho
- Maioria das versões do UNIX e do Windows



Arquitetura em camadas do OpenVMS.

Gerência de Máquinas Virtuais

- Cria um nível intermediário entre o hardware e o sistema operacional.
- Cria várias máquinas virtuais (VM – virtual machine) independentes onde cada uma oferece uma cópia virtual do hardware. Cada VM é independente e é possível que cada VM possua seu próprio sistema operacional e que os usuários executem aplicações como se estivesse dedicado a cada um deles.



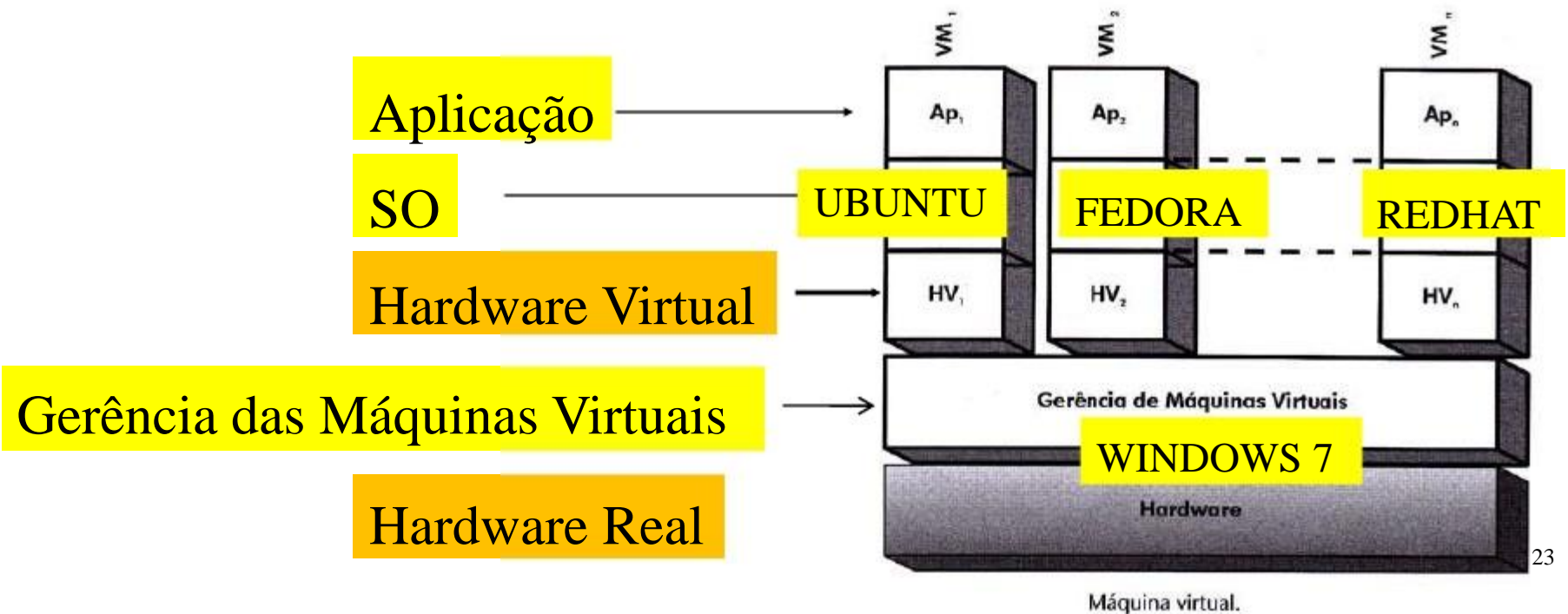
Gerência de Máquinas Virtuais

■ Vantagem

- Cria um isolamento total entre cada VM, oferecendo grande segurança para cada uma delas.

■ Desvantagem:

Necessidade de compartilhar e gerenciar recursos do hardware entre as diversas VM

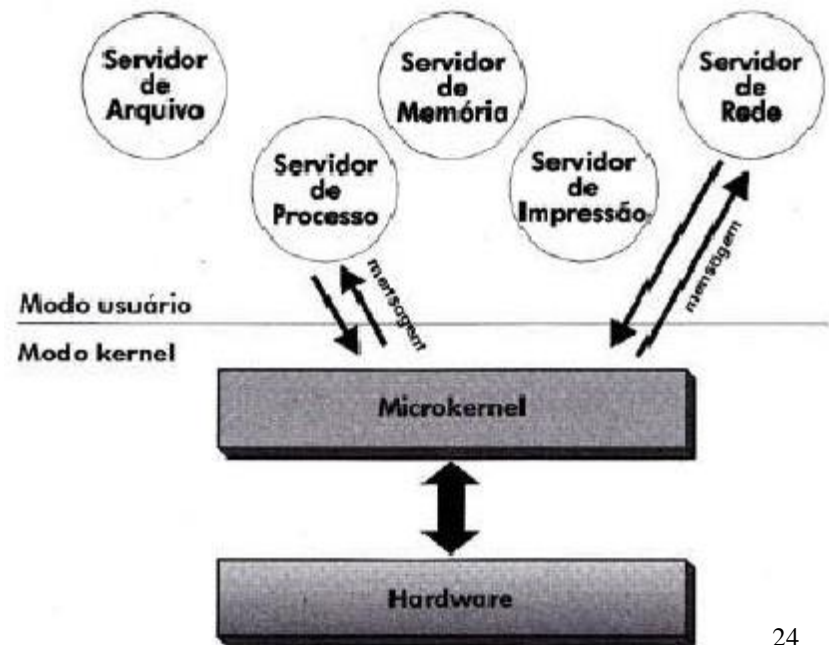


Arquitetura Microkernel

Idéia: tornar o núcleo do SO o mais simples possível.

Os serviços do sistema são disponibilizados através de processos, responsáveis por oferecer um conjunto específico de funções (gerência de arquivos, processos, de memória e escalonamento)

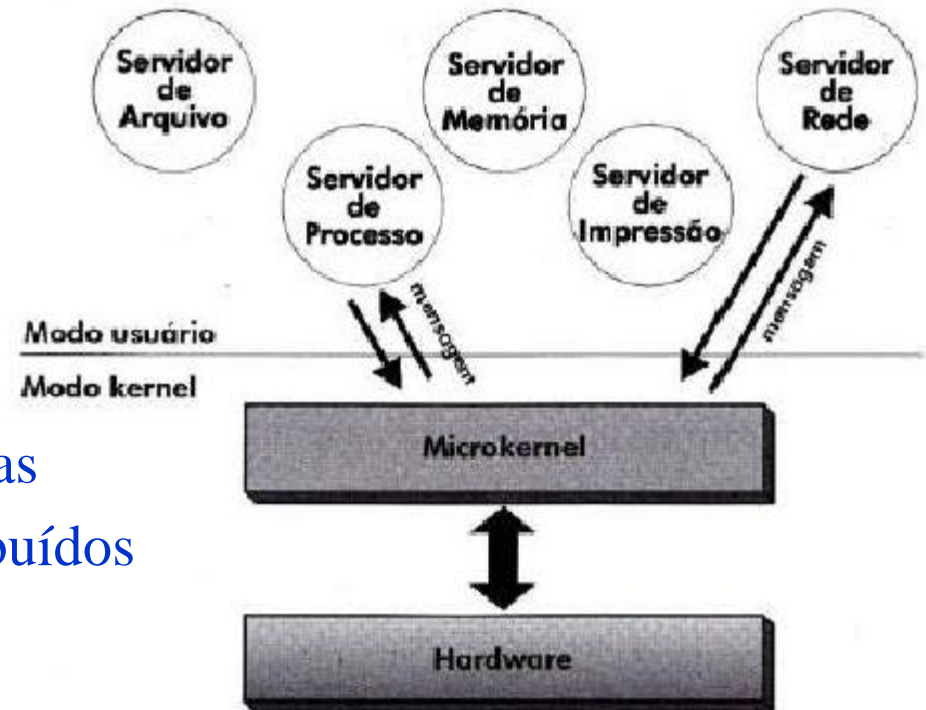
Sempre que uma aplicação deseja algum serviço, deve solicitar ao processo responsável.



Arquitetura microkernel.

Arquitetura Microkernel

- A aplicação que solicita serviço é chamada de cliente e o processo que responde é denominado de servidor.
- A principal função do núcleo é realizar a troca de mensagens entre cliente e o servidor
- Servidores: modo usuário e Núcleo: modo Kernel.
- Vantagem:
Manutibilidade, flexibilidade e portabilidade
- Desvantagem:
– Difícil implementação
- Uso: maioria das iniciativas ligadas ao desenvolvimento de SO distribuídos



Arquitetura microkernel.

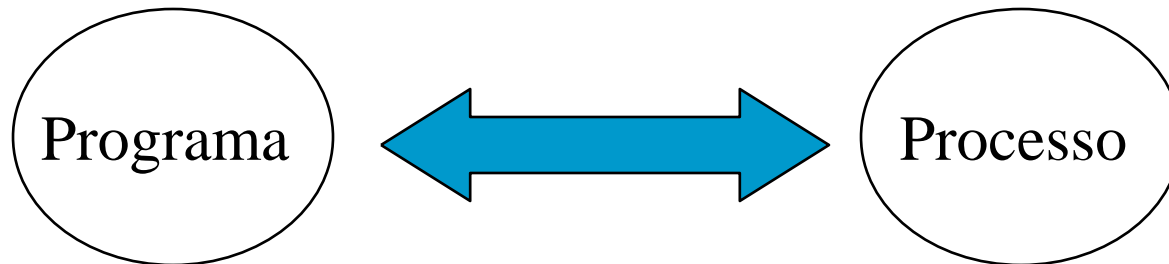


Processos

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.



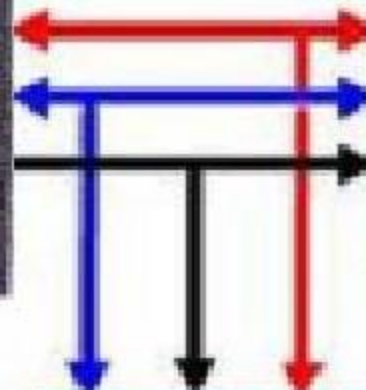
TUDO PROGRAMAAO SER CRIADO JÁ VEM ASSOCIADO A UM PROCESSO

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

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

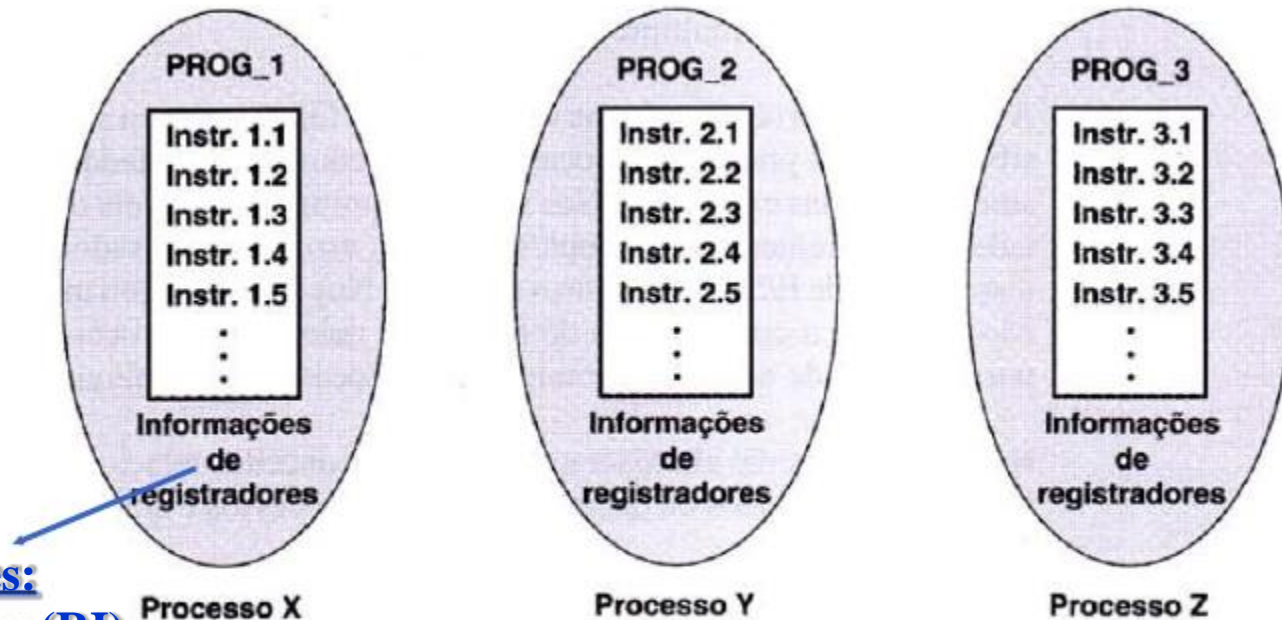
Processo e Concorrência

■ 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)

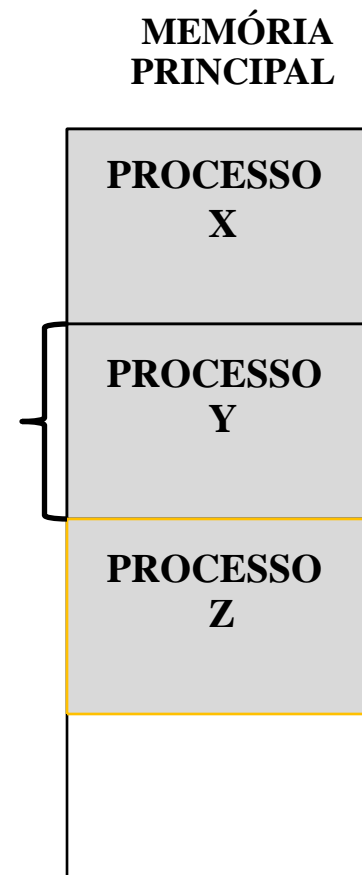
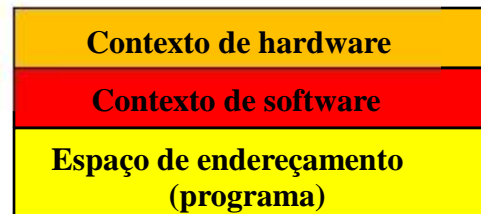
Processo e Concorrência



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

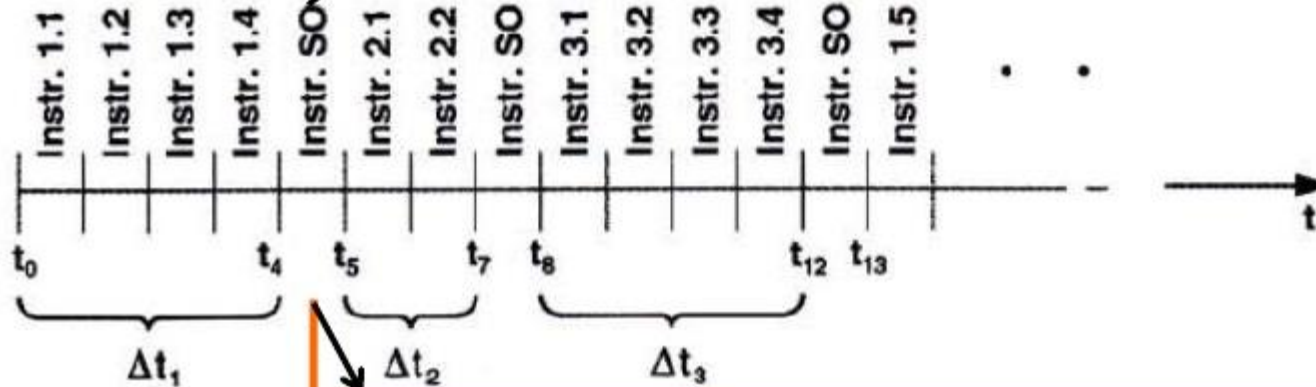


Os registradores mantêm informações sobre o programa em execução por isso suas informações precisam ser guardadas na mudança de contexto



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

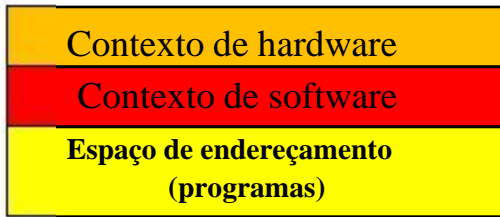
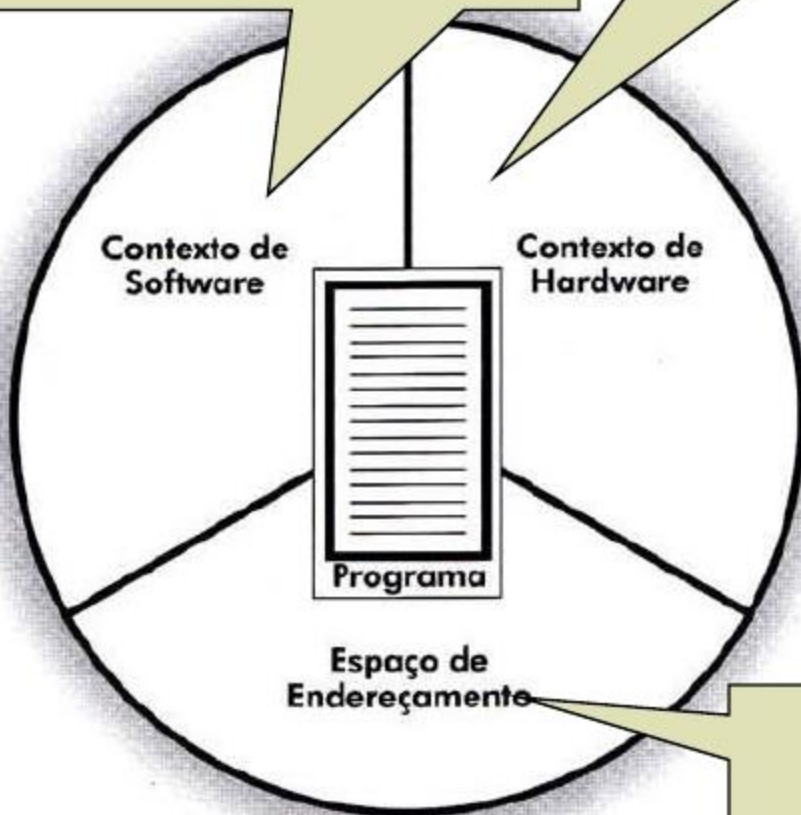
- É formado por três partes (**contexto de hardware**, **de software** e **espaço de endereçamento**) que juntas mantêm informações necessárias a execução de um programa em um sistema em que exista concorrência (multiprogramação).



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)



MEMÓRIA PRINCIPAL

PROCESSO X

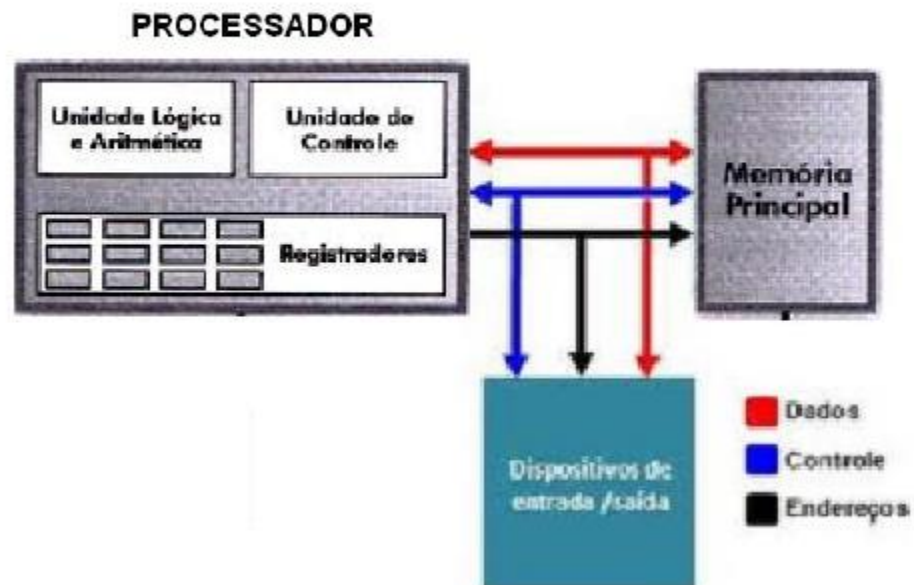
PROCESSO Y

PROCESSO Z

É a área de memória pertencente são armazenados para execução

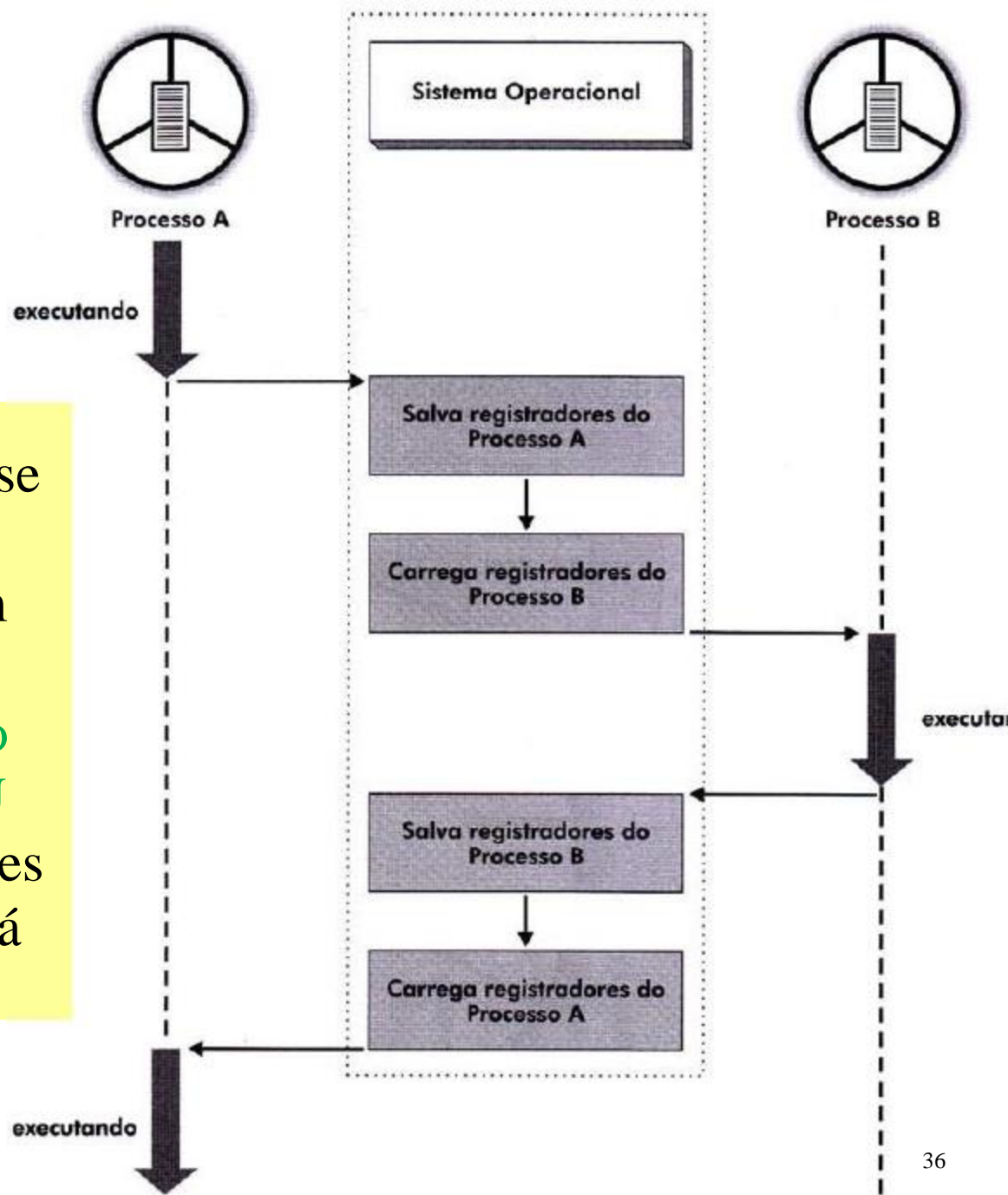
Contexto de Hardware

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

Amudança 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.



Mudança de contexto.

Contexto de Software

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



Identificação

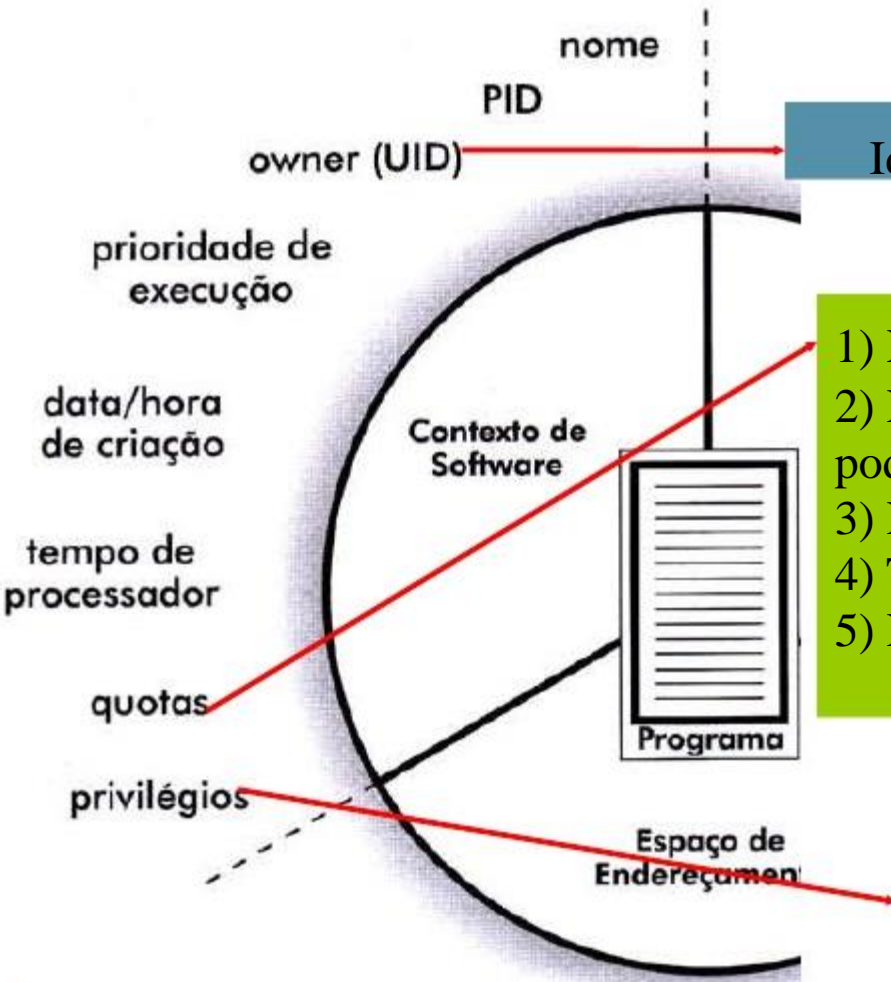
- PID (process identification)
- UID (user identification)

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



Identificação do usuário ou processo que o criou

- 1) Nr Máx de arquivos abertos
- 2) Nr Máx de Mem Pcpal e Mem Sec que o processo pode alocar
- 3) Nr Máx de operações de E/S pendentes
- 4) Tamanho máximo do buffer para operações E/S
- 5) Número máximo de (sub)processo que pode-se criar

Prioridade de execução, limites alocados nas memórias principal e secundária

Listagem de alguns processos (estação com sistema linux)

IDT USUÁRIO

IDT PROCESSO

Tempo de utilização do processador

#	ps	-l	-A	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
F	S	UID												
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	-	414	schedu	?	00:00:00	portmap
5	S	29		619	1	0	85	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	cron

Listagem de alguns processos (prática)



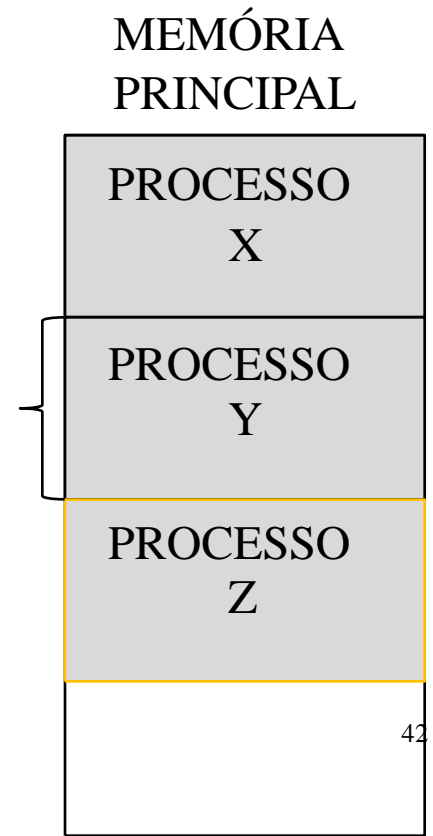
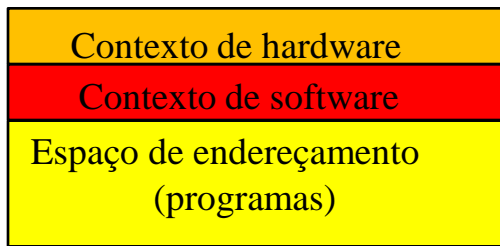
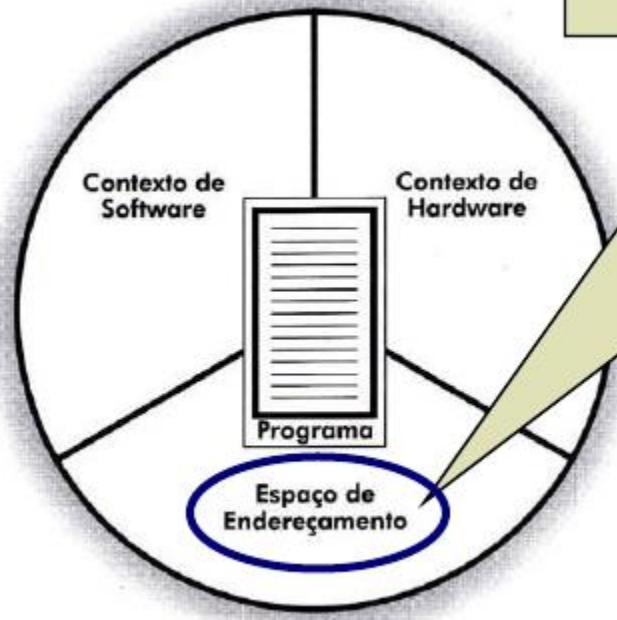
Listagem de alguns processos (prática)

```
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

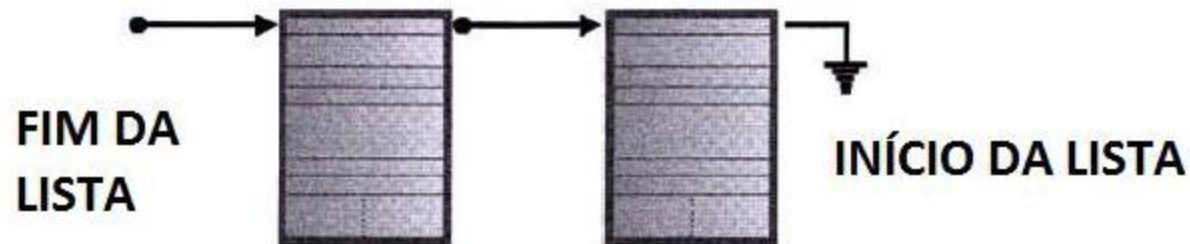
Espaço de Endereçamento

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



LISTA

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



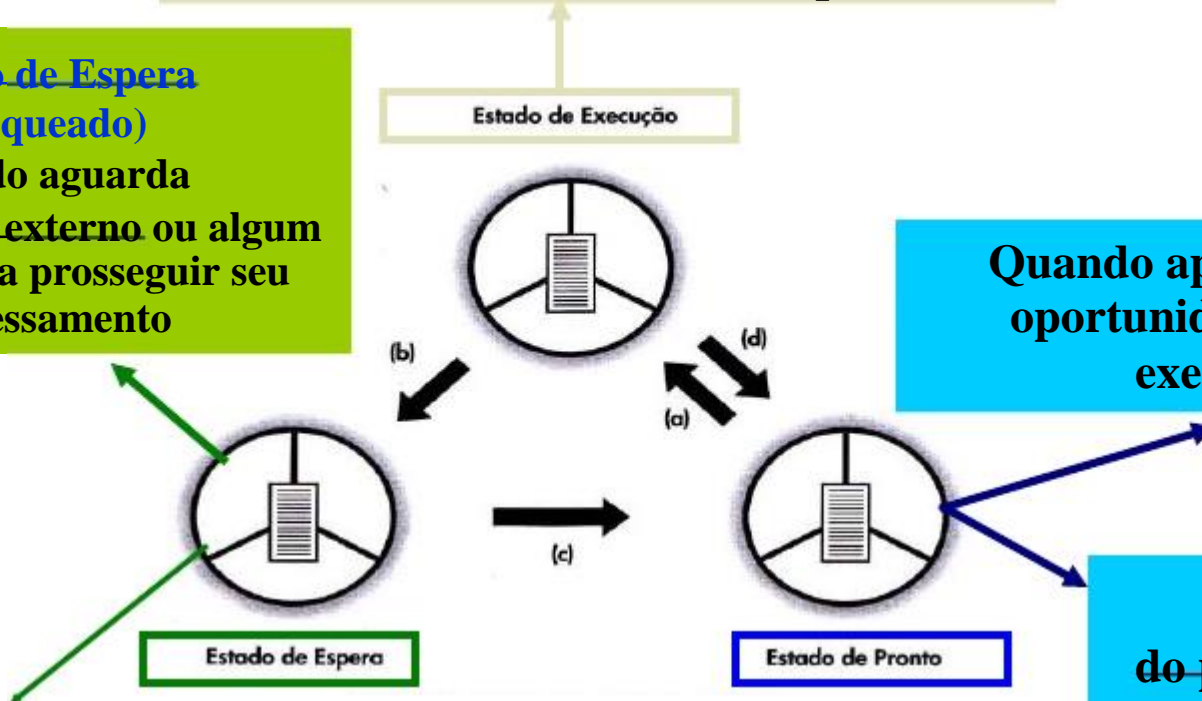
Estados de um processo

Estado de Execução
Quando está sendo executado pela CPU

Estado de Espera (bloqueado)
Quando aguarda algum evento externo ou algum recurso para prosseguir seu processamento

Quando apenas aguarda oportunidade para ser executado

Após criação do processo o mesmo vai para lista de processos em estado de pronto



EXEMPLO

Aguardando o término de operação de E/S
Aguardando data/hora para continuar operação



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)

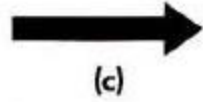
Estado de Execução

(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)

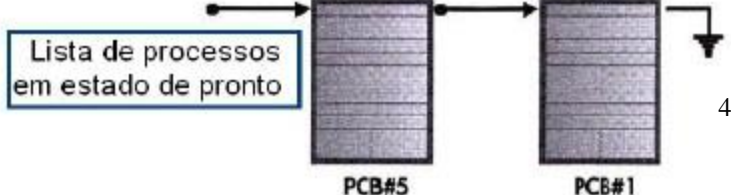
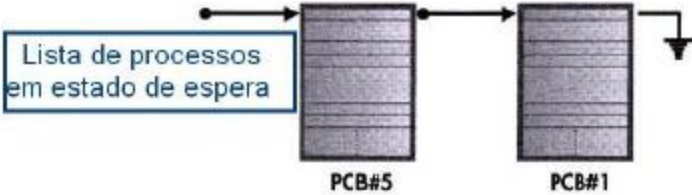


Estado de Espera



Estado de Pronto

(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.



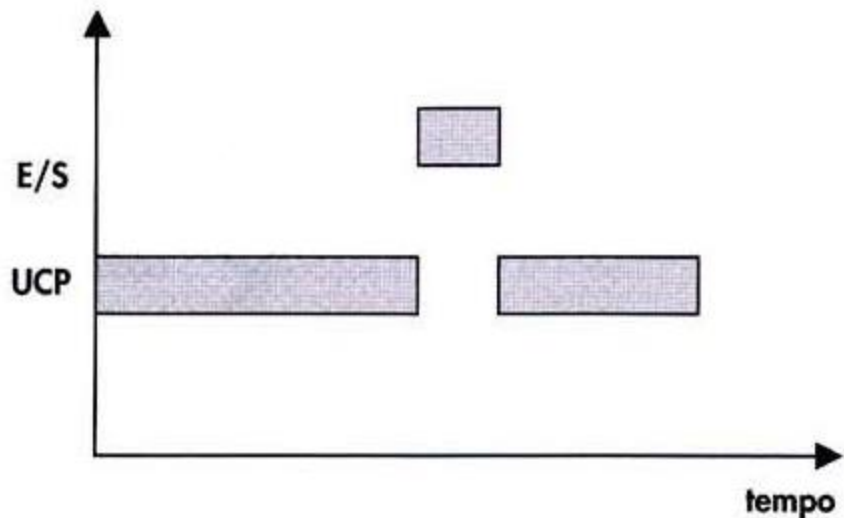


Evento	Fila de pronto	Execução	Fila de Espera
-	J->I->H->G->F->E->D->C	B	A
Fim de B Escalonamento de C	J->I->H->G->F->E->D	C	A
Fim de C Escalonamento de D	J->I->H->G->F->E	D	A
Fim de D Escalonamento de E	J->I->H->G->F	E	A
A obtêm recurso que aguardava	A->J->I->H->G->F	E	
Fim de E Escalonamento de F	A->J->I->H->G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



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			

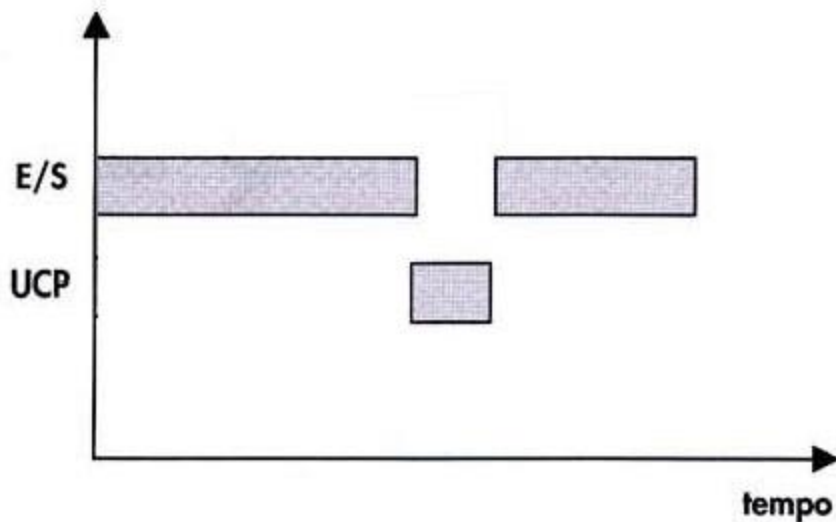
Processos CPU-bound (ligado à UCP)



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)




```
adao@adao-laptop: ~  
Arquivo Editar Ver Terminal Ajuda  
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);
}
```

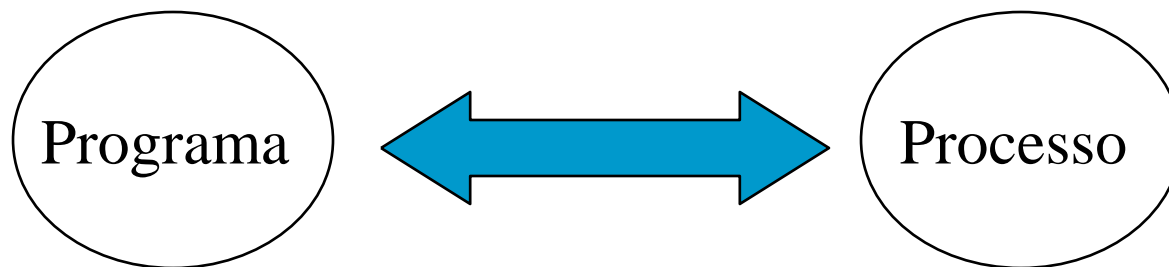
Rotina de criação de um subprocesso
filho

- 
- Processo
 - Subprocesso
 - Threads

PROCESSO

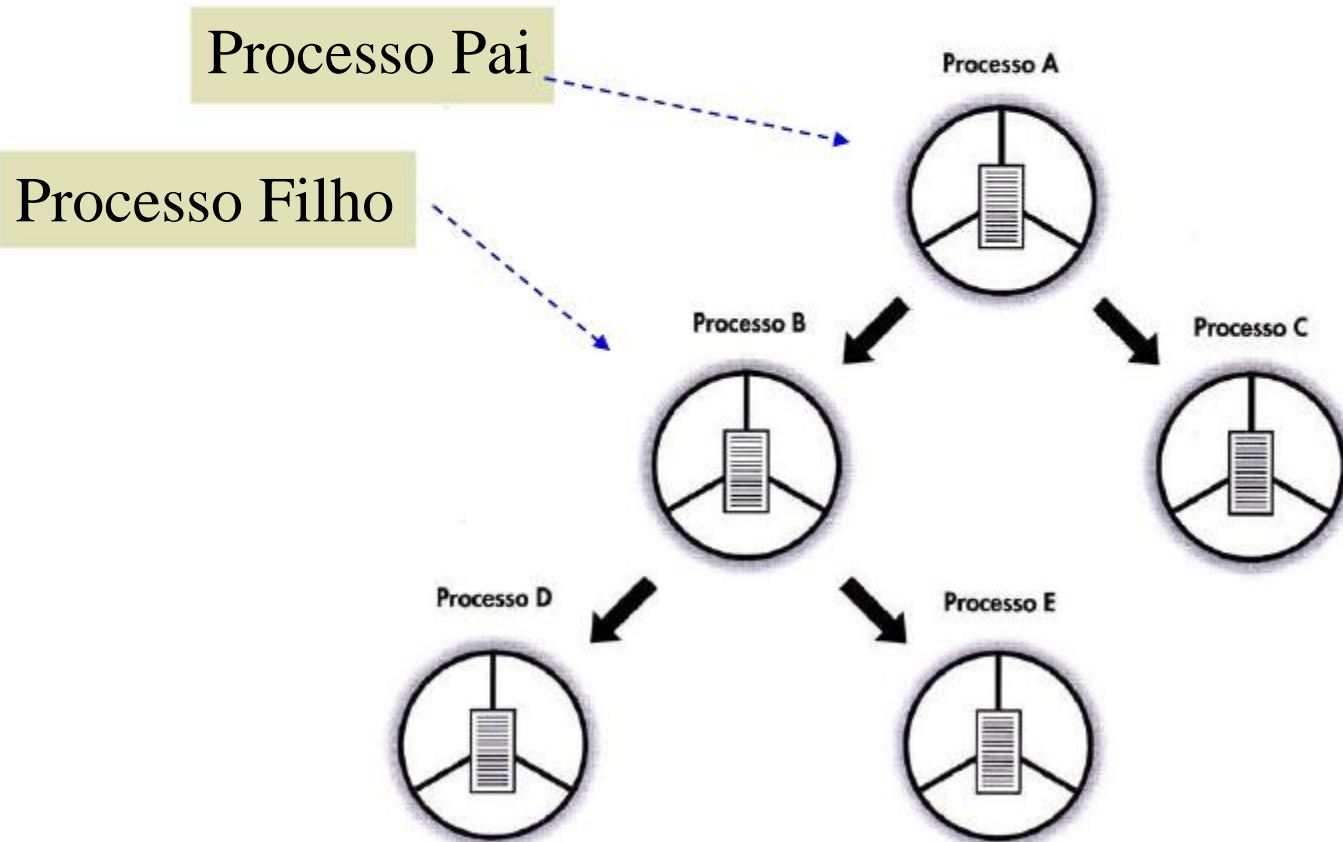
Forma de implementar a concorrência entre programas pelo uso dos recursos do sistema.

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
```

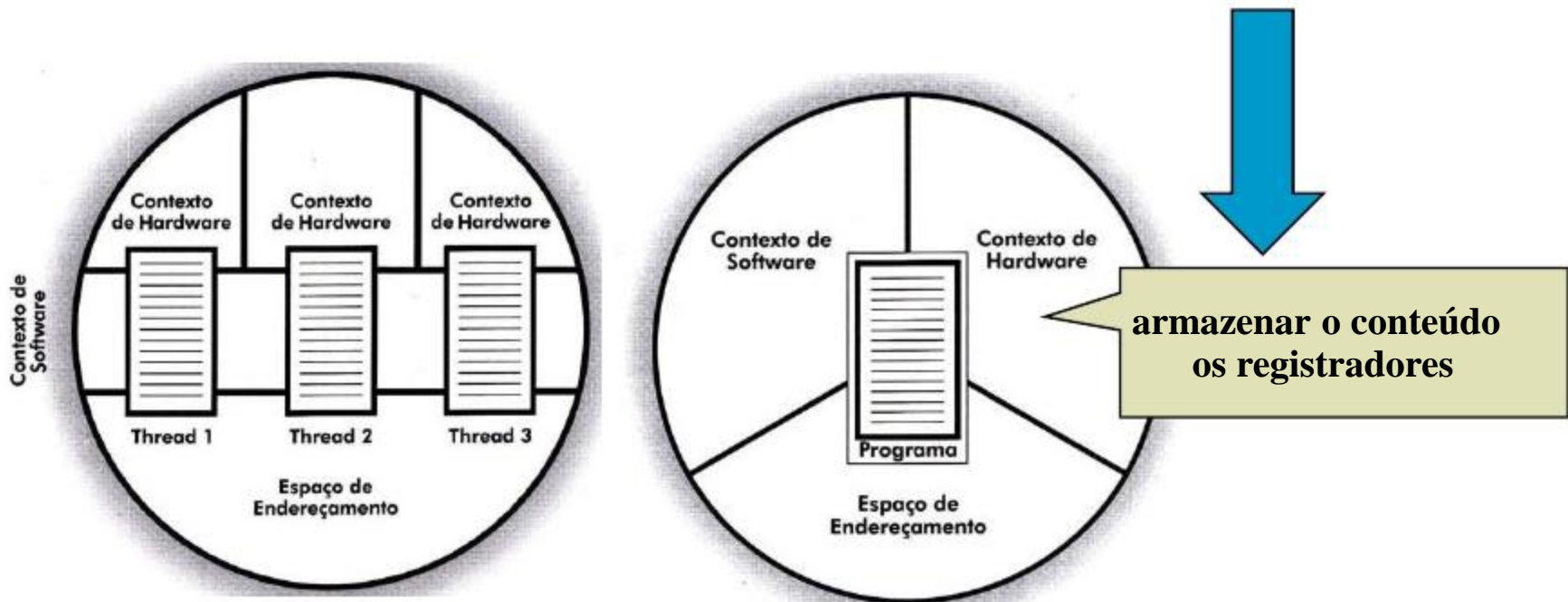
THREADS

UM PROCESSO PODE ARMAZENAR VÁRIAS THREADS

THREADS (objetivos)

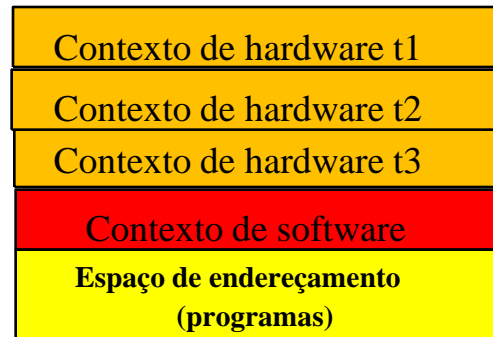
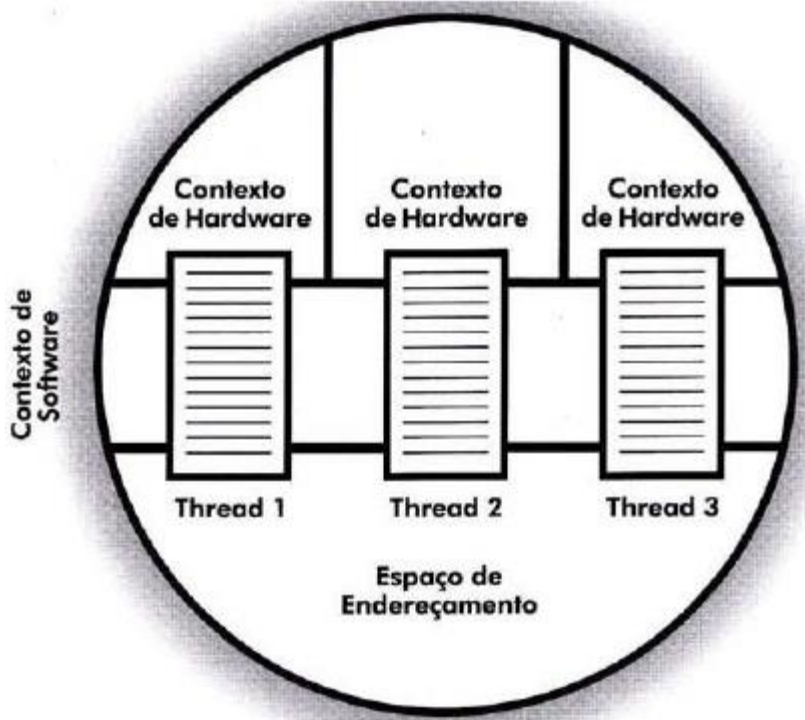
- Reduzir o tempo gasto na criação/eliminação de processos
- Reduzir o tempo gasto na troca de contexto em processos
- Economizar recursos do sistema como um todo

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

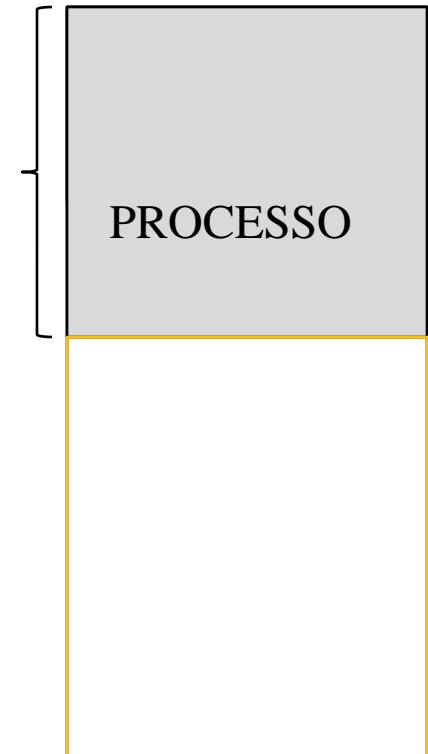


THREADS

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 UM 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

