



Universidade Tecnológica Federal do Paraná – UTFPR
Coordenação de Ciência da Computação - COCIC
Ciência da Computação

BCC34G – Sistemas Operacionais

Prof. Rogério A. Gonçalves
rogerioag@utfpr.edu.br

Aula 004

Processos

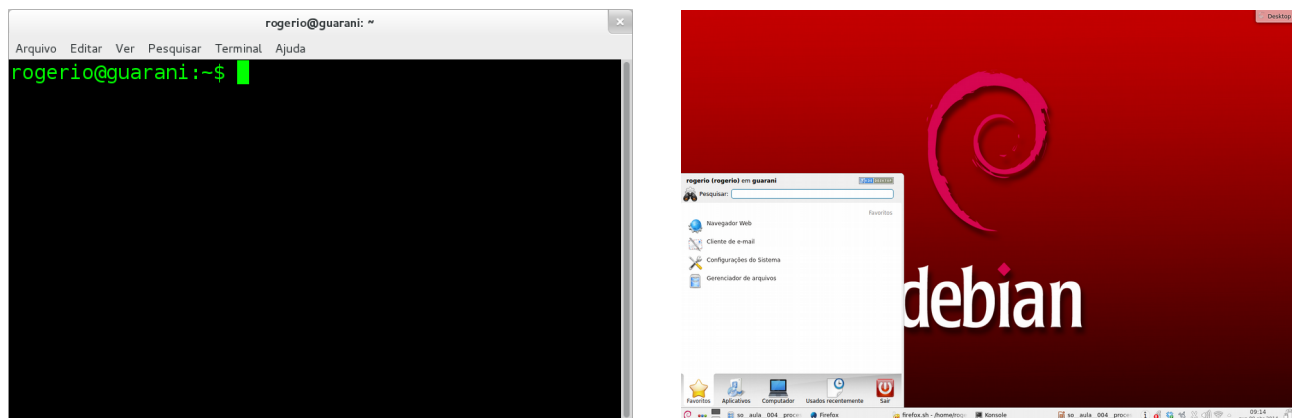
Gerenciamento de Processos

- Vimos que o SO é um gerenciador de recursos
 - Gerenciamento de Memória
 - Gerenciamento de Arquivos
 - Gerenciamento de E/S
 - Gerencia também **Aplicações**
- Aplicações → Processos
- Gerenciar Processos (Escalonar)
 - Além de: Prover Mecanismos de Sincronização e Comunicação entre Processos

Gerenciamento de Processos

- O que é um Processo?
- Quais são os tipos de Processos?
- O que pode estar associado a um Processo?
- Como representar um Processo? (BCP)
- Como gerenciar Processos? (Filas)
- Como escolher Processos? (Algoritmos)

Interação com o sistema



Chamadas de Sistema

SO

Terminal Básico

```
terminal.c x
1  #include <unistd.h>
2  #include <stdio.h>
3
4  #define TRUE 1
5
6  int main() {
7      int pid, status;
8
9      while(TRUE){
10         type_prompt();
11         read_command(command, parameters);
12
13         pid = fork();
14
15         if(pid != 0){ // Código do Pai.
16             waitpid(-1, &status, 0);
17         }
18         else { // Código do processo filho.
19             execvp(command, parameters);
20         }
21     }
22     return 0;
23 }
```

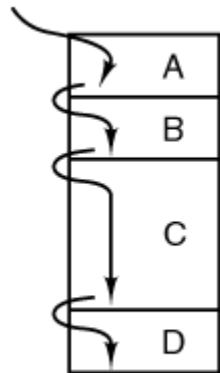
Definição de Processo

■ Um programa em execução

- Um processo tem seu próprio espaço de endereço, que consiste em:
 - **Região de texto**
 - Armazena o código que o processador executa.
 - **Região de dados**
 - Armazena variáveis e memória alocada dinamicamente.
 - **Região de pilha**
 - Armazena instruções e variáveis locais para chamadas ativas ao procedimento.

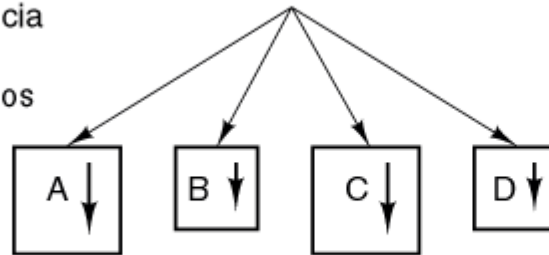
Gerência de Processos: Modelo de Processos

Um contador de programa

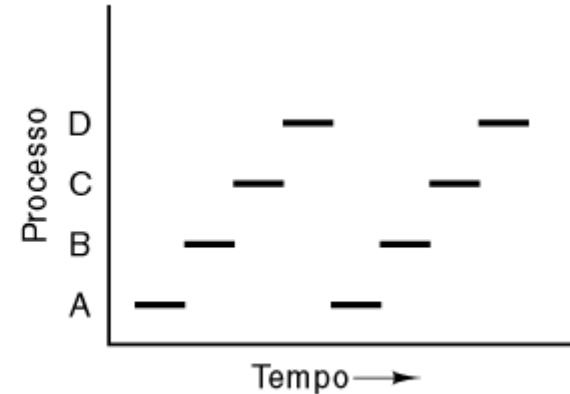


(a)

Quatro contadores de programa



(b)



(c)

- (a) Multiprogramação de quatro programas
- (b) Modelo conceitual de 4 processos sequenciais, independentes
- (c) Somente um programa está ativo a cada momento

Gerenciamento de Processos

■ Os Sistemas Operacionais prestam certos serviços fundamentais. Por exemplo:

- Criam processos.
- Destroem processos.
- Suspendem processos.
- Retomam processos.
- Mudam a prioridade de um processo.
- Bloqueiam processos.
- Acordam (ativam) processos.
- Despacham processos.
- Capacitam os processos à comunicação interprocessos (IPC).

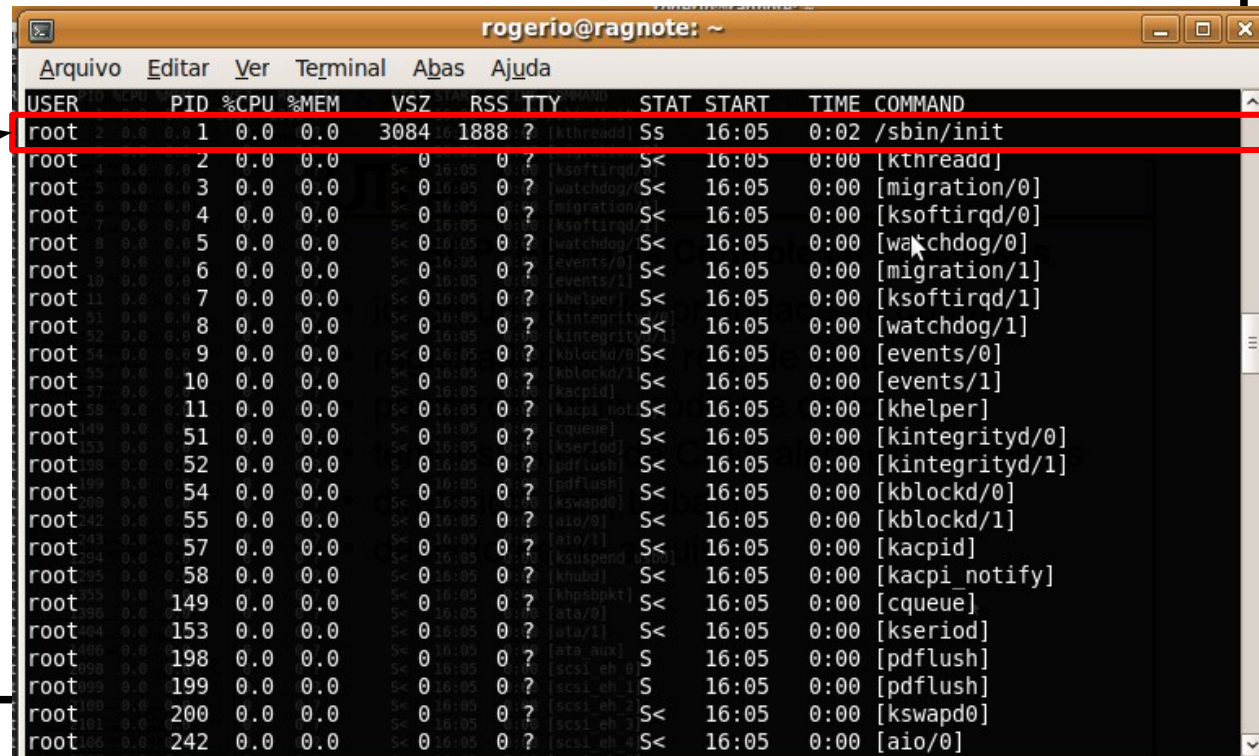
Criação de Processos

- Eventos que levam à criação de processos
 - Início do sistema
 - Execução de chamada ao sistema de criação de processos
 - Solicitação do usuário para criar um novo processo
 - Início de um *job* em lote

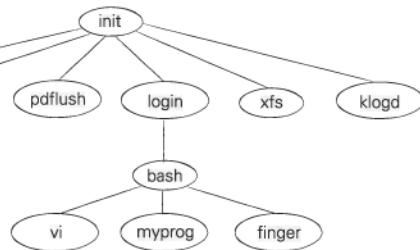
Criação de Processos

- O comando ***ps -aux*** no Linux lista os processos em execução, pela sequência de PID (Identificador) é possível saber a ordem de criação dos processos.

O ***init*** é o primeiro processo que o kernel executa.
Init é o pai de todos os processos.



```
rogerio@ragnote: ~  
Arquivo  Editar  Ver  Terminal  Abas  Ajuda  
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  
root      1  0.0  0.0   3084  1888 ?        Ss   16:05   0:02 /sbin/init  
root      2  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kthreadd]  
root      3  0.0  0.0     0     0 ?        Ss   16:05   0:00 [migration/0]  
root      4  0.0  0.0     0     0 ?        Ss   16:05   0:00 [ksoftirqd/0]  
root      5  0.0  0.0     0     0 ?        Ss   16:05   0:00 [watchdog/0]  
root      6  0.0  0.0     0     0 ?        Ss   16:05   0:00 [events/0]  
root      7  0.0  0.0     0     0 ?        Ss   16:05   0:00 [migration/1]  
root     11  0.0  0.0     0     0 ?        Ss   16:05   0:00 [ksoftirqd/1]  
root     18  0.0  0.0     0     0 ?        Ss   16:05   0:00 [events/1]  
root     21  0.0  0.0     0     0 ?        Ss   16:05   0:00 [khelper]  
root     32  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kintegrityd/0]  
root     34  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kintegrityd/1]  
root     35  0.0  0.0     0     0 ?        Ss   16:05   0:00 [events/0]  
root     37  0.0  0.0     0     0 ?        Ss   16:05   0:00 [events/1]  
root     38  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kacpid]  
root     39  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kacpi_notify]  
root     51  0.0  0.0     0     0 ?        Ss   16:05   0:00 [cqueue]  
root     52  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kseriod]  
root     54  0.0  0.0     0     0 ?        Ss   16:05   0:00 [pdflush]  
root     55  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kswapd0]  
root     57  0.0  0.0     0     0 ?        Ss   16:05   0:00 [aio/0]  
root     58  0.0  0.0     0     0 ?        Ss   16:05   0:00 [aio/1]  
root     59  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kacpid]  
root     60  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kacpi_notify]  
root    149  0.0  0.0     0     0 ?        Ss   16:05   0:00 [cqueue]  
root    153  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kseriod]  
root    198  0.0  0.0     0     0 ?        Ss   16:05   0:00 [pdflush]  
root    199  0.0  0.0     0     0 ?        Ss   16:05   0:00 [pdflush]  
root    200  0.0  0.0     0     0 ?        Ss   16:05   0:00 [kswapd0]  
root    242  0.0  0.0     0     0 ?        Ss   16:05   0:00 [aio/0]
```



Hierarquia de processos no Linux.

Término de Processos

- Condições que levam ao término de processos
 - Saída normal (voluntária)
 - A tarefa a ser executada é finalizada;
 - exit (linux), exitProcess (windows)
 - Saída por erro (voluntária)
 - Arquivo não encontrado
 - Erro fatal (involuntário)
 - Divisão por 0
 - Cancelamento por um outro processo (involuntário)
 - kill, TerminateProcess

Processos

- Processos *CPU-bound* (orientados à CPU): processos que utilizam muito o processador
 - Tempo de execução é definido pelos ciclos de processador;
- Processos *I/O-bound* (orientados à E/S): processos que realizam muito E/S;
 - Tempo de execução é definido pela duração das operações de E/S.
 -
- **IDEAL**: existir um balanceamento entre processos *CPU-bound* e *I/O-bound*;

Escalonador de Processos



**Nível mais baixo
do SO**

**Manipulação de
interrupções e
processos**

Implementação de Processos

Tabela de Processos:

- Cada processo possui uma entrada
- Cada entrada possui um ponteiro para o bloco de controle de processo (BCP) ou descritor de processo
- BCP possui todas as informações do processo → contextos de hardware, software, endereço de memória;

Implementação de Processos

Gerenciamento de processos	Gerenciamento de memória	Gerenciamento de arquivos
Registradores Contador de programa Palavra de estado do programa Ponteiro de pilha Estado do processo Prioridade Parâmetros de escalonamento Identificador (ID) do processo Processo pai Grupo do processo Sinais Momento em que o processo iniciou Tempo usado da CPU Tempo de CPU do filho Momento do próximo alarme	Ponteiro para o segmento de código Ponteiro para o segmento de dados Ponteiro para o segmento de pilha	Diretório-raiz Diretório de trabalho Descritores de arquivos Identificador (ID) do usuário Identificador (ID) do grupo

Campos da entrada de uma tabela de processos

Implementação de Processos

- 1) O hardware empilha o contador de programa etc.
- 2) O hardware carrega o novo contador de programa a partir do vetor de interrupção.
- 3) O procedimento em linguagem de montagem salva os registradores.
- 4) O procedimento em linguagem de montagem configura a nova pilha.
- 5) O serviço de interrupção em C executa (em geral lê e armazena temporariamente a entrada).
- 6) O escalonador decide qual processo é o próximo a executar.
- 7) O procedimento em C retorna para o código em linguagem de montagem.
- 8) O procedimento em linguagem de montagem inicia o novo processo atual.

Esqueleto do que o nível mais baixo do SO
faz quando ocorre uma interrupção

Blocos de Controle de Processo (PCBs) (Descritores de Processo)

- Os PCBs mantêm as informações que o SO precisa para gerenciar o processo.
- Informações importantes para o controle:
 - Número de identificação de processo (PID)
 - Grupo (group)
 - Estado do processo
 - Prioridade de escalonamento
 - Um ponteiro para o processo-pai (id do pai)
 - registradores, Contador de Programa (PC), reg. de status
 - tempos: início, de CPU, alarmes, dos filhos

Blocos de Controle de Processo (PCBs) (Descritores de Processo)

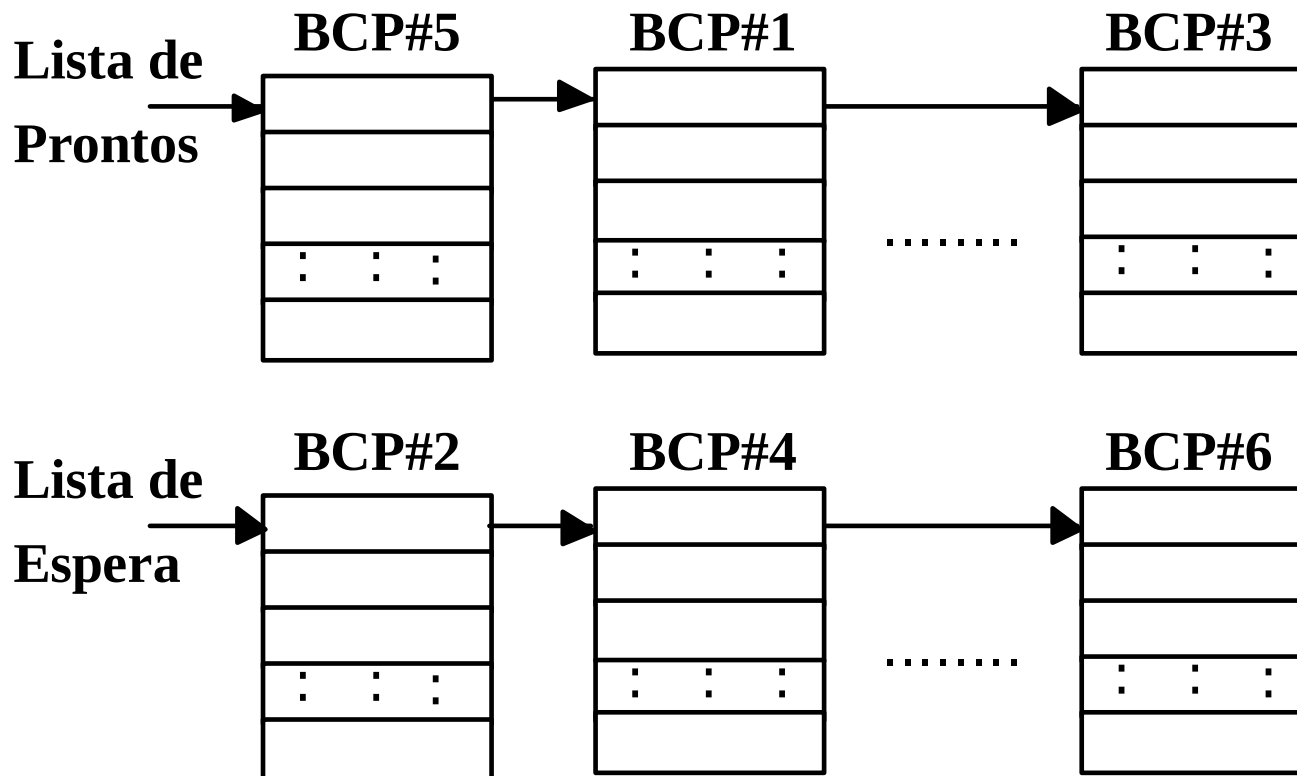
- Informações importantes para o controle:
 - Credenciais
 - Ponteiros para os processos-filho
 - Ponteiros para localizar os dados e instruções do processo na memória (pilha, código e dados)
 - Ponteiros para recursos alocados, descritores de arquivos
 - diretórios: raiz, trabalho

Blocos de Controle de Processo (PCBs) (Descritores de Processo)

■ Tabela de processos

- O SO mantém ponteiros para cada BCP dos processos em uma tabela de processos no âmbito de todo o sistema ou por usuário.
- Permite acesso rápido aos BCPs.
- Quando um processo é encerrado, a tabela de processos retira o processo da tabela e disponibiliza todos os seus recursos.

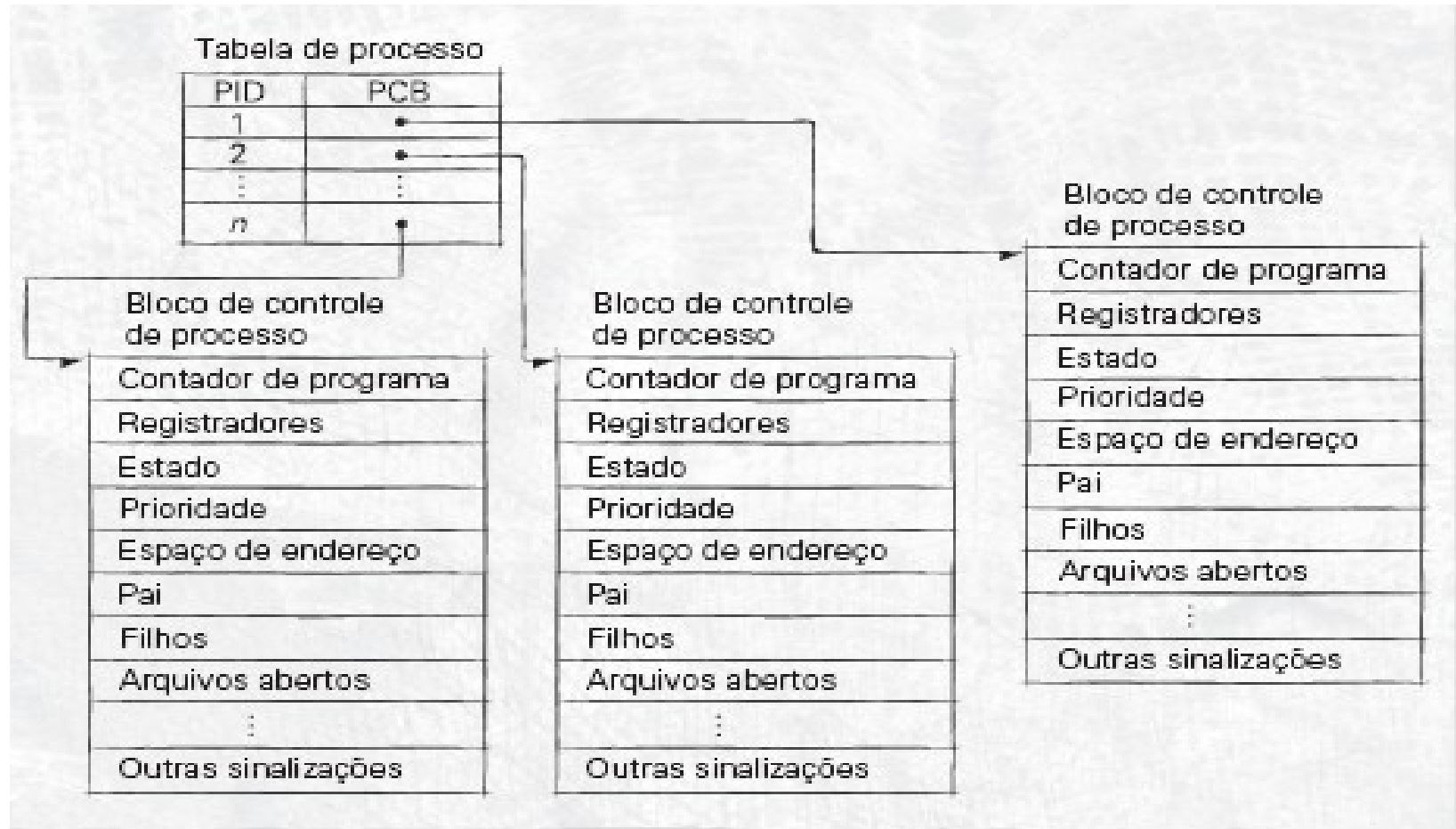
Localização dos Processos



Processos ficam em estruturas que os classificam conforme seu estado. Processos Prontos à serem escalonados para a execução ficam na lista/fila de prontos.

Blocos de Controle de Processo (BCPs) (Descritores de Processo)

Figura 3.2 Tabela de processos e blocos de controle de processo.



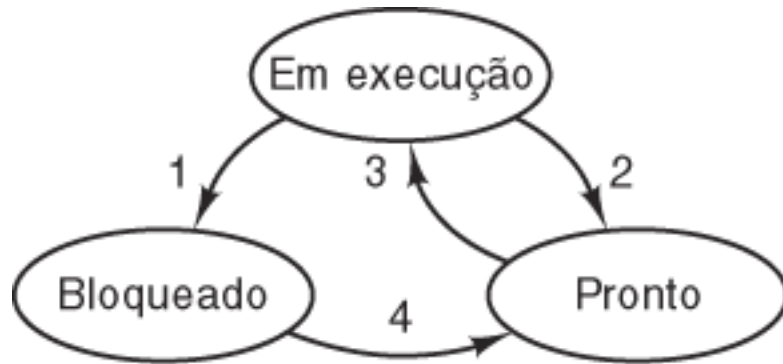
Hierarquia de Processos

- Pai cria um processo filho, processo filho pode criar seu próprio processo
- Formam uma hierarquia
 - UNIX chama isso de “grupo de processos”
- Windows não possui o conceito de hierarquia de processos
 - Todos os processos são criados iguais

Ciclo de vida de um Processo

- **Um processo passa por uma série de estados de processo distintos:**
 - Estado de *execução*
 - O processo está sendo executado em um processador.
 - Estado “*de pronto*”
 - O processo poderia ser executado em um processador se houvesse algum disponível.
 - Estado *bloqueado*
 - O processo está aguardando a ocorrência de algum evento para prosseguir.
- **O sistema operacional mantém uma lista de *prontos* e uma lista de *bloqueados* para armazenar referências a processos que não estão sendo executados.**

Estados de um Processo

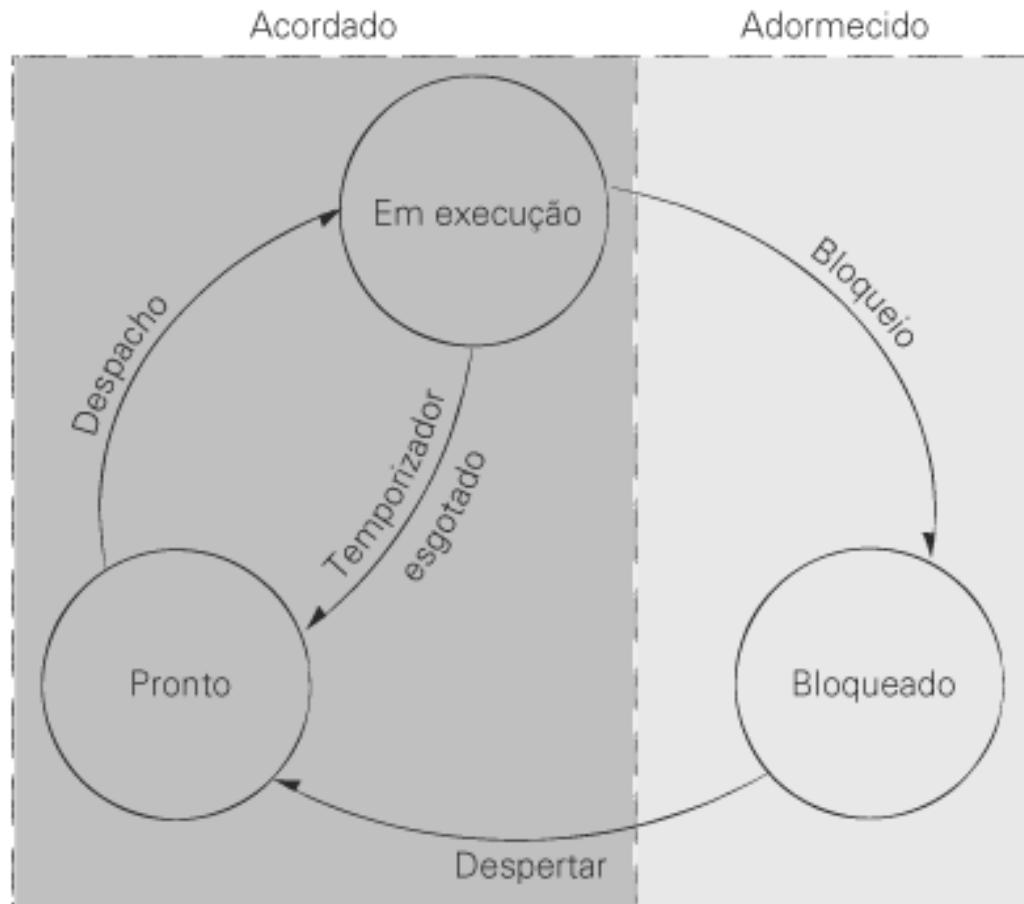


1. O processo bloqueia aguardando uma entrada
2. O escalonador seleciona outro processo
3. O escalonador seleciona esse processo
4. A entrada torna-se disponível

- Possíveis estados de processos
 - em execução
 - bloqueado
 - pronto
- Mostradas as transições entre os estados

Estados de processo e estados de transição

Figura 3.1 Transições de estado de processo.



Estados de processo e estados de transição

■ Estados de processo

- O ato de designar um processador ao processo escolhido da lista de prontos é denominado **despacho**.
- O SO usa um temporizador de intervalo para permitir que um processo seja executado durante um **intervalo de tempo específico**.
- Esse intervalo de tempo é chamado de *quantum* ou *timeslice*.
- Os processos alternam na execução até que todos sejam executados até o fim.

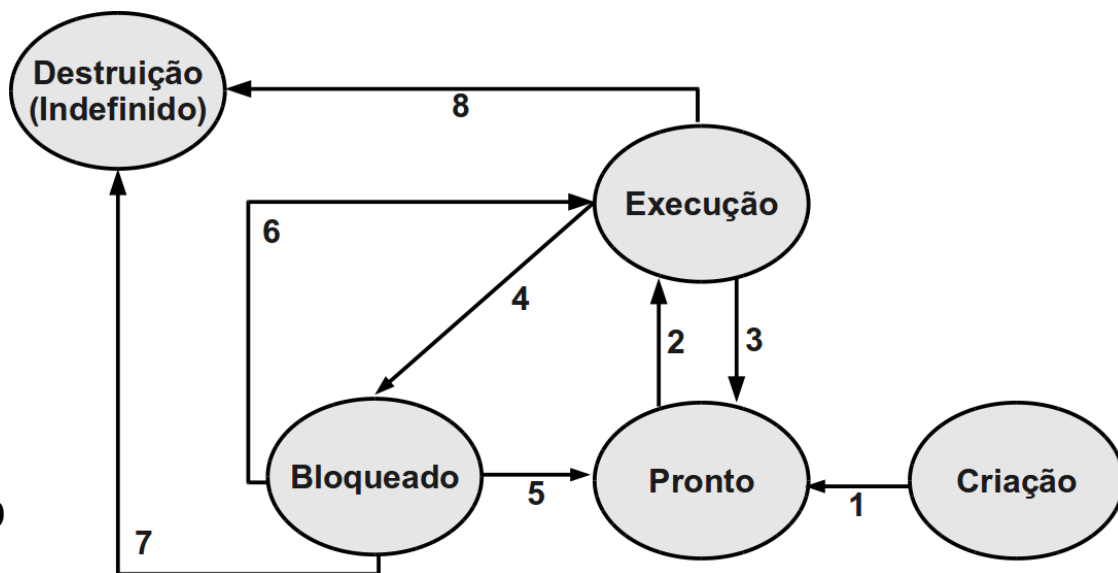
Estados de processo e estados de transição

■ Estados de transição

- Até agora, existem quatro estados de transição possíveis:
 - Quando um processo é **despachado**, ele transita de ***pronto*** → ***execução***.
 - Quando o **quantum** de um processo **expira**, ele transita de ***em execução*** → ***pronto***.
 - Quando um processo é **bloqueado**, ele transita de ***em execução*** → ***bloqueado***.
 - Quando um **evento** ocorre, ele transita de ***bloqueado*** → ***pronto***.

Transição entre os estados

1. Criar
2. Escalonar/Selecionar / Despachar
3. Suspender/Preemptar / Interrupção de Execução
4. Bloquear (chamada de sist.)
5. Acordar/Ocorrência de Evento (interrupção)
6. Retorno Imediato.
7. Destruir/Erro
8. Término/"Auto destruir"



Se diz que um algoritmo/Sistema Operacional é preemptivo quando um processo entra na CPU e o mesmo pode ser retirado da CPU antes do término da execução do mesmo.

Operações de Processo

- **Um processo pode gerar um novo processo.**
 - O processo de criação é chamado de **processo-pai**.
 - O processo criado é chamado de **processo-filho**.
 - Cada processo-filho é criado por exatamente um processo-pai. (1..1)
 - Quando um processo-pai é destruído, os Sistemas Operacionais em geral respondem de uma destas duas formas:
 - Destroem todos os processos-filho desse processo-pai.
 - Permitem que os processos-filho prossigam independentemente dos processos-pai.

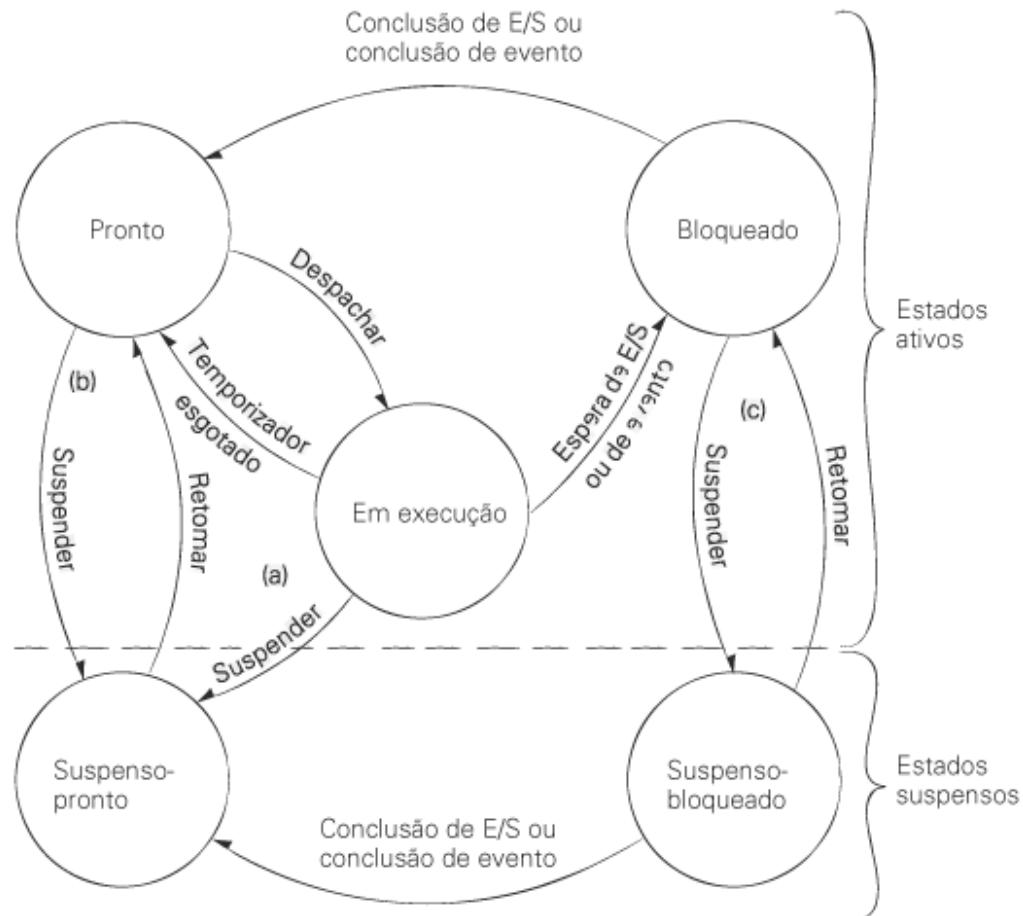
Suspende e retomar

■ Suspensão de um processo

- O processo é retirado indefinidamente da disputa por tempo em um processador, sem ser destruído.
- É útil para detectar ameaças à segurança e para depuração de software.
- A suspensão pode ser iniciada pelo processo que está sendo suspenso ou por outro processo.
- O processo suspenso precisa ser retomado por outro processo.
- Existem dois estados de suspenso:
 - *Suspenso-pronto*
 - *Suspenso-bloqueado*

Suspende e retomar

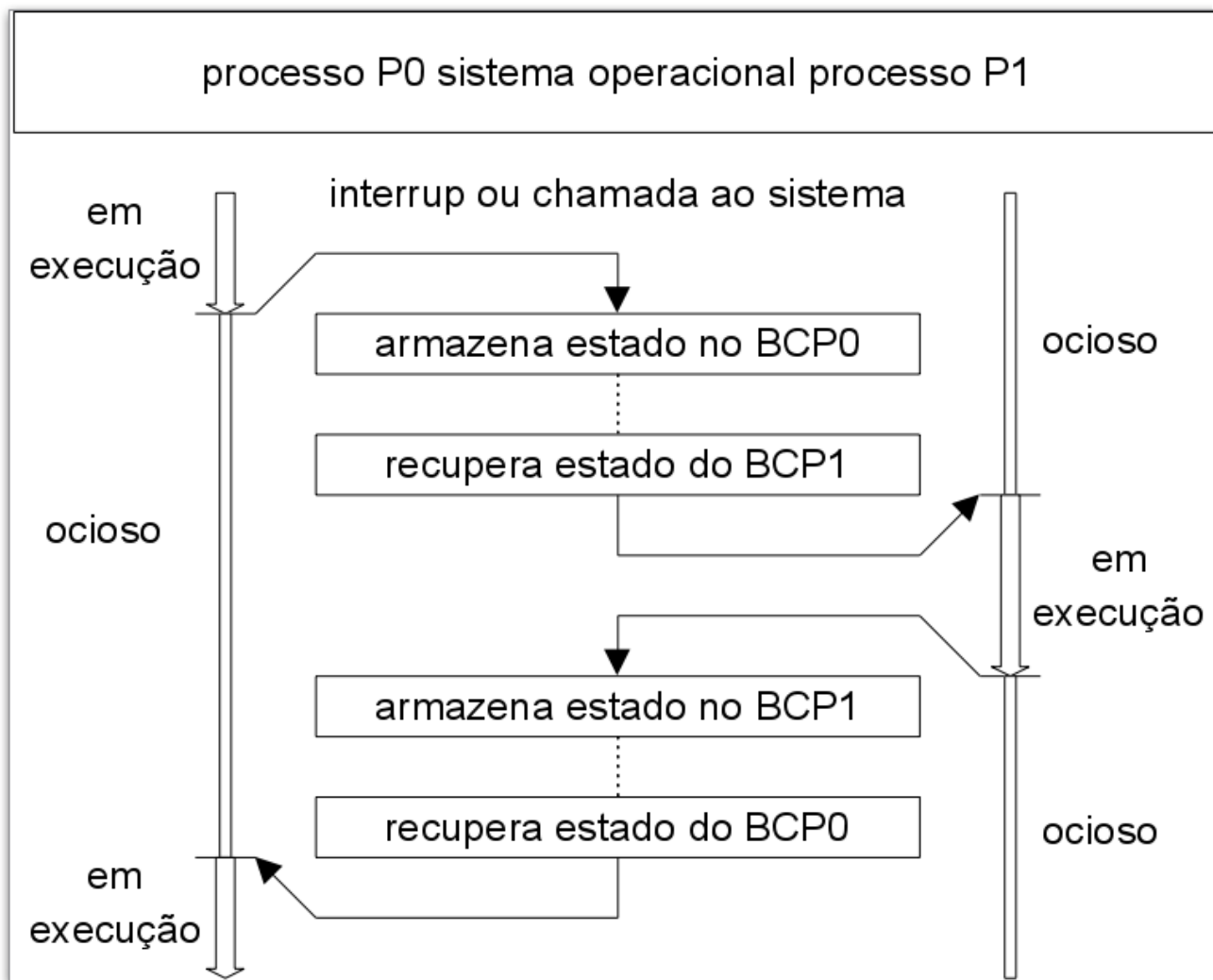
Transições de estado de processo com suspensão e retomada.



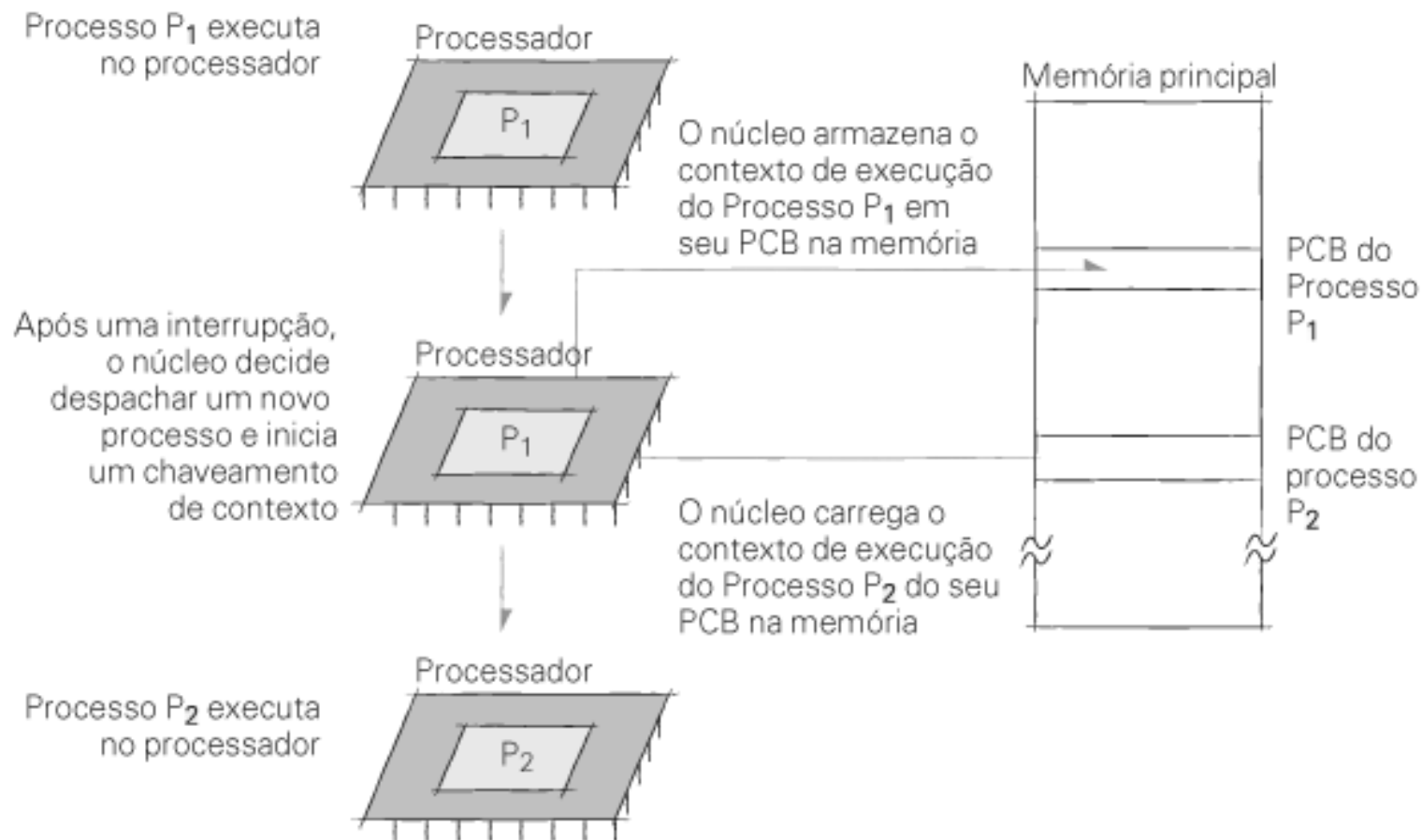
Chaveamento ou Troca de contexto

- O sistema operacional realiza um chaveamento de contexto para interromper um processo em execução e começar a executar um processo previamente pronto.
- Salva o contexto de execução do processo em execução no PCP desse processo.
- Carrega o contexto de execução anterior do processo pronto no BCP desse último.
- Deve ser transparente aos processos.
- O processador não pode realizar nenhuma computação “útil”.
 - O sistema operacional precisa, portanto, diminuir o tempo de chaveamento de contexto.
- É executado no hardware por algumas arquiteturas.

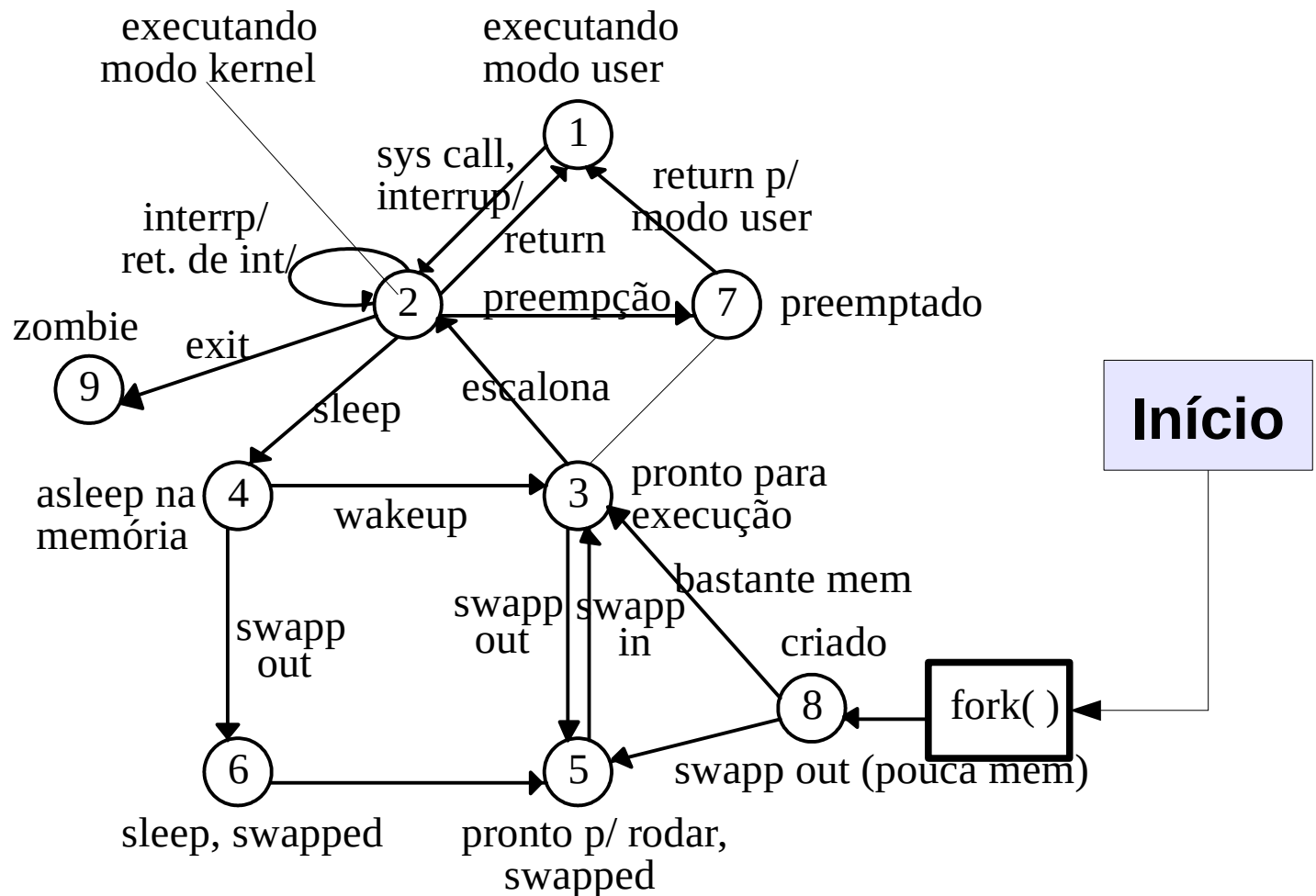
Chaveamento ou Troca de contexto



Chaveamento ou Troca de contexto



No Unix



Processos no Unix

- Todos os processos têm um conjunto de endereços de memória denominado espaço de endereço virtual.
- O núcleo mantém o PCB de um processo em uma região protegida da memória que os processos usuários não podem acessar.
- Em sistemas Unix, um PCB armazena:
 - O conteúdo dos registradores dos processos.
 - O identificador do processo (PID).
 - O contador do programa.
 - A pilha do sistema.
- Todos os processos são relacionados na tabela de processos.

Processos no Unix

- Todos os processos interagem com o sistema operacional por meio de chamadas ao sistema.
- Um processo pode gerar um processo-filho usando a chamada ao sistema *fork*, que cria uma cópia do processo-pai.
 - O processo-filho também recebe uma cópia dos recursos do processo-pai.
 - As prioridades de processo são números inteiros entre -20 e 19 (inclusive).
 - Um valor numérico de prioridade mais baixo indica uma prioridade de escalonamento mais alta.
 - O Unix fornece vários mecanismos que habilitam os processos a trocar dados, como é o caso dos pipes.

Chamadas de Sistema no Unix

Chamada ao Sistema	Descrição
<i>fork</i>	Gera um processo-filho e aloca ao processo uma cópia dos recursos de seu pai
<i>exec</i>	Carrega as instruções e dados de um processo no seu espaço de endereço.
<i>wait</i>	Faz com que o processo que está chamando fique bloqueado até que seu processo-filho termine.
<i>signal</i>	Permite que um processo especifique um tratador de sinal para um tipo de sinal particular.
<i>exit</i>	Termina o processo que está fazendo a chamada.
<i>nice</i>	Modifica a prioridade de escalonamento de um processo.

fork()

- Cria um processo filho que diferencia-se a partir do processo pai somente em suas PID e PPID.
- Arquivos travados e sinais pendentes não são herdados.
- No Linux, **fork** consome o tempo e memória requerida para duplicar as tabelas do processo pai e criar uma estrutura de tarefa única para o filho.

fork()

- Em caso de sucesso, a PID do processo filho é devolvido para o pai, e um 0 é retornado para o filho.
- Caso contrário, um -1 será retornado no contexto pai, e nenhum processo filho será criado, e um erro será selecionado adequadamente.

Criação de Processo no Unix - fork()

Código na linguagem C

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

int main( ){
    int  pai;
    int  valor = 0;

    pai = fork( );
    if (pai){    /* este trecho é executado pelo pai */
        printf("Eu Sou o Processo Pai %d \n", pai);
        valor = 5;
        printf("Valor: %d \n", valor);
    }
    else {    /* este trecho é executado pelo filho */
        printf("Eu Sou o Processo Filho %d \n", pai);
        valor = 10;
        printf("Valor: %d \n", valor);
    }
    exit(0);
}
```

Criação de Processo no Unix - exec()

- A família de funções `exec` substitui a imagem do processo atual por uma imagem de um novo processo.

```
#include <unistd.h>
#include <stdio.h>
int main(){
    printf("Antes do exec\n");
    // execl("/bin/ls", "/bin/ls", "-r", "-t", "-l", (char *) 0);
    printf("Depois do exec\n");
    return 0;
}
```

Note que a linha que contém a chamada à função `execl(...)` está comentada, assim as mensagens “Antes do exec” e “Depois do exec” serão escritas na tela.

Criação de Processo no Unix - exec()

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

int main() {
    printf("Antes do exec\n");
    execl("/bin/ls", "/bin/ls", "-r", "-t", "-l", (char *) 0);
    printf("Depois do exec\n");
    Return 0;
}
```

Resultado da Execução

```
rogerio@chamonix:~$ ./exe_exec
```

Antes do exec

```
total 352
```

lrwxrwxrwx	1	rogerio	rogerio	26	2009-01-22	14:00	Examples
drwxr-xr-x	2	rogerio	rogerio	6	2009-01-22	14:01	Modelos
drwxr-xr-x	2	rogerio	rogerio	6	2009-01-22	14:01	Vídeos

...

E a mensagem "Depois do exec"?

Como ficaria em Java

```
...  
processo = Runtime.getRuntime().exec("ls");  
  
InputStream in = processo.getInputStream();  
int c;  
String saida = "";  
  
while ((c = in.read()) != -1) {  
    saida += ((char) c);  
}  
  
System.out.println(saida);  
in.close();  
...
```

Toda aplicação Java tem uma única instância da classe **Runtime** que fornece à aplicação uma interface com o ambiente nativo no qual está executando. A instância corrente pode ser obtida através do método ***getRuntime()***.

Questões

- Um processo pode passar do estado pronto para bloqueado? Porque?
- E de execução para indefinido?
- Quantos processos podem existir na lista de prontos? Este valor está diretamente relacionado com a quantidade de memória disponível?

Questões

- Quantos processos podem existir na lista de processos em execução? Este valor está diretamente relacionado a que?
- Como funciona os comandos *fork* e *exec* do Unix?
- Explique: Processos, Estados e Transições e Fork.

Questões

- Faça um diagrama de transição de estados para processos que tenham os seguintes estados: indefinido, pronto na memória, pronto no disco, bloqueado por escrita, bloqueado por leitura e em execução. Explique.

Tarefa

- Implementar um terminal básico.
- Pesquisar sobre as chamadas de sistema do Linux/Unix, Windows e Minix.
- Submissões pelo moodle.

Próxima Aula
