

Aula 4

Professores:

Felipe M. G. França
Valmir C. Barbosa

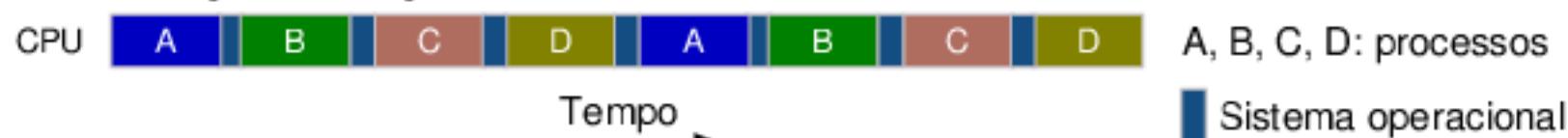
Conteúdo:

Processos

- O modelo de processos
- Gerenciamento e implementação

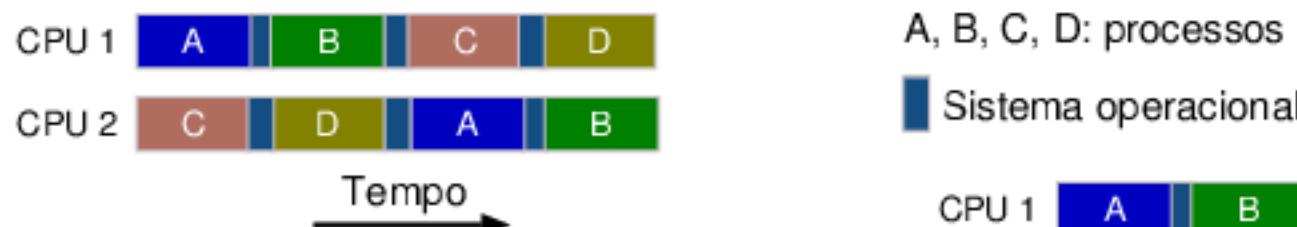
Introdução aos processos

→ Se o computador possuir somente um processador, então teremos um **pseudoparalelismo**:

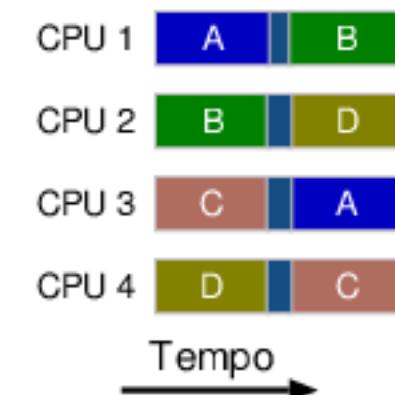


Neste exemplo temos um pseudoparalelismo, pois temos somente um processador (CPU) no computador, e temos quatro processos executando.

→ Teremos um **parallelismo real** quando o computador possuir mais de um processador.

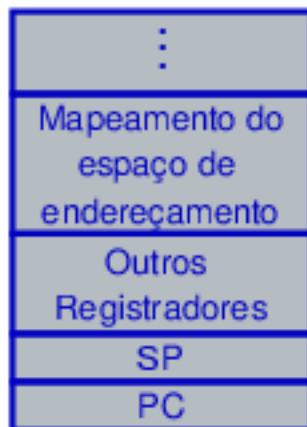


Acima temos algum grau de pseudoparalelismo pois somente dois processos podem executar ao mesmo tempo, mas ao lado não temos um pseudoparalelismo, pois os quatro processos podem executar simultaneamente.



Modelo de processos

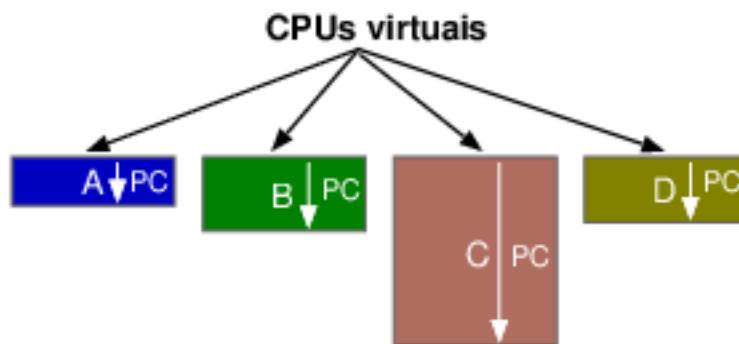
- ➡ Todos os programas rodando no computador são organizados como um conjunto de processos seqüenciais.
- ➡ O sistema operacional também é, em geral, organizado como um conjunto de processos (como por exemplo, o Linux).
- ➡ Cada processo possui um **contexto**, necessário à correta execução do processo, e que contém:
 - O contador de programa (PC), que controla o fluxo de instruções do processo, e os outros registradores da CPU.
 - Outras informações, como o mapeamento do espaço de endereçamento.



O **contexto** do processo contém todas as informações que devem ser salvas ao paramos o processo, para que ao reiniciarmos a sua execução mais tarde, este possa continuar a sua execução sem comprometer o resultado da tarefa.

Modelo de processos

→ O modelo de processos facilita o entendimento do conceito de multiprogramação:



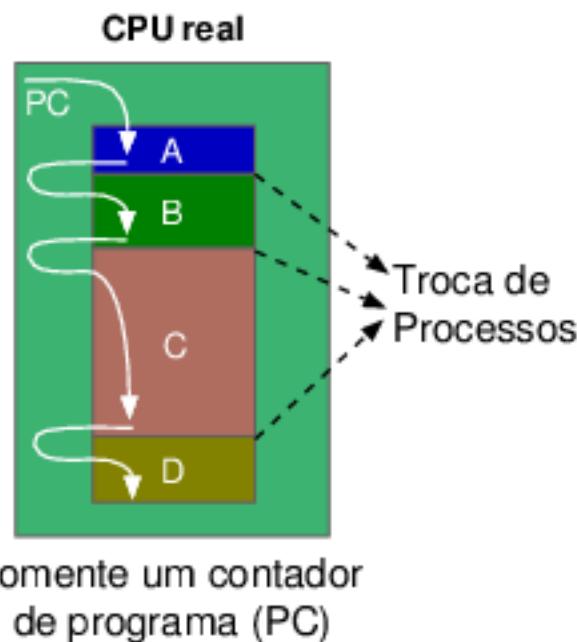
Cada CPU virtual possui o seu próprio contador de programa (PC), que é salvo no contexto do processo

O modelo de processos sequências, e o fato de os registradores do processador serem salvos no contexto do processo, nos permite visualizar cada processo do sistema rodando exclusivamente em um processador virtual, cujos valores dos registradores são iguais aos valores salvos no contexto do processo.



Modelo de processos

→ O modelo de processos facilita o entendimento do conceito de multiprogramação:

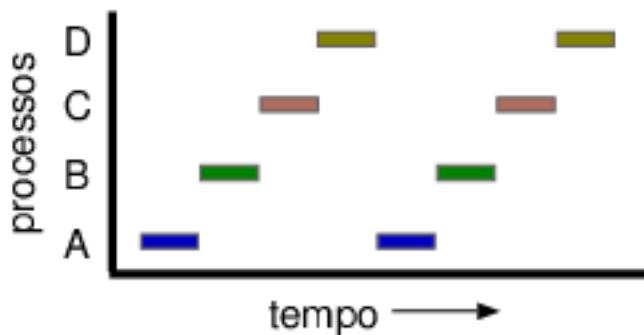


Para implementar o modelo de processos, deveremos dividir o tempo do processador real entre os processadores virtuais associados aos processos. Ao trocarmos o processo que está atualmente executando por um outro, deveremos salvar o seu contexto, e restaurar o contexto do outro processo. No exemplo, devemos compartilhar o processador entre os processos A, B, C, e D.



Modelo de processos

→ O modelo de processos facilita o entendimento do conceito de multiprogramação:



Como o sistema operacional divide o uso do processador entre os processos A, B, C, e D, se avaliarmos o uso do processador em um dado intervalo de tempo, veremos que estes processos usam alternadamente uma parcela igual do tempo do processador.



Modelo de processos

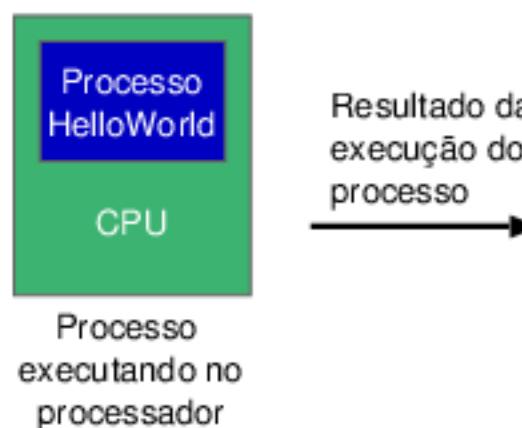
→ A distinção entre um programa e um processo é importante:

- Um **programa** é um conjunto de instruções para executar uma tarefa.
- Um **processo** é uma atividade em execução, e possui um programa, uma entrada, uma saída, um contexto, e um estado.

```
program HelloWorld;

var
  i: integer;

begin
  for i := 1 to 4 do
    writeln("Hello World!");
end;
end.
```



Resultado da execução do processo

Programa: instruções descrevendo como executar uma tarefa.

Processo: uma atividade que implementa as instruções do programa, ao ser executado no processador do computador.

Criação de processos

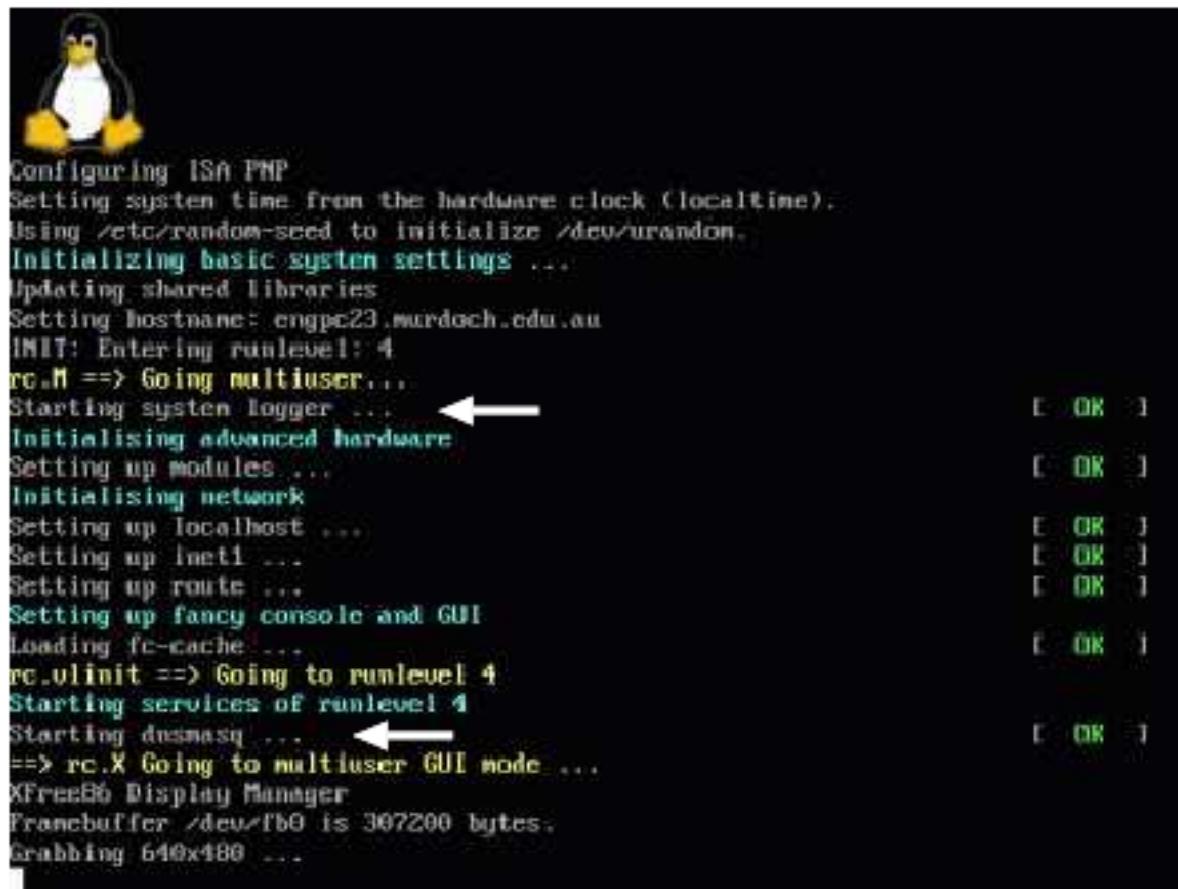
- ➡ Um sistema operacional deve garantir a existência de todos os processos necessários às aplicações executadas pelo usuário.
- ➡ Com isso, o sistema operacional deve fornecer um meio para a criação e o término de um processo.
- ➡ Em geral, um processo pode ser criado em decorrência de um dos seguintes eventos:
 - Quando o sistema operacional é iniciado ao ligar o computador.
 - Quando um processo em execução cria um novo processo.
 - Quando um usuário cria um novo processo para executar um programa.
 - Quando um usuário submete um trabalho a um sistema em lote, sendo que o processo executará este trabalho.

Criação de processos

- ➡ Os processos criados podem ser de dois tipos:
 - **Em primeiro plano (Foreground)**: em geral integram com o usuário para poder executar a sua tarefa.
 - **Em segundo plano (Background)**: não interagem com um usuário para poder executar a sua tarefa.
- ➡ Um **daemon** é um processo em background que recebe requisições de outros processos para tratar de tarefas específicas.
- ➡ Quando um usuário estiver usando um interpretador de comandos, e digitar um comando:
 - Os comandos sem & no final serão executados por processos em foreground.
 - Os comandos com & no final serão executados por processos em background.

Criação de processos

- Em geral, vários processos (em geral, daemons) são criados ao inicializar o sistema operacional.



```

Configuring ISA PNP
Setting system time from the hardware clock (localtime).
Using /etc/random-seed to initialize /dev/urandom.
Initialising basic system settings ...
Updating shared libraries
Setting hostname: engpc23.murdoch.edu.au
INIT: Entering runlevel: 4
rc.M ==> Going multiuser... ←
Starting system logger ...
Initialising advanced hardware
Setting up modules ...
Initialising network
Setting up localhost ...
Setting up ineti ...
Setting up route ...
Setting up fancy console and GUI
Loading fc-cache...
rc.ulimit ==> Going to runlevel 4
Starting services of runlevel 4
Starting dnsmasq ... ←
==> rc.X Going to multiuser GUI mode ...
XFree86 Display Manager
Framebuffer /dev/fb0 is 307200 bytes.
Grabbing 640x480 ...
  
```

Nas distribuições baseadas no sistema operacional Linux, após o computador ser ligado, o núcleo do Linux é lido na memória do computador, e depois de fazer algumas inicializações necessárias ao sistema, o núcleo criará o um processo executando o programa init, que será o responsável por criar todos os outros processos do sistema. Na figura, por exemplo, foram criados, dentre outros, os processos logger e dnsmasq (indicados por uma seta).

Criação de processos

→ A criação de processos por um outro processo:

- Facilita a implementação de uma tarefa que pode ser dividida em tarefas menores.
- Neste caso, um processo será criado para executar cada uma das tarefas menores.
- Em geral, estes processos deverão interagir uns com os outros, para que a tarefa original seja executada.

Processo
make

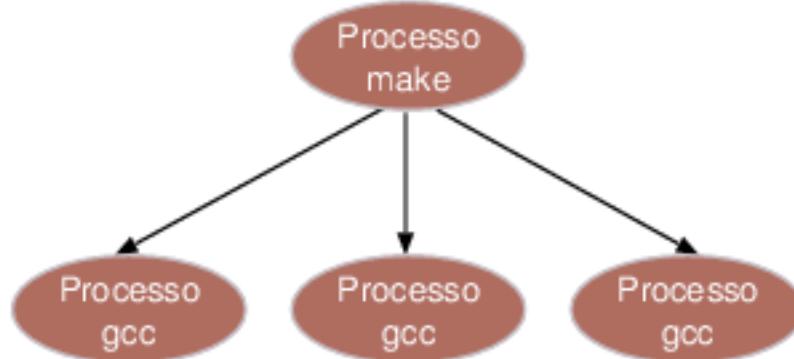
O usuário deseja compilar e instalar algum programa, e para isso, usou o comando make. Ao ser executado, este comando irá criar um processo chamado make.



Criação de processos

→ A criação de processos por um outro processo:

- Facilita a implementação de uma tarefa que pode ser dividida em tarefas menores.
- Neste caso, um processo será criado para executar cada uma das tarefas menores.
- Em geral, estes processos deverão interagir uns com os outros, para que a tarefa original seja executada.



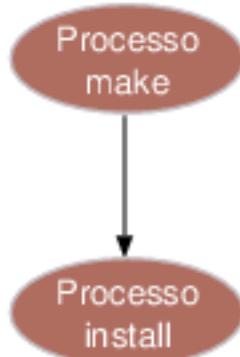
Se supusermos que o programa é composto pelos arquivos main.c, procedures.c, e interfaces.c, então este processo criará três processos executando o comando gcc, um para compilar cada um destes arquivos.



Criação de processos

→ A criação de processos por um outro processo:

- Facilita a implementação de uma tarefa que pode ser dividida em tarefas menores.
- Neste caso, um processo será criado para executar cada uma das tarefas menores.
- Em geral, estes processos deverão interagir uns com os outros, para que a tarefa original seja executada.



Depois de os processos criados pelo processo make compilarem os arquivos, este criará um processo executando o comando install para instalar no sistema o programa que acabou de ser compilado. Depois disso, o processo make terminará a sua execução.



Criação de processos



Nos sistemas que seguem o padrão POSIX, um processo cria outro por meio de uma chamada **fork** ao sistema:

- Cria um processo filho que é uma cópia exata do processo pai.
- O processo filho deverá executar uma outra chamada ao sistema, a **execve**, para poder executar um outro programa.
- A chamada ao sistema retorna, ao processo pai, o PID do filho, e ao processo filho, o valor 0.

```
pid := fork; ←
if (pid = 0) then
begin (* Processo filho *)
(* Inicializações *)
(* Carrega o novo programa *)
execve('/bin/ls', 0, 0);
end;
else
begin (* Processo pai *)
writeln("id do filho = ", pid);
(* Executa outras tarefas *)
end;
```

Processo
pai

Um processo executava alguma atividade descrita por um programa, quando precisou criar um processo filho para auxiliar a executar a atividade. Para isso, este processo usa a função **fork** da biblioteca para executar a chamada ao sistema **fork**.

Código executado pelo processo para criar um filho usando **fork**.



Criação de processos

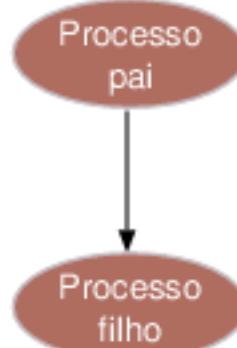


Nos sistemas que seguem o padrão POSIX, um processo cria outro por meio de uma chamada **fork** ao sistema:

- Cria um processo filho que é uma cópia exata do processo pai.
- O processo filho deverá executar uma outra chamada ao sistema, a **execve**, para poder executar um outro programa.
- A chamada ao sistema retorna, ao processo pai, o PID do filho, e ao processo filho, o valor 0.

```
pid := fork;
if (pid = 0) then
begin (* Processo filho *)
(* Inicializações *)
(* Carrega o novo programa *)
execve('/bin/ls', 0, 0);
end;
else
begin (* Processo pai *)
writeln("id do filho = ", pid);
(* Executa outras tarefas *)
end;
```

Código executado pelo processo para criar um filho usando **fork**.



Agora temos dois processos no sistema, o pai e o filho. Depois da chamada ao sistema **fork**, o filho será uma cópia exata do pai, incluindo uma cópia do espaço de endereçamento, e de todos os recursos que o processo usava.



Criação de processos



Nos sistemas que seguem o padrão POSIX, um processo cria outro por meio de uma chamada **fork** ao sistema:

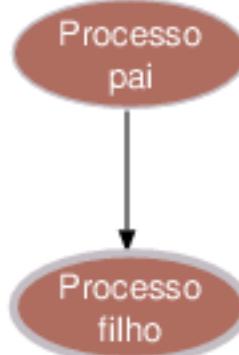
- Cria um processo filho que é uma cópia exata do processo pai.
- O processo filho deverá executar uma outra chamada ao sistema, a **execve**, para poder executar um outro programa.
- A chamada ao sistema retorna, ao processo pai, o PID do filho, e ao processo filho, o valor 0.

```

pid := fork;
if (pid = 0) then
begin (* Processo filho *)
    (* Inicializações *)
    (* Carrega o novo programa *)
    execve('/bin/ls', 0, 0);
end;
else
begin (* Processo pai *)
    writeln("id do filho = ", pid);
    (* Executa outras tarefas *)
end;

```

Código executado pelo processo para criar um filho usando **fork**.



A primeira instrução executada pelo processo filho será o comando **if**. O valor de **pid** será 0, pois este é o processo filho. O processo então faz algumas inicializações, e depois, em geral, o processo filho chama a função **execve**, que implementa a chamada ao sistema **execve** usada para um processo poder executar um novo programa.



Criação de processos



Nos sistemas que seguem o padrão POSIX, um processo cria outro por meio de uma chamada **fork** ao sistema:

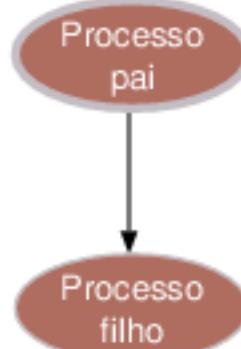
- Cria um processo filho que é uma cópia exata do processo pai.
- O processo filho deverá executar uma outra chamada ao sistema, a **execve**, para poder executar um outro programa.
- A chamada ao sistema retorna, ao processo pai, o PID do filho, e ao processo filho, o valor 0.

```

pid := fork;
if (pid = 0) then
begin (* Processo filho *)
(* Inicializações *)
(* Carrega o novo programa *)
execve('/bin/ls', 0, 0);
end;
else
begin (* Processo pai *)
writeln("id do filho = ", pid);
(* Executa outras tarefas *)
end;

```

Código executado pelo processo para criar um filho usando **fork**.



O processo pai executa a instrução **if**, e como o valor **pid** é positivo, descobre que ele é o pai, e que **pid** é o identificador do processo filho no sistema operacional. Agora, o pai poderá fazer qualquer outra atividade, como esperar pelo término do processo filho.



Término de processos

- ➔ Um processo deverá existir somente enquanto estiver executando a tarefa implementada por um programa.
- ➔ Os seguintes eventos podem causar o término de um processo:
 - O processo terminou voluntariamente a sua execução.
 - O processo foi terminado pelo sistema operacional.
 - O processo foi terminado por algum outro processo.
- ➔ Quando um processo termina voluntariamente, ou executou a sua tarefa, ou não consegui executá-la devido a algum erro:

```
$ls  
helloworld.c
```

```
$gcc -o helloworld helloworld.c
```

```
$ls  
helloworld.c helloworld
```

```
$gcc -o teste teste.c
```

```
gcc: teste.c: No such file or directory  
gcc: no input files  
$
```

O processo executando este comando gcc termina voluntariamente com sucesso, pois o arquivo helloworld.c existe.

Já o segundo processo executando o gcc termina voluntariamente mais sem sucesso, pois o arquivo teste.c não existe.

Término de processos

- ➔ Nos sistemas que seguem o padrão POSIX, a chamada ao sistema para terminar um processo é a **exit**:
 - A chamada possui um parâmetro que o processo pode usar para informar se a tarefa foi executada com sucesso.
 - Por exemplo, a chamada `exit(0)` informa ao interpretador que a tarefa foi executada com sucesso.
- ➔ O segundo caso ocorre quando a execução do processo gera um erro reportado pelo processador do computador, como:
 - Um erro aritimético, como uma divisão por zero.
 - Uma tentativa de executar uma instrução privilegiada, que somente pode ser executada no modo supervisor.
 - Uma tentativa de acessar uma região da memória fora do espaço de endereçamento do processo.

Término de processos

- Ao invés de ser terminado quando o primeiro evento ocorrer, o processo pode usar o mecanismo de sinais do sistema operacional:
- O sistema operacional envia um sinal específico ao processo para tratar cada um dos erros.
 - O processo deverá definir um procedimento para cada um dos sinais que o processo deseja tratar.
 - Os erros não tratados pelo processo irão gerar a ação default, que é a de terminar a execução do processo.

```
program DivisaoPorZero;
var
  i, j, k: integer;
begin
  i := 5;
  j := 0;
  k := i / j;
end;
end.
```

O programa `DivisaoPorZero` possui um possível erro aritmético, uma divisão por zero. Se o programador não definir um tratador para o sinal gerado por este erro, o processo será terminado ao executar o comando "`k := i/j`", e, em caso contrário, este erro talvez possa ser corrigido, e com isso, o processo poderá continuar normalmente a sua execução.

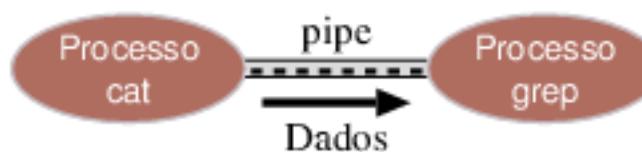
Término de processos

- ➡ O último evento que termina um processo é gerado pela chamada ao sistema **kill**, executada por algum outro processo:
 - O processo que executou a chamada deverá ter autorização para terminar o processo.
 - A tarefa que o processo estava executando não será terminada corretamente.
- ➡ Em geral, quando um processo termina a sua execução, todos os seus descendentes também terminam a sua execução.

Estados de um processo

- Ao executarem uma tarefa em comum, os processos precisam interagir, comunicar, e sincronizar uns com os outros:

Exemplo: cat c1 c2 c3 | grep árvore



Os processos cat e grep estão conectados por um pipe. Enquanto o processo cat gerar dados para o processo grep, os processos cat e grep podem executar no computador.

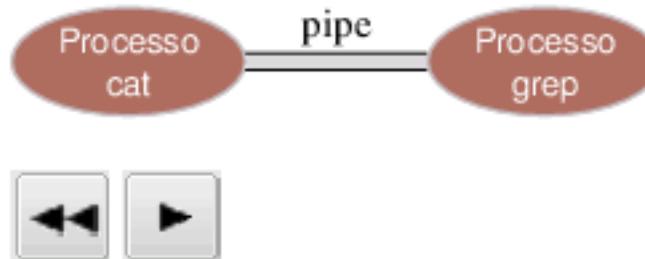
- Um processo pode estar em um dos seguintes estados:

- **Rodando ou Executando:** quando o processo estiver executando no processador.
- **Pronto:** quando o processo estiver suspenso porque o processador está sendo usado por um outro processo.
- **Bloqueado:** quando o processo foi suspenso porque está esperando a ocorrência de um evento externo.

Estados de um processo

- Ao executarem uma tarefa em comum, os processos precisam interagir, comunicar, e sincronizar uns com os outros:

Exemplo: cat c1 c2 c3 | grep árvore



Se o processo grep for mais rápido do que o cat, ele eventualmente processará todos os dados enviados pelo processo cat, e com isso, não terá nenhuma tarefa para executar. Para não ocupar o tempo do processador, o processo grep deverá ser suspenso até que novos dados sejam disponibilizados pelo processo cat.

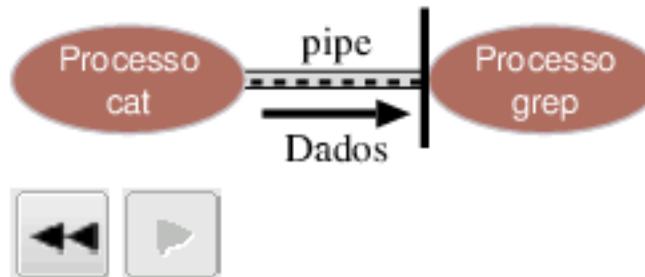
- Um processo pode estar em um dos seguintes estados:

- **Rodando ou Executando:** quando o processo estiver executando no processador.
- **Pronto:** quando o processo estiver suspenso porque o processador está sendo usado por um outro processo.
- **Bloqueado:** quando o processo foi suspenso porque está esperando a ocorrência de um evento externo.

Estados de um processo

- Ao executarem uma tarefa em comum, os processos precisam interagir, comunicar, e sincronizar uns com os outros:

Exemplo: cat c1 c2 c3 | grep árvore



Se o processo cat for mais rápido do que grep, o pipe eventualmente ficará cheio, e este processo não poderá mais colocar dados no pipe. Como antes, para não gastar o tempo do processador, este processo deverá ser suspenso até que o processo grep leia algum dado do pipe.

- Um processo pode estar em um dos seguintes estados:

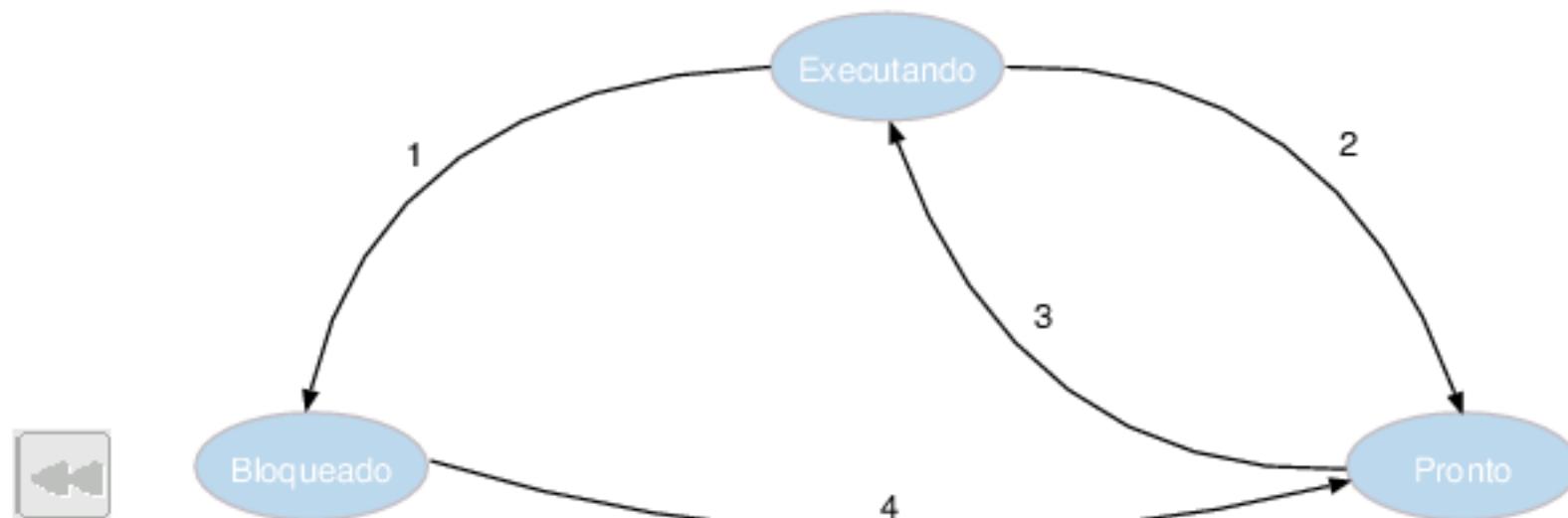
- **Rodando ou Executando:** quando o processo estiver executando no processador.
- **Pronto:** quando o processo estiver suspenso porque o processador está sendo usado por um outro processo.
- **Bloqueado:** quando o processo foi suspenso porque está esperando a ocorrência de um evento externo.

Estados de um processo

- ➡ A diferença entre os estados pronto e bloqueado é importante:
 - Um processo no estado pronto está suspenso por não existir um processador disponível para executá-lo.
 - Um processo no estado bloqueado não pode executar até o evento ocorrer, mesmo se o processador estiver disponível.
- ➡ Os estados pronto e executando são logicamente equivalentes, pois em ambos o processo pode executar.
- ➡ Em alguns sistemas, o processo deverá executar uma chamada ao sistema para ser bloqueado.
- ➡ A parte do sistema que gerencia a alocação do processador aos processos é chamada de o **escalonador** ou o **agendador**.

Estados de um processo

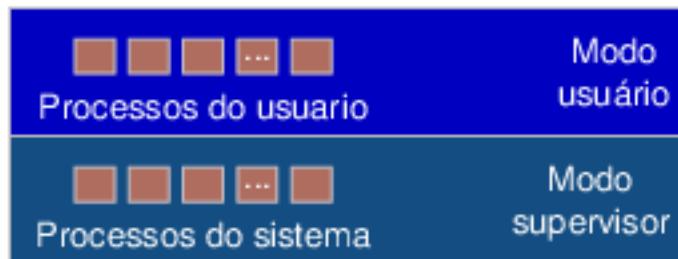
Com base no significado destes três estados, podemos definir o seguinte diagrama de transições:



Pressione o mouse sobre as transições do diagrama (as setas numeradas) para ver o significado de cada uma das transições.

Estados de um processo

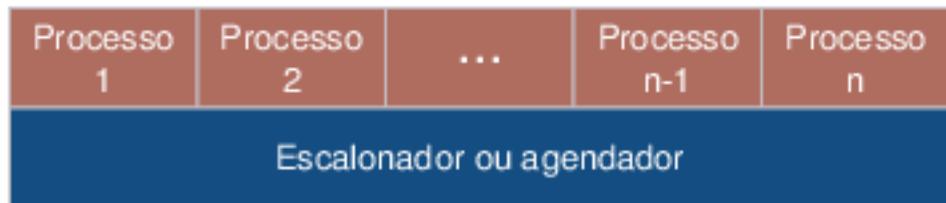
- O modelo de processos e os estados de um processo facilitam o entendimento do funcionamento do sistema operacional:



Usando o modelo de processo podemos visualizar dois tipos de processos executando no computador:

- Processos do usuário:** executam os programas executados pelos usuários.
- Processos do sistema:** ou executam tarefas para os processos dos usuários, ou tratam do gerenciamento do computador pelo sistema operacional.

- O modelo de processos permite a seguinte estruturação para o sistema operacional:

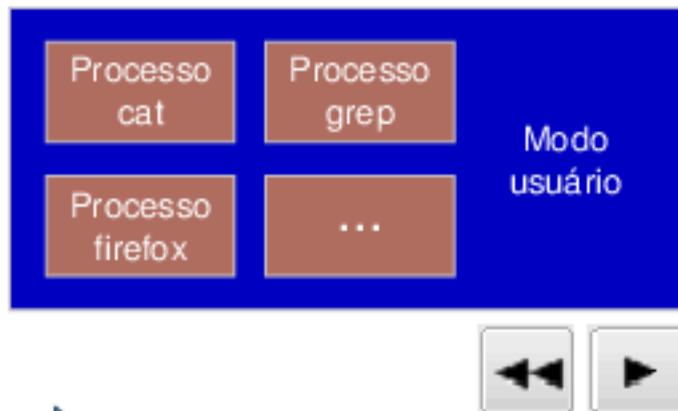


O sistema será composto por um conjunto de processos seqüenciais, que executam no processador no computador.

Os detalhes de como o processador é comutado entre os processos, e de como as interrupções do hardware são tratadas ficam ocultos ao resto do sistema, dentro do escalonador do sistema.

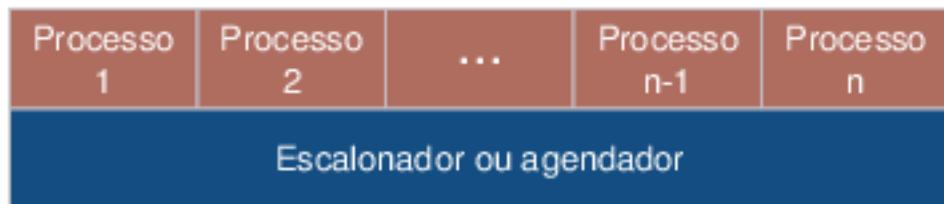
Estados de um processo

- O modelo de processos e os estados de um processo facilitam o entendimento do funcionamento do sistema operacional:



Cada um dos processos de usuário executa algum programa que implementa um comando digitado, em um interpretador de comandos, por algum usuário do sistema. Estes processos executam no modo usuário.

- O modelo de processos permite a seguinte estruturação para o sistema operacional:

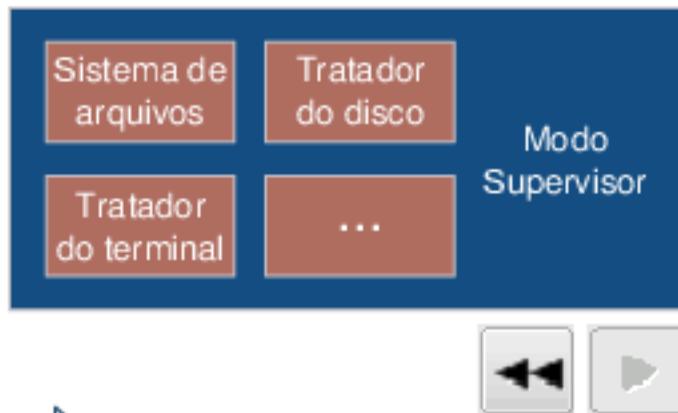


O sistema será composto por um conjunto de processos seqüenciais, que executam no processador no computador.

Os detalhes de como o processador é comutado entre os processos, e de como as interrupções do hardware são tratadas ficam ocultos ao resto do sistema, dentro do escalonador do sistema.

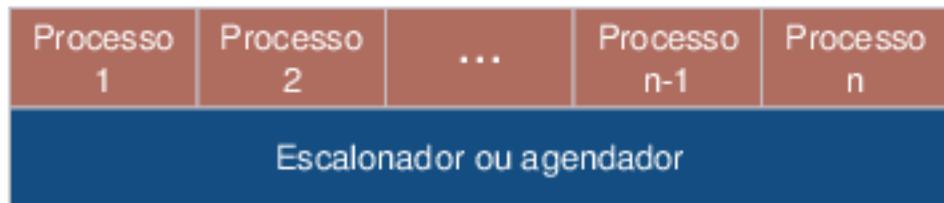
Estados de um processo

- O modelo de processos e os estados de um processo facilitam o entendimento do funcionamento do sistema operacional:



A visão do sistema operacional como um conjunto de processos facilita a compreensão de alguns detalhes internos do sistema, como, por exemplo, o tratamento de interrupções.

- O modelo de processos permite a seguinte estruturação para o sistema operacional:

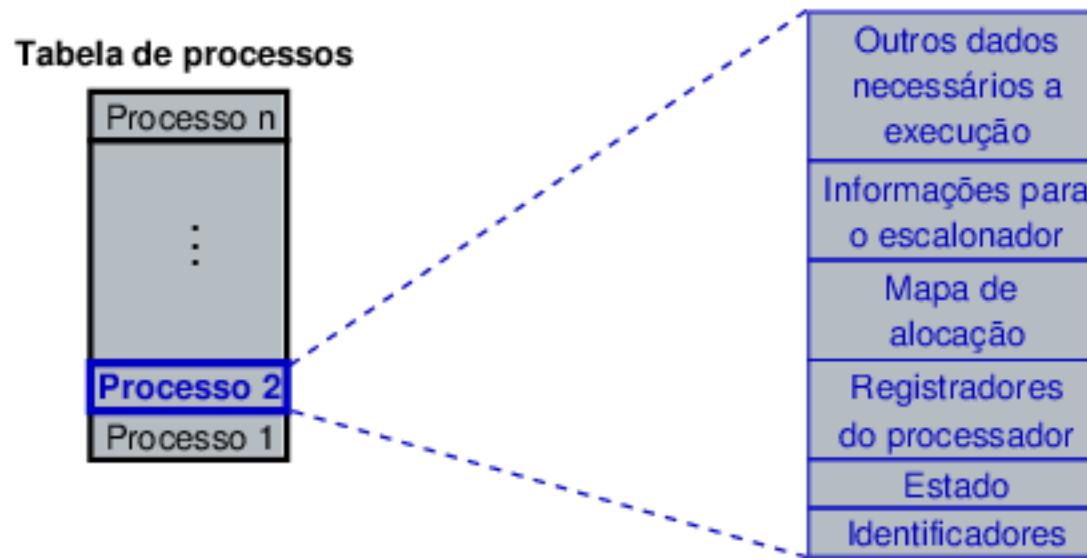


O sistema será composto por um conjunto de processos seqüenciais, que executam no processador no computador.

Os detalhes de como o processador é comutado entre os processos, e de como as interrupções do hardware são tratadas ficam ocultos ao resto do sistema, dentro do escalonador do sistema.

Implementação de processos

- ➡ Para implementar o modelo de processos, precisamos armazenar as informações necessárias a correta execução dos processos.
- ➡ A **tabela de processos** é usada para armazenar as informações:

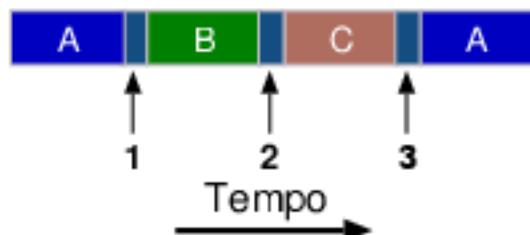


A tabela de processos possui uma entrada para cada processo do sistema, sendo que cada entrada, chamada de **bloco de controle do processo**, possui as informações do contexto do processo.

Implementação de processos

As informações dadas na tabela de processos, que são o contexto do processo:

- Devem ser salvas ao processo passar do estado executando para o estado pronto ou bloqueado.
- Ao voltar ao estado executando, o processo poderá reiniciar a sua execução de onde parou se usarmos o seu contexto.
- Permite que a execução do processo seja similar a executar o programa no processador até o seu término sem interrupção.



- 1: O contexto do processo A é salvo, e o do B é restaurado.
- 2: O contexto do processo B é salvo, e o do C é restaurado.
- 3: O contexto do processo C é salvo, e o do A é restaurado.

Implementação de processos

- ➔ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



O processo A está executando no processador do computador.

Implementação de processos

- ▶ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



O processo A estava continuando a sua execução quando uma interrupção ocorreu, gerada pelo disco rígido do computador. Neste caso:

- O processador, ao notar a interrupção, salvará na pilha os registradores.
- O endereço do tratador da interrupção é obtido do **vetor de interrupção**.
- O processador coloca na pilha um código que identifica o dispositivo.
- O processador finalmente salta para o tratador.

Implementação de processos

- ➔ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



O procedimento tratador da interrupção, escrito em assembler, começa a executar, e salva o contexto do processo A. Depois de retirar da pilha o código do dispositivo que gerou a interrupção, e de criar uma pilha temporária, um procedimento em C é chamado.

Implementação de processos

- ➔ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



O procedimento em C começa a sua execução, e coloca o processo que trata do disco, que estava bloqueado esperando pelo término de alguma operação de E/S, no estado pronto. Depois disso, o escalonador do sistema é chamado para selecionar um novo processo para executar no processador.

Implementação de processos

- ➔ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



O escalonador começa a sua execução, e escolhe um processo no estado pronto para executar no processador, no caso, o processo tratador do disco.

Implementação de processos

- ➔ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



Depois de o escalonador terminar a sua execução, o controle é retornado ao procedimento em C, que depois de executar algumas finalizações, retorna o controle ao procedimento em assembler.

Implementação de processos

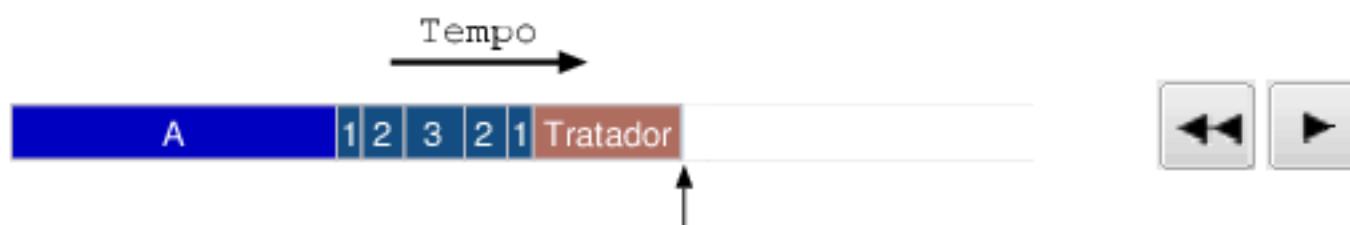
- ➔ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



O procedimento em assembler reinicia a sua execução, e restaura o contexto do processo que trata do disco. Depois disso, o procedimento salta para o endereço da próxima instrução deste processo, que teria sido executada caso o processo não fosse bloqueado.

Implementação de processos

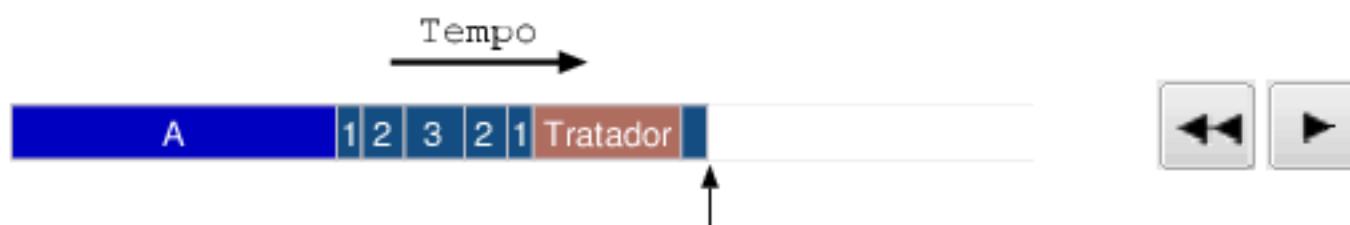
- ➔ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



O procedimento tratador do disco começa a sua execução, e faz as ações necessárias para terminar a operação de E/S que o disco acabou de executar. Depois de terminar estas ações, o processo bloqueia-se para esperar por novas interrupções do disco.

Implementação de processos

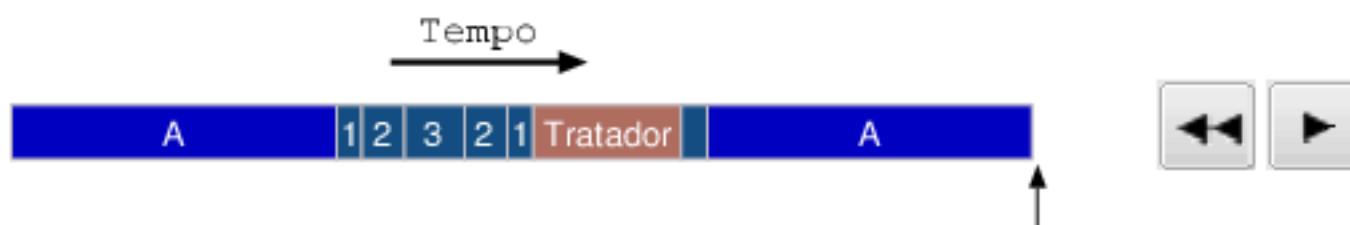
- ➔ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



Como o processo tratador de disco foi bloqueado, o escalonador é executado, e escolhe o processo A para executar. O contexto do processo será então restaurado, permitindo que este continua normalmente a sua execução.

Implementação de processos

- ➔ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



O processo A começa a executar do ponto onde parou antes da interrupção.

Implementação de processos

- ➔ Um sistema com múltiplos processos seqüenciais, e com vários dispositivos de E/S, pode agora ser facilmente entendido:
 - Podemos associar um processo a cada um dos dispositivos.
 - O processo que trata do dispositivo será chamado ao ocorrer uma interrupção gerada pelo dispositivo.



O processo A continua a executar no processador exclusivamente (porque não existe outro processo que deseja usar o processador), até que este finalmente termina a tarefa que deveria executar, e então este processo executará a chamada ao sistema operacional **exit**, que como vimos é usada quando um processo termina voluntariamente a sua execução.