Aula 04 – Processos e Escalonamento

Norton Trevisan Roman
Clodoaldo Aparecido de Moraes Lima

3 de setembro de 2014

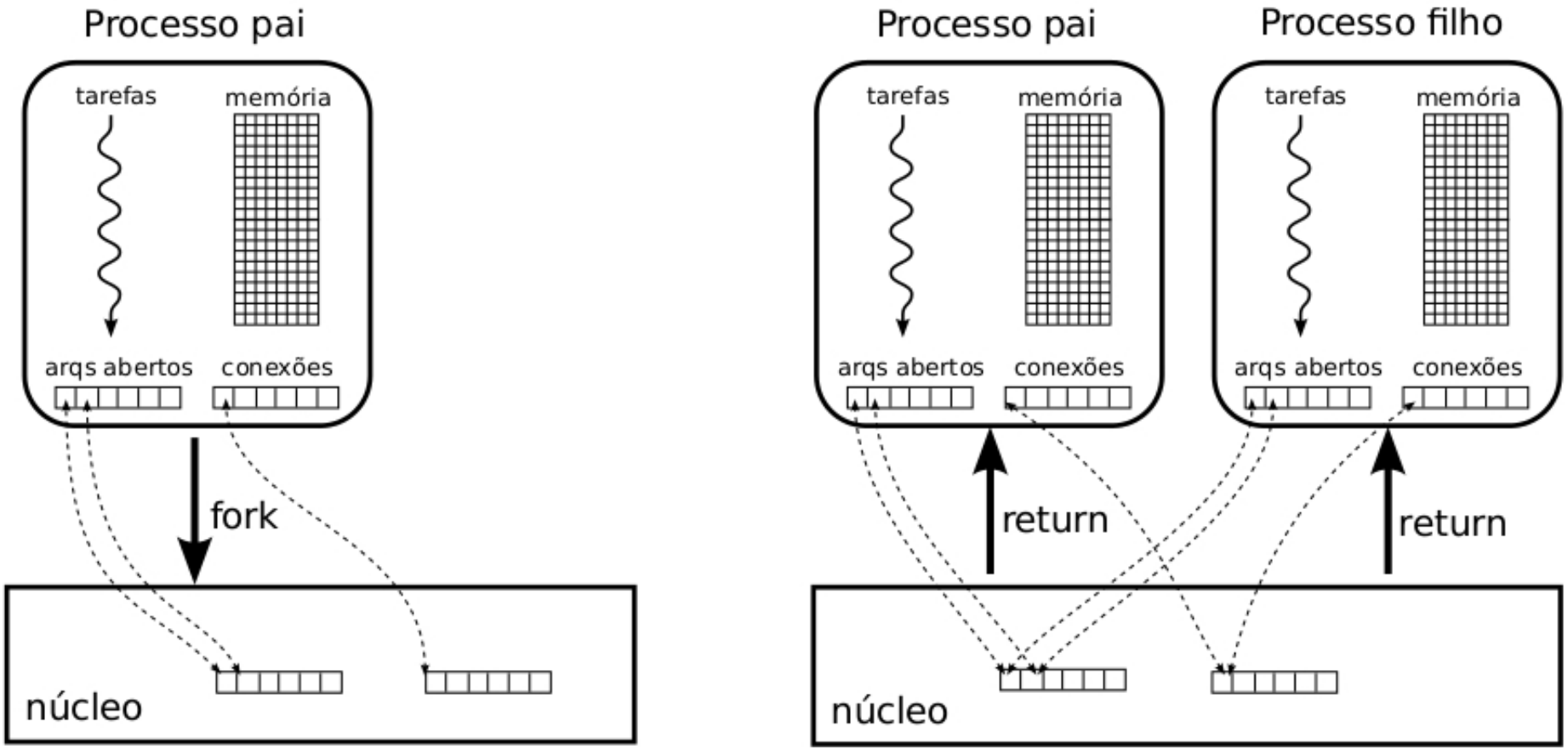
Criação de Processos

- Processos precisam ser criados e finalizados a todo o momento:
  - Inicialização do sistema;
  - Execução de uma chamada de sistema para criação de processo, realizada por algum processo em execução;
  - Requisição de usuário para criar um novo processo;
  - Inicialização de um processo em batch
    - Mainframes com sistemas em batch;

Criação de Processos

- Processos são criados por outros processos, executando uma chamada ao sistema
  - UNIX: Fork
    - Cria clone do processo Pai: cópias exatas na memória, mas com identificadores diferentes
    - Depois o processo Filho executa um novo código, via execve, substituindo o conteúdo da memória
    - Permite que o filho execute algo enquanto ainda é clone (ex: manipular descritores de arquivo, redirecionando entrada e saída)
  - Windows: CreateProcess
    - Cria Filho, já carregando novo programa nele.

Criação de Processos – Fork



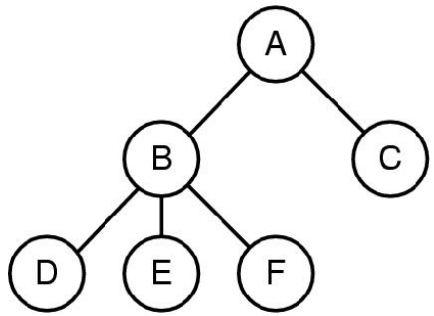
Embora fork seja chamada do pai, ela retorna tanto no pai quanto no filho

Hierarquia de Processos

- Unix: Pai cria processo filho, que cria neto
  - Possibilidade de Hierarquia
  - O comando pstree (linux) permite visualizar a árvore de processos do sistema
- Windows:
  - Todos os processos são iguais
  - Não existe o conceito de hierarquia
    - O Pai controla o filho via um identificador – Handle
    - Ele é livre para passar o identificador a outros processos, deserdando o Filho

Hierarquia de Processos

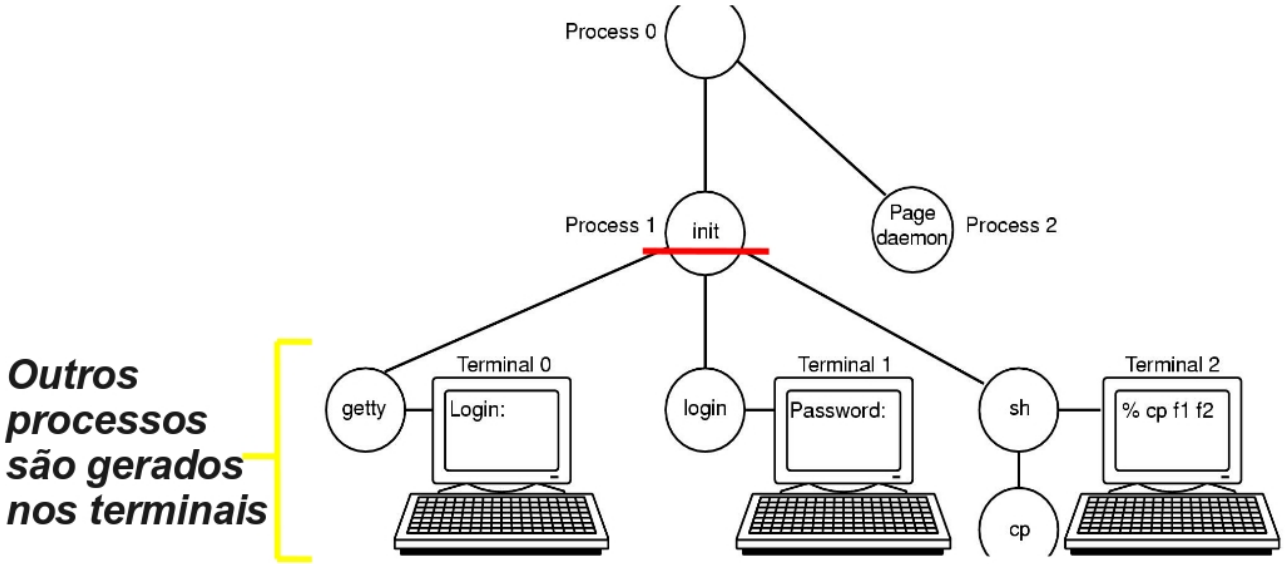
- Unix/Linux:
  - Forma grupos de processos (o processo e seus descendentes)
    - Ex: sinais do teclado são mandados aos membros do grupo associado com o teclado – normalmente todos os processos da janela atual



  - O filho não conhece o processo pai, mas o pai conhece o processo Filho → cabe ao pai abrir um canal de comunicação com o filho
    - O pai não pode passar adiante o identificador do filho → não pode deserdar o Filho

Hierarquia de Processos

- Unix/Linux:
  - Todo processo descende de init
    - init gera vários processos filhos para atender os vários terminais que existem no sistema;



Finalizando Processos

- Condições:
  - Término normal (voluntário):
    - A tarefa a ser executada é finalizada;
    - Ao terminar, o processo executa uma chamada (comunicando ao SO que terminou): exit (UNIX) e ExitProcess (Windows)
  - Término por erro (voluntário):
    - O processo sendo executado não pode ser finalizado
    - Ex: gcc filename.c, o arquivo filename.c não existe;

Finalizando Processos

- Condições:
  - Término com erro fatal (involuntário);
    - Erro causado por algum erro no programa (bug)
    - Ex: Divisão por 0 (zero); Referência a memória inexistente; Execução de uma instrução ilegal;
    - Nesse caso, o processo é interrompido (via signal)
  - Término (involuntário) causado por algum outro processo, via chamada a:
    - Kill (UNIX)
    - TerminateProcess (Windows)
    - O processo pedindo o término deve ter permissão para isso

Processos – Estados

- Um processo possui três estados básicos:
  - Executando: realmente usando a CPU naquele momento
  - Bloqueado: incapaz de executar enquanto um evento externo não ocorrer
  - Pronto: em memória, pronto para executar (ou para continuar sua execução), apenas aguardando a disponibilidade do processador
  - Há também um quarto estado, problemático: zumbi
    - Processo que não deveria existir, mas que mantém cópia em memória
    - Ocorre quando o filho perde sua ligação com o pai

Processos – Estados

Zumbi

- Quando um programa filho termina antes do pai, o núcleo ainda mantém informação sobre ele
  - Como seu exit status, por exemplo
  - Uma entrada na tabela de processos (mais adiante)
- Ao chamar wait(), o pai recebe a informação e o núcleo descarta o processo
  - Nesse meio tempo, entre o término do filho e o wait, ele é um zumbi
  - Todo processo antes passa pelo estado de zumbi

Processos – Estados

Zumbi

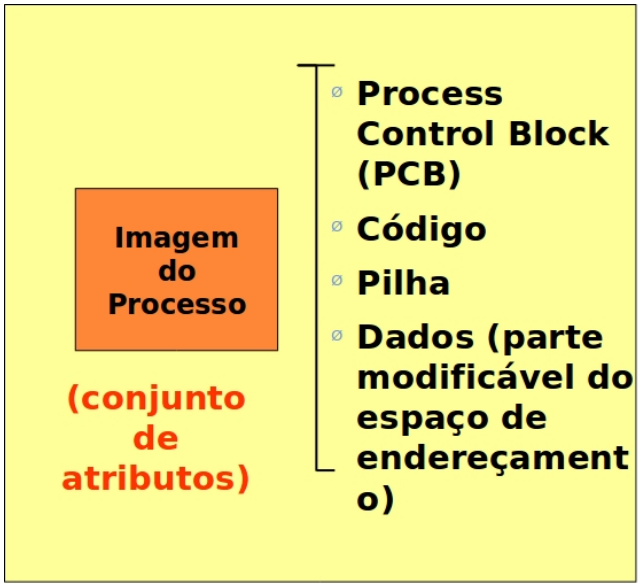
- Mesmo que não rode, ele ocupa uma entrada na tabela
  - Que tem tamanho fixo
  - Um grande número deles, pode impedir a abertura de novos processos
- Se o pai termina sem chamar wait(), o filho é adotado por init
  - Que cuida de limpar o filho
  - Pode-se contudo matar manualmente um zumbi, com kill -18 PPID

Implementação de Processos

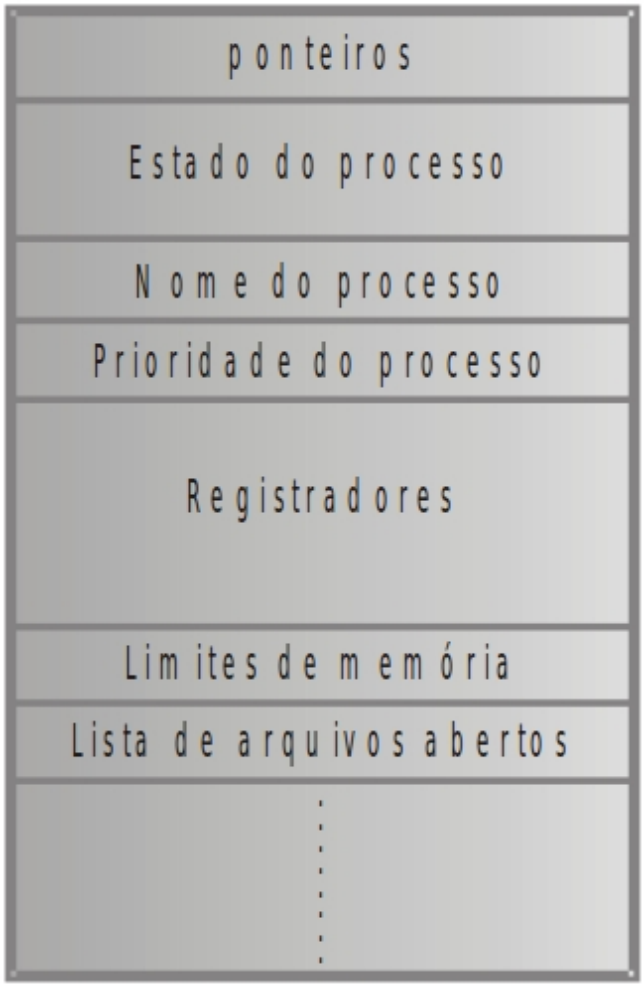
- Todo controle de processos é feito por uma tabela – Tabela de Processos
  - Os diversos processos em um computador são mantidos na Tabela de Processos
  - Cada processo constitui uma entrada na tabela
  - Cada entrada possui um ponteiro para o bloco de controle de processo (BCP) ou descritor de processo
    - BCP possui todas as informações do processo (contextos de hardware, software, endereço de memória – tudo que for necessário para se reiniciar um processo do ponto em que foi interrompido)
    - A tabela é então um arranjo ou lista ligada de BCPs

Implementação de Processos

- BCP (Bloco de Controle de Processo)
  - Contém informações sobre o estado do processo

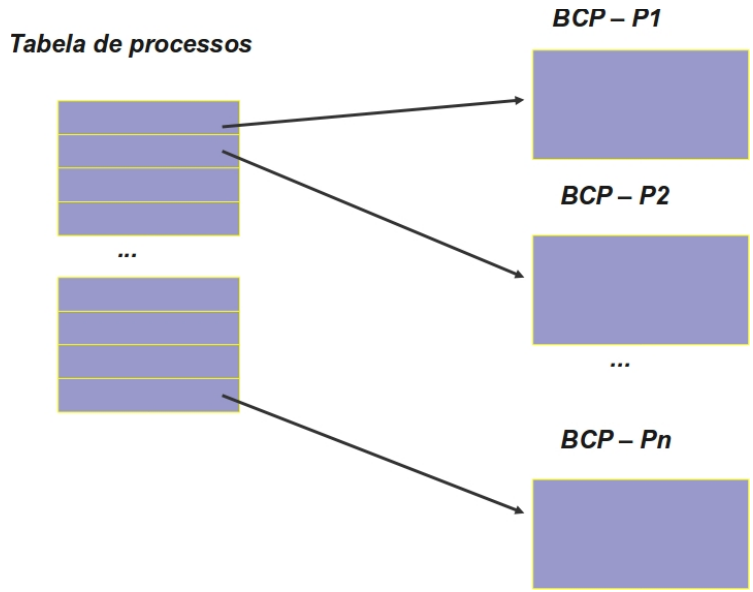


Componentes de um processo



Implementação de Processos

- Toda a informação é mantida no BCP
  - Exceto o conteúdo de seu próprio espaço de endereçamento



- Assim, um processo é constituido de seu espaço de endereçamento e BCP (com seus registradores etc), representando uma entrada na tabela de processos

Implementação de Processos

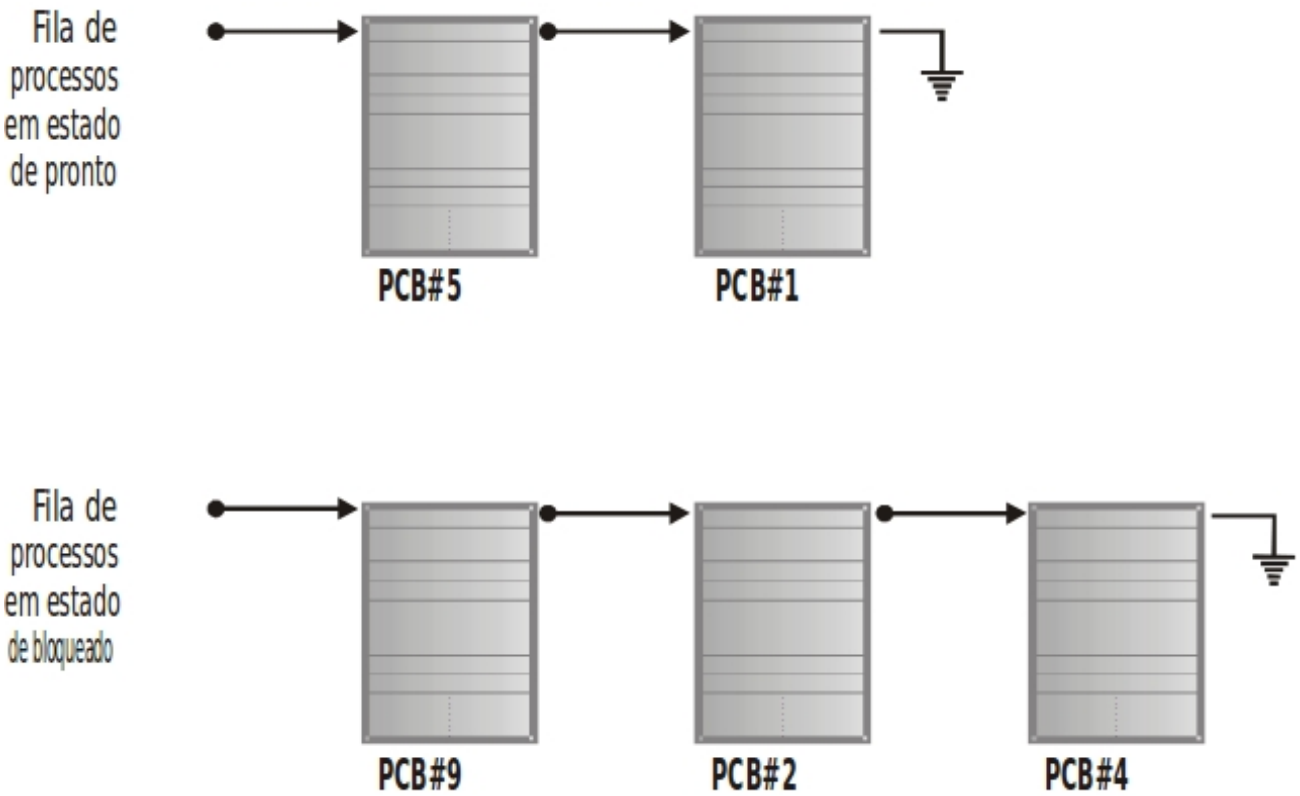
Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Algumas informações do BCP

Etapas de Criação de Processos

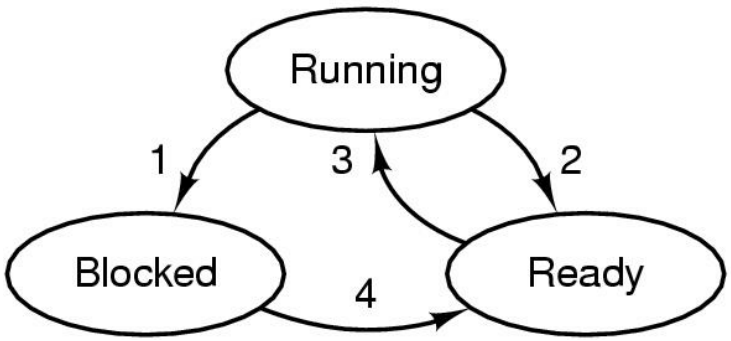
- O SO atribui um identificador único (PID) ao processo
- Aloca uma entrada na tabela de processos
- Aloca espaço para o processo na memória
- Inicializa o BCP
- Coloca o endereço do BCP na fila apropriada (Pronto ou bloqueado)
- Cria estruturas auxiliares

Criação de Processos: Filas



Mudança de Estado de Processos

- Atingir esse balanceamento requer trocar o processo que está rodando, dando um lugar a outro que precise da CPU
  - Requer a mudança do estado do processo



(1) O processo bloqueia aguardando entrada (E/S)

Um processo é bloqueado via (i)

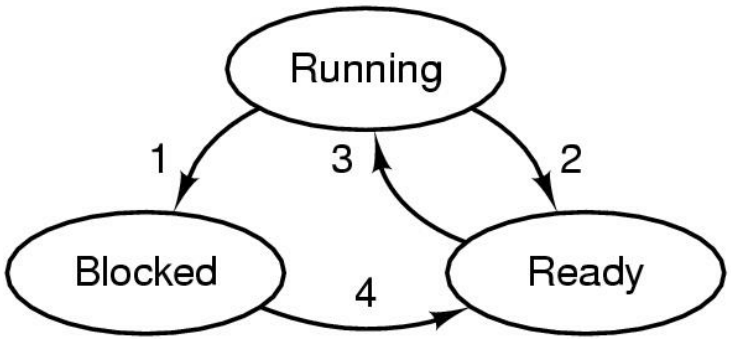
chamada ao sistema: block ou pause;

ou (ii) se não há entrada disponível para

que o processo continue sua execução

Mudança de Estado de Processos

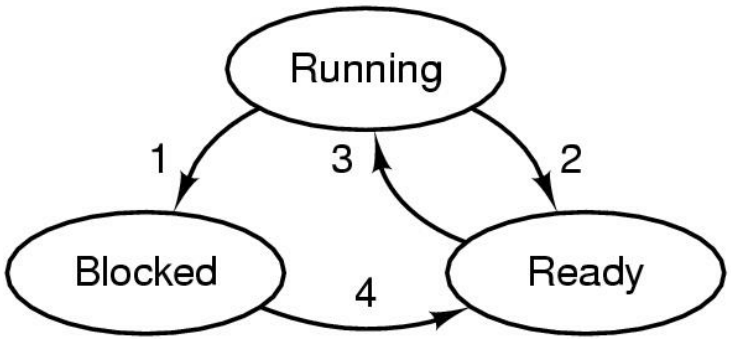
- Atingir esse balanceamento requer trocar o processo que está rodando, dando um lugar a outro que precise da CPU
  - Requer a mudança do estado do processo



As transições 2 e 3 ocorrem durante o escalonamento de processos: o tempo destinado àquele processo acabou e outro processo é colocado no processador

Mudança de Estado de Processos

- Atingir esse balanceamento requer trocar o processo que está rodando, dando um lugar a outro que precise da CPU
  - Requer a mudança do estado do processo



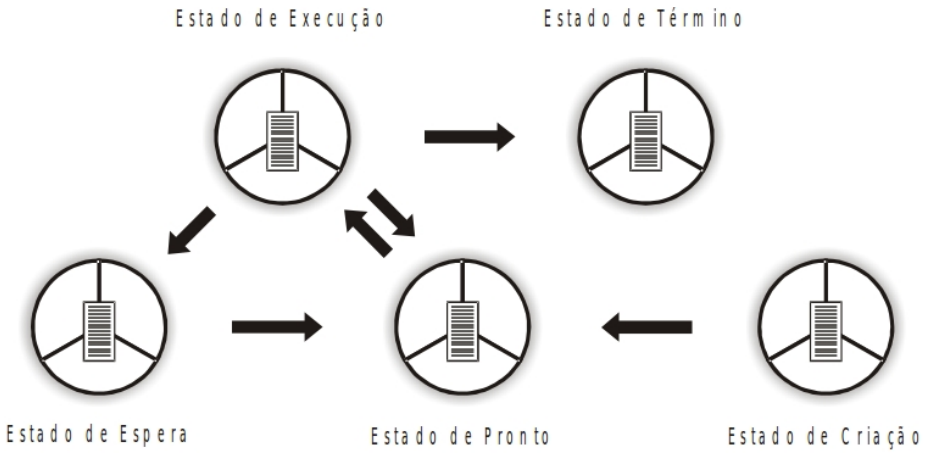
A transição 4 é feita quando o evento esperado pelo processo bloqueado ocorre (Ex: E/S).
Se o processador está parado, o processo é executado imediatamente (3);

Se o processador está ocupado, o

processo deve esperar sua vez

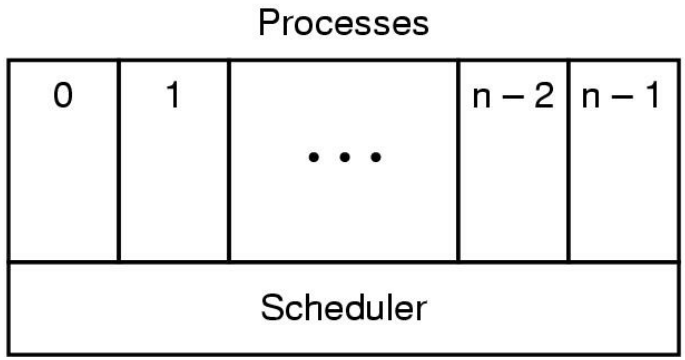
Mudança de Estado de Processos

- Naturalmente, processos não somem nem surgem do nada
  - O modelo com 3 estados tem, na verdade, 5
    - Criação: processo sendo criado $\rightarrow$ código e bibliotecas sendo carregados na memória, e as estruturas de dados do núcleo atualizando para permitir sua execução.
    - Término: o processamento foi encerrado e o processo pode ser removido da memória.



Escalonador

- A troca de processos, por sua vez, é feita pelo Escalonador de Processos
  - Processo que escolhe qual será o próximo processo a ser executado
    - Existem diversas técnicas/algoritmos para escalonamento de processos
  - Nível mais baixo do SO



Escalonador

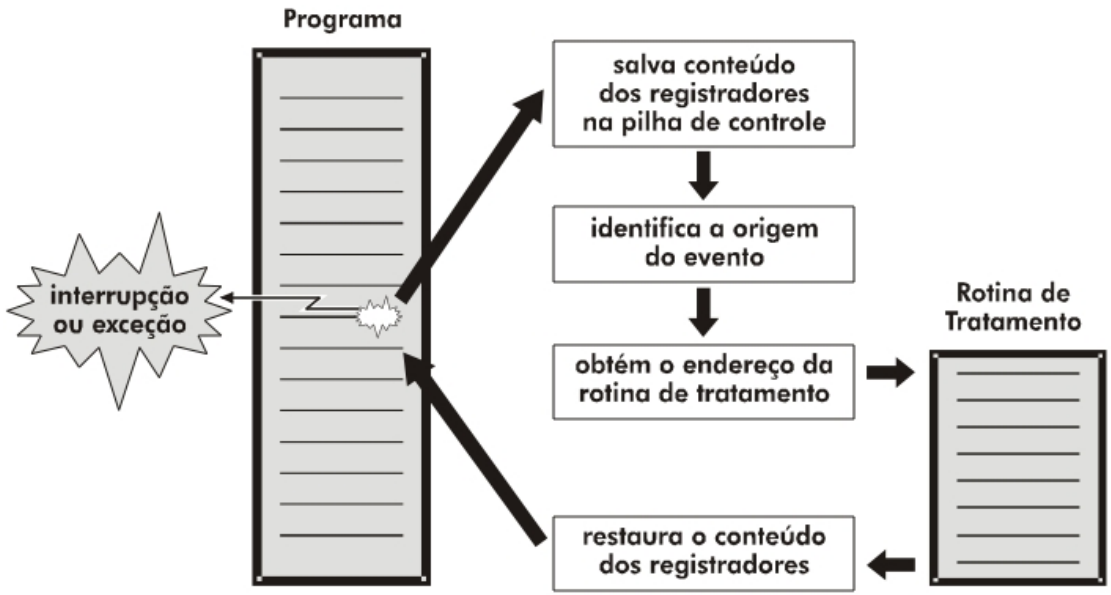
- O escalonamento é realizado com o auxílio do hardware
- O escalonador de Processos escolhe o processo que será executado pela CPU
  - Várias políticas possíveis
  - Veremos mais adiante...
- Contém todo o tratamento de interrupções, além de detalhes sobre iniciação e bloqueio de processos

Interrupções

- Vimos que um software pode interromper seu próprio processo (ao fazer uma chamada ao sistema)
  - Via traps (Interrupções de software ou Exceções)
  - Para isso, o software tem que estar rodando
- Mas como pode o escalonador interromper um determinado processo se ele mesmo não está rodando?
  - Via interrupções de hardware
    - Sinal elétrico no Hardware
    - Causa: dispositivos de E/S ou o clock

Interrupções – Tratamento

- Feito pelo SO, que determina a natureza da interrupção e dispara a Rotina de Serviço adequada para executar as ações que forem necessárias.



Interrupções – Tratamento

- Quando o SO inicia, ele instala um tratador de interrupções em um endereço específico da memória, definido pelo hardware
  - Ex: mips $\rightarrow$ 0x80000180
- Cria também um Arranjo de Interrupções (também chamado de arranjo de traps)
  - Ex: primeiras 256 palavras do segmento decódigo do SO (x8000-x80FF)
    - Cada palavra é um jump (JMP) para uma determinada rotina
    - Ou para uma rotina especial, caso o código da interrupção não esteja nesses 256, caso em que essa rotina mata o processo

Interrupções – Tratamento

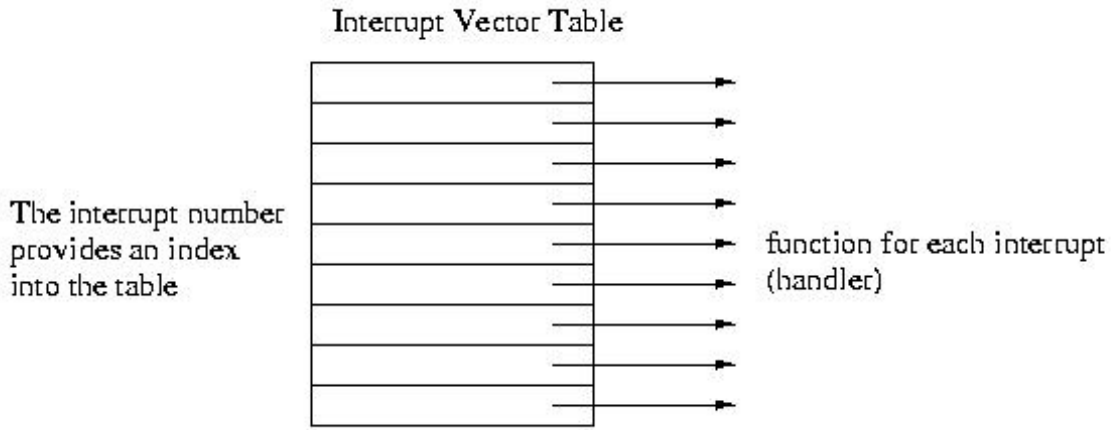
Arranjo de Interrupções

- Locação de memória (geralmente próxima da parte mais baixa da memória), associada a cada classe de dispositivos de E/S (inclusive clock)
  - Cada classe (disco, clock etc) pode ter um arranjo diferente
- É um arranjo usado para gerenciar interrupções dos programas
  - Contém os endereços dos procedimentos dos serviços de interrupção
  - Junto com o BCP, controla a execução dos processos

Interrupções – Tratamento

Arranjo de Interrupções

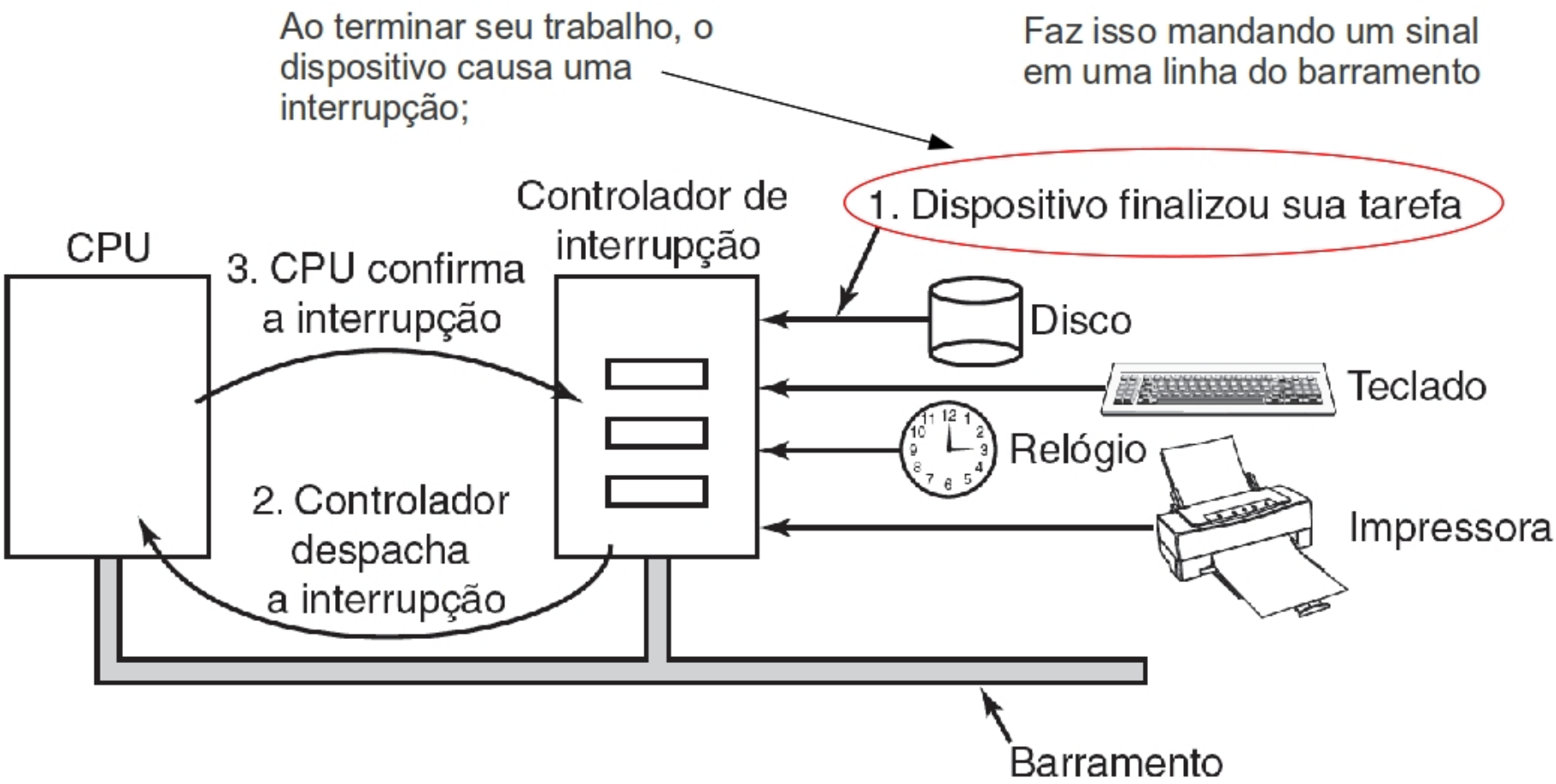
- O código da interrupção (número do dispositivo que a gerou) é usado como indice de busca no arranjo
  - interrupção 0 $\rightarrow$ posição 0
  - interrupção 1 $\rightarrow$ posição 1
  - etc



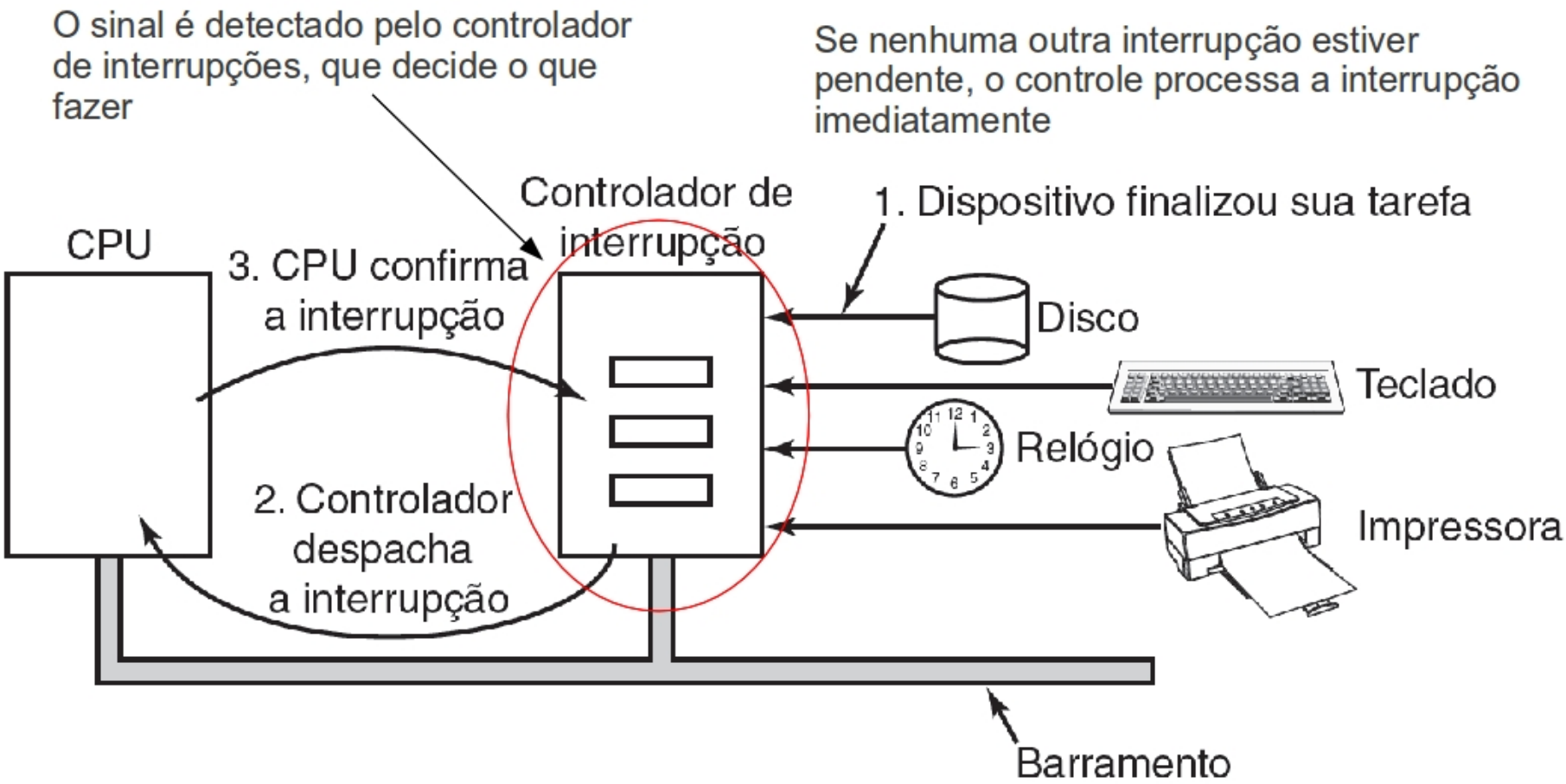
Interrupções × Traps

- Interrupções:
  - Evento externo ao processador
  - Gerados por dispositivos que precisam da atenção do SO
  - Podem não estar relacionadas ao processo que está rodando
- Traps:
  - Evento inesperado vindo de dentro do processador
  - Causadas pelo processo corrente no processador (seja por chamada, seja por instrução ilegal)

Interrupção – Pé no Hardware

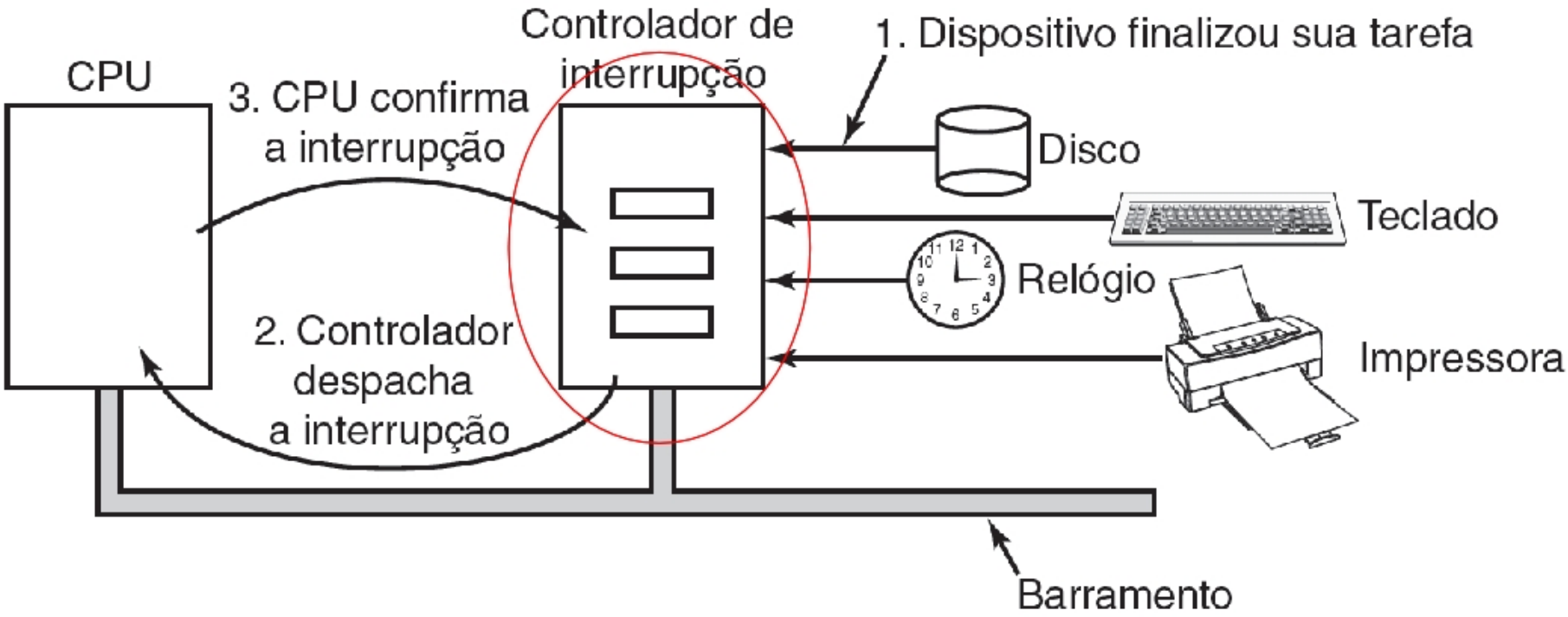


Interrupção – Pé no Hardware



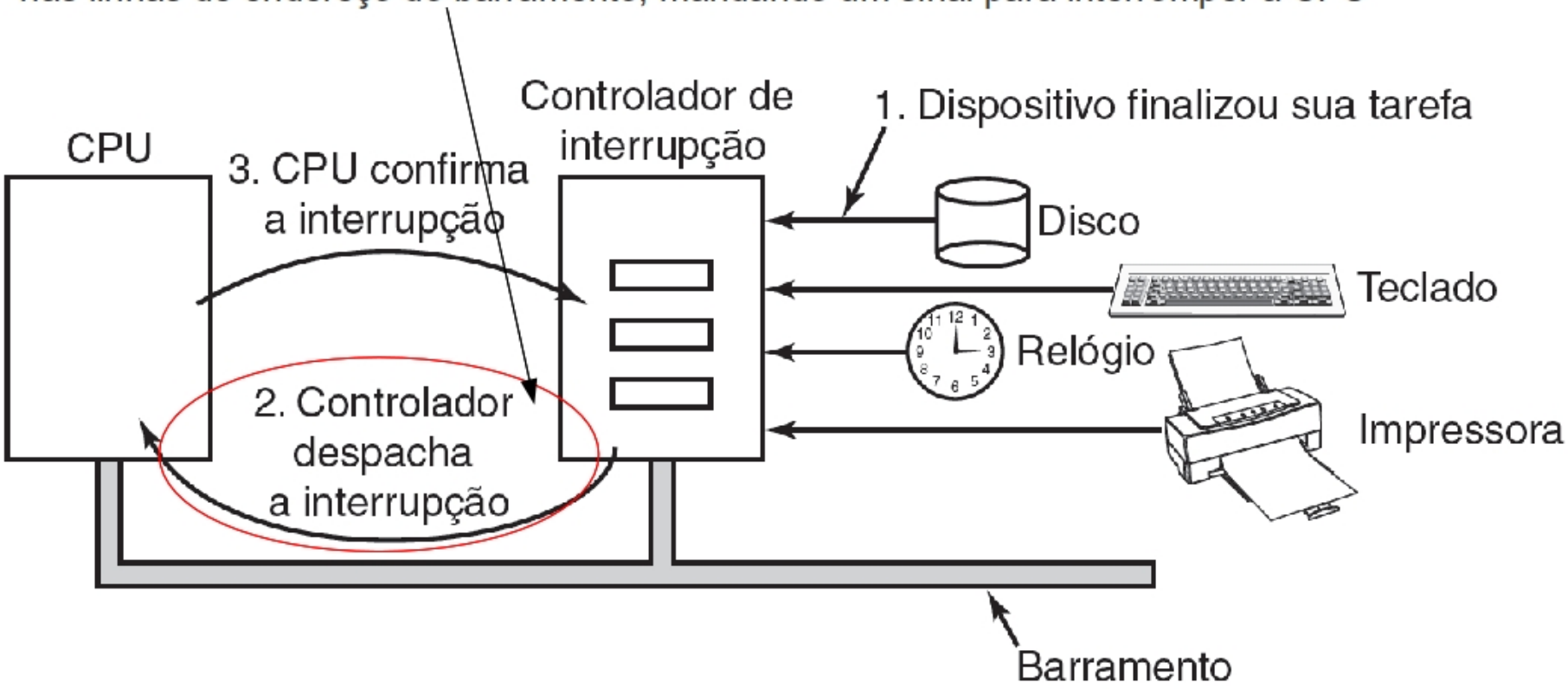
Interrupção – Pé no Hardware

Se alguma outra estiver em progresso, ou outro dispositivo fez um pedido simultâneo, em uma linha de interrupção de maior prioridade no barramento, o dispositivo é ignorado.
Caso em que continua a mandar o sinal de interrupção ao barramento



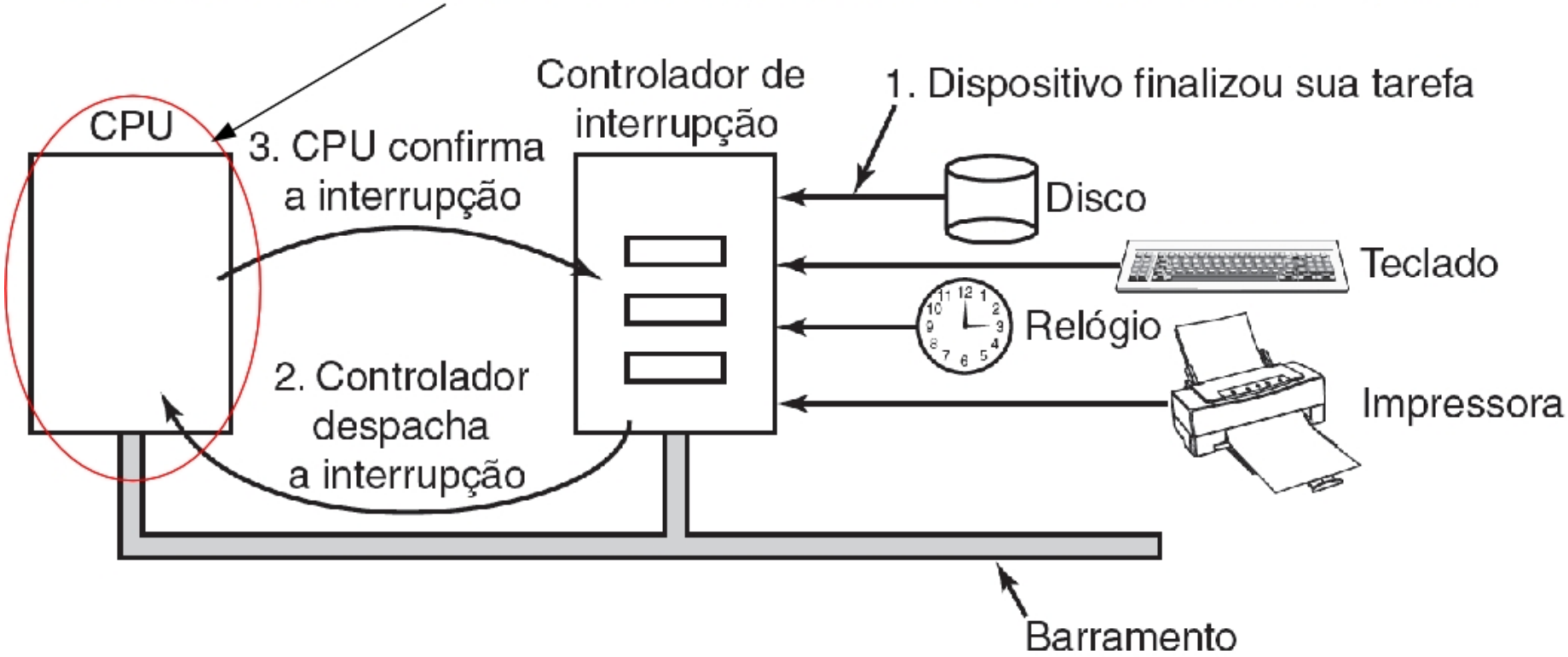
Interrupção – Pé no Hardware

Para tratar a interrupção, o controlador coloca um número, especificando o dispositivo, nas linhas de endereço do barramento, mandando um sinal para interromper a CPU

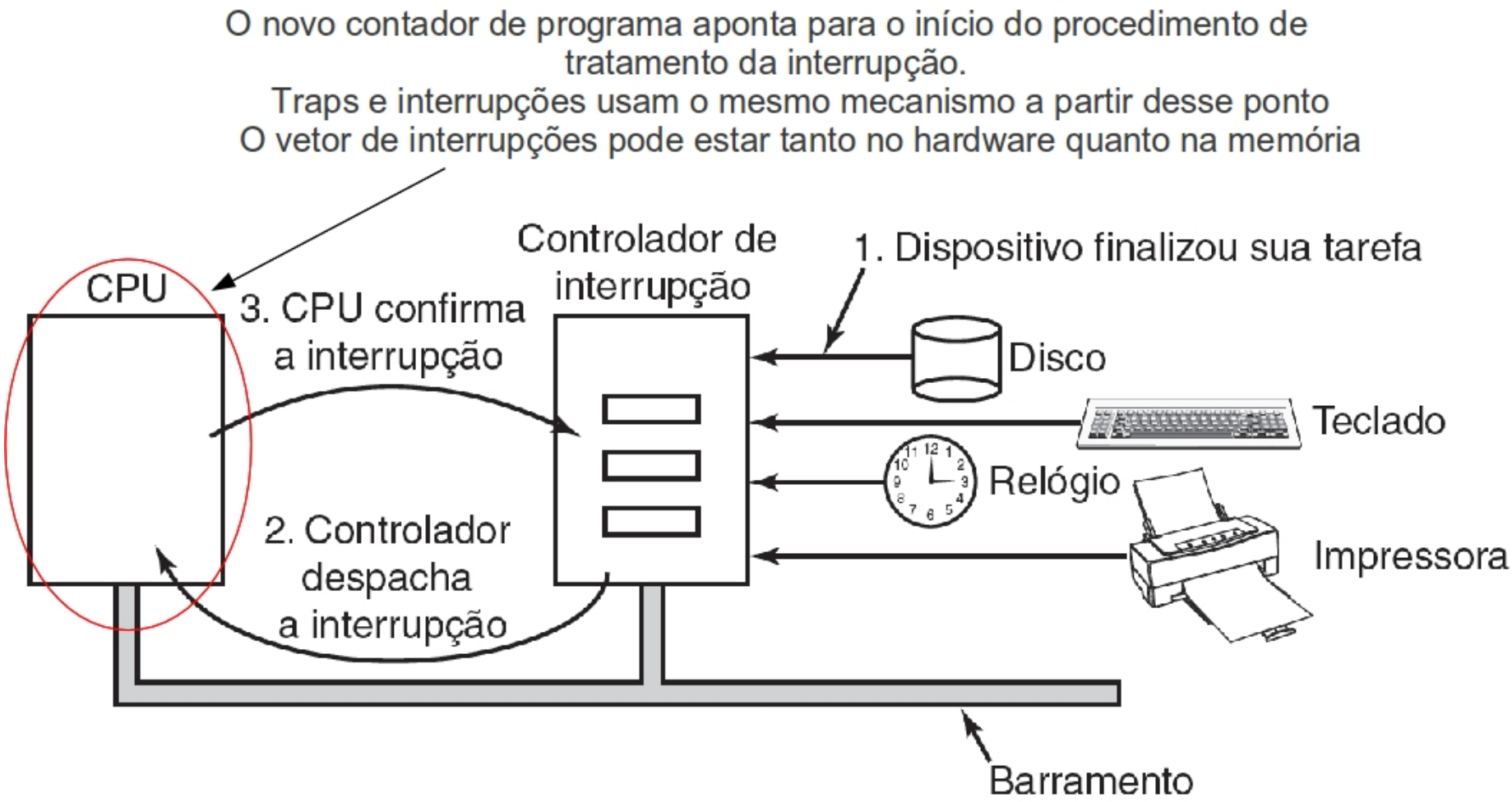


Interrupção – Pé no Hardware

O sinal faz com que a CPU pare o que está fazendo. Ela então usa o número nas linhas de endereço como índice no vetor de interrupções, obtendo um novo contador de programa



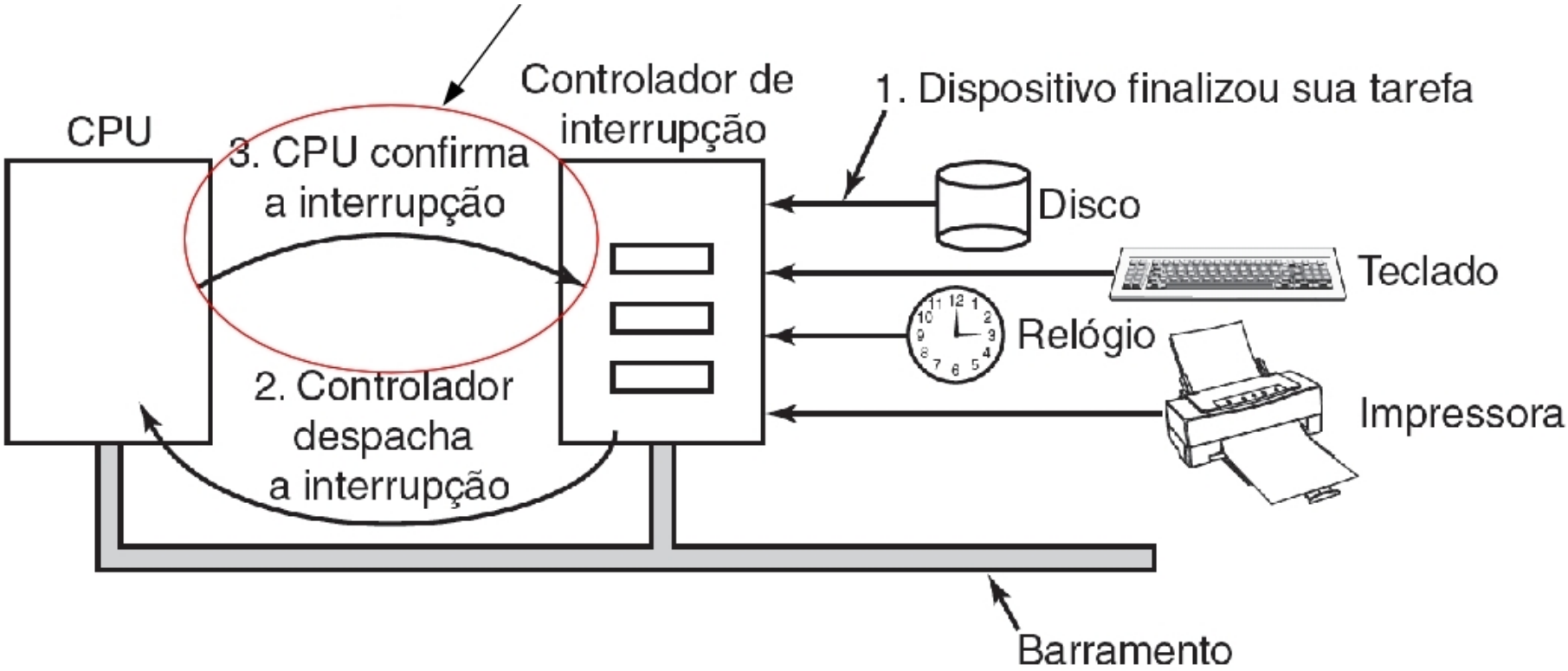
Interrupção – Pé no Hardware



Caso na memória, haverá um registrador da CPU (carregado pelo SO), que aponta para sua origem

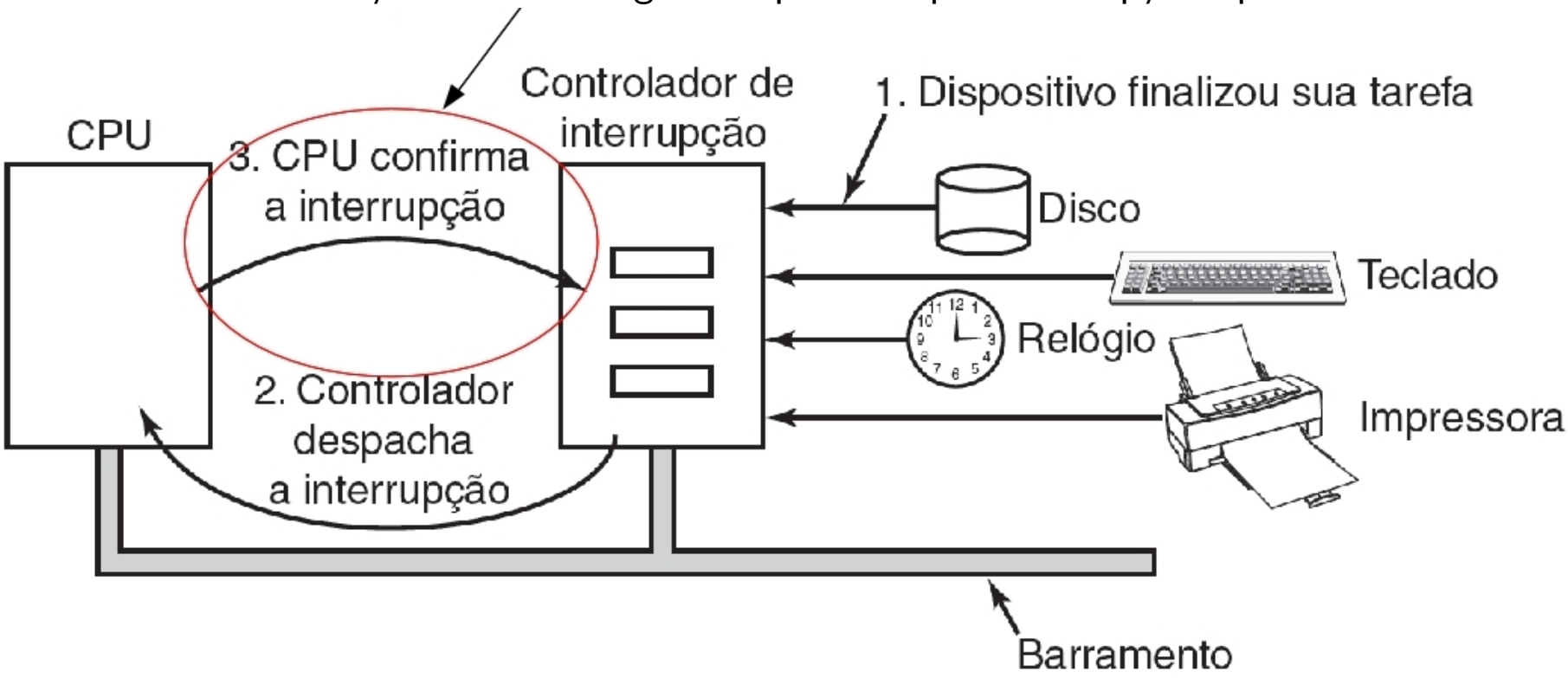
Interrupção – Pé no Hardware

Imediatamente após começar a execução, a rotina de tratamento confirma a interrupção, escrevendo um certo valor em uma das portas de E/S do controlador. Isso diz ao controlador que ele está livre para repassar outra interrupção.



Interrupção – Pé no Hardware

Assim, a CPU atrasa essa confirmação até estar livre para lidar com a próxima interrupção, evitando assim condições de corrida geradas por múltiplas interrupçoẽs quase simultâneas.



Interrupção

- Informação salva pelo hardware antes de iniciar a rotina de tratamento:
  - Pelo menos, o contador de programa
  - Em último caso, todos os registradores gerais e alguns internos
  - O que salva varia de CPU para CPU
- Onde salvar? Não há solução perfeita.
  - Registradores internos
    - Problema: Não há como enviar a confirmação (passo 3) ao controlador de interrupções até que toda a informação neles tenha sido usada (evitando sobrescrita) $\rightarrow$ toma tempo

Interrupção

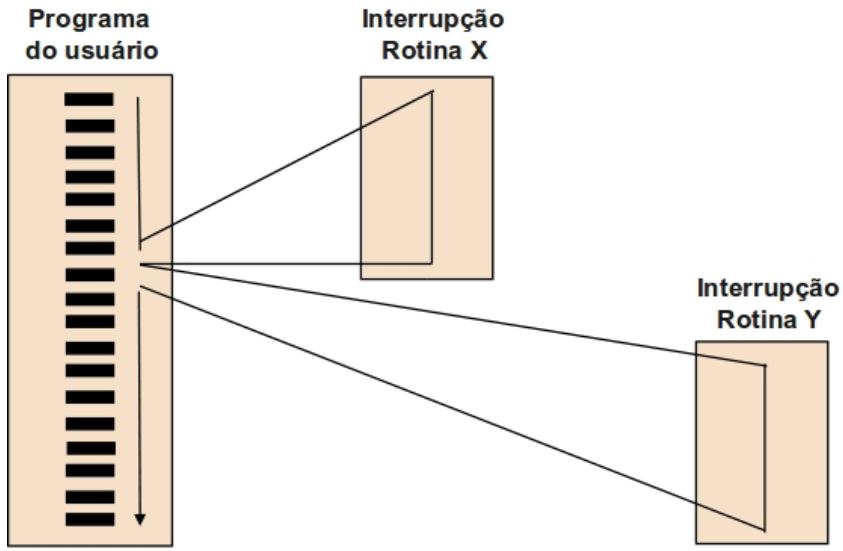
- Onde salvar? Não há solução perfeita.
  - Pilha (do processo do usuário)
    - O ponteiro da pilha pode não ser legal (por escalonamento, página não na memória, por exemplo)
    - Poderia ser o final de uma página, levando a um page fault durante a interrupção (e onde salvar o estado para tratar a page fault?)
  - Pilha do kernel
    - Maior chance do ponteiro ser legal
    - O chaveamento para modo núcleo pode exigir mudança de contexto, pela MMU, podendo mudar cache e TLB $\rightarrow$ toma tempo (veremos mais adiante)

Interrupção

- O sinal (linha) de interrupção é exibido dentro de cada ciclo de instrução do processador
- A cada ciclo de instrução, a CPU:
  - Verifica se existe interrupção. Se não $\rightarrow$ busca próxima instrução, ...
  - Se existe interrupção pendente:
    - Suspende a execução do programa
    - Salva contexto
    - Atualiza PC (Program Counter) $\rightarrow$ PC aponta para a ISR (rotina de atendimento de interrupção)
    - Executa a rotina de tratamento da interrupção
    - Recarrega contexto e continua processo interrompido

Múltiplas Interrupções

- Modelo seqüencial
  - A Rotina de Serviço desabilita as interrupções
    - Uma nova interrupção só é tratada após o retorno da anterior
    - A interrupção pode demorar a ser tratada, o que pode eventualmente ocasionar uma perda de dados
  - Finalizada a Rotina de Serviço de interrupção, o processador checa por interrupções adicionais



Múltiplas Interrupções

- Modelo cascata
  - Interrupções têm prioridade
    - Interrupções com alta prioridade interrompem rotinas de serviço de interrupções de menor prioridade
  - Exemplos de prioridade:
    - impressora (menor)
    - disco
    - comunicação (maior)