Computadores de Programação (DCC/UFRJ)

Aula 16: Fluxo de controle com exceções

Prof. Paulo Aguiar

Transferência de controle

- Durante a operação de um processador, o **ponteiro de programa (%eip)** assume uma sequência de valores $a_0, a_1, \ldots, a_n$
- Cada a_k corresponde ao endereço de uma instrução de máquina I_K

As transições entre os valores a_k assumidos pelo ponteiro de programa (ex., de a_k para a_{k+1}) são chamadas de **transferência de controle**

Fluxo de controle

Uma determinada sequência de transferências de controle é chamada **fluxo de controle** de execução do processador

- O tipo mais simples de sequência de transferência de controle é quando I_k e I_{k+1} são instruções em endereços contínuos da memória (o valor do ponteiro de programa muda incrementado do tamanho da última instrução realizada)
- Quando há instruções de **desvio** (*jumps*), **chamadas** e **retornos de subrotinas**, as transferências de controle deixam de ser incrementais

Instruções de desvio (jumps, call e ret)

As instruções de desvio que vimos até agora permitem que o programa responda a modificações no estado interno do programa, representado pelas **variáveis do programa**

Entretando, os programas também podem reagir a **mudanças no estado do sistema** (não capturadas nas variáveis do programa, e não necessariamente associadas à execução do programa em si)

- ex.: temporizador do sistema, chegada de pacotes de rede, dados recebidos do disco, notificação de término de processos filhos, etc)

Reações a mudanças no estado do sistema

- Os sistemas de computador **reagem a mudanças do estado do sistema** alterando (abruptamente) o fluxo de execução do processador
- Essas alterações são chamadas **fluxo de controle excepcional** (FCE)

Fluxo de controle excepcional (FCE)

FCE ocorre em **todos os níveis de um sistema de computador**:

1. **Exceções**: interseção do hardware com o SO
   A ocorrência de um "evento" de hardware dispara a execução de um tratador do evento
2. **Processos e sinais**: interseção do SO com as aplicações
   O kernel transfere o controle de um processo para o outro
3. **Chamadas de sistema**: interseção das aplicações com o SO
   Permitem que as aplicações requisitem serviços do SO
4. **Desvios não-locais**: interseção entre aplicações
   Um processo pode desviar a execução para outro processo

Fluxo de controle excepcional (FCE)

- **FCE** é o mecanismo básico que os SOs usam para implementar I/O, gerência de processos e de memória virtual
- **FCE** é o mecanismo usado pelas aplicações para interagir com o SO (ex., ler/escrever em arquivos, trocar mensagens, criar processos) e é o que nos permite escrever programas como *servidores Web*
- **FCE** é o mecanismo usado por linguagens como Java e C++ para implementar *mecanismos de exceção* via sentenças `try-catch`

Exceções

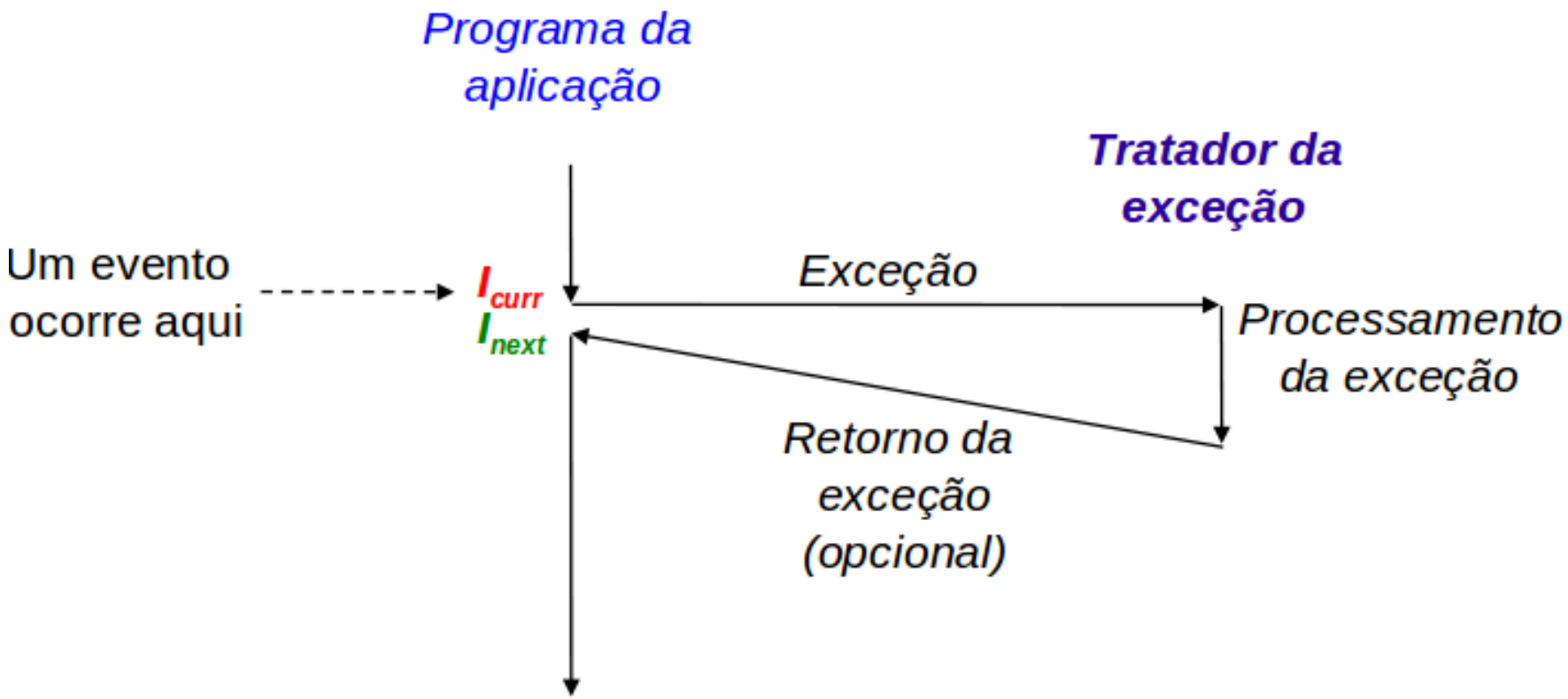
Exceções são uma forma de **controle de fluxo excepcional** implementadas em parte pelo **hardware** e em parte pelo **SO**

Uma **exceção** é uma **alteração abrupta no fluxo de execução** em resposta a uma **mudança de estado** do processador ("evento")

Eventos ou mudanças de estado do processador

- Um **evento** pode estar diretamente associado à execução da instrução corrente (ex., estouro aritmético, falta de página, divisão por zero), ou não (ex., temporizador, I/O)
- Em qualquer caso, quando um evento é detectado pelo processador, uma **chamada indireta de subrotina** é feita para uma subrotina do SO ("tratador do evento") projetada para tratar esse tipo de evento

Tratamento de exceções



Retorno do tratamento de exceções

Quando o tratador do evento (ou da exceção) termina, uma de três possibilidades ocorre:

1. **O controle volta para a instrução corrente I_{atual};**
2. **Ou o controle é passado para a próxima instrução I_{prox};**
3. **Ou o sistema aborta o programa.**

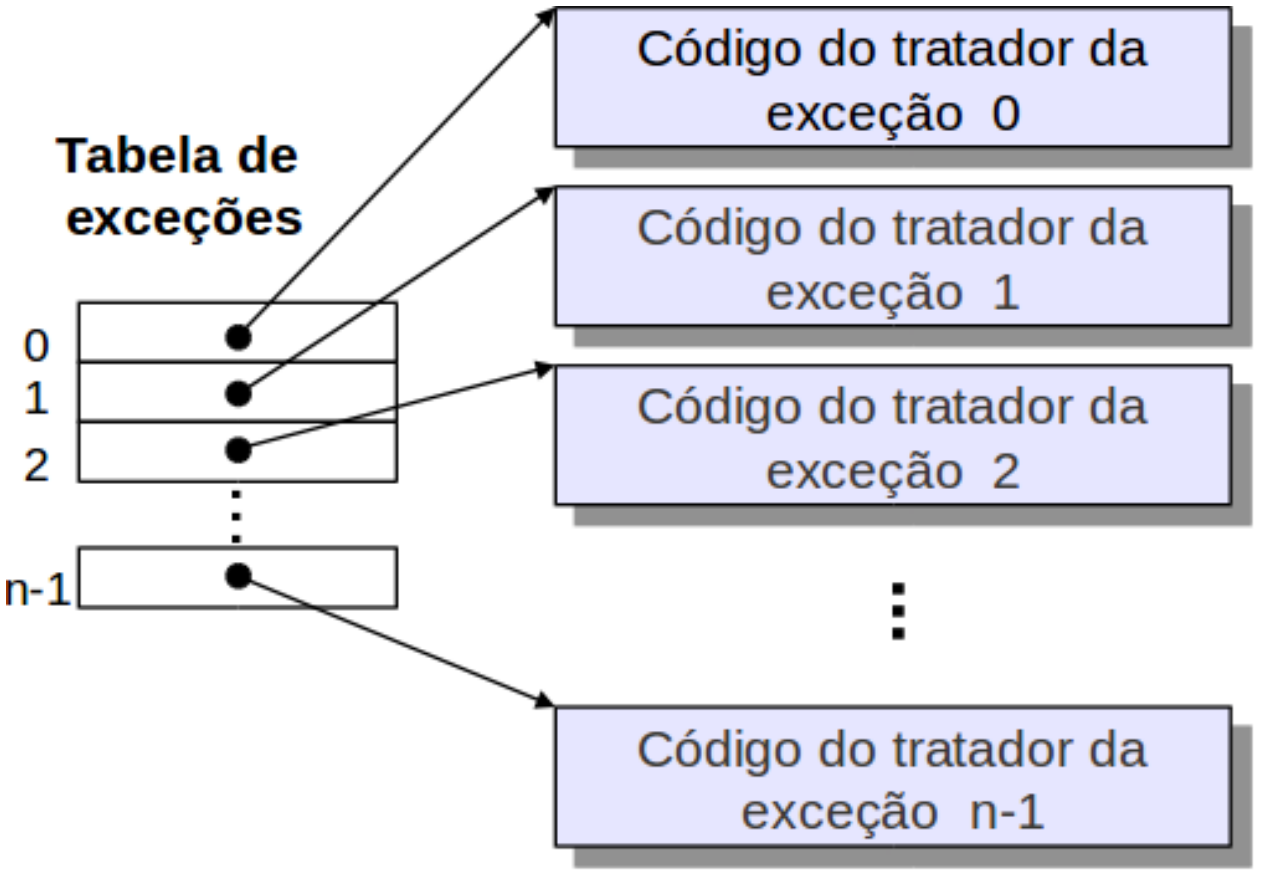
Tabela de exceções

- Cada tipo de exceção recebe um **identificador único** não negativo
- No boot do sistema (quando o computador é inicializado), o SO aloca espaço para a **tabela de exceções**

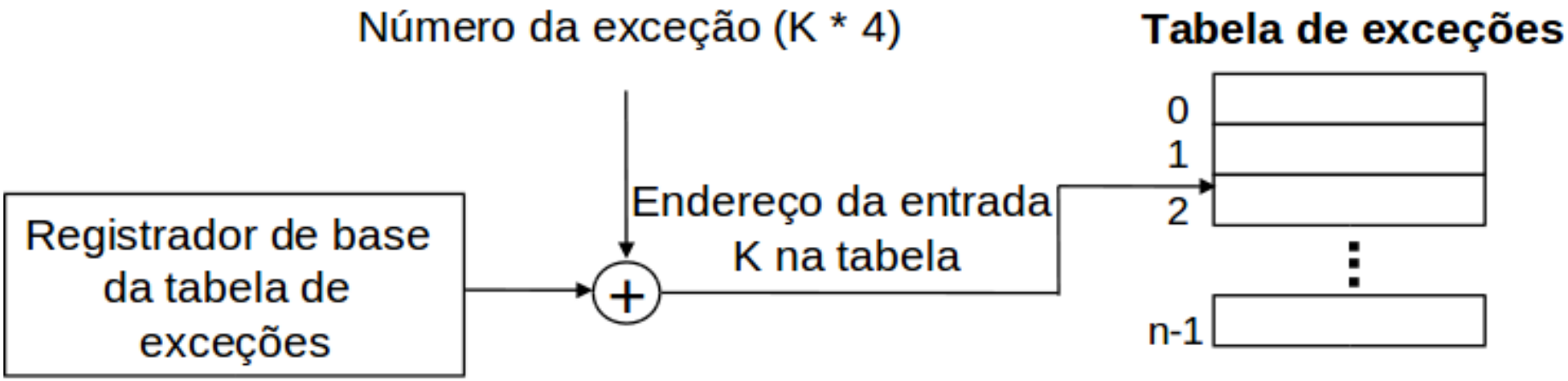
A entrada **k** contém o endereço do **tratador da exceção k**

Quando um evento do sistema ocorre, o processador identifica o número da exceção e dispara uma chamada indireta de subrotina para o tratador correspondente

Tabela de exceções



Cálculo do endereço do tratador da exceção



Tratamento da exceção

Um tratador de exceção é similar a uma chamada de subrotina, mas há algumas diferenças

- Se o segmento de código onde está a rotina de tratamento tem o mesmo nível de privilégio do programa corrente, a rotina usa a mesma pilha;
- Caso contrário, o processador comuta para a pilha de nível de privilégio correspondente.

Mudança de Pilha

Um programa pode estar rodando em **user mode** quando é interrompido e a interrupção precisa ser tratada em **kernel mode**, com completo acesso a todos os recursos do sistema.

Tratamento da exceção

Se a pilha a ser usada é a atual, o processador executa os seguintes passos principais:

- Armazena EFLAGS, CS (segmento de código ou *code segment*) e EIP na pilha;
- Carrega o seletor de segmento para o novo segmento de código e atualiza o EIP;
- Inicia a execução da rotina de tratamento.

Tratamento da exceção

Se a pilha a ser usada é outra, o processador executa os seguintes passos principais:

- Salva internamente o conteúdo de SS (segmento de pilha ou *stack segment*), ESP, EFLAGS, CS e EIP;
- Carrega o seletor de segmento e o ponteiro de pilha para a nova pilha;
- segue os demais passos do caso anterior...

Chamada e retorno de um tratador de exceção

- A instrução **INT n** permite que um programa (ou tarefa) chame explicitamente um tratador de interrupção ou exceção
- A instrução recebe como argumento o número da exceção
- O retorno de uma rotina de tratamento de exceção é feito com a instrução **IRET** (*interrupt return*, similar a RET, exceto que restaura também o conteúdo dos demais registradores salvos)

Classificação das exceções

Assíncronas: eventos de I/O externos ao processador

- **Interrupção** (*interrupt*)

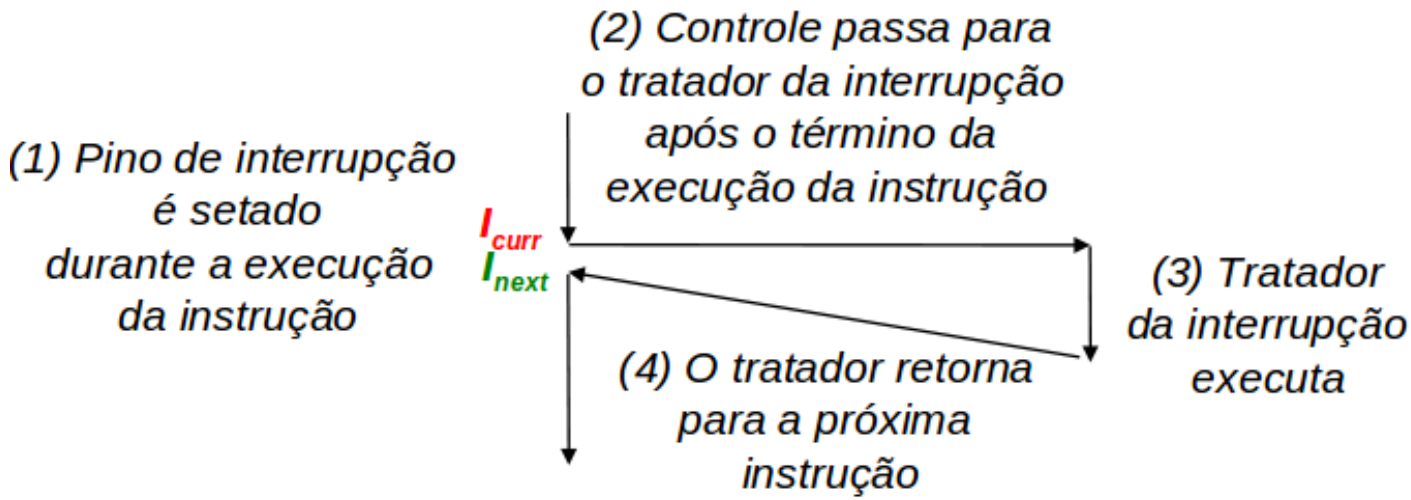
Síncronas: resultado direto da execução de uma instrução

- **Falha** (*fault*)
- **Armadilha** (*trap*)
- **Aborto** (*abort*)

Interrupções

- Ocorrem **assincronamente** como resultado de **sinais de dispositivos de I/O** que são externos ao processador (ex.: adaptador de rede, controlador de disco, temporizador, etc)
- Sinalizam um pino do processador e colocam o número da exceção no barramento do sistema

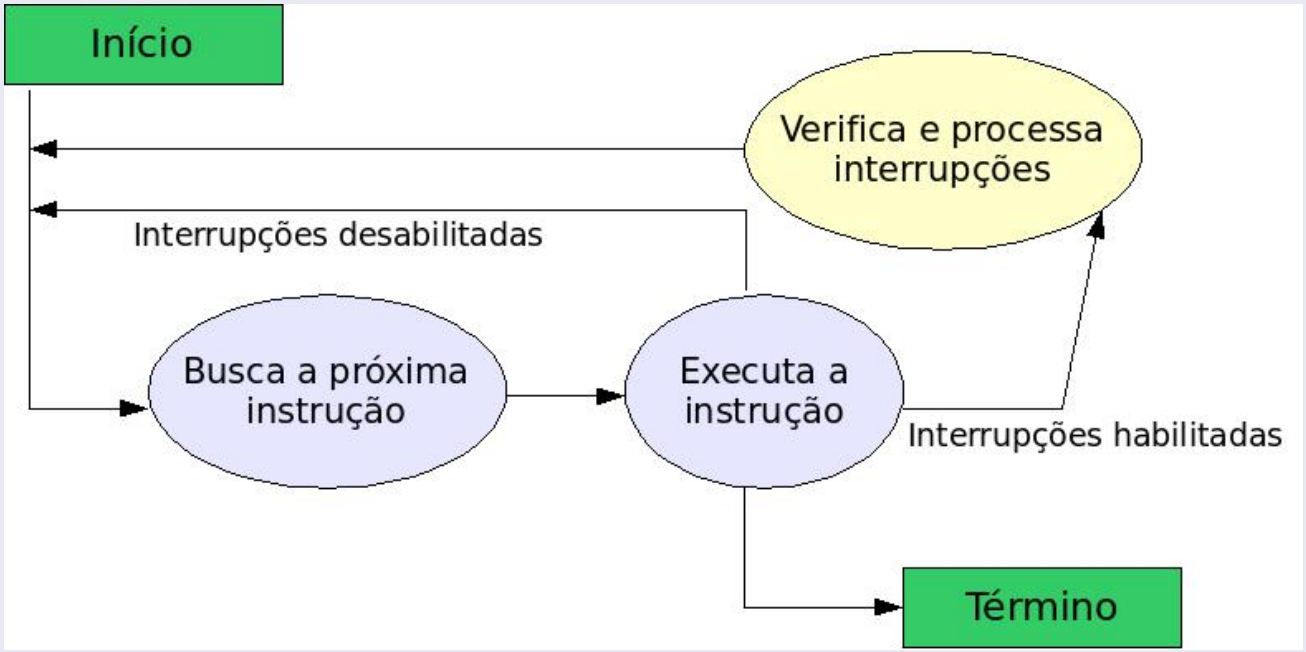
Tratamento de interrupção



Após o tratamento da interrupção sempre retorna para a próxima instrução (aquela que seria executada caso não houvesse ocorrido a interrupção)

Ciclo de instruções com interrupções

Quando a interrupção é tratada?

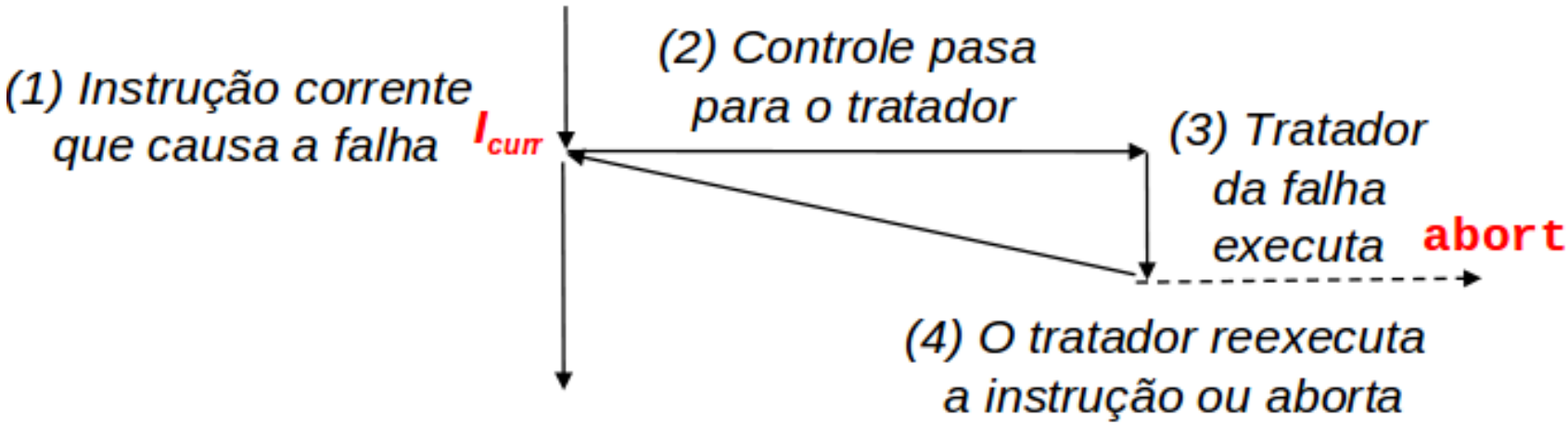


Falhas

Resultam de **condição de erro que o tratador de exceção pode ser capaz de corrigir**

- Se for corrigida, permite que o programa seja retomado sem perda de continuidade (ex., "falha de página"), com o endereço de retorno apontando para a instrução que causou a falha;
- Se não for possível ser corrigida, o programa aborta.

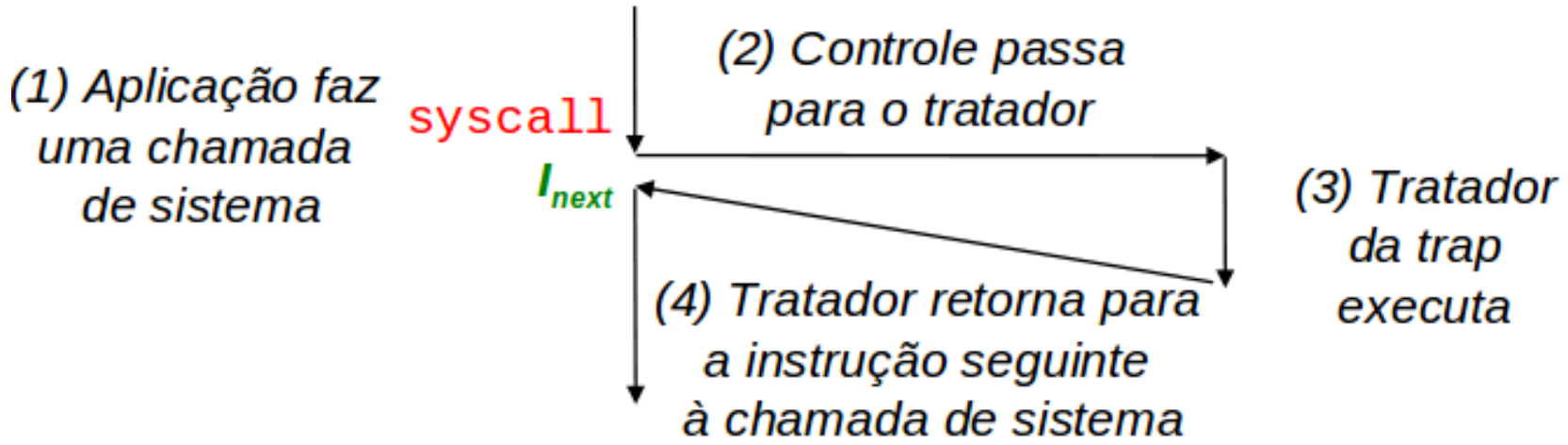
Falhas


(1) Instrução corrente
que causa a falha
I_{curr}
(2) Controle pasa
para o tratador
(3) Tratador
da falha
executa
abort
(4) O tratador reexecuta
a instrução ou aborta

Traps e chamadas de sistema

- São **exceções intencionais** que ocorrem como resultado da execução de uma instrução
- Permite que o programa continue a partir da instrução seguinte à que causou a exceção
- O uso mais importante de *traps* é prover uma interface de chamada de procedimento entre aplicações e kernel do SO, conhecidas como **chamadas de sistema**

Traps


(1) Aplicação faz uma chamada de sistema
syscall
(2) Controle passa para o tratador
(3) Tratador da trap executa
I_{next}
(4) Tratador retorna para a instrução seguinte à chamada de sistema

Chamadas de sistema

- As aplicações normalmente precisam requisitar serviços do SO
  - ex., leitura/escrita de arquivo (`read`), criação de processos (`fork`), carga de um novo programa (`execve`), finalização de programa (`exit`)
- Para organizar o acesso a esses serviços, os processadores oferecem uma **instrução especial** que as aplicações executam quando querem requisitar esses serviços

Ao executar essa instrução especial, uma **trap** é gerada e o tratador de exceção associado é ativado (ele decodifica o argumento que identifica a chamada de sistema e chama a subrotina do kernel apropriada)

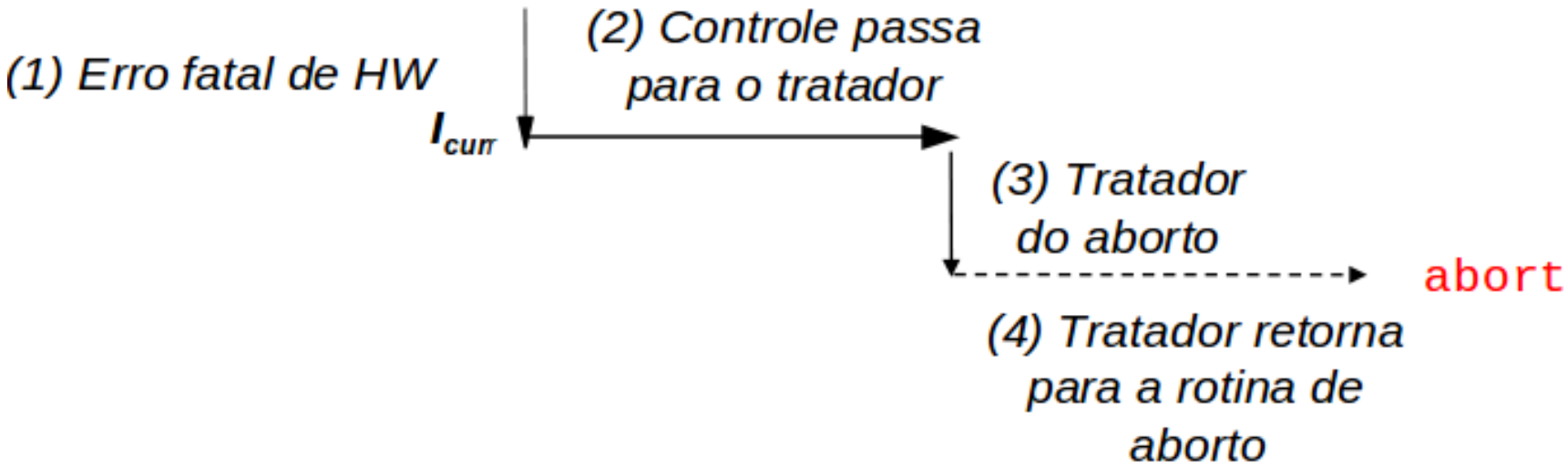
Chamadas de sistema X chamadas de funções

- Do ponto de vista do programador, uma **chamada de sistema** é idêntica a uma **chamada de função**
- A principal diferença é que as chamadas de sistemas executam no **modo kernel**, o qual permite executar instruções especiais e acessar a pilha do kernel

Abortos

- **Exceções causadas por erros graves/fatais**
  - Ex.: erros de paridade em memória, divisão por zero (*floating exception*), tentativa de escrita em área read only (*segmentation fault*), erro de hardware (*machine check*), etc
- Se não é possível indicar a localização precisa da instrução que causou o erro (ex., valores ilegais em uma tabela do sistema), então o programa **não é retomado**

Abortos


(1) Erro fatal de HW
I_{curr}
(2) Controle passa para o tratador
(3) Tratador do aborto
abort
(4) Tratador retorna para a rotina de aborto

Exceções nos sistemas Linux/IA32

Na arquitetura IA32 há 289 tipos diferentes de exceções:

- **0–31**: exceções definidas pelo fabricante (Intel)
- **32–289**: interrupções e traps definidas pelo SO

unistd.h define o número das chamadas no sistema Linux

```
#ifndef _ASM_I386_UNISTD_H_
#define _ASM_I386_UNISTD_H_
#define __NR_restart_syscall      0
#define __NR_exit   1
#define __NR_fork   2
#define __NR_read   3
#define __NR_write   4
#define __NR_open   5
#define __NR_close   6
#define __NR_waitpid   7
#define __NR_creat   8
#define __NR_link   9
#define __NR_unlink  10
#define __NR_execve  11
...
#define __NR_request_key        287
#define __NR_keyctl             288
```

Chamadas de sistema no Linux

- Cada chamada de sistema tem um identificador único que corresponde a um offset na tabela de desvios do kernel
- Na arquitetura IA32 as chamadas de sistemas são acessadas via a instrução especial de trap **int n** (onde "n" é um identificador de exceção)
- Historicamente, chamadas de sistema são providas pela entrada 128 (**0x80**)

Gerando exceções nas instruções do programa

- No Linux, antes de executar a instrução **int 0x80**, o número da chamada de sistema (i.e. a definição do serviço a ser executado) deve ser colocado no registrador **%eax**, e os argumentos necessários (no máximo 6) nos registradores **%ebx, %ecx, %edx, %esi, %edi, %ebp**
- Depois que o SO executa o serviço requisitado, o controle retorna para a instrução seguinte a **int n**, com o valor de retorno em **%eax**

Fazendo chamadas ao SO

Exemplos de código

- (*Ver código anexo nos arquivos* `hello.s`, `copia.c` *e* `chamadas.s`)
- **Tarefa de casa**: implementar as demais funções do arquivo `copia.c` (myread, mywrite e myclose) em linguagem de montagem (ver aqui chamadas de sistema do Linux)

Referências bibliográficas

- Manuais Intel IA-32
- *Understanding the Linux kernel*, D. P. Bovet and M. Cesati, O'Reilly, 3ed, 2006.
- *Computer Systems–A Programmer's Perspective* (**Cap. 8**, seções 8.1.1, 8.1.2, 8.1.3)