Interrupções, Exceções e Chamadas ao SO

Noemi Rodriguez
Ana Lúcia de Moura

http://www.inf.puc-rio.br/~inf1018

Fluxo de Controle

Fluxo de controle de um programa

a_0 a_1 a_2 ... a_n → sequência de endereços

i_0 i_1 i_2 ... i_n → sequência de instruções

Fluxo de Controle

Fluxo de controle de um programa

a_0 a_1 a_2 ... a_n → sequência de endereços

i_0 i_1 i_2 ... i_n → sequência de instruções

Dois mecanismos alteram o fluxo sequencial

- desvios (jumps) e procedimentos (chamada e retorno)

Fluxo de Controle

Fluxo de controle de um programa

a_0 a_1 a_2 ... a_n → sequência de endereços

i_0 i_1 i_2 ... i_n → sequência de instruções

Dois mecanismos alteram o fluxo sequencial

- desvios (jumps) e procedimentos (chamada e retorno)

Outro mecanismo: **exceções/interrupções**

- a CPU **interrompe** o programa em execução e desvia o controle para um **tratador** (procedimento do SO)

Interrupções e Exceções

Eventos que indicam a existência de uma condição que requer a atenção **imediata** do processador

- resultam em uma transferência de controle **forçada** para um **tratador**

Interrupções e Exceções

Eventos que indicam a existência de uma condição que requer a atenção **imediata** do processador

- resultam em uma transferência de controle **forçada** para um **tratador**

Eventos **assíncronos** independem da instrução corrente

- interrupções geradas por dispositivos de E/S: teclado, controlador de disco, interface de rede, ...

Interrupções e Exceções

Eventos que indicam a existência de uma condição que requer a atenção **imediata** do processador

- resultam em uma transferência de controle **forçada** para um **tratador**

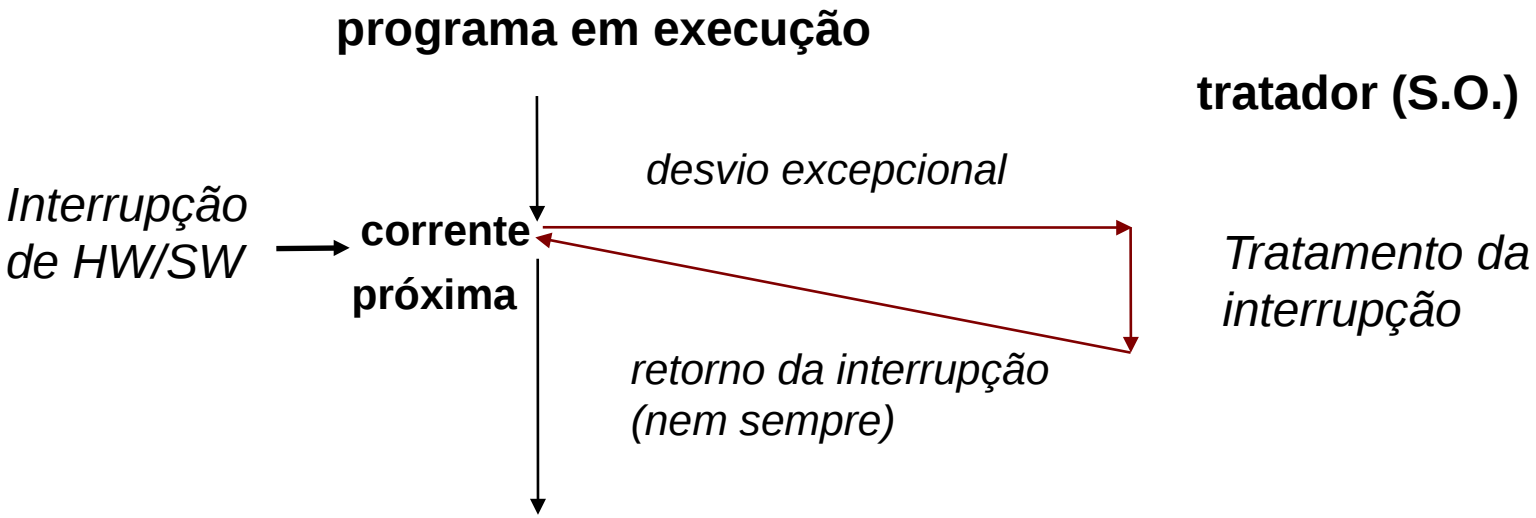
Eventos **assíncronos** independem da instrução corrente

- interrupções geradas por dispositivos de E/S: teclado, controlador de disco, interface de rede, ...

Eventos **síncronos** resultam da instrução executada

- geradas pelo programa (**traps**), geradas pelo hardware (**faults** e **aborts**)

Alteração no Fluxo de Controle

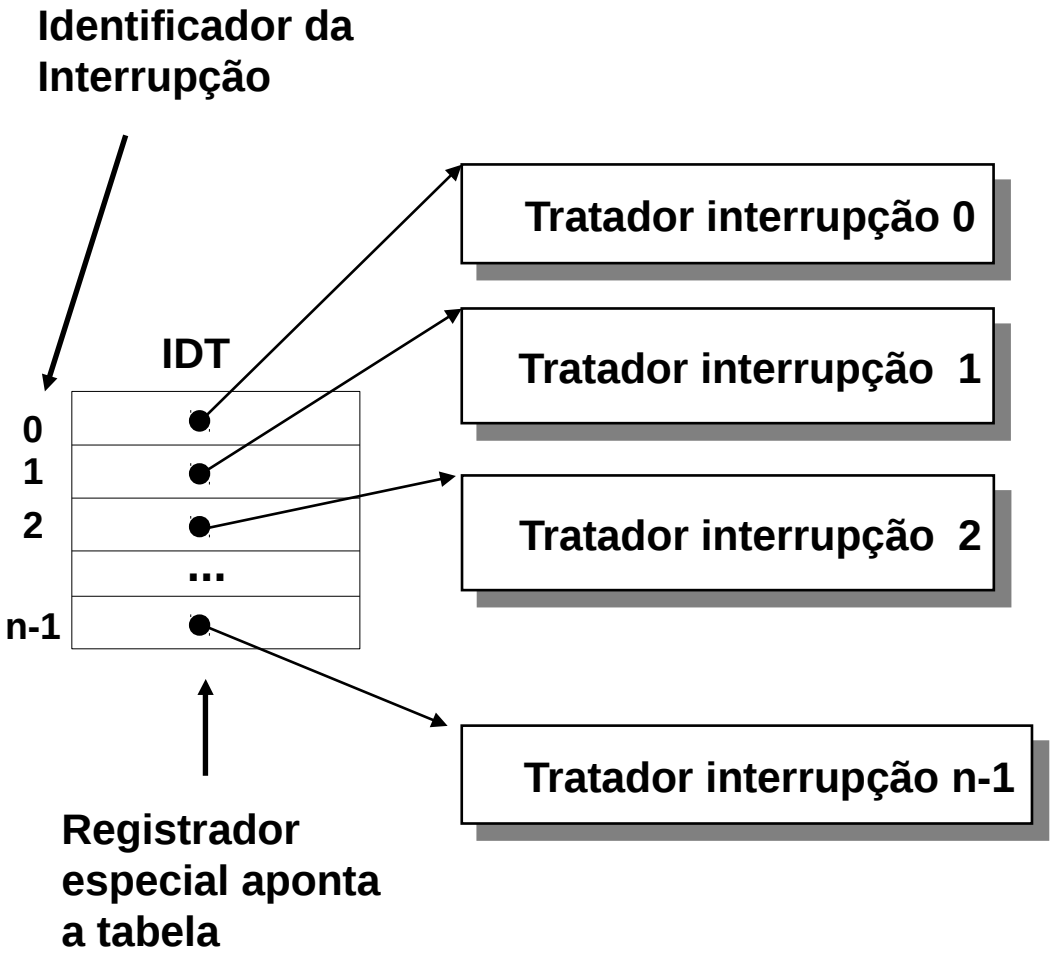


A CPU interrompe o programa em execução e desvia o controle para um tratador

- o tratador executa em modo **privilegiado**

Cada tipo de interrupção é associado a um **identificador**

Tabela de Descritores de Interrupção



- Durante sua inicialização, o SO preenche a IDT com as informações dos tratadores e coloca o endereço da tabela em um registrador especial
- Durante a execução, a CPU consulta a tabela para realizar o desvio para o tratador de uma interrupção
- Alguns identificadores de interrupção são específicos da plataforma
  - *page fault*, div. zero, gpf, opcode inválido, ...

Salvamento de Contexto

Para que o programa interrompido possa prosseguir após o tratamento da interrupção é necessário guardar seu **contexto de execução**

- a CPU salva na **pilha do tratador** o **%rsp**, o **%rip** e o **%rflags** do programa interrompido

Salvamento de Contexto

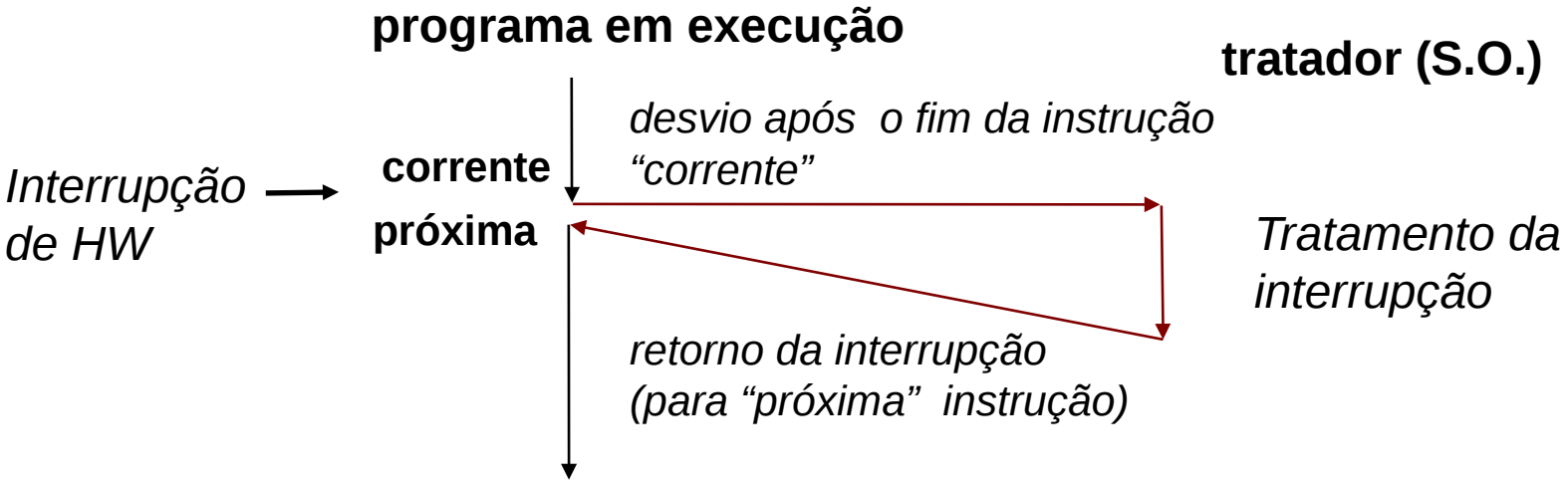
Para que o programa interrompido possa prosseguir após o tratamento da interrupção é necessário guardar seu **contexto de execução**

- a CPU salva na **pilha do tratador** o **%rsp**, o **%rip** e o **%rflags** do programa interrompido

Ao terminar sua execução, o tratador restaura o contexto através de uma instrução (privilegiada) especial

- a CPU recupera os registradores salvos, e restaura a pilha do programa interrompido

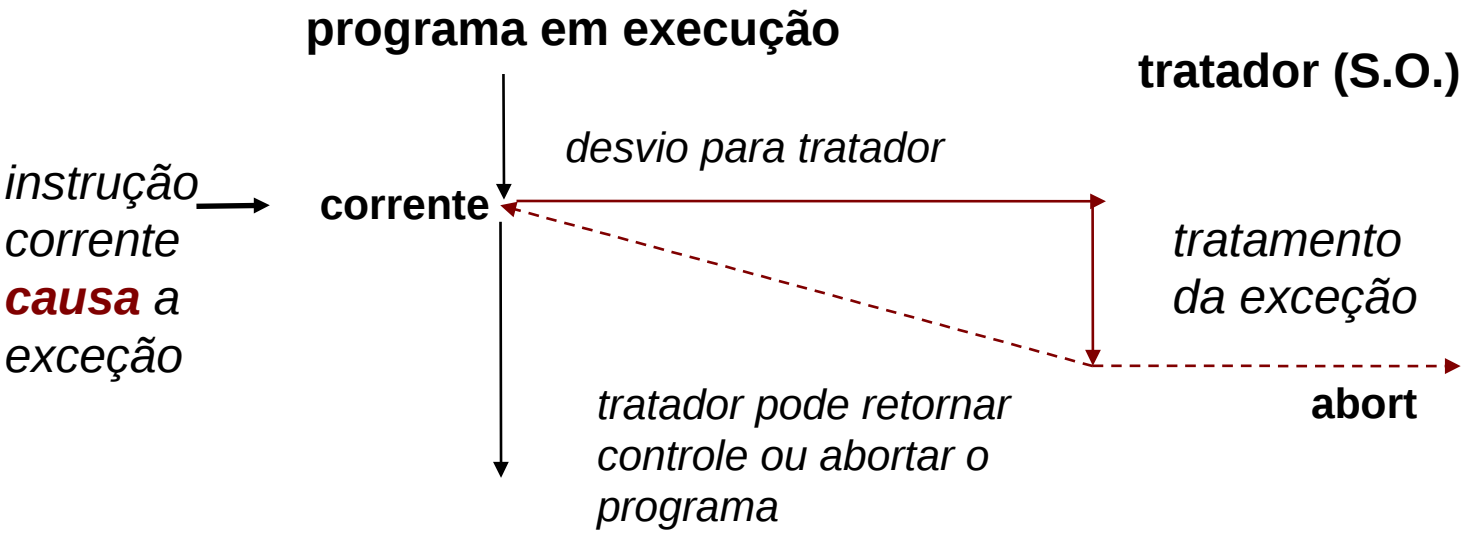
Interrupções Assíncronas



A execução do tratador da interrupção é **transparente** para o programa interrompido

- após o tratamento da interrupção o programa continua sua execução como se a interrupção não houvesse ocorrido
- a menos de situações especiais, como o acionamento de ctrl-C!

Exceções: falhas e erros fatais



Se uma falha não pode ser corrigida, o SO aborta o programa

- divisão por zero, acesso inválido à memória (GPF), instrução inválida

Se o tratador corrigiu a falha, o controle retorna para a instrução "corrente" (que é re-executada!)

- ***page fault***: a instrução referencia um ***endereço virtual*** cuja página correspondente não está na memória principal

Chamadas ao Sistema Operacional

Uma chamada ao SO (**syscall**) corresponde a um serviço executado por um procedimento do Sistema Operacional (*read, write, exit, ...)*

- uma **syscall** não pode ser invocada com o mecanismo de chamada convencional de funções (**call**)
- SO deve executar em **modo privilegiado**!

Chamadas ao Sistema Operacional

Uma chamada ao SO (**syscall**) corresponde a um serviço executado por um procedimento do Sistema Operacional (*read, write, exit, ...)*

- uma **syscall** não pode ser invocada com o mecanismo de chamada convencional de funções (**call**)
- SO deve executar em **modo privilegiado**!

A arquitetura x86-64 provê uma instrução para um programa de usuário invocar um procedimento do SO (**syscall**)

- um registrador especial armazena o endereço do **tratador** de chamadas ao Sistema Operacional
- o tratador termina sua execução usando uma instrução privilegiada (**sysret**)

Linux x86-64: interface com syscall

Cada chamada é associada a um identificador

- esse identificador é passado no **%rax**

Linux x86-64: interface com syscall

Cada chamada é associada a um identificador

- esse identificador é passado no **%rax**

Parâmetros e retorno em registradores

- parâmetros em %rdi, %rsi, %rdx, **%r10**, %r8, %r9
- valor de retorno no **%rax**

Linux x86-64: interface com syscall

Cada chamada é associada a um identificador

- esse identificador é passado no **%rax**

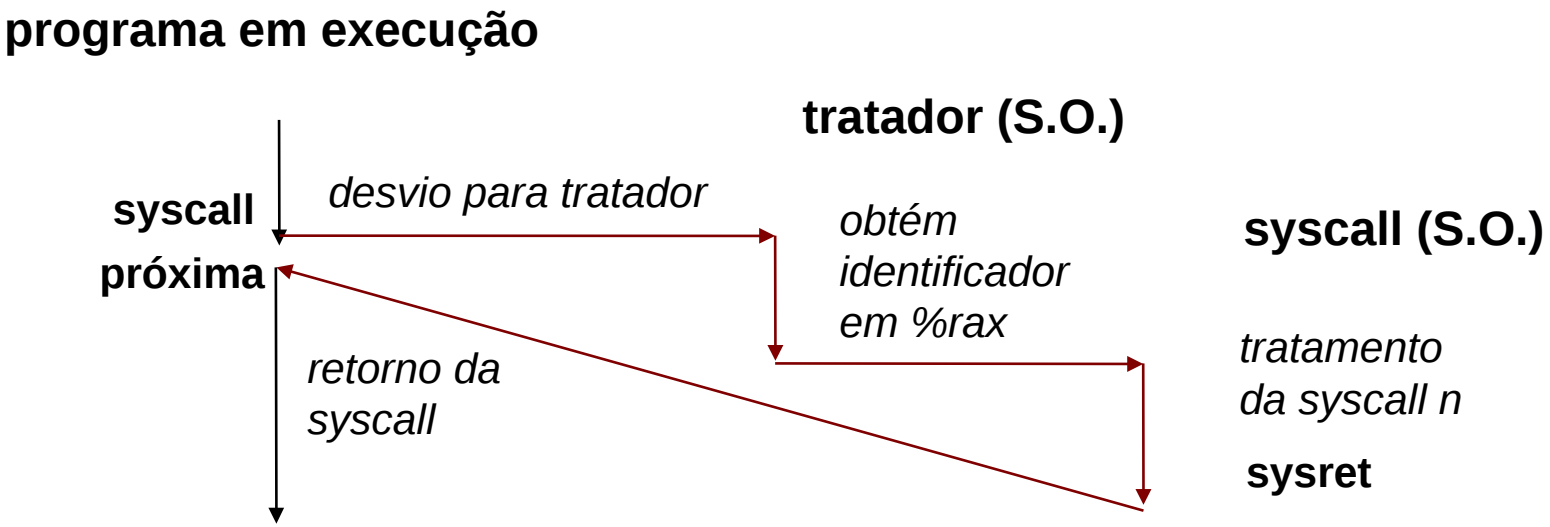
Parâmetros e retorno em registradores

- parâmetros em %rdi, %rsi, %rdx, **%r10**, %r8, %r9
- valor de retorno no **%rax**

Antes de desviar o controle para o tratador, a CPU guarda o contexto do programa de usuário

- %rip em %rcx e %rflags em %r11

Tratamento de syscall



O programa envia o código da syscall (n) em %rax e os argumentos nos outros registradores

Quando o controle é transferido de volta ao programa (“*próxima instrução*”) o resultado da chamada está em %rax

Algumas syscalls

%rax	nome	%rdi	%rsi	%rdx	retorno (%rax)
60	sys_exit	**int status**	-	-	-
0	sys_read	**int fd**	**void *buf**	**size_t count**	**ssize_t ou -1**
1	sys_write	**int fd**	**void *buf**	**size_t nbyte**	**ssize_t ou -1**
2	sys_open	**char *path**	**int flags**	(opcional)	**int (fd) ou -1**
3	sys_close	**int fd**			**int (0 ou -1)**
8	sys_lseek	**int fd**	**off_t offset**	**int whence**	**off_t ou -1**

Exemplo de syscall

```
int f (...) {
   ...

   if (...)
    exit (2);

   ...
}
```



```
f:
 pushq %rbp
 movq %rsp, %rbp
 ...
 movq $60,%rax /* sys_exit */
 movl $2,%edi
 syscall
 ...
```

Funções “wrapper”

A biblioteca padrão de C provê funções “wrapper” para a maioria das chamadas ao sistema

- preparação dos argumentos (se necessário) e acionamento da syscall
- protótipos e definições em unistd.h