Software Básico

Interrupções e Chamadas ao SO

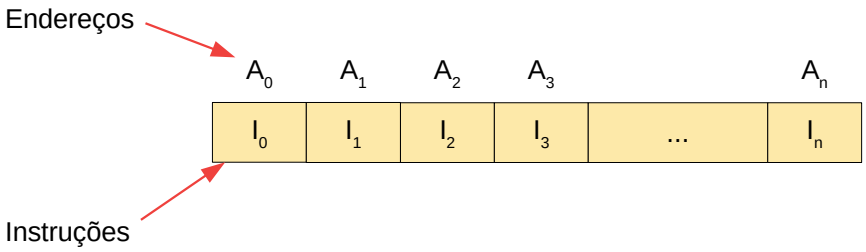
INSTITUTO DE INFORMÁTICA - UFG

Reconhecimento

- Material produzido por:
  - Noemi Rodriguez – PUC-Rio
  - Ana Lúcia de Moura – PUC-Rio
- Adaptação
  - Bruno Silvestre – UFG

Fluxo de Controle

- Fluxo de controle de um programa
  - Ciclo do processador: *fetch-decode-execute*



Fluxo de Controle

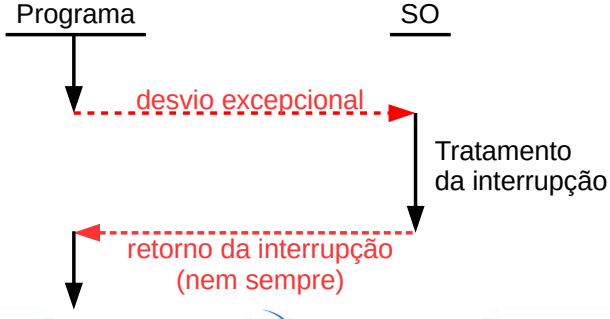
- Dois mecanismos alteram o fluxo sequencial
  - Desvios (*jumps*) e procedimentos (chamada e retorno)
- Outro mecanismo: Interrupções e exceções
  - A CPU interrompe o programa em execução e desvia o controle para um tratador (procedimento) do SO

Interrupções e Exceções

- Interrupções e exceções são eventos que indicam a existência de uma condição que requer a atenção imediata do SO
  - Resultam em uma transferência de controle forçada para um tratador do SO
- Eventos podem ser:
  - Síncronos
  - Assíncronos

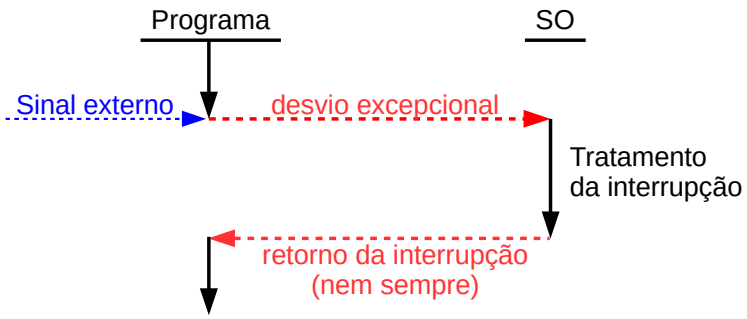
Interrupções e Exceções

- Eventos síncronos são causados de alguma forma pela instrução que está sendo executada
  - Geradas pelo programa (*traps*): abrir arquivo, alocar memória, etc.
  - Geradas pelo hardware (*faults* e *aborts*): paginação



Interrupções e Exceções

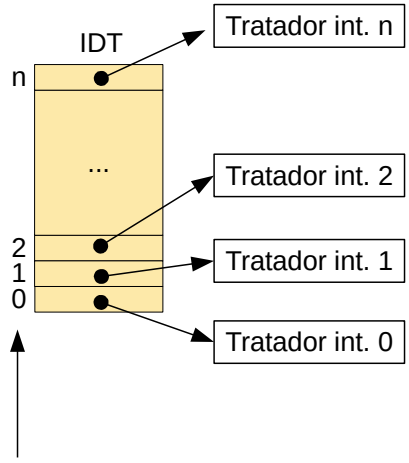
- Eventos assíncronos independem da instrução corrente, é um evento externo
  - Interrupções geradas por dispositivos de E/S: teclado, controlador de disco, interface de rede, etc.



Interrupções e Exceções

- A CPU interrompe o programa em execução e desvia o controle para um tratador do SO
  - O tratador executa em modo privilegiado
- Do ponto de vista do programa, é como se nada tivesse acontecido, ou seja, ele não nota que o processamento foi interrompido e retomado
- Cada interrupção possui um identificar para diferenciá-las

Tabela de Descritores de Interrupção (IDT)



- Durante a 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, ...

Troca 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 do tratador, o SO restaura o contexto através de uma instrução (privilegiada) especial
  - A CPU recupera os registradores salvos, restaurando a execução do programa interrompido

Chamada ao
Sistema Operacional

Chamadas ao Sistema Operacional

- Uma chamada ao SO (*syscall*) corresponde ao pedido para executar um serviço oferecido pelo 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

- A arquitetura x86-64 provê uma instrução para um programa de usuário invocar um procedimento do SO: *syscall*
- Um registrador armazena o endereço do do tratador de *syscalls* do SO
  - Ou seja, todas as *syscalls* são direcionadas a um único tratador
- O tratador termina sua execução usando uma instrução privilegiada: *sysret*

Linux x86-64: Interface com *syscall*

- Cada serviço do SO tem um identificador associado
  - 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 da syscall, a CPU guarda o contexto do programa de usuário:
  - %rcx = %rip
  - %r11 = %rflags

syscalls

Syscall (%rax)	Nome	Descrição	%rdi	%rsi	%rdx	Retorno (%rax)
0	sys_read	Lê do descritor	int fd	void *buf	size_t count	ssize_t ou -1
1	sys_write	Escreve no descritor	int fd	void *buf	size_t count	ssize_t ou -1
2	sys_open	Abre um arquivo	char *path	int flags	int mode	fd ou -1
3	sys_close	Fecha um descritor	int fd	-	-	0 ou -1
60	sys_exit	Termina o programa	int status	-	-	-

Ver mais em http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

Exemplo de syscall

```
void foo() {
  exit(2);
}
```

```
foo:
    pushq %rbp
    movq  %rsp, %rbp
    movq  $60,%rax /* sys_exit */
    movl  $2,%edi
    syscall
    leave
    ret
```

Funções *wrapper*

- A biblioteca padrão de C provê funções *wrappers* para a maioria das chamadas ao sistema operacional
  - Prepara os argumentos (se necessário) e aciona a syscall
  - Protótipos e definições em “unistd.h”