

# Sistemas Hardware-Software

Aula 17 - Sinais II: recebimento e concorrência

**Engenharia**

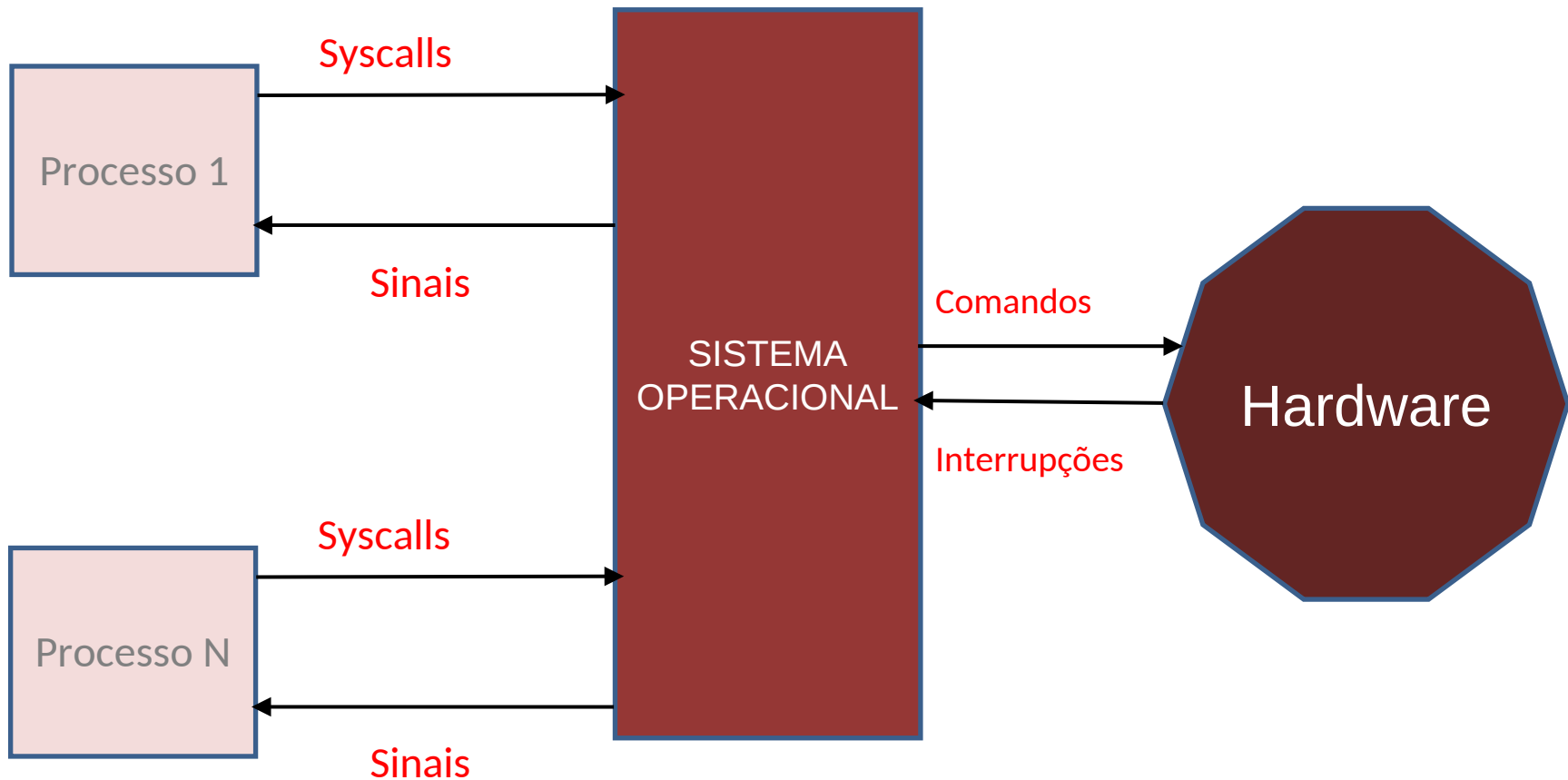
Fabio Lubacheski

Maciel C. Vidal

Igor Montagner

Fábio Ayres

# Interação do Kernel com processos



# (Alguns) Sinais POSIX

Signal	Default Action	Description
SIGABRT	Terminate (core dump)	Process abort signal
SIGALRM	Terminate	Alarm clock
SIGCHLD	Ignore	Child process terminated, stopped, or continued.
SIGFPE	Terminate (core dump)	Erroneous arithmetic operation.
SIGSEGV	Terminate (core dump)	Segmentation fault.
SIGINT	Terminate	Terminal interrupt signal. (Ctrl+C)
SIGKILL	Terminate	Kill (cannot be caught or ignored).
SIGTERM	Terminate	Termination signal.

# Exemplos de usos de sinais

- **Ctrl+C** envia um sinal **SIGINT (2)** para o processo. Ele pode ser capturado e fazer com que o programa feche conexões e arquivos abertos, por exemplo.
- O sinal **SIGSTOP (19)** é usado para deixar um processo em background. Ele fica parado até ser resumido por SIGCONT
- O sinal **SIGKILL (9)** interrompe um processo imediatamente. Ele não pode ser ignorado.
- O sinal **SIGTERM (15)** ele também causa o término do processo, como em SIGKILL(9), porém **pode ser tratado ou ignorado pelo processo.**

# Enviando um sinal

- **Kernel** detectou um evento de sistema, tal como uma divisão-por-zero (**SIGFPE**) ou término de um processo filho (**SIGCHLD**)
- Outro processo invocou a chamada de sistema **kill** para explicitamente pedir ao **Kernel** que envie um sinal ao processo destinatário.

# Correção

## **Enviando sinais II (20 minutos)**

1. A chamada de sistema kill

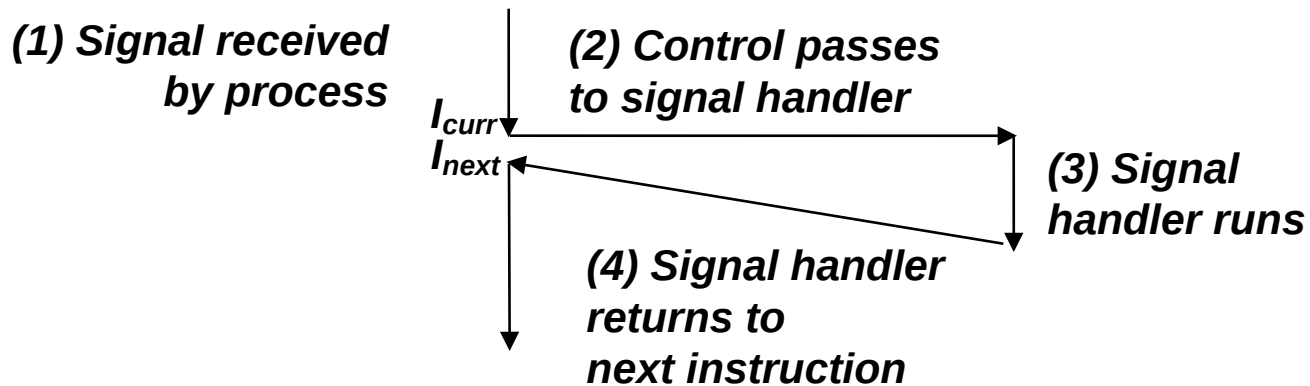
# Recebendo um sinal

O Kernel força o processo destinatário a reagir de alguma forma à entrega do sinal. O destinatário pode:

- **Ignorar:** terminar operação que não pode ser interrompida
- **Terminar:** Adicionar tempo limite ao processo e depois **terminar** o processo.
- **Capturar** o sinal e executar uma função definida no tratador do sinal definido pelo usuário (handler).

# Captura de sinais

Similar a uma exceção de hardware sendo chamada em resposta a um evento assíncrono





# Capturar um sinal

```
#include <signal.h>
int sigaction(int signum, const struct
               sigaction *act,
               struct sigaction *oldact);
```

Função permite modificar e/ou associar uma ação associada a um sinal informado em **signum**.

Se **act** for diferente de NULL, a nova ação para o sinal **signum** é executada a partir de **act**. Se **oldact** é diferente de NULL, a ação anterior é salva em **oldact**.

Retorna **0** se **OK** e **-1** em caso de **erro**.

# Capturar um sinal

```
struct sigaction {
```

```
    void      (*sa_handler)(int);
```

- **SIG\_IGN** para ignorar
- **SIG\_DFL** para padrão
- Nome de uma função

```
    void      (*sa_sigaction)(int,  
siginfo_t*, void *);
```

```
    sigset_t   sa_mask;
```

Lista de sinais que devem ser bloqueados e não serão entregues até sa\_handler finalizar.

```
    int        sa_flags;
```

Opções de recepção. Usaremos 0 sempre aqui.

```
    void      (*sa_restorer)(void);
```

```
};
```

# Capturando um sinal

```
#include <stdio.h>
int b=0;
int main()
{
    int i = 3/b;
    printf("fim do programa.\n");
    return 0;
}
```

O que acontecia  
nesse exemplo ??

# Capturando um sinal

```
#include <stdio.h>
int b=0;
int main()
{
    int i = 3/b;
    printf("fim do programa.\n");
    return 0;
}
```

O que acontecia  
nesse exemplo ??

Exceção de ponto flutuante (imagem do núcleo gravada)

# Capturando um sinal

```
int b=0;
void trata_div_by_zero(int num){
    // num = numero do sinal
    printf("Divisao por zero!!.\n");
    printf("Vou esperar pra sair;\n");
    sleep(1);
    exit(0);
}
int main()
{
    struct sigaction sig_div_by_zero;
    sig_div_by_zero.sa_handler = trata_div_by_zero;
    sig_div_by_zero.sa_flags = 0;
    sigemptyset(&sig_div_by_zero.sa_mask);
    sigaction(SIGFPE,&sig_div_by_zero, NULL);
    int i = 3/b;
    printf("fim do programa.\n");
    return 0;
}
```



# Atividade prática

## Capturando sinais (20 minutos)

1. Chamada **sigaction** e seu uso para receber sinais

# Sinais – tentativa 1

```
volatile int count = 0;
void sig_handler(int num) {
    printf("Chamou Ctrl+C\n");
    count++;
}

int main() {
    struct sigaction s;
    s.sa_handler = sig_handler;
    sigemptyset(&s.sa_mask);
    s.sa_flags = 0;
    sigaction(SIGINT, &s, NULL);

    if(count >= 3 ) return 0;

    printf("Meu pid: %d\n", getpid());

    while(1){
        sleep(1);
    }
    return 0;
}
```

Será que funciona essa estratégia ?

# Sinais – tentativa 1

```
volatile int count = 0;
void sig_handler(int num) {
    printf("Chamou Ctrl+C\n");
    count++;
}


int main() {
    struct sigaction s;
    s.sa_handler = sig_handler;
    sigemptyset(&s.sa_mask);
    s.sa_flags = 0;
    sigaction(SIGINT, &s, NULL);

    if(count >= 3 ) return 0;

    printf("Meu pid: %d\n", getpid());

    while(1){
        sleep(1);
    }
    return 0;
}
```

E se o código já tiver passado deste ponto?





# Sinais – tentativa 2

```
volatile int flag = 0;
void sig_handler(int num) {
    printf("Chamou Ctrl+C\n");
    flag = 1;
}

int main() {
    int count = 0;
    struct sigaction s;
    s.sa_handler = sig_handler;
    sigemptyset(&s.sa_mask);
    s.sa_flags = 0;
    sigaction(SIGINT, &s, NULL);
    printf("Meu pid: %d\n", getpid());
    while(count<3){
        sleep(1);
        if(flag){
            count++;
            printf("Passei aqui.\n");
            flag = 0;
        }
    }
    return 0;
}
```

Será que funciona essa estratégia ?

# Sinais – tentativa 2

```
volatile int flag = 0;
void sig_handler(int num) {
    printf("Chamou Ctrl+C\n");
    flag = 1;
}

int main() {
    int count = 0;
    struct sigaction s;
    s.sa_handler = sig_handler;
    sigemptyset(&s.sa_mask);
    s.sa_flags = 0;
    sigaction(SIGINT, &s, NULL);
    printf("Meu pid: %d\n", getpid());
    while(count<3){
        sleep(1);
        if(flag){
            count++;
            printf("Passei aqui.\n");
            flag = 0;
        }
    }
    return 0;
}
```

Tenho que incluir essa  
checagem  
em várias partes do programa?

# Sinais – tentativa 2

```
volatile int flag = 0;
void sig_handler(int num) {
    printf("Chamou Ctrl+C\n");
    flag = 1;
}
```

```
int main() {
    int count = 0;
    struct sigaction s;
    s.sa_handler = sig_handler;
    sigemptyset(&s.sa_mask);
    s.sa_flags = 0;
    sigaction(SIGINT, &s, NULL);
    printf("Pressione Ctrl+C\n");
    while(1) {
        sleep(1);
        if(flag == 1) {
            flag = 0;
        }
    }
    return 0;
}
```

Tenho que incluir essa

**Erro conceitual: O programa principal espera informações vindas do handler.**

do programa?

**Correto: o handler deveria ser auto contido**



# Atividade prática

## Sinais e concorrência (20 minutos)

1. Chamada sigaction e seu uso para receber sinais
2. Sinais diferentes sendo capturados pelo mesmo processo

# Problemas de concorrência!

O que acontece se dois handlers tentam

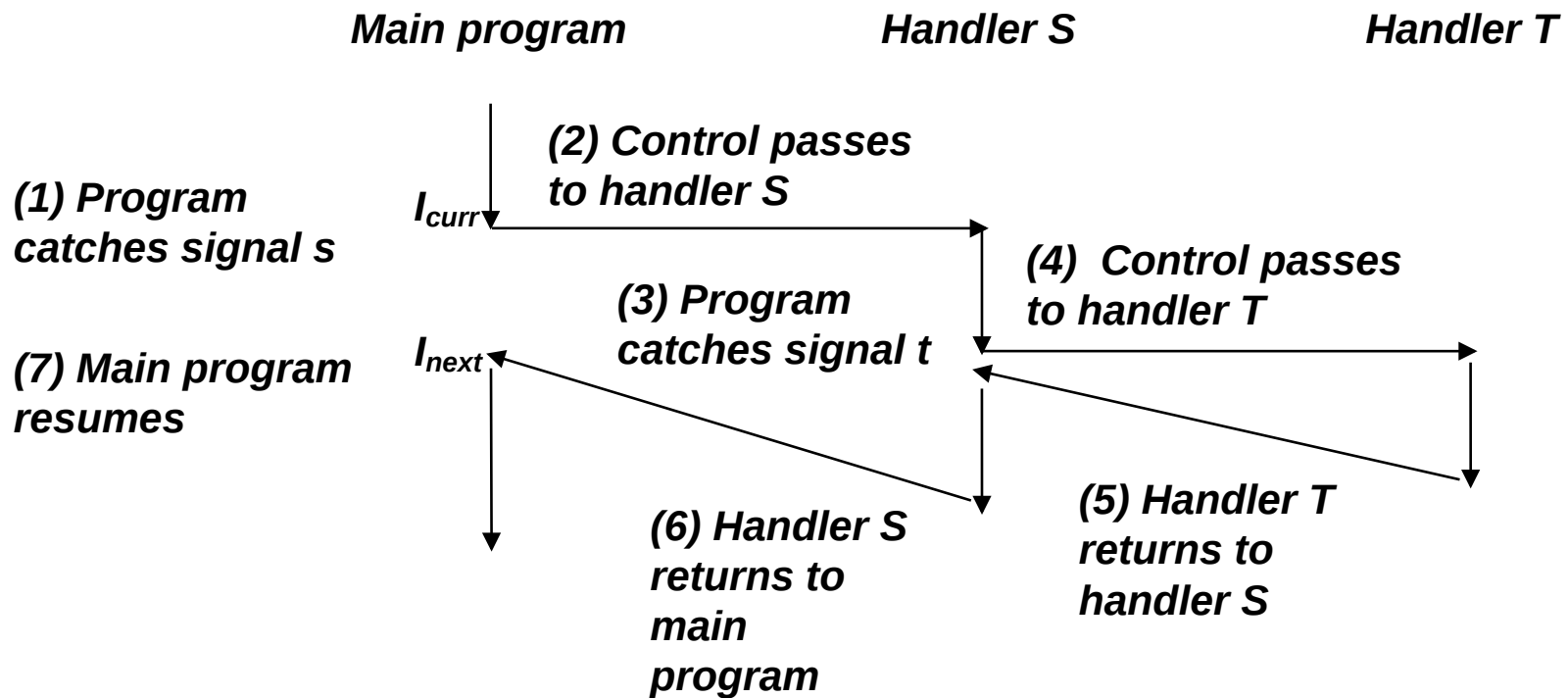
- mexer na mesma variável?
- chamar printf?
- usar a global errno?

Um handler que trata um sinal **A** só pode ser interrompido pela chegada de um outro sinal **B != A**.

Temos que ser cuidadosos ao tratar sinais!

# Handlers aninhados

Handlers podem ser interrompidos por outros handlers!



Mas não pode haver mais de um handler do mesmo sinal rodando!

# Bloqueio de sinais

Podemos "bloquear" o recebimento de um sinal:

- O sinal bloqueado fica pendente até que seja desbloqueado
- Quando for desbloqueado ele será recebido normalmente pelo processo!

**Bloquear um sinal é algo "temporário" e não implica na recepção do sinal**

# Recebendo um sinal

```
struct sigaction {  
    void (*sa_handler)(int);
```

- SIG\_IGN para ignorar
- SIG\_DFL para padrão
- Nome de uma função

```
        void (*sa_sigaction)(int,  
siginfo_t*, void *);
```

```
    sigset_t sa_mask;
```

Sinais a serem bloqueados durante a execução de sa\_handler

```
    int sa_flags;
```

Opções de recepção. Usaremos 0 sempre aqui.

```
        void (*sa_restorer)(void);
```

```
};
```





# Atividade prática

## **Bloqueando sinais (15 minutos)**

1. Sinais diferentes sendo capturados pelo mesmo processo
2. Bloqueando sinais durante a execução do handler

# Insper

[www.insper.edu.br](http://www.insper.edu.br)