

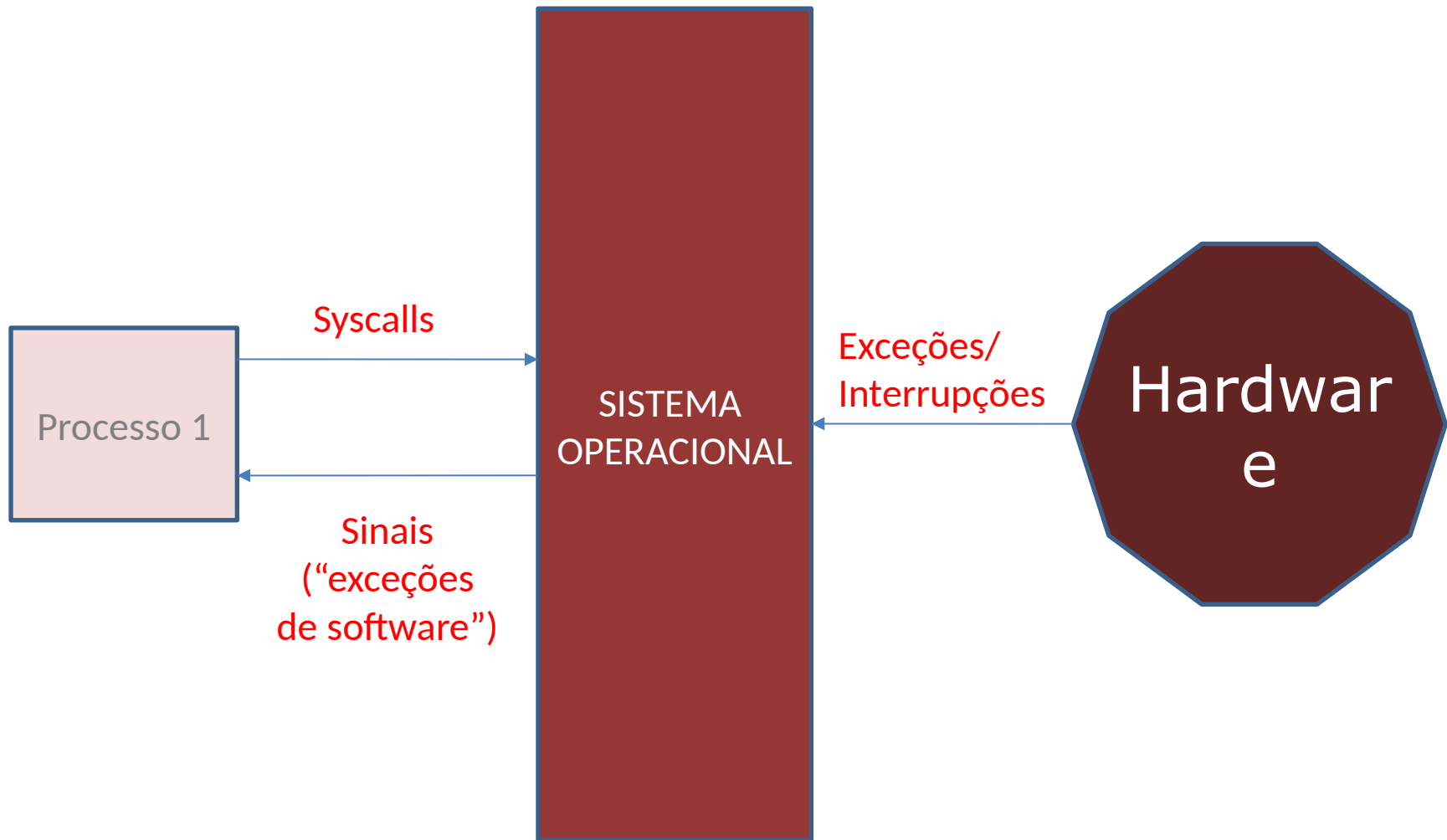
# **Sistemas Hardware-Software**

## Aula 21– Sinais POSIX 2

2017 – Engenharia

Igor Montagner, Fábio Ayres

# Interação do SO com seus processos



# Sinais POSIX

SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating-point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers; see pipe(7)
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at terminal
SIGTTIN	21,21,26	Stop	Terminal input for background process
SIGTTOU	22,22,27	Stop	Terminal output for background process

# Sinais

- Enviados por processos
- Eventos excepcionais externos
- Não carregam informação
- Comportamento padrão
  - Ignorar, Bloquear, Handler
- Uso opcional

# Interrupções (Embarcados)

- Conectados a periféricos
- Entrada de dados
- Handlers muito rápidos
- Parte do fluxo do programa
- Essenciais

# Sinais – versão Embarcados 1

```
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(1) {
        sleep(1);
        if (flag) {
            count++;
            flag = 0;
        }
    }
    return 0;
}
```

# Sinais – versão Embarcados 1

```
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(1) {
        sleep(1);
        if (flag) {
            count++;
            flag = 0;
        }
    }
    return 0;
}
```

Tenho que incluir essa checagem em várias partes do programa?

# Sinais – versão Embarcados 1

```
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);
```

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

```
    printf("Meu programa está rodando\n");
    while(1) {
        sleep(1);
        if (flag) {
            count++;
            flag = 0;
        }
    }
    return 0;
}
```

Correto: o handler deveria ser auto contido

# Sinais – versão Embarcados 2

```
volatile int count = 0;

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

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);

    if (count >= 3) return 0;

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

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



# Sinais – versão Embarcados 2

```
volatile int count = 0;

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

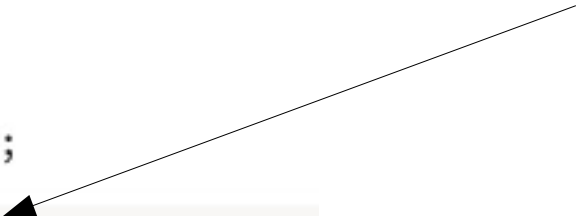
int main() {
    int count = 0;
    struct sigaction s;
    s.sa_handler = sig_handler;
    sigemptyset(&s.sa_mask);
    s.sa_flags = 0;
    sigaction(SIGINT, &s, NULL);

    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 – versão Embarcados 2

```
volatile int count = 0;

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

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);

    if (count == 0)
        printf("Mensagem enviada para o handler\n");

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

Erro conceitual: O programa principal tenta se sincronizar com o handler

Correto: o handler pode ocorrer a qualquer momento.

# Sinais – versão correta 0

```
int num_vezes = 0;
void sig_handler(int num) {
    printf("Chamou Ctrl+C: %d\n", num_vezes);
    if (num_vezes == 3) {
        exit(-1);
    }
    num_vezes++;
}

int main() {
    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(1) {
        sleep(1);
    }

    return 0;
}
```

# Sinais – versão correta 1

```
int num_vezes = 0;
void sig_handler(int num) {
    printf("Chamou Ctrl+C: %d\n", num_vezes);
    if (num_vezes == 1) {
        struct sigaction s;
        s.sa_handler = SIG_DFL;
        sigemptyset(&s.sa_mask);
        printf("Chamou sigaction! %d\n", num_vezes);
        sigaction(SIGINT, &s, NULL);
    }
    num_vezes++;
}

int main() {
    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(1) {
        sleep(1);
    }

    return 0;
}
```

# Hoje

- Revisão de sinais
- Bloqueio de sinais
- Atividade 6 – mini shell

# Enviando sinais

Chamada kill

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);
```

# Recebendo um sinal

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

- **Ignorar** o sinal (não faz nada)
- **Terminar** o processo (opcional: core dump)
- **Capturar** o sinal e executar, como usuário, um signal handler

# Recebendo um sinal

```
#include <signal.h>
```

```
int sigaction(int signum, const struct  
sigaction *act, struct sigaction *oldact);
```

```
...
```

If act is non-NULL, the new action for signal signum is installed from act. If oldact is non-NULL, the previous action is saved in oldact.



# 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;
```

→ Quais sinais serão bloqueados durante a execução do handler

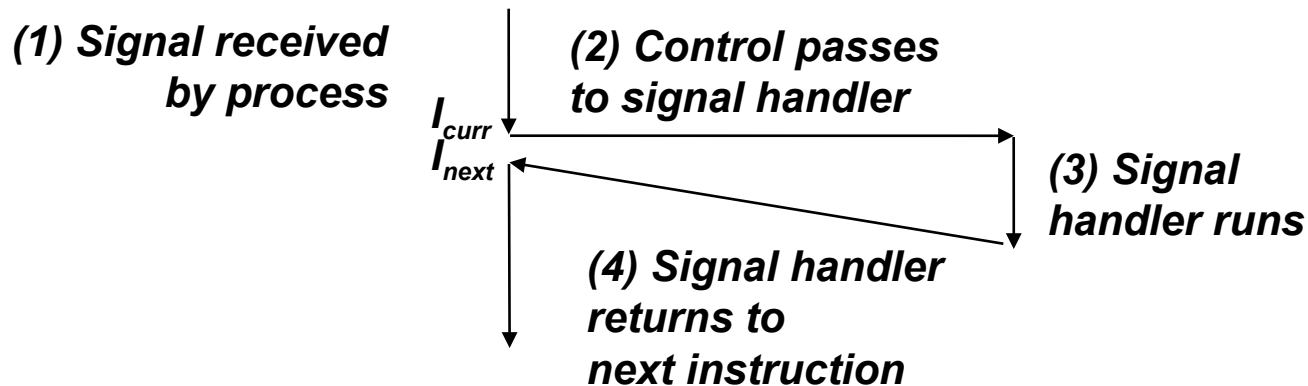
```
    int sa_flags;
```

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

```
};
```

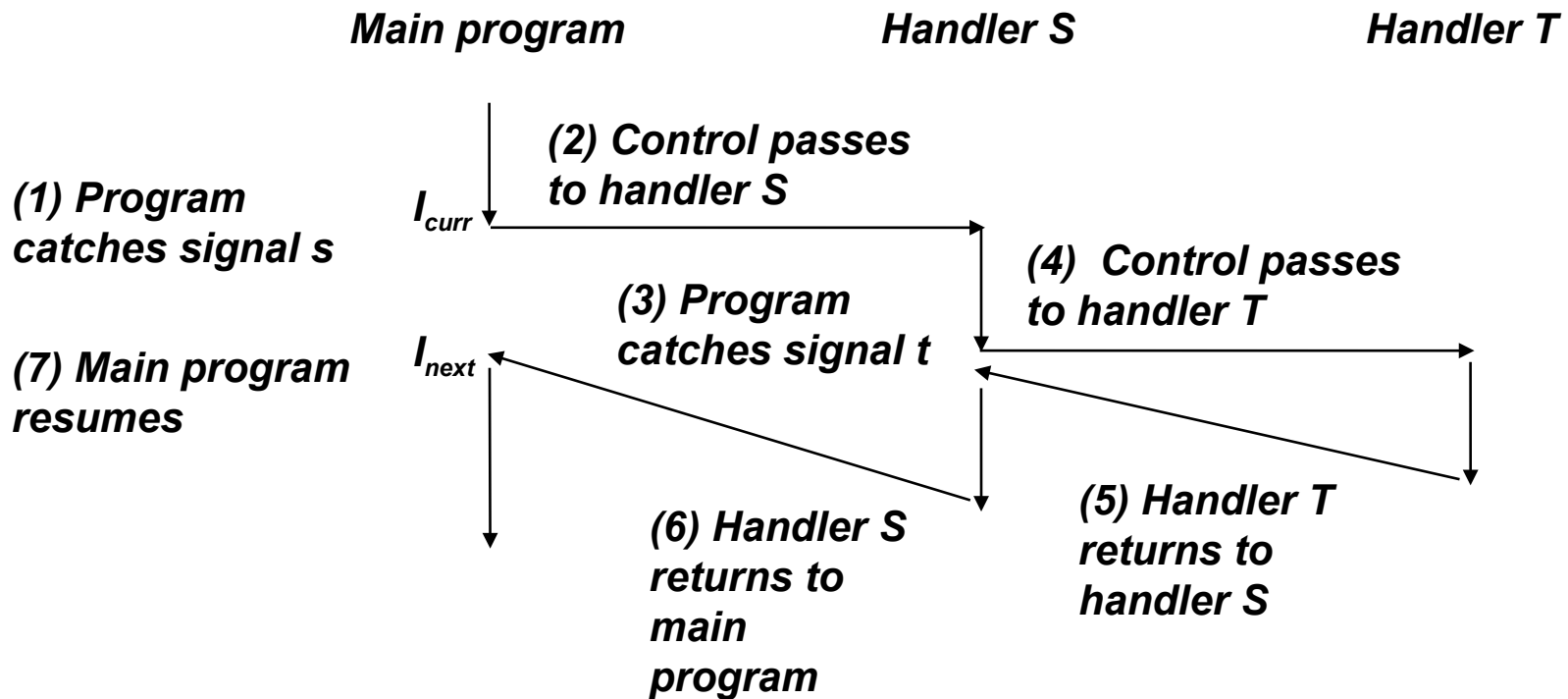
# Captura de sinais

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



# Handlers aninhados

Handlers podem ser interrompidos por outros handlers!



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

# 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!

# Bloqueio de sinais

A chamada `sigprocmask` pode bloquear o recebimento de alguns sinais

- O sinal bloqueado fica pendente até que seja desbloqueado
- Um sinal ignorado é recebido pelo processo e executa um handler vazio
- Usa um vetor de bits do tipo `sigset_t`

```
man sigprocmask  
man sigsetops
```

# Agora

- Handout de hoje: recebendo, ignorando e mascarando sinais
- Revisão de fork, wait e kill.

# Insper

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