

Sistemas Hardware-Software

Aula 17 - Sinais II: recebimento e concorrência

Engenharia

Fabio Lubacheski

Maciel C. Vidal

Igor Montagner

Fábio Ayres

Correção

Envio de sinais via terminal e programa (20 minutos)

1. Recuperando sinais com **wait** (parte1.c);
2. Enviando **kill -9 PID** pelo terminal (parte2.c);
3. Enviando **kill** pelo processo pai (parte3.c);e
4. Entendendo wait não bloqueante (parte4.c).

Recebendo um sinal

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

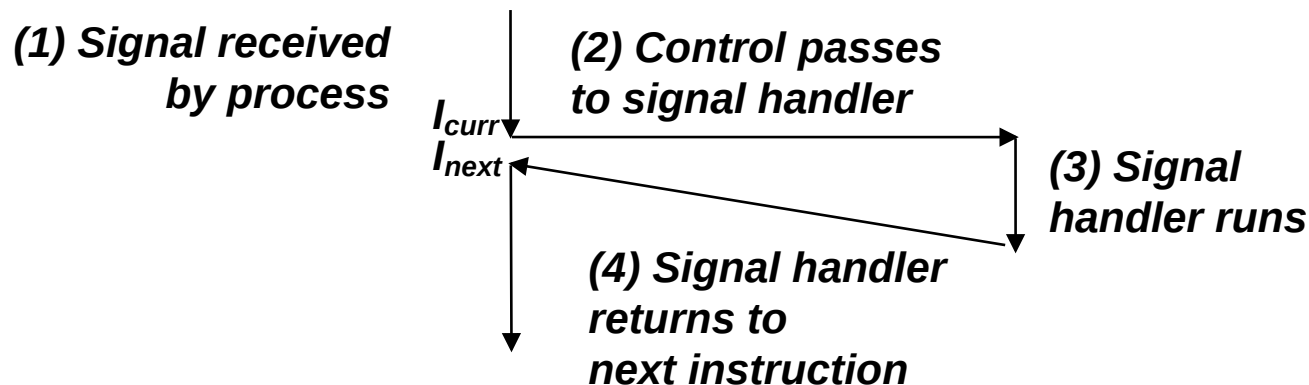
- **Capturar** o sinal, quando o processo recebe o sinal ele executa uma função definida no tratador do sinal definido pelo usuário (handler). Isso permite que você tome alguma ação, como salvar arquivos, limpar recursos,
- **Ignorar** o sinal (não faz nada), o programa **não executa o padrão do daquele sinal** (SIG_IGN).
- **Terminar**: o processo (SIGSEGV, SIGFPE, SIGKILL).

Ações permitidas para Sinais

Sinal	Código	Capt	Ign	Bloq	Descrição
SIGINT	2	✓	✓	✓	Interrupção (Ctrl+C)
SIGFPE	8	✓	✓	✓	Erro de ponto flutuante (divisão por zero...)
SIGKILL	9	✗	✗	✗	Terminação forçada, não pode ser capturado ou ignorado.
SIGSEGV	11	✓	✗	✓	Violação de segmento (acesso inválido)
SIGPIPE	13	✓	✓	✓	Pipe quebrado (escrever sem leitor)
SIGALRM	14	✓	✓	✓	Alarme de timer expirado.
SIGTERM	15	✓	✓	✓	Solicitação de término
SIGSTOP	19	✗	✗	✗	Para processo imediatamente, não pode ser capturado ou ignorado.
SIGTSTP	20	✓	✓	✓	Parada interativa (Ctrl+Z)

Capturar de sinal

Significa que, quando o processo recebe o sinal, ele executa uma função (**handler**) que você definiu.



Funções para tratar sinal:

signal(): mais simples, mas **menos robusta/portável**.

sigaction(): **mais poderosa, moderna e preferida**.

Programa terminando com sinal ex1_slide.c (aula passada)

```
#include <stdio.h>
int main() {
    int *px = (int*) 0x01010101;
    *px = 0;
    printf("fim do programa.\n");
    return 0;
}
```

Nesse exemplo o programa termina com sinal **SIGSEGV**

Como capturar o sinal ??

Falha de segmentação (imagem do núcleo gravada)

Por padrão, **SIGSEGV** encerra o processo imediatamente. Mas você pode **capturar** esse sinal e fazer algo antes que o processo termine — como salvar dados ou exibir uma mensagem.

Capturando e tratando sinal

```
void trata_falha_memoria(int num){
    printf("Falha no acesso a memoria!!.\n");
    printf("Vou esperar 1 segundo para sair;\n");
    sleep(1);
    exit(0); // o que acontece se tirar exit(0) ?
}

int main()
{ // Preenche estrutura para capturar SIGSEGV
    struct sigaction sa;
    sa.sa_handler = trata_falha_memoria;
    sa.sa_flags = 0;
    sigemptyset(&sa.sa_mask);
    sigaction(SIGSEGV, &sa, NULL);
    // Força falha de segmentacao
    int *px = (int*) 0x01010101;
    *px = 0;
    printf("fim do programa.\n");
    return 0;
}
```

Capturar um sinal – função **sigaction**

```
#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** (ex: **SIGSEGV**, **SIGINT**).

Argumento **act** é um ponteiro para uma struct sigaction que define o novo comportamento do sinal.

Argumento **oldact** guarda o comportamento anterior, se não quiser salvar passe **NULL**.

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

Capturar um sinal – **struct sigaction**

Principais campos da estrutura:

```
struct sigaction {  
    // Ponteiro para a função (handler) que irá tratar  
    // o sinal, o campo também pode receber SIG_IGN para  
    // ignorar SIG_DFL para comportamento padrão  
    void (*sa_handler)(int);  
    // Conjunto de sinais a bloquear temporariamente  
    // enquanto a função (handler) está rodando.  
    sigset_t    sa_mask;  
    // permite configurar o comportamento avançado do  
    // tratador de sinais, nos nossos exemplos será  
    // sempre sa_flags = 0;  
    int sa_flags;  
};
```

Capturar um sinal – função **sigemptyset**

A função **sigemptyset(&sa.sa_mask)** garante que nenhum outro sinal será **bloqueado** automaticamente durante o tratamento de **SIGSEGV**.

Se você quiser bloquear outros sinais temporariamente (por exemplo **SIGTERM**) durante a execução do **handler** (função), pode usar **sigaddset()** como abaixo:

```
....  
sigemptyset(&sa.sa_mask);  
// Adiciona SIGTERM à máscara  
sigaddset(&sa.sa_mask, SIGTERM);  
sigaction(SIGSEGV, &sa, NULL);  
....
```

Ignorando um sinal (ex2_ignora.c)

Para um processo ignorar um sinal (se possível) basta preencher o **handler** da estrutura com a constante **SIG_IGN**.

```
int main() {  
    struct sigaction sa;  
    // Zera a estrutura  
    memset(&sa, 0, sizeof(sa));  
    // Define a ação: ignorar o sinal  
    sa.sa_handler = SIG_IGN;  
    sigaction(SIGINT, &sa, NULL);  
}
```

Cuidado: **Ignorar** é diferente de **bloquear**, quando o sinal é **bloqueado** ele não é perdido, mas fica pendente enquanto esteve bloqueado

Atividade prática

Capturando sinais - a chamada sigaction¶ (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 para interromper o programa.\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

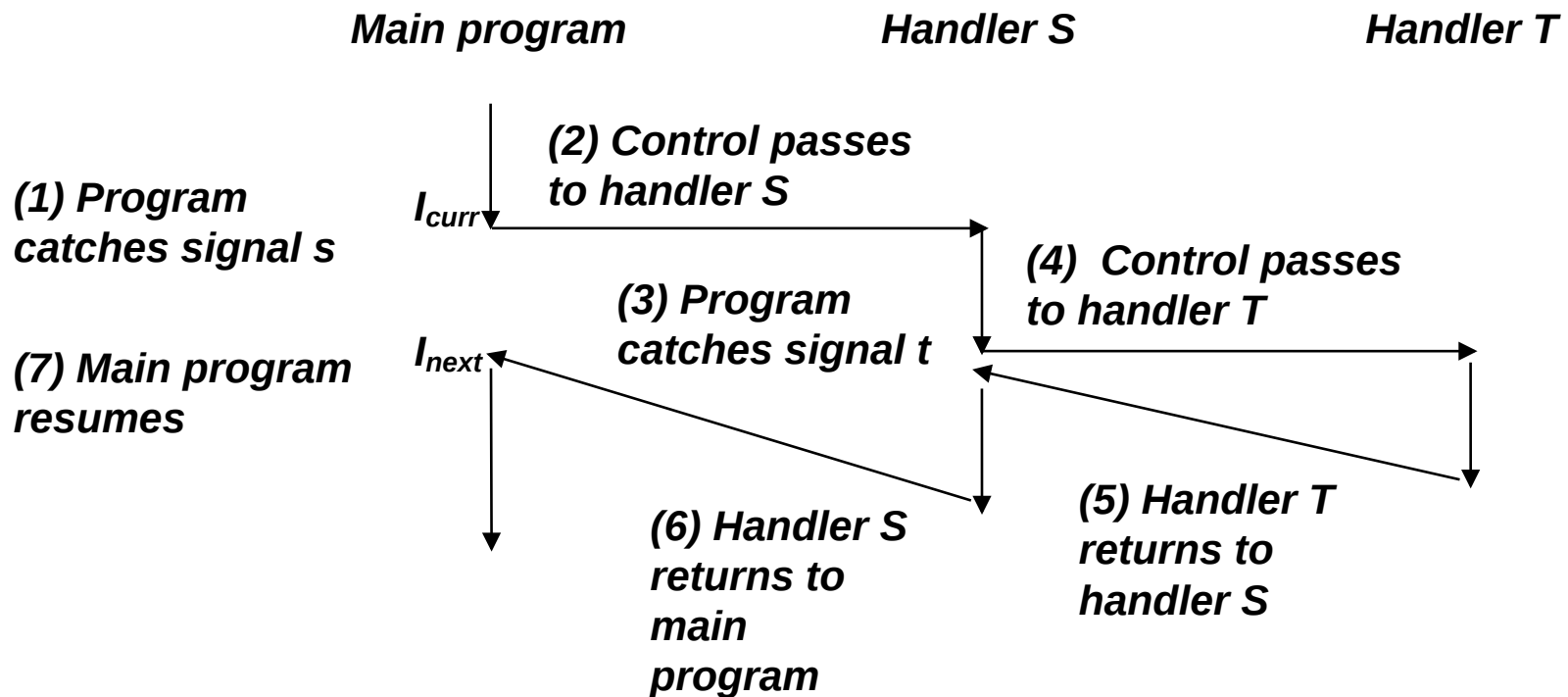
- mexer na mesma variável?
- chamar printf?
- usar a global errno?

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

- Para bloquear um sinal pesquise sobre a função `sigaddset(&sa.mask, sinal);`

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