

Mestrado Integrado em Engenharia Informática

# Guião 7

## Sinais

Sistemas Operativos 2018/2019

Universidade do Minho

# Sinais

- Um sinal é um evento assíncrono enviado a um processo.
- Por assíncrono, entende-se que o evento pode ocorrer a qualquer momento e de forma independente à execução do programa.
- Alguns sinais estão relacionados com exceções provocadas pelo hardware e operações de erro:
  - Violação da proteção no endereçamento (segmentation fault)
  - Erros de cálculo matemático, como divisão por zero
  - Instruções ilegais
  - Interação do utilizador: Ctrl-c, Ctrl-z, ...
- Podem ser utilizados para comunicação entre processos (IPC).

# Exemplos de sinais

Sinal	Causa/Descrição
<b>SIGINT</b>	Interrupção a partir do teclado (Ctrl+c)
<b>SIGQUIT</b>	Terminação a partir do teclado (Ctrl+\)
<b>SIGFPE</b>	Erro de cálculo matemático
<b>SIGKILL</b>	Forçar terminação
<b>SIGSEGV</b>	Acesso inválido (Segmentation fault)
<b>SIGPIPE</b>	Escrita em PIPE sem leitores
<b>SIGALRM</b>	Sinal de temporizador ( <i>alarm(2)</i> )
<b>SIGCHLD</b>	Processo filho parou ou terminou
<b>SIGCONT</b>	Continuar execução, se parado
<b>SIGUSR1</b>	<i>Sinal para uso personalizado</i>
<b>SIGUSR2</b>	<i>Sinal para uso personalizado</i>

# Exemplo - SIGCHLD

**PID 2034**  
(pai)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

**PID 2035**  
(filho)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

# Exemplo - SIGCHLD

PID 2034  
(pai)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

PID 2035  
(filho)

```
1 int main() {
2     pid_t pid;
3
4     _exit(0);
5
6     // código processo-filho
7     _exit(0);
8 } else {
9     // código processo-pai
10    pid_t child = wait(&status);
11    printf("filho saiu %d \
12          erro: %d\n", child, status);
13 }
14 }
```

**\_exit()** termina o processo atual com código passado por argumento.

# Exemplo - SIGCHLD

PID 2034  
(pai)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

PID 2035  
(filho)

```
1 int main() {
2     pid_t pid;
3     O processo filho termina, o SO
4     acorda o processo pai e
5     envia-lhe o sinal SIGCHLD.
6     // código processo-filho
7     _exit(0);
8 } else {
9     // código processo-pai
10    pid_t child = wait(&status);
11    printf("filho saiu %d \
12          erro: %d\n", child, status);
13 }
14 }
```

# Exemplo - SIGCHLD

**PID 2034**  
**(pai)**

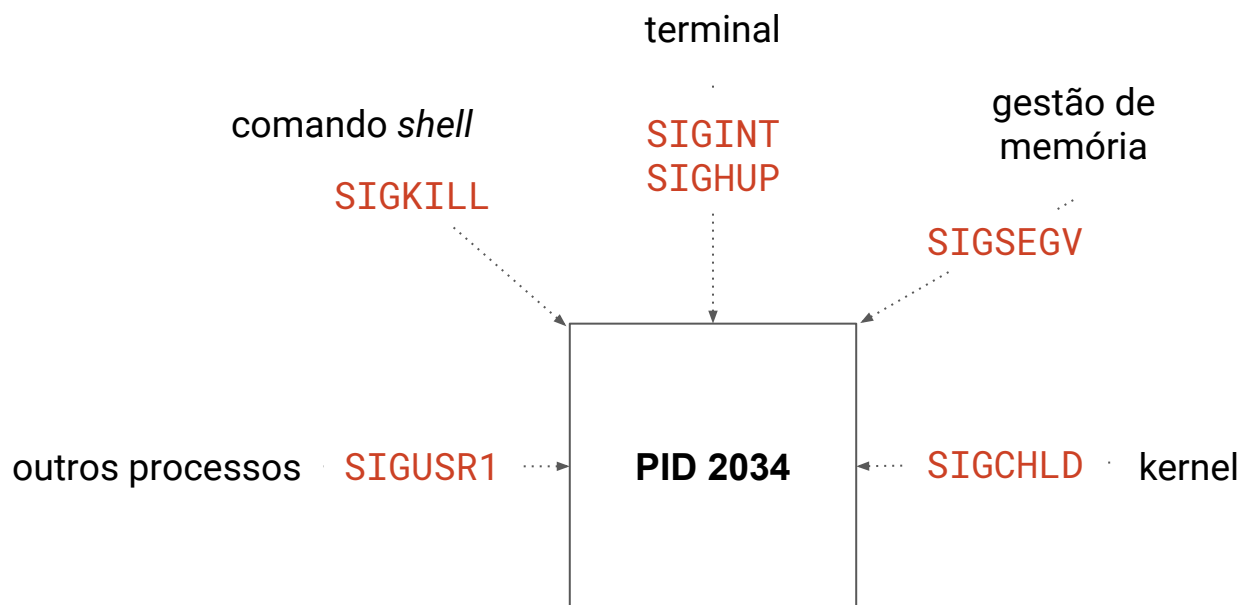
```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

O processo pai continua a execução após a terminação do processo-filho.

**PID 2035**  
**(filho)**

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

# Fontes de sinais





## Envio de sinais (via shell)

```
kill -s <SIGNAL> <pid>
```

Envia o sinal <SIGNAL> ao processo <pid>.

Se o sinal não for especificado, é enviado **SIGTERM**.

```
kill -SIGQUIT <pid>
```

```
kill -SIGKILL <pid>
```

# Envio de sinais (em C) - Chamadas ao sistema

```
int kill(pid_t pid, int sig);
```

Envia um sinal com o código **sig** para o processo **pid**.

Os sinais disponíveis encontram-se em “man 7 signal”.

Retorna o valor **0**, em caso de sucesso, e **-1** em caso de erro.

# Tratamento de sinais

Um processo pode tomar uma das seguintes decisões, ao receber sinais:

- Desempenhar o comportamento definido pelo sistema operativo.
  - Terminar o processo é o comportamento mais comum.
  - “**man 7 signal**” (MacOS) ou “**man signal**” (Linux)
- Ignorar o sinal.
- Especificar uma rotina de tratamento do sinal.

# Tratamento de sinais - Rotina de tratamento

- É possível especificar uma função que executa quando um sinal é recebido, por exemplo:
  - SIGUSR1: Iniciar uma cópia de segurança.
  - SIGUSR2: Imprimir o estado do programa.
- A execução do processo é imediatamente interrompida e o sinal é tratado. Após o tratamento do sinal, a execução retoma no ponto deixado anteriormente.
- **Não é possível tratar os sinais SIGSTOP e SIGKILL.**

# Tratamento de sinais - Chamadas ao sistema

```
typedef void (*sighandler_t)(int);  
sighandler_t signal(int signum, sighandler_t handler);
```

Associa a rotina de tratamento **handler** ao sinal **signum**.

Retorna o valor **SIG\_ERR**, em caso de insucesso e o valor associado ao signal **handler** anterior em caso de sucesso.

```
1 void print_status(int signum) {  
2     ...  
3 }  
4  
5 int main(int argc, char * argv[]) {  
6     signal(SIGCONT, print_status);  
7 }
```

# Tratamento de sinais - Exemplo

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         perror("SIGINT failed");
11     }
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

# Tratamento de sinais - Exemplo

```
1 int Define-se a rotina de tratamento.  
2  
3 void ctrl_c_handler(int signum) {  
4     printf("CTRL+C\n");  
5     ctrl_c_counter++;  
6 }  
7  
8 int main(int argc, char * argv[]) {  
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {  
10         perror("SIGINT failed");  
11     }  
12  
13     while(ctrl_c_counter < 3) {  
14         printf("working...\n");  
15         sleep(1);  
16     }  
17  
18     return 0;  
19 }
```

# Tratamento de sinais - Exemplo

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         perror("SIGINT failed");
11     }
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

Regista a rotina de tratamento para o sinal do tipo **SIGINT**.



# Tratamento de sinais - Exemplo

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         printf("Error\n");
11     }
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

O programa continuará a executar até receber três vezes o sinal **SIGINT** (Ctrl+C).

Em tratamento

--

Fila de sinais


Sinais enviados


# Tratamento de sinais - Exemplo

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         perror("SIGINT failed");
11     }
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

**Em tratamento**

--

**Fila de sinais**


**Sinais enviados**

SIGINT

# Tratamento de sinais - Exemplo

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         perror("SIGINT failed");
11     }
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

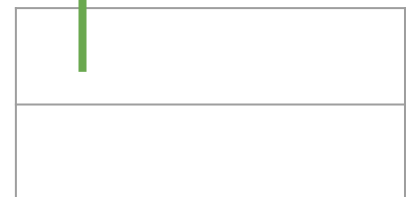
**Em tratamento**



**Fila de sinais**



**Sinais enviados**



# Tratamento de sinais - Exemplo

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         perror("SIGINT failed");
11
12         A chamada sleep(...) é
13         interrompida quando um sinal
14         é recebido.
15     }
16     printf("working...\n");
17     sleep(1);
18 }
19
20 return 0;
21 }
```

**Em tratamento**

--

**Fila de sinais**

SIGINT

**Sinais enviados**


# Tratamento de sinais - Exemplo

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         perror("SIGINT failed");
11     }
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

## Em tratamento

Por processo, apenas existe um sinal de cada tipo na fila de sinais, por processo.

## Fila de sinais

SIGINT

## Sinais enviados

SIGINT

# Tratamento de sinais - Exemplo

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         perror("SIGINT failed");
11     }
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

## Em tratamento

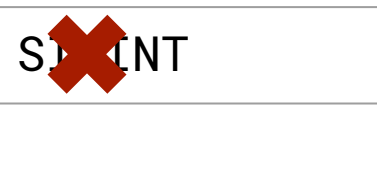


## Fila de sinais

SIGINT

Os sinais do mesmo tipo são descartados, enquanto existir um sinal na fila de sinais.

## Sinais enviados



# Tratamento de sinais - Exemplo

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         perror("SIGINT failed");
11     }
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

## Em tratamento

--

## Fila de sinais

SIGINT

## Sinais enviados


# Tratamento de sinais - Exemplo

A execução do programa é interrompida e a execução da rotina de tratamento do sinal é iniciada.

```
1
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         perror("SIGINT failed");
11     }
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

**Em tratamento**

SIGINT

**Fila de sinais**

**Sinais enviados**



# Tratamento de sinais - Exemplo

Incrementa o contador global de Ctrl+C.

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int num) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 int main(int argc, char * argv[]) {
9     if(signal(SIGINT, ctrl_c_handler) == SIG_ERR) {
10         perror("SIGINT failed");
11     }
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

**Em tratamento**

SIGINT


**Fila de sinais**


**Sinais enviados**


# Tratamento de sinais - Exemplo

```
1 int ctrl_c_counter = 0;
2
3 void ctrl_c_handler(int signum) {
4     printf("CTRL+C\n");
5     ctrl_c_counter++;
6 }
7
8 {
9     ler) == SIG_ERR) {
10
11
12
13     while(ctrl_c_counter < 3) {
14         printf("working...\n");
15         sleep(1);
16     }
17
18     return 0;
19 }
```

O programa retoma a execução.  
Como o sleep(...) foi interrompido, prossegue para a próxima iteração.



**Em tratamento**

--

**Fila de sinais**


**Sinais enviados**


# Notas

- As funções `sleep(...)` e `pause()` são interrompidas e não são retomadas após um sinal.
- As chamadas bloqueantes que temos vindo a utilizar são retomadas após o tratamento sinal.
  - `read(...)`, `write(...)`, `open(...)`, etc.
- Tal como os descritores de ficheiros, a associação de rotinas de tratamento de sinais é também herdada por processos filho.
- O tratamento de sinais por omissão varia de acordo com a distribuição.

# Notas

```
int pause();
```

Força o processo a entrar em modo de pausa até que um sinal seja recebido (p.ex., via a função **kill()** ou via um **alarme** interno).

Retorna sempre **-1**.

# Notas

```
unsigned int alarm(unsigned int seconds);
```

Envia um sinal (SIGALRM) após o número de segundos especificados como argumento.

Retorna o tempo que ainda faltava esperar caso a função seja interrompida antes do valor especificado.

Se um alarme já estiver registrado, uma outra chamada à função irá reiniciar o temporizador.

# Sumário

## **kill**

- Envia um sinal a um determinado processo.

## **signal**

- Associa uma rotina de tratamento a um tipo de sinal.
- Usar **SIG\_DFL** para restaurar o comportamento por omissão.
- Usar **SIG\_IGN** para ignorar um sinal.

# Material de apoio

- José Alves Marques, Paulo Ferreira, Carlos Nuno da Cruz Ribeiro, Luís Veiga e Rodrigo Rodrigues, Sistemas Operativos., Oct 2012, FCA, ISBN 978-972-722-756-3.
- <https://pt.slideshare.net/tusharkute/signal-handling-in-linux>