# Sinais

## Primeira aula

José Pedro Oliveira
(jpo@di.uminho.pt)

Grupo de Sistemas Distribuídos
Departamento de Informática
Escola de Engenharia
Universidade do Minho

Sistemas Operativos I
2006-2007





## Conteúdo







## Introdução

### Sinais

Sinais são parte integrante de um sistema UNIX/POSIX. Sinais são utilizados para diversos fins, tais como:

- tratamento de excepções (divisão por zero, etc)
- notificação de ocorrência de eventos assíncronos (disparo de um timer, finalização de uma tarefa de E/S, etc.)
- finalização de processos sob circunstâncias anormais
- emulação de multitasking
- comunicação entre processos





## Introdução

### Conceitos

Um sinal POSIX é o equivalente software de uma interrupção ou da ocorrência de uma excepção. A recepção de um sinal notifica o processo de que algo importante aconteceu e de que é necessária a sua atenção.

### Conceitos

- cada sinal têm um nome
- os nomes dos sinais começam pelos caracteres SIG
- estes nomes são todos definidos por constantes inteiras positivas (número do sinal)
- nenhum sinal têm o número 0





## Acções associadas a um sinal

### Sumário

- Um sinal ocorre em momentos aparentemente aleatórios para o processo
- Um processo não pode simplesmente testar o valor de uma variável para determinar se um sinal ocorreu; em vez disso, o processo deve especificar ao kernel o que fazer quando um sinal ocorrer.

### Acções associadas a um sinal

1. ignorar o sinal (`SIG_IGN`)
2. apanhar o sinal (*catch the signal*) através da execução de uma rotina de atendimento
3. acção por omissão (`SIG_DFL`); a acção por omissão para a grande maioria dos sinais é terminar o processo





## Tipos de sinais (1/4)

### Detecção de erros

| | |
|---|---|
| SIGBUS | - access to undefined portion of a memory object (A) |
| SIGFPE | - erroneous arithmetic operation (A) |
| SIGILL | - illegal instruction (A) |
| SIGPIPE | - write on a pipe with no reader (T) |
| SIGSEGV | - invalid memory reference (A) |
| SIGSYS | - bad system call (A) |
| SIGXCPU | - CPU-time limit exceeded (A) |
| SIGXFSZ | - file-size limit exceeded (A) |





## Tipos de sinais (2/4)

### Gerados por aplicações/utilizadores

| | |
|---|---|
| SIGABRT | - call to `abort` (A) |
| SIGHUP | - hangup (T) |
| SIGINT | - interrupt (from keyboard) (T) |
| SIGKILL | - kill; synthetic only (T) |
| SIGQUIT | - quit (from keyboard) (A) |
| SIGTERM | - termination; synthetic only (T) |
| SIGUSR1 | - user signal 1; synthetic only (T) |
| SIGUSR2 | - user signal 2; synthetic only (T) |





## Tipos de sinais (3/4)

### Controlo de processos

| | |
|---|---|
| SIGCHLD | - child process terminated or expired (I) |
| SIGCONT | - continue executing (from keyboard) (C) |
| SIGSTOP | - stop executing; synthetic only (S) |
| SIGTSTP | - terminal stop executing (from keyboard) (S) |
| SIGTTIN | - background process attempting read (S) |
| SIGTTOU | - background process attempting write (S) |

## Tipos de sinais (4/4)

### Timer

SIGALRM - alarm clock expired (T)
SIGVYALRM - virtual timer expired (T)
SIGPROF - profiling timer expired (T)

### Miscellaneous

SIGPOLL - pollable event (T)
SIGTRAP - trace/breakpoint trap (A)
SIGURG - out-of-band data available at socket (I)





## Sinais: acções por omissão

### Acções por omissão

**I** - o sinal é ignorado
**T** - o processo é terminado
**A** - o mesmo que **T** mas com acções adicionais tais como a geração de ficheiros core
**S** - stop
**C** - continuar depois de um stop

### Permitir a geração de ficheiros **core**

```
$ ulimit -c unlimited
```





## Linux: kernel 2.4 e acções por omissão

### Linux: sinais disponíveis e acções por omissão

```
$ man 7 signal
```

### Exercício

Descarregue o último kernel versão 2.4 (linux-2.4.33.3.tar.bz2) e faça o levantamento das acções por omissão dos sinais através da análise dos ficheiros `signal.c`.

- `linux-2.4.33.3/kernel/signal.c`
- `linux-2.4.33.3/arch/i386/kernel/signal.c`





## Comando **kill**

### Comando **kill**

O comando kill permite enviar um sinal para um processo ou um grupo de processos. Se nenhum sinal for especificado, o sinal **TERM** é enviado por omissão. O sinal **TERM** mata todos os processos que não o interceptarem. Nalguns casos pode ser necessário enviar o sinal **KILL** (9), dado que este sinal não pode ser interceptado.

### Synopsis

```
kill [opções] [pid] ...
```





## Comando **kill**: listar sinais

### $ kill -l

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1
36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4  39) SIGRTMIN+5
...
60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2  63) SIGRTMAX-1
64) SIGRTMAX
```





## Comando **kill**: sinais

### Alguns sinais

SIGHUP (1) - Hangup.
SIGINT (2) - Interrupt.
(gerado pela sequência de teclas CTRL+C)
SIGQUIT (3) - Quit.
(gerado pela sequência de teclas CTRL+\)
SIGKILL (9) - Kill, unblockable.
Este sinal não pode ser interceptado pelo processo.
SIGTERM (15) - Termination.
Sinal enviado por omissão pelo comando **kill**.





## Comando **kill**: exemplos

### $ kill pid

Envia o sinal **TERM** para o processo *pid*. O processo ao receber este sinal deverá terminar a sua execução.

### $ kill -9 pid

O sistema operativo termina o processo *pid*, não lhe dando qualquer hipótese de executar código de limpeza.

### $ kill -INT pid

O sinal **INT** é enviado para o processo *pid*.





## Conteúdo



## Chamada ao sistema: **signal**

### Sumário

Permite instalar uma nova rotina de atendimento do sinal `signum`.

### Synopsis

```
#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

### Handlers pré-definidos

- SIG_IGN - ignorar o sinal
- SIG_DFL - acção por omissão





## signal - registar rotina de atendimento

### Exemplo 1 - signal.c

```
1  /* Incluir: stdio.h, stdlib.h, signal.h */
2
3  void rotina_atendimento( int sig )
4  {
5      ...
6  }
7
8  int main(void)
9  {
10     ...
11     if (signal(SIGINT, rotina_atendimento) == SIG_ERR) {
12         perror("signal\n");  exit(SIGINT);
13     }
14     ...
15 }
```





## signal - ignorar sinal

### Exemplo 2 - ignorar.c

```
1  /* Incluir: stdio.h, stdlib.h, unistd.h, signal.h */
2
3  int main(void)
4  {
5      int i = 0;
6      if (signal(SIGINT, SIG_IGN) == SIG_ERR) {
7          perror("signal");  exit(SIGINT);
8      }
9
10     printf("CTRL+C (SIGINT) ou CTRL+\\ (SIGQUIT)\n");
11
12     while (1) {
13         printf("%d\n", i++);  sleep(3);
14     }
15     return 0;
16 }
```





## signal - mesma rotina de atendimento para vários sinais

### Exemplo 3 - mesma rotina de atendimento

```
1  void rotina_atendimento(int sinal)
2  {
3      switch(sinal) {
4          case SIGINT:
5              ...
6              break;
7          case SIGUSR1:
8              ...
9              break;
10     }
11 }
12
13 int main(void)
14 {
15     signal(SIGINT, rotina_atendimento);
16     signal(SIGUSR1, rotina_atendimento);
17     ...
18     return 0;
19 }
```





## Chamada ao sistema: **kill**

### Sumário

Permite enviar um sinal para um processo ou para um grupo de processos.

### Synopsis

```
#include <sys/types.h>
#include <unistd.h>

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

### Permissões

O envio de sinais é permitido se:

- a conta do processo emissor for privilegiada
- a conta do processo emissor for igual à conta do processo alvo





## Chamada ao sistema: **kill**

### Parâmetro **pid**

- $> 0$ - o sinal **sig** é enviado ao processo **pid**
- $== 0$ - o sinal **sig** é enviado a todos os processos do grupo ao qual pertence o processo corrente
- $== -1$ - o sinal **sig** é enviado a todos os processos para os quais o processo corrente tem permissões de o fazer (com a excepção do processo 1)
- $< -1$ - o sinal **sig** é enviado a todos os processos do grupo **-pid**

### Parâmetro **sig**

- $== 0$ - o sinal não é enviado mas o tratamento de erros é efectuado





## Função: **raise**

### Sumário

Envia um sinal para o processo corrente.

### Synopsis

```
#include <signal.h>

int raise(int sig);
```

### Nota

A função `raise(sig)` é equivalente a:

```
kill(getpid(), sig);
```





## Chamada ao sistema: **pause**

### Sumário

Coloca o processo corrente a dormir até este receber um sinal que o termine ou cause a invocação de uma rotina de atendimento.

### Synopsis

```
#include <unistd.h>

int pause(void);
```

## Chamada ao sistema: **alarm**

### Sumário

Programa a entrega de um sinal **SIGALRM** ao processo corrente ao fim de *n* segundos.

### Synopsis

```
#include <unistd.h>

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

### Notas

- A invocação de **alarm** devolve o número de segundos que ainda faltariam para completar o alarme pendente (zero se não houver nenhum alarme pendente).
- Se o argumento for zero, o alarme alarme pendente é desactivado (caso aplicável).





## Alarme periódico

### Exemplo - alarme_periodico.c

```
1  /* Incluir: stdio.h, stdlib.h, unistd.h, signal.h */
2
3  void rotina(int signo)
4  {
5      alarm(1);       /* printf("1 segundo\n"); */
6  }
7
8  int main(void)
9  {
10     signal(SIGALRM, rotina);
11
12     alarm(1);
13     while (1) {
14         pause();
15     }
16     return 0;
17 }
```





## Função: **sleep**

### Sumário

Coloca o processo corrente a dormir até decorrerem os *n* segundos ou até o processo receber um sinal que não seja ignorado.

### Synopsis

```
#include <unistd.h>

unsigned int sleep(unsigned int seconds);
```





## Exercício

### Enunciado

Pretende-se monitorizar (todos os minutos) o custo de comunicação em função de um tarifário apresentado em baixo. O estabelecimento e o fim da ligação são assinalados através de `SIGUSR1` e `SIGUSR2` respectivamente. Apresente o programa **monitor** recorrendo às primitivas de sinais do UNIX SYSTEM V.

| Duração | Custo |
|---|---|
| $t < 5min.$ | 5 cent./min. |
| $t \geq 5min.$ | 4 cent./min. |





## Conteúdo







## Conteúdo







## Exercícios

### Mini-interpretador de comandos

- Impedir que o interpretador termine ao receber os sinais **SIGINT**, **SIGUSR1** e **SIGUSR2**.
- Permitir que o interpretador termine correctamente ao receber o sinal **SIGTERM**.
- Criar um sistema de alerta de eventos que permita ter mais do que um pedido pendente:
  - alarm list - listar alertas pendentes
  - alarm add nminutos mensagem - adicionar novo evento





## Referências

### Bibliografia


- **Advanced Programming in the UNIX Environment, 2nd ed.**
  W. Richard Steven, Stephen A. Rago
  - Capítulo 10 - Signals
- **Advanced UNIX Programming, 2nd ed.**
  Marc J. Rochkind
  - Capítulo 9 - Signals and Timers
- **Linux Programming by Example: The Fundamentals**
  Arnold Robbins
  - Capítulo 10 - Signals
- **The Design of the Unix Operating System**
  Maurice J. Bach
  - Capítulo 7 - Secção 7.2 - Signals