

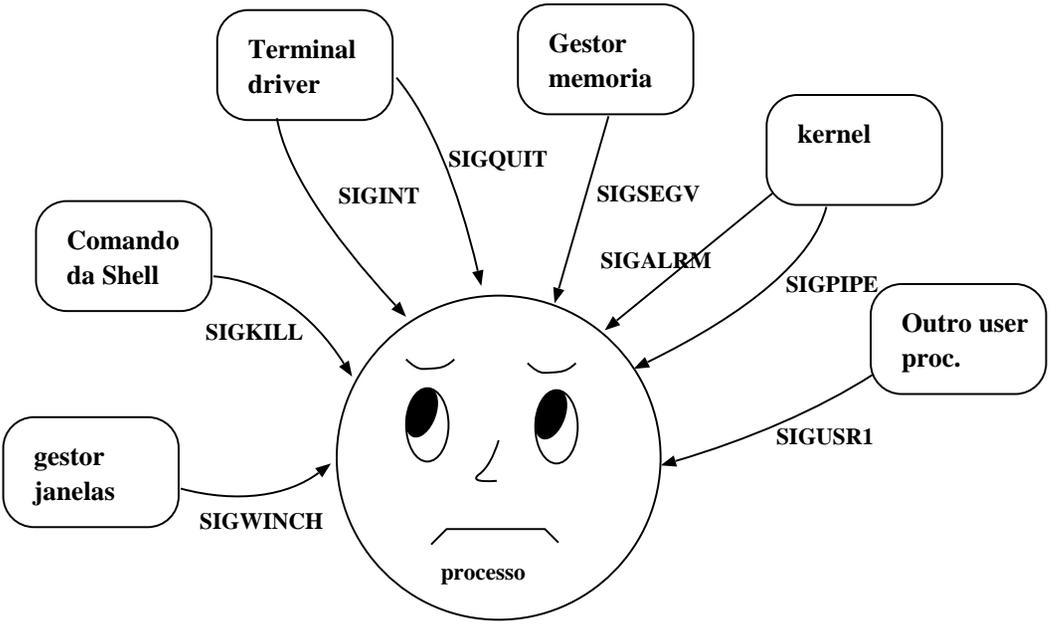
Sinais: eventos assíncronos

- Um sinal é um evento assíncrono que pode ser enviado a um processo, avisando-o de que algo de “inesperado” ou “anormal” aconteceu.
- *Evento Assíncrono* significa que pode ocorrer a qualquer momento.
- Tipos de sinais: (vê `/usr/include/signal.h`)

Nome	Ação Default	Descrição/Causa
SIGINT	Termina proc.	Interrupção gerada pelo teclado
SIGQUIT	Imagem core	Abortar a execução (por teclado)
SIGKILL	Termina proc.	Resultado de um <code>kill -9</code>
SIGSEGV	Imagem core	Ref. inválida à memória
SIGPIPE	Termina proc.	Escrita na pipe sem ninguém para ler
SIGALRM	Termina proc.	Sinal gerado por um despertador
SIGFPE	Imagem core	Excepção floating-point (divisão por zero)
SIGUSR1	Termina proc.	Sinal definido pelo utilizador

Sinais: eventos assíncronos

- De onde vêm os sinais?



Como é que um processo responde a um sinal?

Um processo pode escolher como responder à ocorrência de um sinal. Pode:

- *ignorar o sinal*: e.g. um processo pode proteger-se e ignorar sinais de interrupção.
- *apanhar o sinal*, executar uma função (signal-handler) em modo kernel e depois continuar a sua execução.
- aceitar a acção *default* do sinal, que na maioria dos casos termina o processo.

● Envio explícito de um sinal

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

envia o sinal definido por `sig` ao processo identificado por `pid`.

Um processo só pode enviar sinais aos processos com o mesmo UID.

Sinais: eventos assíncronos

Apanhar ou ignorar sinais

```
void *signal(int signum, void *handler())
```

onde `signum` define uma acção ou um sinal que quando ocorrer é apanhado e tratado pela função `handler()`. A função retorna o endereço da função que estava activa para o sinal indicado.

- A função `handler()` pode ser definida pelo utilizador ou uma das seguintes: `SIG_IGN` – para ignorar o sinal; ou `SIG_DFL` – para repôr o default para o sinal.

Outras funções ligadas a sinais

- Um processo pode requerer ao SO para que seja gerado um *timeout* enquanto faz outras tarefas. I.e. pede que lhe seja enviado um SIGALRM ao fim de um certo tempo (`nsecs` segundos):

```
int alarm(unsigned int nsecs);
```

- Um processo pode voluntariamente adormecer durante um certo tempo:

```
int sleep(unsigned int nsecs);
```

- Um processo também pode suspender voluntariamente a execução até receber um sinal:

```
int pause();
```

Sinais: eventos assíncronos

O que fazer com sinais?

- Ignorar a acção de um dado sinal (ver os exemplos).
- **Reconfiguração dinâmica.** Um programa que leia dados de configuração de um ficheiro, normalmente fá-lo no início da execução, e caso ocorra alteração no ficheiro de configuração é necessário recomeçar o programa. Com um handler especial isso não seria necessário.

```
void le_fich_configuracao(int sig) {
    int fd=open("`myconfig'", O_RDONLY);
    /* ... leitura ... */
    close(fd);
    signal(SIGHUP, le_fich_configuracao);
}
main() {
    le_fich_configuracao();
    while (1) { ... }
}
```

Exemplo: apanhar o sinal SIGINT.

O que fazer com sinais? (cont.)

- **Libertar memória antes de terminar.** Ao apanhar o SIGINT chama uma função que liberta segmentos de memória partilhada, ou remove ficheiros temporários.
- **Ligar e desligar debugging.**

Exemplo: apanhar o sinal SIGINT.

- Programa que se protege contra o sinal SIGINT, ie contra o sinal control-c.

```
#include <signal.h>

main()
{
    int (*f0)();

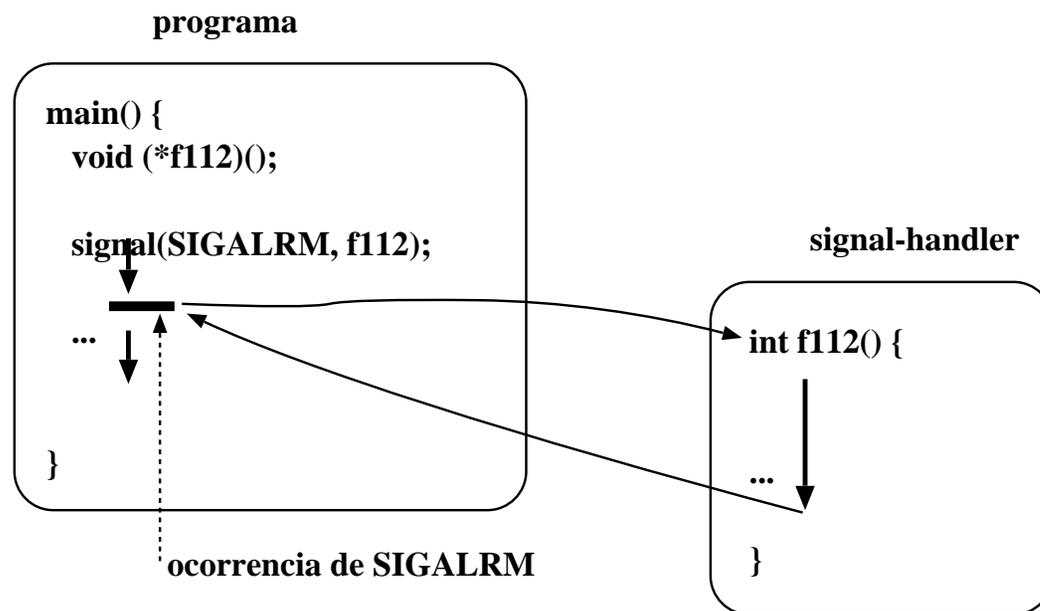
    printf(``Control-c activo\n'');
    sleep(3);
    f0= signal(SIGINT, SIG_IGN);
    printf(``Control-c inactivo\n'');
    sleep(3);
    signal(SIGINT, f0);
    printf(``Control-c activo\n'');
    sleep(3);
    printf(``Adeusinho!\n'');
}
```

Exemplo: apanhar o sinal SIGALRM.

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

int flag=0;
void f112();

main()
{
    signal(SIGALRM, f112);
    alarm(5);
    printf(``Esperando...\n'');
    while (!flag)
        pause();
    printf(``terminei.\n'');
}
void f112()
{
    printf(``Recebi sinal de alarme\n'');
    flag=1;
}
```



Mais um exemplo de sincronização por sinais

Escrever um programa que crie um processo filho e ambos, pai e filho, devem imprimir alternadamente linhas, de modo a obter uma sequência alternada:

Filho: 1

Pai: 2

Filho: 3

...

Pai: 20

Mais um exemplo de sincronização por sinais: Solução 1

```
void my_handler2() { signal(SIGUSR1, my_handler2);}
void sinc_sinais() {
    pid_t new_pid;
    int n=0;

    signal(SIGUSR1, my_handler2);
    if ((new_pid = fork())==0) { // filho
        new_pid = getppid();
        n = 1;
        while (n <= 20) {
            sleep(1);
            printf(``Filho: %d\n'', n);
            n += 2;
            kill(new_pid, SIGUSR1);
            pause();
        }
    } else // pai
        while (n <= 20) {
            pause();
            n += 2;
            printf(``Pai: %d\n'', n);
            kill(new_pid, SIGUSR1);
        }
}
```

PAI

n = 0

signal

fork

pause

n = 2

printf 2

kill (envia sinal ao filho)

pause

.....

FILHO

herda n = 0 do pai

herda a tab de sinais

new_pid = pid do pai

n = 1

sleep

printf 1

n = 3

kill (envia sinal ao pai)

pause

sleep

printf 3

n = 5

.....

Mais um exemplo de sincronização por sinais: Solução 2

```
void sinc_sinais(void) {
    int newpid, n2;
    n2=ctr=0;
    signal(SIGUSR1, my_handle
    if ((newpid=fork())==0) {
        ctr=1;
        while (ctr < LIMITE) {
            printf("Filho %d\n", ctr);
            n2=ctr;
            kill(getppid(), SIGUSR1);
            // esperar que o outro processo intervenha
            if(ctr==n2) pause();
        }
    } else { // pai
        while (ctr < LIMITE) {
            // esperar que o outro processo intervenha
            if(ctr==n2) pause();
            printf("Pai %d\n", ctr);
            n2=ctr;
            kill(newpid, SIGUSR1);
        }
        kill(newpid, SIGKILL);
    }
}
```

```
#define LIMITE 20
int ctr; // var. global

void my_handler(int sinal) {
    ctr+=2;
    signal(SIGUSR1, my_handler2);
    if ( sinal==SIGKILL) exit(0);
}
```

PAI

n2 = ctr = 0

signal

fork

pause

ctr +=2 -> 2

printf 2

n2 = 2

kill (envia sinal ao filho)

pause

.....

FILHO

herda n2 = ctr = 0 do pai

herda a tab de sinais

ctr = 1

printf 1

n2 = 1

kill (envia sinal ao pai)

pause

ctr = 3

printf 3

n2 = 3

kill (envia sinal ao pai)

.....