Programação de sistema UNIX

Sinais

Definição e tipos

Sinais – Espécie de interrupções enviadas aos processos, na ocorrência de certos eventos

Cada processo pode definir uma função para responder a um sinal específico

Nome	Descrição	Origem	Acção por defeito
`SIGABRT`	Terminação anormal	`abort()`	Terminar
`SIGALRM`	Alarme	`alarm()`	Terminar
`SIGCHLD`	Filho terminou ou foi suspenso	S.O.	Ignorar
`SIGCONT`	Continuar processo suspenso	shell (fg, bg)	Continuar
`SIGFPE`	Excepção aritmética	hardware	Terminar
`SIGILL`	Instrução ilegal	hardware	Terminar
`SIGINT`	Interrupção	teclado (^C)	Terminar
`SIGKILL`	Terminação (*non catchable*)	S.O.	Terminar
`SIGPIPE`	Escrever num *pipe* sem leitor	S.O.	Terminar
`SIGQUIT`	Saída	teclado (^|)	Terminar
`SIGSEGV`	Referência a memória inválida	hardware	Terminar
`SIGSTOP`	Stop (*non catchable*)	S.O. (shell - stop)	Suspender
`SIGTERM`	Terminação	teclado (^U)	Terminar
`SIGTSTP`	Stop	teclado (^Y, ^Z)	Suspender
`SIGTTIN`	Leitura do teclado em *backgd*	S.O. (shell)	Suspender
`SIGTTOU`	Escrita no écran em *backgd*	S.O. (shell)	Suspender
`SIGUSR1`	Utilizador	de 1 proc. para outro	Terminar
`SIGUSR2`	Utilizador	idem	Terminar

Tratamento dos sinais

Acção por defeito – a inicial (ver tabela anterior)
Ignorar – o processo "descarta" o sinal
Catch – o processo instala um handler (função) que é chamado quando chega o sinal

Para modificar o tratamento dos sinais:

instala handler
ou ignora
ou coloca o default

```
#include <signal.h>

void ( * signal(int signo, void (*handler)(int)) )(int);
```

```
typedef void sigfunc(int);

sigfunc * signal(int signo, sigfunc *handler);
```

handler
SIG_IGN
SIG_DFL

```
#include <signal.h>

int raise(int signo);
```

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

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

Envio de sinais

Outros serviços com sinais

Envia o sinal SIGALRM após *seconds* segundos

```
#include <unistd.h>
unsigned int alarm(unsigned int seconds);
```

Bloqueia o processo até ao recebimento de um sinal não ignorado

```
#include <unistd.h>
int pause(void);
```

Envia ao processo o sinal SIGABRT. Este sinal, por defeito, termina o processo sem executar os *handlers* atexit( )

```
#include <stdlib.h>
void abort(void);
```

Bloqueia o processo durante *seconds* segundos. Podem existir outras variantes mais precisas

```
#include <unistd.h>
unsigned int sleep(unsigned int seconds);
```

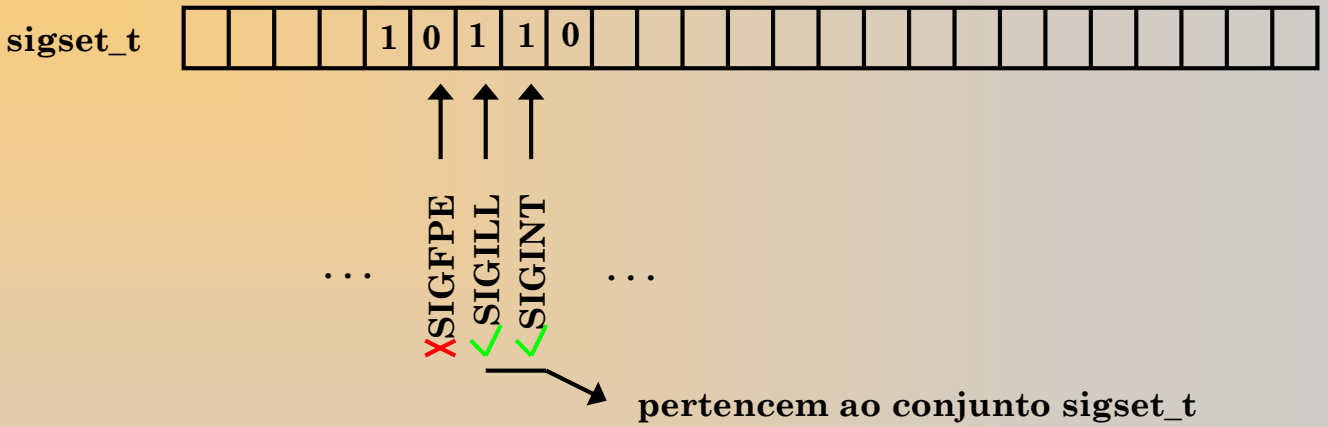
Um exemplo

Exemplo - Estabelecimento de um alarme e respectivo handler

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

int alarmflag = 0;
void alarmhandler(int signo);

void main(void)
{
   signal (SIGALRM, alarmhandler);
   alarm(5);
   printf (“Looping ...\n”);
   while (!alarmflag)
      pause();
   printf (“Ending ...\n”);
}

void alarmhandler (int signo)
{
   printf(“Alarm received ...\n”);
   alarmflag = 1;
}
```

Bloqueio de sinais (1)

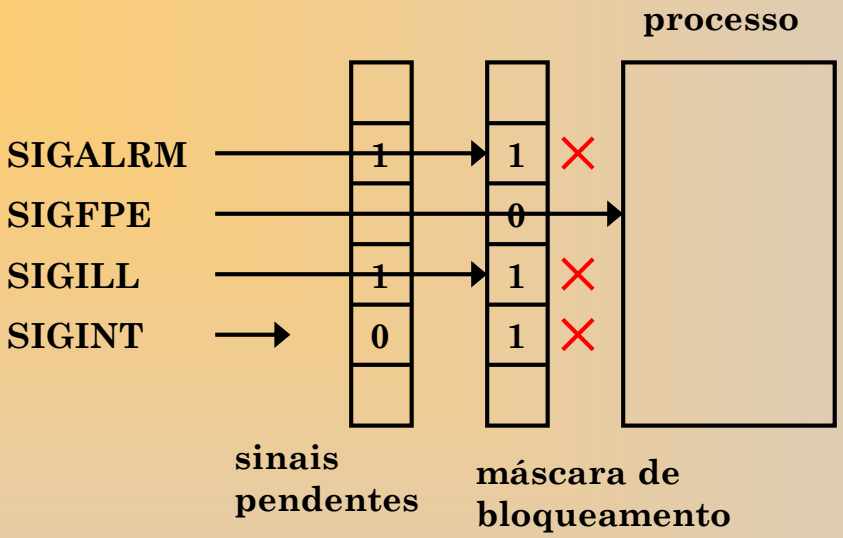
Máscara de sinais – conjunto ao qual pertencem alguns sinais e outros não




```
#include <signal.h>

int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
int sigismember(const sigset_t *set, int signo);
```

esvazia o conjunto
preenche completamente
adiciona um sinal
retira um sinal
testa a pertença

Bloqueio de sinais (2)



modifica a máscara de bloqueamento do processo

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
```

máscara anterior

SIG_BLOCK
SIG_UNBLOCK
SIG_SETMASK

operando de entrada

```
#include <signal.h>
int sigpending(sigset_t *set);
```

preenche conjunto dos sinais pendentes

Handlers e máscaras

```
#include <signal.h>
int sigaction(int signo, const struct sigaction *act, struct sigaction *oact);
```

```
struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
}
```

Estabelece simultaneamente um handler para um sinal e uma máscara, activa durante a execução desse handler

disposições antigas para o sinal *signo*

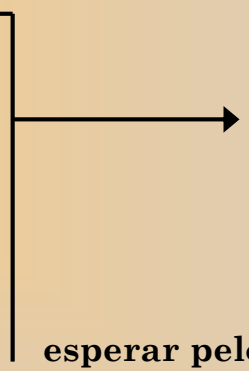
Exemplo:

```
char msg[] = "Control - C pressed!\n";
void catch_ctrl_c(int signo)
{
    write(STDERR_FILENO, msg, strlen(msg));
}
...
struct sigaction act;
...
act.sa_handler = catch_ctrl_c;
sigfillset(&act.sa_mask);
act.sa_flags = 0;
sigaction(SIGINT, &act, NULL);
...
```

Bloqueio melhor do que pause()

```
int flag = 0;
   ...
   while (flag = = 0)
      pause();
   ...

void handler(int signo)
{
   if (signo == SIGINT) {
      ...
      flag = 1;
   }
}
```

```
sigset_t mask;

...
sigfillset(&mask);
sigdelset(&mask, SIGINT);
sigsuspend(&mask);
...
```



esperar pelo sinal SIGINT

```
#include <signal.h>
int sigsuspend(const sigset_t *sigmask);
```

Bloqueia o processo até à chegada de um sinal, mas colocando a máscara sigmask activa, durante a espera