

Fonaments dels Sistemes Operatius (FSO)

Departament d'Informàtica de Sistemes i Computadores (DISCA)

Universitat Politècnica de València

Bloc Temàtic 2: Gestió de Processos

Unitat temàtica 3

SUT3: Crides al sistema UNIX per
a processos

fSO

DISCA



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

- Objectius:
 - Conèixer els **serveis** proporcionats pel Sistema Operatiu **UNIX per a crear processos**
 - Presentar **exercicis en llenguatge C** amb crides al sistema
 - Conèixer el **concepte de senyal**
- Bibliografia:
 - “**UNIX Programación Práctica**”, Kay A. Robbins, Steven Robbins. Prentice Hall. ISBN 968-880-959-4

- Identificació de processos
- Creació de processos
- Espera de processos
- Acabament de processos
- Senyals

	Processos
fork	Creació d'un procés fill
exit	Acabament del procés en execució
wait	Espera l'acabament d'un procés
exec	Canvia la imatge de memòria per la d'un executable (executa programa)
getpid	Obté atributs d'un procés
setsid	Modifica atributs d'un procés

	Senyals
kill	Enviar senyals
alarm	Generar una alarma (senyal de rellotge)
sigemptyset	Iniciar una màscara per a que no tinga senyals seleccionades
sigfillset	Iniciar una màscara per a que continga tots els senyals
sigaddset	Afegir un senyal concret a un conjunt de senyals
sigdelset	Esborrar un senyal concret d'un conjunt de senyals
sigismember	Consultar si un senyal concret pertany a un conjunt de senyals
sigprocmask	Examinar /Modificar /Establir una màscara de senyals
sigaction	Capturar/Manejar un senyal
sigsuspend	Esperar la captura de senyals

- **Identificació de processos**
- Creació de processos
- Espera de processos
- Acabament de processos
- Senyals

- **PID del procés**

- Cada procés ha de tindre un ID.
- El procés creador és el *pare*, mentre que el procés creat és el *fill*. Per a conèixer-los:

- **PID**: identitat del procés amb **getpid()**
- **PPID**: identitat del pare del procés

```
pid_t getpid(void);
```

```
pid_t getppid(void);
```

```
#include <stdio.h>
int main(void)
{
    printf("\nID del procés: %ld\n", (long)getpid());
    printf("ID del pare: %ld\n", (long)getppid());
    while(1);
    return 0;
}
```

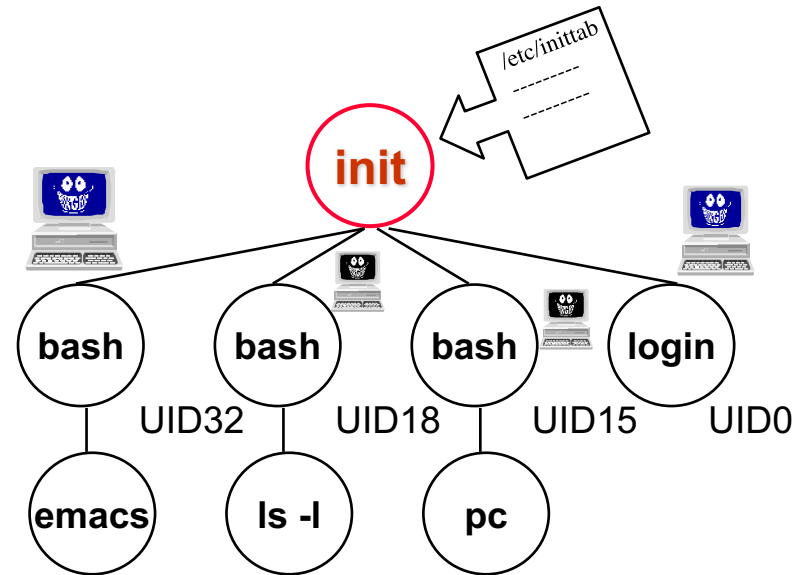
```
$ ./a.out &
[1] 2959
$
ID del procés: 2959
ID del pare: 1060
$ ps -l
```

UID	PID	PPID	F	CPU	PRI	NI	SZ	RSS	WCHAN	S	ADDR	TTY	TIME	CMD
501	1060	1059	4	006	0	31	0	2435548	1088 -	S	ffffff80136e3d50	ttys000	0:00.06	-bash
501	2959	1060	4	006	0	31	0	2434832	340 -	R	ffffff80140d8300	ttys000	0:04.65	./a.out

- Identificació de processos
- **Creació de processos**
- Espera de processos
- Acabament de processos
- Senyals

- Unix utilitza un mecanisme de **creació per còpia**
 - El procés **fill** és una **rèplica exacta** del seu procés **pare**
 - El procés **fill hereta** la majoria d'**atributs** del procés **pare**:
 - imatge de memòria
 - UID, GID
 - directori actual
 - descriptors de fitxers oberts
 - Unix assigna un identificador a cada procés denominat PID en el moment de la creació del mateix
 - Tot procés conèix l'identificador del seu procés pare, PPID
 - L'execució del fill és concurrent i independent
 - En UNIX existeix una **jerarquia de processos**

```
pid_t fork(void);
```



- **fork ()** : creació de processos

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

pid_t fork(void)
```

– Descripció

- Crea un procés fill que és un “clon” del pare: hereta gran part dels seus atributs.
- Atributs heretables: tots excepte **PID**, **PPID**, senyals pendents, temps/contabilitat.

– Valor de retorn

- 0 al fill
- PID del fill al pare
- -1 al pare si error

– Errores

- Insuficiència de recursos per a crear el procés

ETSINF-UPV DISCA

Fonaments dels Sistemes Operatius

```
$ ./ej2
Proces 3242 crea altre proces
Proces 3242 con pare 1060
Proces 3243 con pare 3242
```



- Pare i fill executen codi distint

```
//ej3_fork.c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
    pid_t val;
    int var = 0;
    printf("PID before fork(): %d\n", (long) getpid());
    val = fork()
    if (val > 0) {
        printf("Parent PID: %d\n", (int) getpid());
        var++;
    } else {
        printf("Child PID: %d\n", (long) getpid());
    }
    printf("Process [%d]-> var=%d\n", (long) getpid(), var);
    return 0;
}
```

Quants processos imprimeixen aquest missatge?

Quin valor(s) de "var" s'hi mostren?

- Exemple

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

```
int main(void)
{
```

```
    pid_t pid=fork();
```

```
    switch (pid) {
```

```
        case -1:
```

```
            printf("No s'ha pogut crear el procés fill\n");
            break;
```

```
        case 0:
```

```
            printf("Sóc el fill amb PID %ld i PPID %ld\n",
                   (long)getpid(), (long)getppid());
            break;
```

```
        default:
```

```
            printf("Sóc el pare amb PID %ld i el meu fill és %d\n",
                   (long)getpid(), pid);
```

```
    }
```

```
    sleep(5);
```

```
    return 0;
```

```
}
```

```
$ gcc ej3_fork.c
```

```
$ ./a.out
```

```
Sóc el pare amb PID 3702 i el meu fill és 3704
```

```
Sóc el fill amb PID 3704 i PPID 3702
```

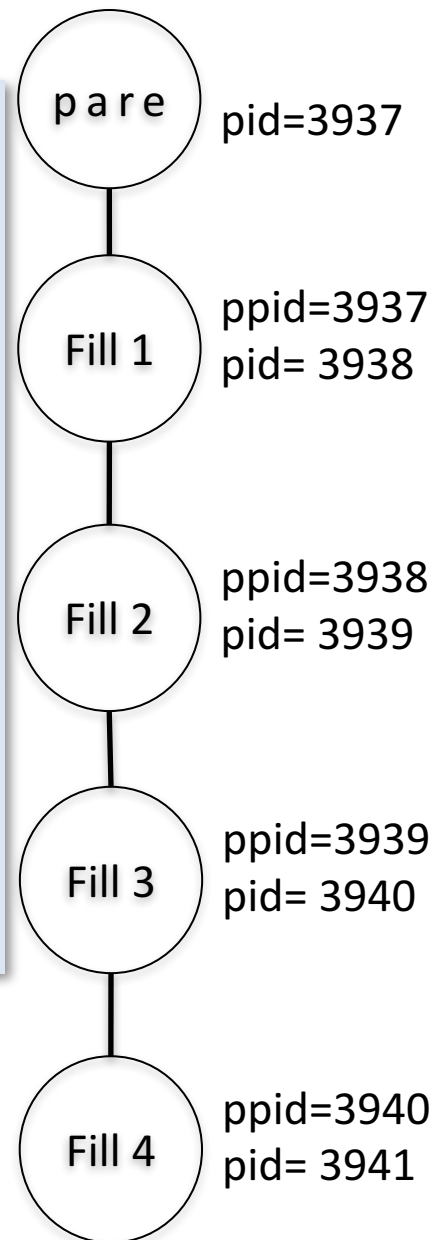
- Creació de processos en cadena

```
#include <stdio.h>
#include <sys/types.h>
#define NPROCESSOS 4
int main(void)
{
    pid_t pid;
    int i;

    for (i=0; i<NPROCESSOS; i++) {
        pid=fork();
        if (pid!=0)break;
        printf("Sóc el fill amb PID %ld amb pare %ld\n",
            (long)getpid(), (long)getppid());
    }
    sleep(5);
    return 0;
}
```

Variants

```
if (pid > 0)
if (pid==0)
if (pid<0)
```



```
$ ./a.out
Sóc el fill amb PID 3938 amb pare 3937
Sóc el fill amb PID 3939 amb pare 3938
Sóc el fill amb PID 3940 amb pare 3939
Sóc el fill amb PID 3941 amb pare 3940
```

- **exec ()**

- La crida a **fork ()** crea un fill que és una còpia del procés que el crida.
- Per a canviar el codi d'un procés s'utilitzen del crides **exec**.
- Existeixen diferents versions en funció dels paràmetres a especificar.

```
#include <unistd.h>
int execl(const char *path, const char *arg0, ...,
          const char *argn, char * /*NULL*/);
int execlp(const char *file, const char *arg, ... ,
          const char *argn, char * /*NULL*/);
int execlx(const char *path, const char *arg, ...,
          const char *argn, char * /*NULL*/ char * const envp[]);
int execv(const char *path, char *const argv[] );
int execvp(const char *file, char *const argv[]);
int execve(const char *path, char *const argv[], char *const envp[]);
```

- Variant **l**: els arguments es proporcionen per separat (llista)
- Variant **v**: els arguments es proporcionen amb un punter a vector
- Variant **p**: es cerca la ruta de **file** en el **PATH**
- Variant **e**: es proporciona l'entorn del fill mitjançant **envp**, no l'hereta del pare

- **exec ()**
 - Canvia la imatge de memòria d'un procés per la definida en un fitxer executable.
 - El fitxer executable s'expressa amb el nom **file** o la ruta completa **path**.
 - Alguns atributs del procés es conserven i, en particular:
 - El maneig de senyals, excepte els senyals capturats amb tractament per defecte.
 - El **PID**, **PPID**
 - Temps (comptabilitat)
 - Els descriptors de fitxer
 - El directori de treball, el directori arrel, la màscara del mode de creació de fitxers.
 - Si el bit **SETUID** del fitxer executable està activat, **exec** fica com a **UID** efectiu del procés al **UID** del propietari del fitxer executable.
 - Ídem amb el bit **SETGID** i el **GID** efectiu
 - Errors
 - Fitxer no existent o no executable
 - Permisos
 - Arguments incorrectes
 - Memòria o recursos insuficients
 - Valor de retorn
 - Si EXEC retorna al programa que el va cridar es que ha ocorregut un error; el valor de retorn és -1.

- Exemple: El procés fill llista el contingut del directori actual

```
//ej6_exec.c
#include <stdio.h>
#include <sys/types.h>
int main(void)
{
    int status;
    pid_t pid=fork();
    char* arguments [] = { "ls", "-l", 0 }
    switch (pid) {
        case -1:
            printf("No s'ha pogut crear el procés fill\n");
            break;
        case 0:
            printf("Sóc el fill amb PID %ld i llistaré el directori\n",
                (long)getpid());
            if (execvp("ls",arguments)==-1){
                printf("Error en exec\n");
                exit(0);
            }
            break;
        default:
            printf("Sóc el pare amb PID %ld i el meu fill es %d.\n",
                (long)getpid(), pid)
    }
    return 0;
}
```

Variants:

```
exec1("/bin/ls", "ls", "-l", NULL)
```


- Identificació de processos
- Creació de processos
- **Espera de processos**
- Acabament de processos
- Senyals

- Un pare ha d'esperar fins que el fill acabe:

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

- Crida **wait**
 - Deté l'execució del procés invocant fins que un fill acabe, o fins que el procés invocant reba un senyal
 - En `status` guarda l'estat retornat pel fill. Existeixen macros per a analitzar-lo.
 - Retorn
 - Torna el PID del fill,
 - -1 si error o no hi ha fills

- **wait()**: espera terminació

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

- **wait() / waitpid()** : Suspen l'execució del procés que l'invoca, fins que finalitza algun dels fills (wait) o un fill en concret (waitpid).
- Si existeix un fill zombie, wait finalitza immediatament. Si no, es deté.
- Quan *status* no és el punter NULL, conté:
 - Fill acaba amb **exit**:

MSB: status definit per exit()

LSB: 0

- Fill acaba per senyal:

MSB: 0

LSB: nº de señal (bit mes pes 1: core dump)

- Exemple (ej7_wait.c):

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

int main(void){
    int status;
    pid_t pid=fork();
    switch (pid) {
        case -1:
            printf("No s'ha pogut crear el procés fill\n");
            break;
        case 0:
            printf("Sóc el fill amb PID %ld i el meu pare es %ld\n",
                (long) getpid(), (long) getppid());
            sleep(20);
            printf("Ya he terminado\n");
            break;
        default:
            printf("Sóc el pare amb PID %ld i el meu fill és %d. Esperant...\n",
                (long) getpid(), pid);
            if (wait(&status)!=-1) printf("El meu fill ha acabat normalment\n")
    }
    return 0;
}
```

- **waitpid**

```
pid_t waitpid(pid_t pid, int *status, int options);
```

- Esperar a un fill en particular.
- Paràmetres:
 - **pid**: PID del fill a esperar. Si pid val `-1`, s'espera al primer que acabe (com `wait`).
 - **status**: estat del fill que retorna
 - **options**: per exemple, **WNOHANG** fa que la crida siga no bloquejant. En l'assignatura utilitzarem normalment la versió bloquejant d'aquesta crida: camp *options* igual a 0
 - **Valor de retorn**: Si 0, no ha acabat cap procés (versió no bloquejant). Si és `-1` indica error. Si és major que 0 el valor retornat és el pid del procés fill retornat.

Crida `waitpid()`

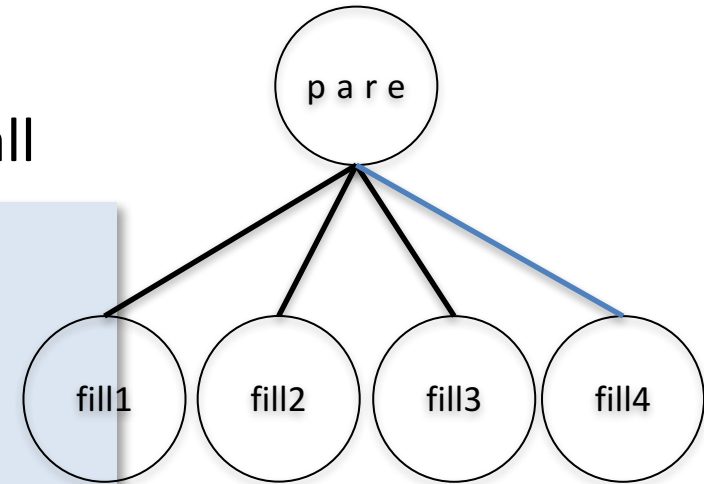
- Exemple: creació de processos en ventall

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

#define NPROCESSOS 4

int main(void)
{
    pid_t pid[NPROCESSOS];
    int i, status;

    for (i=0; i<NPROCESSOS; i++) {
        pid[i]=fork();
        if (pid[i]==0){
            printf("Sóc el fill %ld amb pare %ld\n",
                (long)getpid(), (long)getppid());
            sleep(i+1*5);
            exit(0);
        }
    }
    //Ara a esperar al tercer fill
    if (waitpid(pid[2],&status,0)==pid[2])
        printf("El meu tercer fill ja ha acabat \n");
    return 0;
}
```



```
$gcc -o ej8 ej8_waitpid.c
```

- Identificació de processos
- Creació de processos
- Espera de processos
- **Acabament de processos**
- Senyals

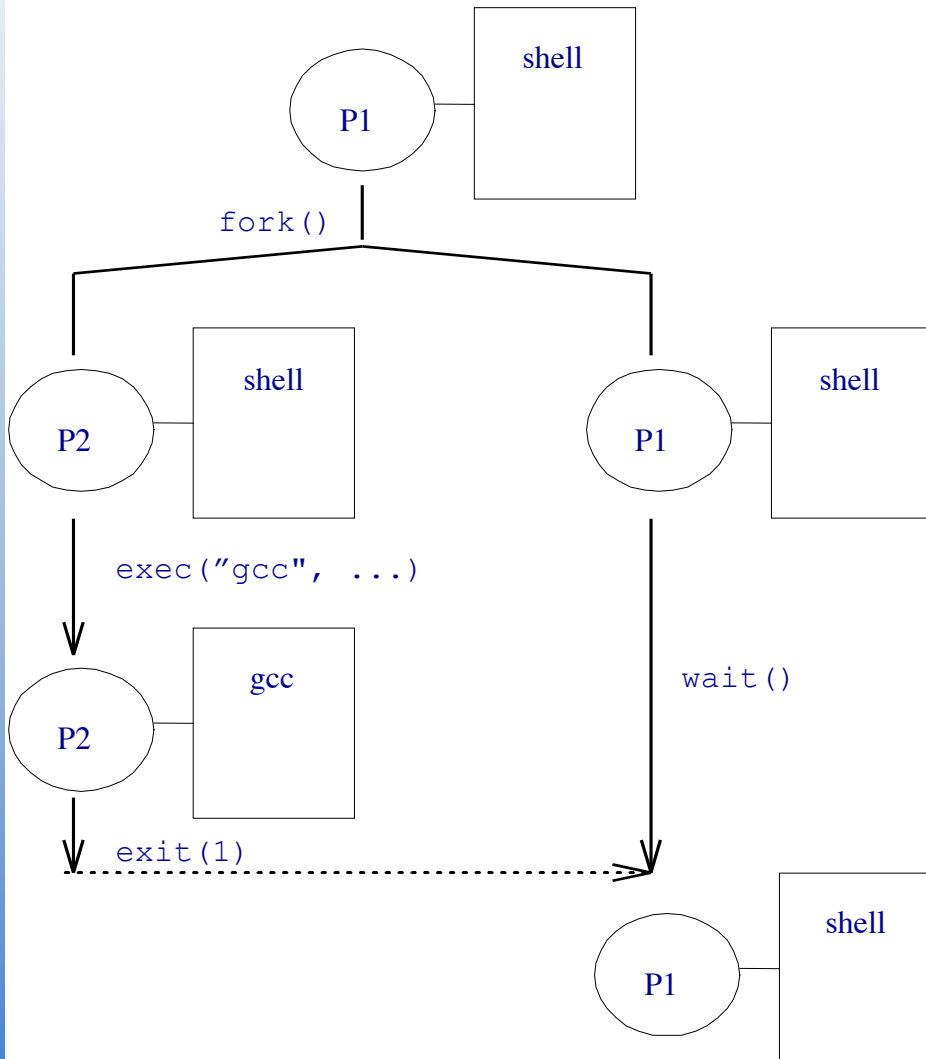
- **exit()**
 - Un procés acaba completament quan:
 - El procés en sí finalitza (normalment o anormal)
 - El seu pare ha realitzat una crida a `wait`.
 - L'acabament normal es realitza mitjançant la crida a **exit**.

```
void exit (int status)
```

- El valor del paràmetre **status** s'utilitza per a comunicar al procés pare la forma en que el procés fill acaba.
- Per conveni, aquest valor sol ser 0 si el procés acaba correctament i qualsevol altre valor en cas d'acabament anormal.
- El procés pare pot obtindre aquest valor a través de la crida al sistema `wait`.

- **Acabament anormal:**
 - El procés acaba per iniciativa del sistema operatiu al detectar alguna condició d'error (violació de límits, errors aritmètics) o per iniciativa d'algun altre procés
 - Senyals
- **Procés zombie:** Si el procés finaliza abans que el seu pare cride a **wait**
- **Procés orfre :** Si el procés pare finalitza abans que el fill
 - Un procés orfre és **adoptat** pel procés **init()**

- El shell de unix (estructura simple)



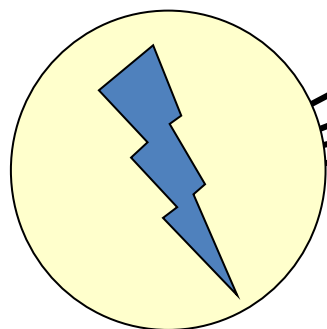
```
while(TRUE) {
    imprimir_prompt();
    leer_orden(orden, param);

    p=fork();                /* crear fill */

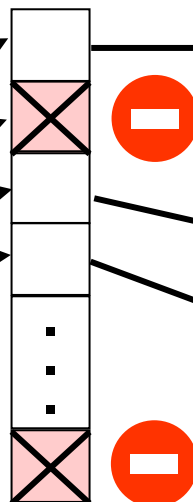
    if (p != 0) {            /* codi del pare */
        waitpid(-1, &status, 0); /* esperar fill */
    }else {                  /* codi del fill */
        exec(orden, params, 0); /* canviar imatge */
                                /* de memòria */
        error("No es pot exec. L'ordre");
        exit(1);
    }
}
```

- Identificació de processos
- Creació de processos
- Espera de processos
- Acabament de processos
- **Senyals**

Generació de senyals



Màscara de senyals



Maneig de senyals

`manejador(...)`

S'executa el manejador

`SIG_IGN`

Se ignora

`SIG_DFL`

Acció per defecte (exit: mata el procés)

⋮

- Terminal
- Errors
- Software

```
kill(...)
alarm(..)
```

Definició de la màscara

```
sigemptyset(...)
sigfillset(..)
sigaddset(..)
sigdelset(..)
sigismember(...)
```

Establiment de màscara

```
sigprocmask(...)
```

Establir manejador

```
sigaction(...)
```

Espera de senyals

```
pause(...)
sigsuspend(...)
```

- Un **senyal** és el **mecanisme** que utilitza el SO per a **informar als processos** de determinats **esdeveniments**
 - La crida `wait` deté al procés invocant fins que un fill acabe o es detinga, ***o fins que el procés invocant reba un senyal***
- Tots segueixen el mateix **patró**:
 - Es genera degut a l'ocurrència d'un esdeveniment
 - Es suministra al procés
 - Ha de rebre un tractament per part d'una rutina de tractament per defecte o be per una específica definida pel procés
- Un senyal pot:
 - **Manejar-se**: cal instal·lar el manejador
 - **Bloquejar-se**: es diferèix el seu tractament un temps
 - **Ignorar-se**: no s'informa de la seua ocurrència

- Crides UNIX per a senyals

	Senyals
kill	Enviar senyals
alarm	Generar una alarma (senyal de rellotge)
sigemptyset	Iniciar una màscara per a que no tinga senyals seleccionades
sigfillset	Iniciar una màscara per a que continga tots els senyals
sigaddset	Afegir un senyal concret a un conjunt de senyals
sigdelset	Esborrar un senyal concret d'un conjunt de senyals
sigismember	Consultar si un senyal concret pertany a un conjunt de senyals
sigprocmask	Examinar /Modificar /Establir una màscara de senyals
sigaction	Capturar/Manejar un senyal
sigsuspend	Esperar la captura de senyals

- Llista de d'alguns senyals de linux (en total hi ha 64):

Nº	Señal	Descripción	DFL.
1	SIGHUP	Penjat/mort terminal control	exit
2	SIGINT	CTRL-C de teclat (interrupció)	exit
3	SIGQUIT	CTRL-\ de teclat (int. amb <i>core</i>)	core
6	SIGABRT	Acabament anormal com amb <i>abort</i>	core
9	SIGKILL	Terminació. No manejable ni ignorable	exit
13	SIGPIPE	Esriptura en tub sense lector	exit
14	SIGALRM	Alarma temporitzada programada per <i>alarm</i>	exit
15	SIGTERM	Acabament per software	exit
-	SIGUSR1	Senyal 1 definida per l'usuari	exit


Mostra nº dels senyals:
\$kill -l

Mostra combinació de tecles per a cada senyal
\$ stty -a

- Exemple: Enviament de SIGALRM

Envia senyal SIGALRM als 10 segons

```
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
int main(void)
{
    alarm(10);
    while(1);
}
```



- Tractament del senyal: *sigaction*

```
#include <signal.h>

int sigaction(int signo, const struct sigaction *act,
               struct sigaction *oact)

struct sigaction{
    void (*sa_handler) (); /*SIG_DFL, SIG_IGN o funcion */
    sigset_t sa_mask;      /* els senyals addicionals
                           serán bloquejades durant
                           l'execució del manejador */
    int sa_flags;          /* indicadors i opcions */
}
```


- `sigaction`

- Instal·la els manejadors de senyal d'un procés.
- El senyal s'especifica en `signo` i els manejadors en `act` que es una estructura de tipus `struct sigaction`. El manejador anterior es retorna en `oact`.
- Si `act=NULL`, el manejador no canvia. Si no es necessita `oact` pot especificar-se `NULL`.
- Els manejadors són “permanents” (no es necessari tornar-los a instal·lar després de l'ocurrència d'un senyal)
- Als manejadores no se'ls pot passar paràmetres.
- `SIG_DFL`: instal·la l'acció per defecte (`exit`, `core`, ...)
- `SIG_IGN`: es maneja el senyal ignorant-lo (!= bloquejar-lo)
- Valor de retorn
 - 0 si funciona i -1 si error
- Errors
 - Senyal invàlid o no manejable (`SIGKILL`, `SIGSTOP`)

- Tractament del senyal: Als 10 s. s'imprimeix un missatge

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

void manejador()
{
    printf("Rebut el senyal\n");
}

int main(void)
{
    sigset_t conjunt_buit;
    struct sigaction accio_nova, accio_vella;

    sigemptyset(&conjunt_buit);
    accio_nova.sa_handler=manejador;
    accio_nova.sa_mask=conjunt_buit;
    accio_nova.sa_flags=0;

    if(sigaction(SIGALRM,&accio_nova,&accio_vella)==-1) return -1;

    alarm(10);
    while(1);
}
```

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

void manejador(){}

int main(void) {
    sigset_t conjunt_buit;
    struct sigaction accio_nova, accio_vella;
    sigemptyset(&conjunt_buit);
    accion_nueva.sa_handler=manejador;
    accion_nueva.sa_mask=conjunt_buit;
    accion_nueva.sa_flags=0;
    if(sigaction(SIGALRM,&accio_nova,&accio_vella)==-1) return -1;

    int status;
    pid_t pid=fork();
    if (pid!=0) {
        printf("Sóc el pare amb PID %ld i el meu fill és %ld\n",
            (long) getpid(), pid);
        alarm(10);
        if (wait(&status)!=-1) printf("El meu fill ha acabat normalment\n");
        else printf("Error en espera del fill\n");
        if (errno==EINTR) printf("Espera interrompuda per un senyal\n");
    } else {
        printf("Sóc el fill amb PID %ld i el meu pare és %ld\n",
            (long) getpid(), (long) getppid());
        sleep(60);
    }
    return 0;
}
```