

Prozesuen kontrolerako sistema-deiak Unix-en

kepa.bengoetxea@ehu.es

Prozesuen kontrolerako sistema-deiak Unix-en

Prozesuen identifikazioa

getpid, getppid, getuid

Prozesuen sorrera

fork, exec

Prozesuen bukaera/sinkronizazioa

exit, wait, kill

Seinaleen kontrola

kill, alarm, pause, signal

Denboraren kontrola

sleep, time, ctime

Prozesuen identifikazioa

int `getpid()`;

- Prozesuaren identifikadorea (pid) bueltatzen du

int `getppid()`;

- gurasoaren identifikadorea (pid) bueltatzen du

int `getuid()`;

- Prozesuaren “jabea” den erabiltzailearen identifikadorea (uid) bueltatzen du

Prozesuen identifikazioa

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

```
int main () {
    int id_prozesua, id_gurasoa, id_erabiltzailea;

    id_prozesua = getpid();
    id_gurasoa = getppid();
    id_erabiltzailea = getuid();

    printf("Prozesua: %d\n", id_prozesua);
    printf("gurasoa: %d\n", id_gurasoa);
    printf("Erabiltzailea: %d\n", id_erabiltzailea);
    exit(0);
}
```

```
$ gcc -o prozesuak prozesuak.c
$ ./prozesuak
Prozesua: 4542
gurasoa: 4535
Erabiltzailea: 1000
```

Prozesuen sorrera

int `fork()`;

- Sistema-dei honek prozesu berri bat sortzen du, prozesu deitzailearen “klona” dena. Prozesu deitzaileari gurasoa deituko diogu, eta sortutako prozesuari umea
- umeak bere exekuzioa `fork`-aren hurrengo agindutik hasten du (gurasoa ere bai)
- umeak gurasotik dena heredatzen du, baina umeak pid desberdina du!
- `fork` deiak honakoa bueltatzen du:
 - umeari: 0 (zero)
 - gurasoari: umearen pid-a
 - -1 errore bat gertatuz gero

Prozesuen sorrera

```
int main ()
{
    int pid;
    pid = fork();
    if (pid == -1)
        {perror(fork);
         exit(-1);
        }

    if (pid == 0) /* umea */
        {printf("%d umea naiz, %d gurasoarena\n",
                getpid(), getppid());}
    else /* gurasoa */
        {printf("%d gurasoa naiz, %d umearena\n",
                getpid(), pid);}

    printf("Bukatzera noa %d\n", getpid()); /*biak*/
    exit(0);
}
```

Prozesuen sorrera

```
2121 gurasoa naiz, 3456 umearena  
3456 umea naiz, 2121 gurasoarena  
Bukatzera noa 2121  
Bukatzera noa 3456
```

```
3456 umea naiz, 2121 gurasoarena  
2121 gurasoa naiz, 3456 umearena  
Bukatzera noa 2121  
Bukatzera noa 3456
```

```
2121 gurasoa naiz, 3456 umearena  
Bukatzera noa 2121  
3456 umea naiz, 1 gurasoarena  
Bukatzera noa 3456
```

... “1” da bere gurasoa
bukatu duelako, eta
Unix-en umezurtz
prozesuak “init”-ek
adoptatzen dituelako

fork sistema-deiaren proba

```
main()
```

```
{
```

```
    int pid;
```

```
    printf("fork deiaren proba\n");
```

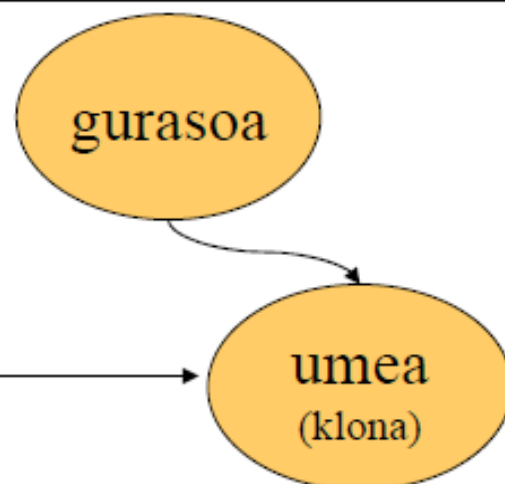
```
    pid = fork();
```

```
    printf("kontrola itzuli zaio ");
```

```
    if (pid == 0) printf("umeari\n");
```

```
    else printf("gurasoari, %d izanik ume berriaren pid-a\n", pid);
```

```
}
```



gurasoaren irteera

fork deiaren proba

kontrola itzuli zaio gurasoari, 999 izanik ume berriaren pid-a

umearen irteera

kontrola itzuli zaio **umeari**

•gurasaia

•**umea**

•gurasaia + umea

fork sistema-deiaren proba

Irteeraren kasu erreal posible batzuk

gurasoa

umea

gurasoa + umea

fork deiaren proba

kontrola itzuli zaio gurasoari, 999 izanik umearen pid-a

kontrola itzuli zaio umeari

fork deiaren proba

kontrola itzuli zaio umeari

kontrola itzuli zaio gurasoari, 999 izanik umearen pid-a

fork deiaren proba


kontrola itzuli zaio kontrola itzuli zaio umeari

gurasoari, 999 izanik umearen pid-a

...

571

```
Int i = 3, c_pid = -1;  
c_pid = fork();  
if(c_pid == 0)  
    printf("child process");  
else if(c_pid > 0)  
    printf("parent process");
```




Data

i = 3
c_pid = 574

574

```
Int i = 3, c_pid = -1;  
c_pid = fork();  
if(c_pid == 0)  
    printf("child process");  
else if(c_pid > 0)  
    printf("parent process");
```



Data

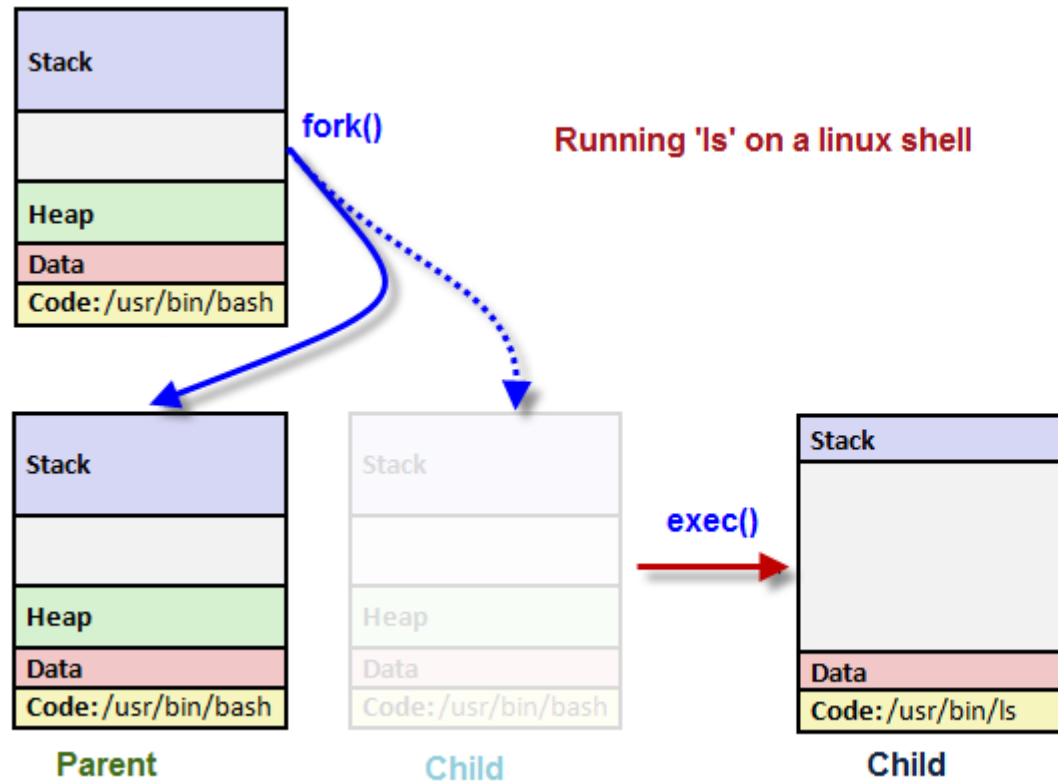
i = 3
c_pid = 0

Prozesuen sorrera

int **exec??**(...);

- Prozesua exekutatzen ari den programa aldatzen du
 1. Prozesuaren edukia husten du, testuingurua mantenduz
 2. Programa berria kargatzen du
- **exec??** deia ongi burutzen bada, ez du ezer bueltatzen... **programa aldatu egin delako!**
- Erroreren bat gertatzen bada, **-1** bueltatzen du

Prozesuen sorrera

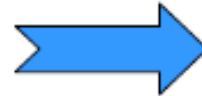


Prozesuen sorrera

2121

```
main() /* adibide3 */
{
    int pid;

    pid = fork();
    switch (pid){
        case -1: /* errorea */
            exit(-1);
        case 0: /* umea */
            execlp("ls", "ls", "-al", NULL);
            errore("execlp");
            break;
        default: /* gurasoa */
            printf("%d gurasoa, %d umearena\n",
                getpid(), pid);
    }
}
```



3456

```
main() /* ls */
{
    /* programa berria */
}
```

Prozesuen sorrera

exec funtzio familia:

parametro zerrenda:

int **execl** (char *path, char *arg0, ..., NULL);

parametro bektore:

int **execv** (char *path, char *arg[]);

parametro zerrenda + environment:

int **execle** (char *path, char *arg0, ..., char *envp[]);

parametro bektore + environment:

int **execve** (char *path, char *arg[], char *envp[]);

parametro zerrenda (izena path barik PATH-aldagaian bilatu):

int **execlp** (char *file, char *arg0, ..., NULL);

parametro bektore (izena path barik PATH-aldagaian bilatu):

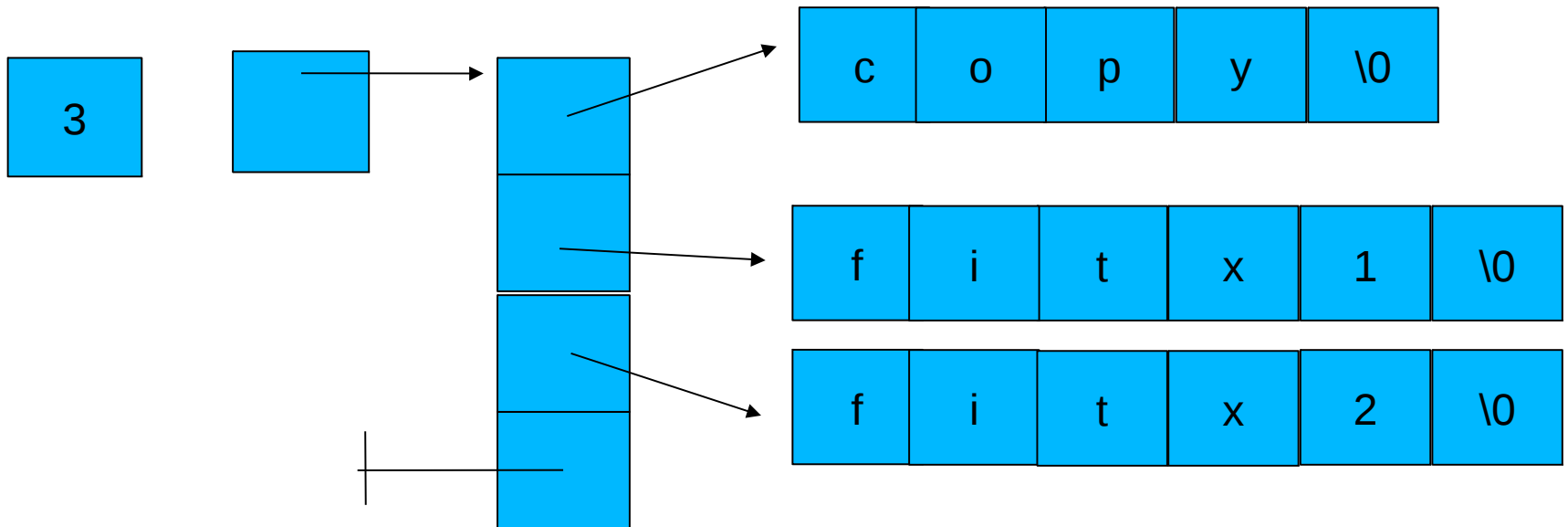
int **execvp** (char *file, char *arg[]);

- Demagun C lengoaian idatzita dagoen copy.c programa dugula. Bere main() metodoan:

```
int main(int argc, char * argv[])
```

```
argv[0]="copy"  
argv[1]="fitx1"  
argv[2]="fitx2"
```

Deia: copy fitx1 fitx2



Prozesuen Sorrera

Exec deien adibideak:

```
execl("/bin/ls", "ls", "-l", NULL);  
execle("/bin/ls", "ls", "-l", NULL, NULL);  
execlp ("ls", "ls", "-l", NULL);
```

```
char *v[] = {"ls", "-l", NULL};  
execv ("/bin/ls", v);  
execve("/bin/ls", v, NULL);  
execvp("ls", v);
```


Prozesuen sorrera:exec

```
int main() /* execlp aginduaren adibidea */
{
    int pid;
    pid = fork();
    switch (pid){
        case -1: /* errorea */
            perror("execlp:");
            exit(-1);
        case 0: /* umea */
            execlp("ls", "ls", "-al", NULL);
            printf("Ezin izan da execlp egikaritu\n");
            exit(-1);
        default: /* gurasoa */
            printf("%d gurasoa, %d umearena\n", getpid(), pid);
    }
    exit(0);
}
```

Prozesuen sorrera

```
kepa@cox:/tmp/c$ ./execlp
```

```
15316 gurasoa, 15317 umearena
```

```
drwxr-xr-x 2 kepa kepa 4096 2009-11-14 16:42 .
```

```
drwxrwxrwt 19 root root 524288 2009-11-14 16:42 ..
```

```
-rwxr-xr-x 1 kepa kepa 9232 2009-11-14 16:42 execlp
```

```
-rw-r--r-- 1 kepa kepa 512 2009-11-14 16:42 execlp.c
```

```
kepa@cox:/tmp/c$ ./execlp
```

```
drwxr-xr-x 2 kepa kepa 4096 2009-11-14 16:42 .
```

```
drwxrwxrwt 19 root root 524288 2009-11-14 16:42 ..
```

```
-rwxr-xr-x 1 kepa kepa 9232 2009-11-14 16:42 execlp
```

```
15320 gurasoa, 15321 umearena
```

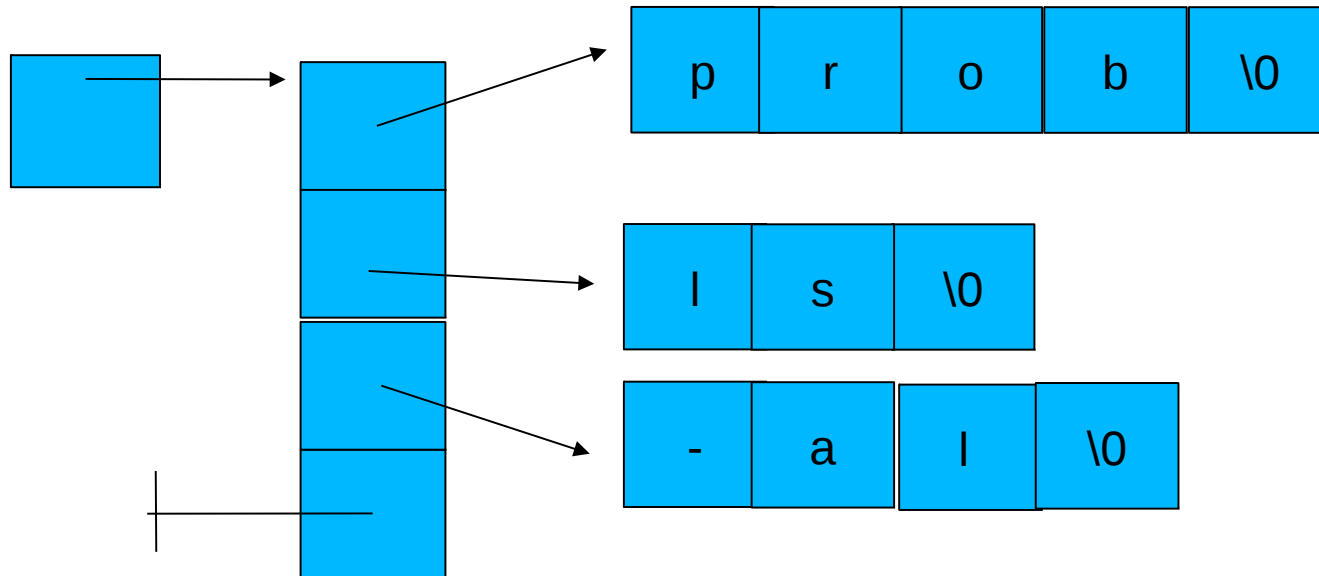
```
-rw-r--r-- 1 kepa kepa 512 2009-11-14 16:42 execlp.c
```

Prozesuen sorrera

```
Int main(int argc, char *argv[]) /* prob */
{
    int pid;
    pid = fork();
    switch (pid){
        case -1: /* errorea */
            perror("fork:");
            exit(-1);
        case 0: /* umea */
            execvp(argv[1], &(argv[1]));
            perror("execvp:");
            exit(-1);
        default: /* gurasoa */
            printf("%d gurasoa, %d umearena\n", getpid(),
                                                           pid);
    }
    exit(0);
}
```

Prozesuen sorrera

- Zergatik `execvp(argv[1], &argv[1])`?



Prozesuen sorrera:exec

Gedit guraso.c

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

```
#include <unistd.h> /*man fork,exec */
```

```
#include <sys/types.h> /*man 2 wait */
```

```
#include <sys/wait.h> /*man 2 wait */
```

```
int main() {
```

```
int pid,status=1;
```

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

Prozesuen sorrera:exec

```
if (pid == 0) { /* Umea */  
    if (execl("umea","umea", "izena", "-a", NULL) == -1) {  
        perror("execl");  
        exit(1);} }  
else { /* Guraso */  
    wait(&status);  
    printf("\nSemearen bukaerako egoera: %d\n", status);  
    exit(0);}  
}
```

Prozesuen sorrera:exec

Gedit umea.c

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

```
int main(int argc, char *argv[]) {
```

```
    int i = 0;
```

```
    for (i = 0; i < argc; i++) printf("\nArgumentua [%d]: %s", i, argv[i]);
```

```
    exit(0);}
```

```
gcc -o guraso guraso.c
```

```
gcc -o umea umea.c
```

```
./guraso
```

Prozesuen sorrera:exec

./guraso

Argumentua [0]: umea

Argumentua [1]: izena

Argumentua [2]: -a

Semearen bukaerako egoera: 0

Prozesuen bukaera/sinkronizazioa

void **exit**(int egoera);

- Prozesuaren bukaera “kontrolatua”
- Unix-ek bukaera-kodea (egoera) gorde egiten du gurasoak `wait()` bukatu arte

int **wait**(int *egoera);

- Deitzailea bere umearen bukaera arte geldiarazten du
- Umerik ez badu, -1 bueltatzen du, deitzailea blokeatu gabe
- Bukatzen duen ume-prozesuaren identifikadorea bueltatzen du
- **egoera** umea bueltatutako bukaera-kodea da

int **kill**(int pid, int SIGKILL);

- SIGKILL seinalea pid prozesuari bidaltzen dio, bukaraziz

Prozesuen bukaera/sinkronizazioa

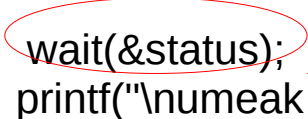
```
#include <stdio.h>
#include <unistd.h> /*man 2 fork */
#include <sys/types.h> /*man 2 wait */
#include <sys/wait.h>
#include <stdlib.h>

int main() {
    int pid = 0, status = -1;

    if ((pid = fork()) == -1)
    { printf('Errorea umea sortzerakoan \n');exit(1); }

    if (pid == 0)
    { printf('Nire gurasoaren PID zenbakia: %d\n',getppid()); exit(0); }

    else
    { printf("Nire PID zenbakia: %d eta nire
                                     umearena: %d\n", getpid(), pid);
      wait(&status);
      printf("\n umeak itzuli duen egoera zenbakia: %d \n", status);
      exit(0);
    }
}
```



umea bukatu arte itxaron

Prozesuen bukaera/sinkronizazioa

```
kepa@cox:/tmp$ ./wait  
Nire PID zenbakia: 22079 eta nire umearena: 22080  
Nire gurasoaren PID zenbakia: 22079
```

```
umeak itzuli duen egoera zenbakia: 0
```

```
kepa@cox:/tmp$ ./wait  
Nire PID zenbakia: 22081 eta nire umearena: 22082  
Nire gurasoaren PID zenbakia: 22081
```

```
umeak itzuli duen egoera zenbakia: 0
```

```
kepa@cox:/tmp$ ./wait  
Nire gurasoaren PID zenbakia: 22083  
Nire PID zenbakia: 22083 eta nire umearena: 22084
```

```
umeak itzuli duen egoera zenbakia: 0
```

Prozesuen bukaera/sinkronizazioa

- **kill:**

- `#include <sys/types.h> eta <signal.h>: int kill(pid_t pid, int sig);`
- Prozesu bati SIGXXXX seinale bat bidaltzeko.
- 0 itzuliko du ondo joanez gero eta -1 errore bat gertatuz gero.
- `Adb:kill(pid,SIGTERM)`

Prozesu bati seinaleak bidali. Adibidea:

```
#include <sys/types.h> /*kill y wait*/

#include <sys/wait.h> /*wait*/

#include <signal.h> /*kill*/

#include <stdlib.h> /*puts y exit*/

#include <stdio.h> /*printf*/

#include <unistd.h> /*fork*/

int main(void) { pid_t hijo; int condicion, valor_retornado;

hijo=fork();

if (hijo==-1) {perror("fork");exit(1);}

if (hijo==0) {sleep(1000);exit(0);}

else {valor_retornado=kill(hijo,SIGKILL);

        if (valor_retornado==-1) {perror("kill");wait(&condicion);}

        else {printf("%d ezabatua \n",hijo);}

exit(0); }}
```

Prozesuen bukaera/sinkronizazioa

```
$gcc -o killer killer.c
```

```
$ ./killer
```

6680 ezabatua

Prozesuen bukaera/sinkronizazioa

- Prozesu baten exekuzioan, edozein momentuan eman ahal den gertaerari seinalea deritzo (Seinalea = Software etena)
- Seinale bakoitza izen bat du (kill -l edo man 7 signal) eta seinaleekin lan egiten dituzten funtzioak `</usr/include/signal.h>` daude.
- Prozesu batek seinale bat jasoz gero :
 - Seinalea kontuan ez hartu
 - Seinale horri dagokion funtzio lehenetsia (gehienetan exit bat core edo core barik)
 - Seinale horri lotu ahal diogu beste funtzio bat (handler), eta gero exit edo gertaera gertatu aurreko instrukziora itzuli.

Prozesuen bukaera/sinkronizazioa

\$ man 7 signal

| Signal | Value | Action | Comment |
|----------------|----------|--------|-------------------------------------------------------------------------|
| SIGHUP | 1 | Term | Hangup detected on controlling terminal or death of controlling process |
| SIGINT | 2 | Term | Interrupt from keyboard |
| SIGQUIT | 3 | Core | Quit from keyboard |
| SIGILL | 4 | Core | Illegal Instruction |
| SIGABRT | 6 | Core | Abort signal from abort (3) |
| SIGFPE | 8 | Core | Floating point exception |
| SIGKILL | 9 | Term | Kill signal |
| SIGSEGV | 11 | Core | Invalid memory reference |
| SIGPIPE | 13 | Term | Broken pipe: write to pipe with no readers |
| SIGALRM | 14 | Term | Timer signal from alarm (2) |
| SIGTERM | 15 | Term | Termination signal |
| SIGUSR1 | 30,10,16 | Term | User-defined signal 1 |
| SIGUSR2 | 31,12,17 | Term | User-defined signal 2 |
| SIGCHLD | 20,17,18 | Ign | Child stopped or terminated |
| SIGCONT | 19,18,25 | Cont | Continue if stopped |
| SIGSTOP | 17,19,23 | Stop | Stop process |
| SIGTSTP | 18,20,24 | Stop | Stop typed at tty |
| SIGTTIN | 21,21,26 | Stop | tty input for background process |
| SIGTTOU | 22,22,27 | Stop | tty output for background process |

Prozesuen bukaera/sinkronizazioa

man 7 signal

signal - lista de las señales disponibles

| Señal | Valor | Acción | Comentario |
|-------|-------|--------|------------|
|-------|-------|--------|------------|

| | | | |
|---------|-------------------|-----|-------------------------------------------------------------|
| SIGINT | 2 | A | Interrupción procedente del teclado(Ctrl-C)(salida) |
| SIGQUIT | 3 | C | Terminación procedente del teclado(Ctrl-4)(salida con core) |
| SIGABRT | 6 | C | Señal de aborto procedente de abort(3) |
| SIGKILL | 9 | AEF | Señal de matar |
| SIGALRM | 14 | A | Señal de alarma de alarm(2) |
| SIGTERM | 15 | A | Señal de terminación |
| SIGUSR1 | 30, <u>10</u> ,16 | A | Señal definida por usuario 1 |
| SIGCONT | 19, <u>18</u> ,25 | | Continuar si estaba parado |
| SIGSTOP | 17, <u>19</u> ,23 | DEF | Parar proceso |

Prozesuen bukaera/sinkronizazioa

A-La acción por defecto es terminar el proceso

B-La acción por defecto es ignorar la señal

C-La acción por defecto es terminar el proceso con core dump

D-La acción por defecto es parar la ejecución del proceso

E-La señal no puede ser capturada por el programa(manipulada)

F-La señal no puede ser ignorada

***core dump: volcado de memoria del contexto del proceso a la carpeta del proceso, para poder ver con un programa de depuración como gdb, sdb o adb**

Seinaleen kontrola

`int kill(int pid, int SIGKILL);`

SIGKILL seinalea pid prozesuari bidaltzen dio, bukaraziz

`void pause();`

Funtzio hau seinale bat jaso arte lo geratzen da. -1 itzultzen du, seinaleari dagokion funtzioa exekutatu ostean.

`int signal(int seinalea, void funtzioa());`

Seinalea jasoz gero exekutatuko den funtzioa

Seinaleen kontrola

Adibidez: gedit pausa.c

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

```
#include <stdlib.h>
```

```
int main(void){pause();
```

```
exit(0);}
```

```
$gcc -o pausa pausa.c
```

```
$/pausa
```

```
Ctrl-4 edo Ctrl-C edo kill -USR1 <PID>
```

Irten

Seinaleen kontrola

```
int signal(int seinala, void funtz());
```

- Seinaleari funtz funtzioa lotzen dio . Signal funtzioak 2 parametro jasotzen ditu:
 - Jaso nahi dugun seinala zenbakia. Zeintzuk? “kill -l”
 - Seinala hori jaso ostean exekutatuko den funtzioaren erakuslea. Hurrengo makroak ere erabili ahal dira: SIG_IGN (seinala kontuan ez hartu) edo SIG_DFL(dagokion portaera lehenetsia uztea)
- Errore bat gertatuz gero SIG_ERR itzultzen du

Seinaleen kontrola

- Funtzio baten erakuslea:
 - C-n funtzio bat konpilatzerakoan bere kodean sarrera puntu bat sortzen da.
 - Funtzio baten deia egiterakoan, kode sarrera puntuari dei bat egiten ari gara.
 - Kode sarrera puntua ere erabili ahal da funtzioari dei egiteko.
 - Kode sarrera puntuaren helbidea funtzioaren izena da. (arraian bezala)

Adibidez, “printf” funtzioaren erakusle bat erabiltzeko. Jakinda: printf-ren erazagupena “int printf(const char *format, ...);” dela eta funtzio erakuslea horrela erazagutzen dela “funtzioak_itzultako_mota (*p) ();” ,non “p” erakuslearen izena den:

Erazagupena: int (*p)();

Esleipena: p=printf;

Deia: (*p)(“kaixo”);

Seinaleen kontrola

- **Adibidez:**

```
#include <stdio.h>
#include <signal.h>
int sig;
void jaso (sig)
{printf("jasotako seinalea:
    %d\n",sig);}
int main(){
/*USR1 seinaleari jaso funtzioa
    esleitu */
signal(SIGUSR1,jaso);
```

```
kill(getpid(),SIGUSR1);

/*seinalea kontuan ez hartu*/

signal(SIGUSR1,SIG_IGN);

kill(getpid(),SIGUSR1);

/*dagokion portaera lehenetsia uztea*/

signal(SIGUSR1,SIG_DFL);

kill(getpid(),SIGUSR1);

return 0;

}
```

Denboraren kontrola

alarm: unsigned int alarm(unsigned int seg); SEak deitzaileari SIGALARM seinalea bidaltzen dio seg segundo igaro ondoren. Funtzio honek >0 balore bat itzuliz gero, jadanik alarma bat programatua dagoela adieraziko digu.

sleep: int sleep(unsigned int seconds); SEak prozesua seconds segundu lotan utziko du. Adb:sleep(30)

time: time_t time(time_t *t); devuelve el tiempo transcurrido, medido en segundos, desde 1 de enero de 1970. Si t no es el puntero nulo, el valor devuelto también se guarda en la zona de memoria a la que apunte. Adb:segundoak=time(0);

Denboraren kontrola

Adb: gedit alarma.c

```
#include ...
```

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

```
if ((alarm(5))>0)
```

```
    {printf("jadanik alarma bat programatua dago");exit(0);}
```

```
sleep(30);
```

```
printf("Nola liteke, hemendik igarotzea?");
```

```
exit(1); }
```

```
$gcc -o alarma alarma.c
```

```
$ ./alarma
```

Alarm clock

Denboraren kontrola

Adibide-programa

Guraso-prozesu batek programa baten exekuzioa abiarazten du, honen izena eta argumentuak bigarren parametrotik pasatzen zaiolarik. Lehen argumentuak programaren exekuzio-denbora maximoa adieraziko du; denbora hori pasa eta programak bukatu ez badu, guraso prozesuak umearen exekuzioa amaiaraziko du. Guraso-prozesuak bere identifikadorea eta abiarazten duen programarena ere idazten ditu. Programa berriak amaitzean, gurasoak bere iraupena idatzi eta itzulera-kodea itzultzen du, edo 1 balioa programa amaiarazi badu.

Exekuzio-adibidea:

> ./guraso 60 umea

```
umea.c
#include <unistd.h> /*sleep*/
int main(){
sleep(6);
return 0;
}
```

```
erlojua.c
#include <stdlib.h> /*atoi and exit*/
#include <stdio.h> /*printf*/
#include <unistd.h> /*alarm*/
#include <signal.h> /*signal and SIGALRM*/

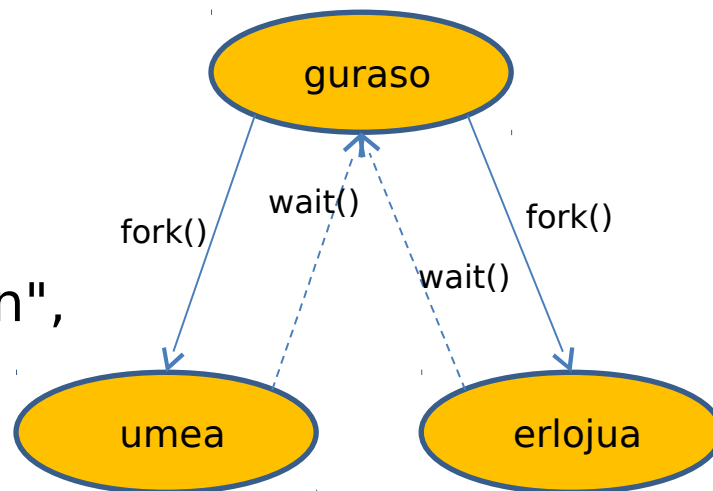
void xtimer(){
printf("time expired.\n");
}

int main(int argc, char *argv[]){
unsigned int sec;
sec=atoi(argv[1]);
printf("time is %u\n",sec);
signal(SIGALRM,xtimer);
alarm(sec);
pause();
return 0;
}
```

gurasoa.c

```
int main(int argc, char *argv[]){
int idHijo, idReloj, id, t1, status1,
status2;
id = getpid();
printf("Proceso padre: %d\n", id);
idReloj = fork();
if(idReloj == 0) { /* hijo Reloj */
execl("/home/kepa/erlojua",
"erlojua", argv[1], NULL);
}
if((idHijo = fork()) == 0) {
execv(argv[2], &argv[2]);
}
printf("Proceso hijo Perezoso:
%d\n", idHijo);
printf("Proceso hijo Reloj: %d\n",
idReloj);
t1 = time(0);
```

```
id = wait(&status1)
if((id == idHijo) {
kill(idReloj, SIGKILL);
//Erlojuaren egoera jaso
wait(&status2);
} else {
kill(idHijo, SIGKILL);
//Semearen egoera jaso
wait(&status2);
status1 = 1;}
t1=time(0)-t1;
printf("Tiempo del proceso hijo : %d\n", t1);
exit(status1);
}
```



```
gcc guraso.c -o guraso
gcc erloju.c -o erloju
gcc umea.c -o umea
./guraso 20 umea
Proceso padre: 3327
Proceso hijo Perezoso: 3329
Proceso hijo Reloj: 3328
time is 20
Tiempo del proceso hijo : 6
$ echo $?
0
```

```
./guraso 3 umea
Proceso padre: 3315
Proceso hijo Perezoso: 3317
Proceso hijo Reloj: 3316
time is 3
time expired.
Tiempo del proceso hijo : 3
echo $?
1
```