```c
#include <stdlib.h>         //    Miscellanious (https://pubs.opengroup.org/onlinepubs/009604599/basedefs/stdlib.h.html)
#include <stdio.h>          //    I/O and miscellanious (https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/stdio.h.html)
#include <string.h>         //    strcpy, strlen (https://pubs.opengroup.org/onlinepubs/7908799/xsh/string.h.html)
#include <unistd.h>         //    fork, pipe, dup2, dup, close, open, execlp, read, write (and many others) (https://pubs.opengroup.c
#include <sys/wait.h>       //    waitpid, wait... (https://pubs.opengroup.org/onlinepubs/009696699/basedefs/sys/wait.h.html)
#include <sys/types.h>      //    Data types (https://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/types.h.html#tag_13_67)
#include <sys/stat.h>       //    S_IRUSR, S_IWUSR, S_IXUSR (https://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/stat.h.html
#include <sys/un.h>         //    Main socket functions (https://pubs.opengroup.org/onlinepubs/009696699/basedefs/sys/un.h.html)
#include <fcntl.h>          //    O_TRUNC, O_CREATE, O_RDONLYM O_RDWR, O_WRONLY... (https://pubs.opengroup.org/onlinepubs/009695399/b
#include <time.h>           //    Clock instruccions (https://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/time.h.html)
#include <errno.h>          //    errno, EEXIST (https://pubs.opengroup.org/onlinepubs/009696899/basedefs/errno.h.html)

//////////////////////////
//    Functions use     //
//////////////////////////

int fd[2];
int fd, fd_substituit, ret;
char buffer[128];

pipe(fd);                           // Crea una pipe sense nom i indica la sortida d'aquesta a df[0] i l'entrada a fd[1]

dub2(fd, fd_substituit);            // Sobreescriu fd a fd_substituit (mirar "Conectar pipe amb terminal o altres pipes" més avall)

dub(fd);      // Ocupa el primer canal lliure amb una copia de fd

close(fd);    // Tanca el canal fd, això es important sobretot per tancar els escriptors i així
              // evitar que els lectors es quedin en un bucle infinit

ret = read(fd, buffer, sizeof(buffer));   // fd -> canal d'on volem llegir, buffer -> on volem guardar lo llegit,
                                          // sizeof(buffer) -> nº bytes a llegir, ret -> bits llegits
                                          // Si s'intenta llegir una pipe buida es quedarà bloquejat fins que hi hagin dades
                                          // En una pipe read només retornarà 0 quan no hi hagin escriptors d'aquesta (per això és important tancar-los)

ret = write(fd, buffer, sizeof(buffer));  // fd -> canal on volem escriure, buffer -> el que volem escriure,
                                          // sizeof(buffer) -> nº bytes a escriuer, ret -> bits escrits
                                          // Si s'intenta escriure en una pipe plena es quedarà bloquejat fins que es buida suficientment

fd = open("nom", acces_mode, permision_flags);    // El "nom" pot ser un fitxer o una pipe (amb nom clar), fd -> canal al que s'assgina el fitxer
                                                  // acces_mode:  O_RDONLY, O_WRONLY, O_RDWR (un d'aquest és sempre necessari)
                                                  //              O_CREAT -> crea; O_TRUNC -> Sobreescriu; O_APPEND -> afageix al final
                                                  //              Podem conquetenar els modes d'accés amb una '|'
                                                  // permision_flags: si renim que <rwxrwxrwx> (en grups de tres fan referencia a usuari, grup i others)
                                                  //                  podem ficar un 1 als que volem activar i convertir-ho en octal
                                                  // per exemple si volem read & write per l'usuari, fariem: 110000000 -> 600

int nova_posicio = lseek(fd, desplaçament, relatiu_a);  // Només per fitxers, fd és el canal del fitxer
                                                        // Posició accedida (nova_posicio) = relatiu_a + desplaçament (pot ser negatiu)
                                                        // relatiu_a: SEEK_SET (com un .begin()), SEEK_CUR (posició actual), SEEK_END (com un .end())

mknod("nompipe", S_IFIFO | 600, 0);                     // "nompipe" (autodescriptiu);  S_IFIFO (volem una pipe basicament, si vols +info ves al segon
                                                        //                              link de <sys/stat.h> per veure les altres possibilitats)
                                                        // 600 -> fa referencia als permisos, obtinguts de la mateixa forma que "permision_flags"
                                                        // El 0 del final no sé que vol dir pero no el tocaria, si tu ho saps pots canviar aquesta frase (

//////////////////////////
//  Creació pipe bàsica //
//////////////////////////

{
  int fd[2];

  pipe (fd);  //Retorna -1 si hi ha hagut algun error en crear la pipe

  write (fd[1], "Hola", 4); //Escriure a la pipe

  char buff[128];
  read (fd[0], buff, strlen(buff)); //Llegir de la pipe (buff = "Hola"). Retorna 0 si la pipe no té escriptors
}
```

```
////////////////////////////
//  Creació pipe amb nom  //
////////////////////////////

{

if (mknod("npipe", S_IFIFO | 600, 0) < 0 && errno != EEXIST)
{
  perror("Error creant la pipe");
  exit(1);
}

int fd = open("nom", O_RDONLY); //Guardem el canal de lectura de la pipe a fd;
//{...}
close (fd);
}
```

```
//////////////////////
//  Pipe amb Forks  //
//////////////////////

{
  int pid = fork(); //Retorna -1 si hi ha hagut error;

  //Procés del fill
  if (pid == 0) {
    close (fd[1]); //Tanquem el canal d'escriptura a la pipe per poder llegir
    char c;
    while (read (fd[0], &c, sizeof(c)) < 0) {
      write (1, &c, sizeof(c)); //Escrivim a la terminal el que anem llegint de la pipe
    }
    close (fd[0]);
    exit (0);
  }
  close (fd[1]); //Tanquem el canal d'escriptura a la pipe perquè el fill pugui llegir!!
  close (fd[0]); //Tanquem el canal de lectura ja que no el necessitem
  wait (NULL); //Esperem el fill
  exit (0);
}
```

```
////////////////////////////////////////////
//  Obrir un fitxer per llegir i/o escriure //
////////////////////////////////////////////

fd = open ("Nom_fitxer", O_RDONLY);   //Canal de lectura al fitxer
fd = open ("Nom_fitxer", O_WRONLY);   //Canal d'escriptura al fitxer
fd = open ("Nom_fitxer", O_RDWR);     //Canal d'escriptura al fitxer



////////////////////////////////////////////////
//  Conectar pipe amb terminal o altres pipes //
////////////////////////////////////////////////

dup2 (fd[0], 0); //El canal de lectura de la pipe passa a ser el canal de lectura de la terminal
close (fd[0]);

dup2 (fd[1], 1); //El canal d'escriptura de la pipe passa a ser el canal d'escriptura de la terminal
close (fd[1]);

dup2 (fd1[0], fd2[1]); //El canal de lectura de la pipe 1 passa a ser el canal d'escriptura de la pipe 2

int newfd = dup(fd); //Copia el canal fd al primer canal disponible i indica quin és aquest (ident. newfd)
```

```
//////////////////////////////////////////
//  Accés directe a posicions de fitxers  //
//////////////////////////////////////////

fd = open ("Nom_fitxer", O_RDWR);
lseek (fd, desplaçament, [*]);      // Mira a sota

//[*]:
    // SEEK_SET: punter = desplaçament
    // SEEK_CUR: punter = punter + desplaçament
    // SEEK_END: punter = file.size() + desplaçament
    // Si tenim:
              lseek (fd, -1, SEEK_END);
              read(fd, buffer, sizeof(char));
    // Llegirem l'últim char del fitxer
```

```
//////////////////////////////////////
//         Exemples de pipes        //
//////////////////////////////////////

// Un procés escriu per una pipe i llegeix el contingut d'aquesta
//-----------------------------------------------------------------

int main()
{
  int pipefd[2], r;
  char c, buff[8];
  if (pipe(pipefd) < 0) {
    perror("Error en crear la pipe");
    exit(1);
  }
  if ((r = write(pipefd[1], "hola!\n", 6)) < 0) {
    perror("Error en escriure a la pipe");
    exit(2);
  }
  if ((r = read(pipefd[0], buff, r)) < 0) {
    perror("Error en llegir de la pipe");
    exit(3);
  }
  write (1, buff, r);
  exit (0) ;
}
```

```
// El pare escriu a la pipe, el fill llegeix de la pipe i escriu per stdout
//-------------------------------------------------------------------------

int main () {
  int pipefd[2] , pid;
  char c;
  char buff[]= "Codificacio Cesar de 3.\n Text pla : hola !\nText xifrat : ";
  write(1, buff, strlen(buff));
  if (pipe(pipefd) < 0) {
    perror(" Error en crear la pipe");
    exit (1) ;
  }
  if (write(pipefd[1], "hola!", 5) < 0) {
    perror("Error en escriure a la pipe") ;
    exit(2);
  }

  pid = fork() ;
  if (pid < 0) {
    perror("Error de fork");
    exit(1);
  }
  if (pid == 0) {
    close(pipefd[1]); /* important !!!! */
    while (read(pipefd[0], &c, 1) > 0) {
      c +=3;
      if (write (1, &c, 1) < 0) {
        perror("Error en escriure a la stdout");
        exit(2);
      }
    }
    close(pipefd[0]);
    write (1, "\n", 1) ;
    exit(0);
  }
  close(pipefd[1]); /* important !!!! */
  close(pipefd[0]);
  wait(NULL);
  exit(0);
}
```

Funcions

```
void func(void) {
    // ...
}
```

Headers

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

#include "lo_que_sea.h"
```

Constants

```
#define PI 3.14

#define CONSTANT value
```

Parámetros

```
// Si ejecutamos ./program

int main(int argc, char *argv[]) {
    // argc -> 1
    // argv[0] -> "./program"
    return 0;
}
```

```
// Si ejecutamos ./program a b c

int main(int argc, char *argv[]) {
    // argc -> 4
    // argv[0] -> "./program"
    // argv[1] -> "a"
    // argv[2] -> "b"
    // argv[3] -> "c"
    return 0;
}
```

Variables globales

Las podemos usar en cualquier función del programa

```
int a, b;
char c;
int vector[CONSTANT];
char buff[CONSTANT];
```

Asignación

```
int x = 4;
char a = 'A';
int *p = &x;
int v[10];
v[0] = 1;
v[1] = 2;
v[9] = 19;
```

Comparaciones

- Igual (x == y)
- No igual (x != y)
- Mayor (x > y)
- Mayor o igual (x >= y)

Variables

- Simples: `type variable_name;`
- Vectores: `type vector_name[vector_size];`

El tamaño de un vector puede ser un número, una constante definida anteriormente o se puede dejar en blanco así: `int vec[] = {1,2,3};` Adoptará el tamaño de aquello a lo que se asigne.

Tipos

- **int** (entero)
- **char** (carácter)
- **punteros**
  - int *
  - char *
  - void *
  - ...
- No existen **bool**, usamos ints. **1** -> **true**, **0** -> **false**
- Los **string** son un **char*** terminado con el carácter invisible **'\0'**

Strings

- Vectores de chars acabados en **'\0'**
- **'\n'** -> Salto de línea
- **strlen(s)** -> devuelve el tamaño de s (un string)
- **sprintf(s, "Hello World %d", 3)** -> Guarda el string "Hello world 3" en la variable s. Puede usar diversos formatos
  - **%d** -> int
  - **%c** -> char
  - **%s** -> string
  - ...
- **strcmp(s1, s2)** -> Compara las strings devuelve **0** si son iguales, **>0** si s2 es mayor, **<0** si s2 es menor.

int -> string

```
int x = 1000;
char buffer[64];
sprintf(buffer, "%d", x);
write(1, buffer, strlen(buff));

// OUT: 1000
```

string -> int

```
char *s = "314";
int x = atoi(s);

// x = 314;
```

```c
/* Esperar fills */
while(waitpid (-1, NULL, 0) > 0) // waitpid (-1, NULL, 0) == wait()
//                                  [1]   [2]   [3]
// [1] -> Normalment -1, no preguntis
// [2] -> NULL o &res (per saber com ha mort el fill - mirar exemple de sota)
// [3] -> 0 o WNOHANG (comprova si hi ha algun fill zombie, if(fills zombie): es 'recull' el fill i retorna l'estat de sortida)
//                                                                                                  else(si no hi ha fills 'acabats') retorna 0
//                                                                                                  else retorna 1
// El waitpid(-1, NULL, WNOHANG) és útil per saber si ha mort algun fill recentment sense a que morin

// +info [WNOHANG](https://stackoverflow.com/questions/33508997/waitpid-wnohang-wuntraced-how-do-i-use-these#34845669)


/* Motiu pel qual ha mort un fill */
int res
pid = waitpid (-1, &res, 0); // Ara avaluem si hi ha hagut error o no
if (WIFEXITED(res) && WEXITSTATUS(res) != 0) // WIFEXITED (res) returns true si el fill ha acabat normalment (exit())
{...error...}                                // WEXITSTATUS(res) retorna el valor d'exit (exit(valor))
                                             // https://www.ibm.com/docs/en/zos/2.4.0?topic=functions-waitpid-wait-specific-child-process-end
                                                                                    // Per poder cridar WIFEXITSTATUS(res) és necessari haver cridat abans WIFEXITED(res)
                                                                                    i ha d'haver retornat un numero diferent de 0

// _____________ Senyals ______________

struct sigaction sa;
sigset_t mask;

/* Evitar que ens arribi una senyal */
sigemptyset (&mask);                    // Buidem la mascara
sigaddset (&mask, SIGALRM);             // Afegim les senyals que volguem bloquejar
sigprocmask (SIG_BLOCK, &mask, NULL);   // Bloquejem les senyals (també serveix per desbloquejar-les)


/* Que només pugui arribi una senyal */

sigfillset (&mask);                   // Emplenem la mascara
sigdelset (&mask, SIGINT);            // En aquest cas només podria arribar SIGINT

/* Reprogramació d'una senyal */
sa.sa_handler = &my_function;      // Pots indicar la teva funció, SIG_IGN per, no fer res o SIG_DFL (funcio per defecte)
sa.sa_flags = SA_RESTART;          // 0 (default) || SA_RESETHAND (restaurarà tractament antic) || SA_RESTART (reinicia la call que l'ha bloquejat)
sigfillset (&sa.sa_mask);          // Afegim els signals que hem modificat a la màscara

if (sigaction (SIGALRM, &sa, NULL) < 0) // Modifiquem el tractament de SIGALRM p.e.
  perror("Missatge d'error");                 // Tractament d'errors
  exit(1);

/* Esperar fins una senyal en concret */

sigfillset (&mask);
sigdelset (&mask, SIGALRM);
sigsuspend (&mask);             // Ens esperem aquí

/* Enviar una senyal a un pid */

kill (pid, SIGINT); // Per fer-ho des de la terminal: #kill -SIGUSR1 pid

/* __________ Taula de senyañs i tractaments _________
 _____________________________________________________________
|___Nom___|__Acció per defecte_|___________Esdeveniment____________|
| SIGCHLD | IGNORAR            | Un fill ha acabat o l'han matat   |
| SIGCONT |                    | Continua si està parat (NB)       |
| SIGSTOP | STOP               | Para un procés (Ctrl+Z)           |
| SIGINT  | TERMINAR (Acabar)  | Interrupció desde teclat (Ctrl+C) |
| SIGALRM | TERMINAR (Acabar)  | El temporitzador ha acabat        |
| SIGKILL | TERMINAR (Acabar)  | Acaba un procés (NB)              |
| SIGSEGV | CORE               | Referència invàlida a memòria     |
| SIGUSR1 | TERMINAR (Acabar)  | Definit per l'usuari              |
| SIGUSR2 | TERMINAR (Acabar)  | Definit per l'usuari              |
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
NB = No és bloquejable
```

```
// ___________ Què s'hereda quan? ______________________________

FORK ----> El fill:
                ----> Hereda:
                                - Copia de les variables
                                - Taula d'accions dels signals
                                - Màscara de signals bloquejats

                ----> No hereda:
                                - Signals pendents ni alarmes
                                - PID

EXECLP ----> Muta:
                ----> Hereda:
                                - Senyals pendents
                                - Màscara de signals bloquejats
                                - PID

                ----> No hereda:
                                - Taula d'accions dels signals ni alarmes
                                - Les variables (canvien de dirrecció)
```

Seguridad

- La seguridad ha de considerarse a cuatro niveles:
- Físico
  - Las máquinas y los terminales de acceso deben encontrarse en un habitaciones/edificios seguros.
- Humano
  - Es importante controlar a quien se concede el acceso a los sistemas y concienciar a los usuarios de no facilitar que otras personas puedan acceder a sus cuentas de usuario
- Sistema Operativo
  - Evitar que un proceso(s) sature el sistema
  - Asegurar que determinados servicios están siempre funcionando
  - Asegurar que determinados puertos de acceso no están operativos
  - Controlar que los procesos no puedan acceder fuera de su propio espacio de direcciones
- Red
  - La mayoría de datos hoy en día se mueven por la red. Este componente de los sistemas es normalmente el más atacado.

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

void error_y_exit(char *s, int error) {
    perror(s);
    exit(error);
}

int main(int argc, char* argv[]) {
    for (int i = 1; i < argc; ++i) {
        int ret = fork();
        char s[50];
        switch (ret) {
            case 0:
                sprintf(s, "Soy el proceso HIJO: %d de %s\n", getpid(), argv[i]);
                write(1, s, strlen(s));
                exit(0);
                break;
            case -1:
                sprintf(s,"Ha fallado el fork del proceso: %d\n", getpid());
                error_y_exit(s, 1);
                break;
            default:
                waitpid(-1, NULL, 0);
                break;
        }
    }
}
```

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void error_y_exit(char *s, int error) {
    perror(s);
    exit(error);
}

/* Ejecuta el comando ps -u username mediante la llamada al sistema execlp */
/* Devuelve: el código de error en el caso de que no se haya podido mutar */
void muta_a_PS(char *username) {
    execlp("ps", "ps", "-u", username, (char*)NULL);
    error_y_exit("Ha fallado la mutación al ps", 1);
}

int main(int argc, char* argv[]) {
    if (argc == 2) {
        int ret = fork();
        char s[50];
        switch (ret) {
            case 0:
                sprintf(s, "Soy el proceso HIJO: %d de %s\n", getpid(), argv[1]);
                write(1, s, strlen(s));
                muta_a_PS(argv[1]);
                while(1);
                break;
            case -1:
                sprintf(s,"Ha fallado el fork del proceso: %d\n", getpid());
                error_y_exit(s, 1);
                break;
            default:
                sprintf(s, "Soy el proceso PADRE: %d\n", getpid());
                write(1, s, strlen(s));
                while(1);
                break;
```

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

void error_y_exit(char *s, int error) {
    perror(s);
    exit(error);
}

int main(int argc, char* argv[]) {
    for (int i = 0; i < 4; ++i) {
        int pid = fork();
        switch (pid) {
            char s[50];
            case 0:
                if (i == 0) execlp("./listaParametros", "a", "b", (char*)0);
                if (i == 1) execlp("./listaParametros", (char*)0);
                if (i == 2) execlp("./listaParametros", "25", "4", (char*)0);
                if (i == 3) execlp("./listaParametros", "1024", "hola", "adios", (char*)0);
                break;
            case -1:
                sprintf(s,"Ha fallado el fork del proceso: %d\n", getpid());
                error_y_exit(s, 1);
                break;
            default:
                break;
        }
    }
    while (wait(NULL) > 0);
}
```

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

void error_y_exit(char *s, int error) {
    perror(s);
    exit(error);
}

int main(int argc, char* argv[]) {
    for (int i = 1; i < argc; ++i) {
        int ret = fork();
        char s[50];
        switch (ret) {
            case 0:
                sprintf(s, "Soy el proceso HIJO: %d de %s\n", getpid(), argv[i]);
                write(1, s, strlen(s));
                exit(0);
                break;
            case -1:
                sprintf(s,"Ha fallado el fork del proceso: %d\n", getpid());
                error_y_exit(s, 1);
                break;
            default:
                sprintf(s, "Soy el proceso PADRE: %d\n", getpid());
                write(1, s, strlen(s));
                break;
        }
    }
    while(waitpid(-1,NULL,0) > 0);
    char c;
    read(1,&c,sizeof(char));
```

Constantes

- const tipo_dato nombre_variable=valor;
- #define nombre_variable valor

Decompose_time

```
...clude <iostream>
using namespace std;
// This program reads a natural number that represents an amount
// of time in seconds and writes the decomposition in hours,
// minutes and seconds
int main() {
int N;
cin >> N;
int h = N / 3600;
int m = (N % 3600) / 60;
int s = N % 60;
cout << h << " hours, " << m << " minutes and "
<< s << " seconds" << endl;
}
```

```
#include <stdio.h>
#include <string.h>
#define STDOUT 1
// This program receives a natural number that represents an amount
// of time in seconds and writes the decomposition in hours,
// minutes and seconds
int main(int argc, char *argv[]) {
int N;
N=atoi(argv[1]);
int h = N / 3600;
int m = (N % 3600) / 60;
int s = N % 60;
char buff[128];
sprintf(buff,"%d hours, %d minutes and %d seconds\n",h,m,s);
write(STDOUT,buff,strlen(buff));
}
```

Primer programa: suma

```
#include <iostream>
using namespace std;
// This program reads two numbers and
// writes their sum
int main() {
  int x, y;
  cin >> x >> y;
  int s = x + y;
  cout << s << endl;
}
```

```
#include <stdio.h>
#include <string.h>
#define STDOUT 1
//This program receives two numbers //and writes their sum
int main(int argc,char *argv[])
{
      int x,y;
      char buff[128];
      x=atoi(argv[1]);
      y=atoi(argv[2]);
      int s=x+y;
      sprintf(buff,"%d\n",s);
      write(STDOUT,buff,strlen(buff));
}
```

```
// Pre: A is a non-empty vector
// Post: returns the min value of the vector
int minimum(const vector<int>& A) {
int n = A.size();
int m = A[0]; // visits A[0]
// loop to visit A[1..n-1]
for (int i = 1; i < n; ++i) {
if (A[i] < m) m = A[i];
}
return m;
```

```
// Pre: A is a non-empty vector
// Post: returns the min value of the vector
int minimum(int *A, int size_A) {
int n = size_A
int m = A[0]; // visits A[0]
// loop to visit A[1..n-1]
for (int i = 1; i < n; ++i) {
if (A[i] < m) m = A[i];
}
return m;
}
```

- int x,i,j;
  - Arithmetic operators: +, -, *,/, %
- char a,b,c;
- bool A;
- string

- int x,i,j;
  - Arithmetic operators: +, -, *,/, %
- char a,b,c;
- ~~bool A;~~ // no existe
- ~~String~~ // No existe

No se pueden aplicar operadores básicos de string en C, hay que usar:
- strlen :para calcular la longitud "usada" de un string, (es diferente del tamaño)
- strcmp: para comparar dos strings

- char(i), int('a')ç
- Visibilidad
- Vectores
  - vector<type> name(n);
  - vector<int> S(n);
  - int x=S[0];

- (char) i, (int)'a'
- Visibilidad (igual)
- Vectores
  - tipo name[n];
  - int S[n];
  - int x=S[0];

No existe vector.h en C, sólo hay operaciones básicas.
Para conocer el tamaño en BYTES de cualquier variable tenemos la función sizeof

```c
void error_y_exit(char *msg,int exit_status)
{
    perror(msg);
    exit(exit_status);
}

/* ESTA VARIABLE SE ACCEDE DESDE LA FUNCION DE ATENCION AL SIGNAL Y DESDE EL MAIN */
int segundos=0;
/* FUNCION DE ATENCION AL SIGNAL SIGALRM */
void funcion_alarma(int s)
{
    if (s == SIGALRM) segundos=segundos+10;
    else {
        char buff[256];
        sprintf(buff, "ALARMA pid=%d: %d segundos\n",getpid(),segundos);
        write(1, buff, strlen(buff));
    }
}
int main (int argc,char * argv[])
{
    struct sigaction sa;
    sigset_t mask;

    sigemptyset(&mask);
    sigaddset(&mask, SIGALRM);
    sigaddset(&mask, SIGUSR1);
    sigprocmask(SIG_BLOCK,&mask, NULL);

    /* REPROGRAMAMOS EL SIGNAL SIGALRM */
    sa.sa_handler = &funcion_alarma;
    sa.sa_flags = SA_RESTART;
    sigfillset(&sa.sa_mask);

    if (sigaction(SIGALRM, &sa, NULL) < 0 || sigaction(SIGUSR1, &sa, NULL) < 0) error_y_exit("sigaction", 1);

    while (segundos<100)
    {
    alarm(10); »/* Programamos la alarma para dentro de 10 segundos */
    »     » /* Nos bloqueamos a esperar que nos llegue un evento */
        sigfillset(&mask);
        sigdelset(&mask, SIGALRM);
        sigdelset(&mask, SIGINT);
        sigdelset(&mask, SIGUSR1);
        sigsuspend(&mask);
    }
    exit(1);
}
```

```
all: ejemplo_alarm2 bucleInfinito ejemplo_alarm3 ejemplo_signal ejemplo_signal2 eventos eventos2

ejemplo_alarm2: ejemplo_alarm2.c
»       gcc -o ejemplo_alarm2 ejemplo_alarm2.c
bucleInfinito: bucleInfinito.c
»       gcc -o bucleInfinito bucleInfinito.c
ejemplo_alarm3: ejemplo_alarm3.c
»       gcc -o ejemplo_alarm3 ejemplo_alarm3.c
ejemplo_signal: ejemplo_signal.c
»       gcc -o ejemplo_signal ejemplo_signal.c
ejemplo_signal2: ejemplo_signal2.c
»       gcc -o ejemplo_signal2 ejemplo_signal2.c
eventos: eventos.c
»       gcc -o eventos eventos.c
eventos2: eventos2.c
»       gcc -o eventos2 eventos2.c
signal_perdido2: signal_perdido2.c
»       gcc -o signal_perdido2 signal_perdido2.c
»
clean:
»       rm ejemplo_alarm2
»       rm bucleInfinito
»       rm ejemplo_alarm3
»       rm ejemplo_signal
»       rm ejemplo_signal2
»       rm eventos
»       rm eventos2
»       rm signal_perdido2
```

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>

void error_y_exit(char *msg,int exit_status)
{
    perror(msg);
    exit(exit_status);
}

int segundos=0;
void funcion_alarma(int signal)
{
»   char buff[256];
»   segundos=segundos+10;
»   sprintf(buff,"ALARMA pid=%d: %d segundos\n",getpid(),segundos);
»   write(1, buff, strlen(buff) );
}

int main (int argc,char * argv[])
{
    struct sigaction sa;
    sigset_t mask;

    /* EVITAMOS QUE NOS LLEGUE EL SIGALRM FUERA DEL SIGSUSPEND */
    sigemptyset(&mask);
    sigaddset(&mask, SIGALRM);
    sigprocmask(SIG_BLOCK,&mask, NULL);

int pid;
while (segundos<100) {
»   alarm(10);
    pid = fork();
    if (pid < 0) error_y_exit("fork", 1);
    else if (pid == 0) {
        /*
»   »   //REPROGRAMAMOS EL SIGNAL SIGALRM
        sa.sa_handler = &funcion_alarma;
        sa.sa_flags = SA_RESTART;
        sigfillset(&sa.sa_mask);
        if (sigaction(SIGALRM, &sa, NULL) < 0) error_y_exit("sigaction", 1);
»   »   */
»   »   execlp("./bucleInfinito", "./bucleInfinito", NULL);
    }

    sigfillset(&mask);
    sigdelset(&mask, SIGALRM);
    sigdelset(&mask, SIGINT);
    sigsuspend(&mask);
}
exit(1);
```

```c
int contador = 0;
int hijos = 0;

void error_y_exit(char* msg, int exit_status)
{
    perror(msg);
    exit(exit_status);
}

void trata_hijo(int s) {
    int pid, exit_code;
    char buff[256];
    while ((pid = waitpid(-1, &exit_code, WNOHANG)) > 0) {
        if (WIFEXITED(exit_code)) {
            int statcode = WEXITSTATUS(exit_code);
            sprintf(buff,"Termina el proceso %d com exit code %d\n",pid, statcode);
        }
        else {
            int signcode = WTERMSIG(exit_code);
            sprintf(buff,"Han matado al proceso %d antes de acabar alarm por el signal %d\n",pid, signcode);
        }
        write(1, buff, strlen(buff));
        hijos--;
        ++contador;
    }
}


void trata_alarma(int s)
{
}
```

```c
int main(int argc, char* argv[])
{
    int pid, res;
    char buff[256];
    struct sigaction sa;
    sigset_t mask;
    int pid_vec[10];

    /* Evitamos recibir el SIGALRM fuera del sigsuspend */

    sigemptyset(&mask);
    sigaddset(&mask, SIGALRM);
    sigprocmask(SIG_BLOCK, &mask, NULL);

    for (hijos = 0; hijos < 10; hijos++) {
        sprintf(buff, "Creando el hijo numero %d\n", hijos);
        write(1, buff, strlen(buff));

        pid = fork();
        if (pid == 0) /* Esta linea la ejecutan tanto el padre como el hijo */
        {
            sa.sa_handler = &trata_alarma;
            sa.sa_flags = SA_RESTART;
            sigfillset(&sa.sa_mask);
            if (sigaction(SIGALRM, &sa, NULL) < 0)
                error_y_exit("sigaction", 1);

            /* Escribe aqui el codigo del proceso hijo */
            sprintf(buff, "Hola, soy %d\n", getpid());
            write(1, buff, strlen(buff));

            alarm(2);
            sigfillset(&mask);
            sigdelset(&mask, SIGALRM);
            sigdelset(&mask, SIGINT);
            sigsuspend(&mask);

            /* Termina su ejecución */
            exit(0);
        } else if (pid < 0) {
            /* Se ha producido un error */
            error_y_exit("Error en el fork", 1);
        } else pid_vec[hijos] = pid;
    }
    /* Esperamos que acaben los hijos */
    sa.sa_handler = &trata_hijo;
    sa.sa_flags = SA_RESTART;
    sigaction(SIGCHLD, &sa, NULL);
    for (int i = 0; i < 10; ++i) kill(pid_vec[i], SIGUSR1);
    while (hijos > 0);
    sprintf(buff, "Valor del contador %d\n", contador);
    write(1, buff, strlen(buff));
    return 0;
```

```c
int contador = 0;


void trata(int s)
{
    if (s == SIGALRM) contador += 1;
    if (s == SIGUSR1) contador = 0;
    else if (s == SIGUSR2) {
        char buf[80];
        sprintf(buf, "Valor contador: %d\n", contador);
        write(1, buf, strlen(buf));
    }

}



int main(int argc,char *argv[])
{
    sigset_t mask;

    sigemptyset(&mask);
    sigaddset(&mask, SIGALRM);
    sigaddset(&mask, SIGUSR1);
    sigaddset(&mask, SIGUSR2);
    sigprocmask(SIG_BLOCK, &mask, NULL);


    struct sigaction sa;
    sa.sa_handler = &trata;
    sa.sa_flags = SA_RESTART | SA_RESETHAND;
    sigfillset(&sa.sa_mask);


    sigaction(SIGALRM, &sa, NULL);
    sigaction(SIGUSR1, &sa, NULL);
    sigaction(SIGUSR2, &sa, NULL);
    while (1) {
    alarm(1);
    sigfillset(&mask);
    sigdelset(&mask, SIGALRM);
    sigdelset(&mask, SIGUSR1);
    sigdelset(&mask, SIGUSR2);
    sigdelset(&mask, SIGINT);
    sigsuspend(&mask);
}
return 0;
```

```c
void error_y_exit(char *msg,int exit_status)
{
    perror(msg);
    exit(exit_status);
}

void trat_sigusr1(int s) {
    char buf[80];

    sprintf(buf, "Hijo: SIGUSR1 recibido \n");
    write(1, buf,strlen(buf));
}

void trat_sigalrm(int s) {
    char buf[80];

    sprintf(buf, "Padre: voy a mandar SIGUSR1 \n");
    write(1, buf,strlen(buf));
}

void main(int argc, char *argv[]) {
    int i,pid_h;
    char buf [80];
    int delay;
    struct sigaction sa, sa2;
    sigset_t mask;

    if (argc !=2) {
»   sprintf(buf, "Usage: %s delayParent \n delayParent: 0|1\n",argv[0]);
»   write(2,buf,strlen(buf));
»   exit(1);
    }

    delay = atoi(argv[1]);
    //signal (SIGUSR1, trat_sigusr1);
    sa.sa_handler = &trat_sigusr1;
    sa.sa_flags = SA_RESTART;
    sigfillset(&sa.sa_mask);
    if (sigaction(SIGUSR1, &sa, NULL) < 0) error_y_exit("sigaction", 1);
```

```c
//signal (SIGALRM, trat_sigalrm);
sa2.sa_handler = &trat_sigalrm;
sa2.sa_flags = SA_RESTART;
sigfillset(&sa2.sa_mask);
if (sigaction(SIGALRM, &sa2, NULL) < 0) error_y_exit("si

pid_h = fork ();

if (pid_h == 0) {
sigset_t delay_mask;
sigemptyset(&delay_mask);
sigaddset(&delay_mask, SIGUSR1);
sigprocmask(SIG_BLOCK, &delay_mask, NULL);

sprintf(buf, "Hijo entrando al pause\n");
write(1,buf,strlen(buf));
//pause();

sigfillset(&mask);
    sigdelset(&mask, SIGUSR1);
    sigdelset(&mask, SIGINT);
    sigsuspend(&mask);

sigprocmask(SIG_UNBLOCK, &delay_mask, NULL);
sprintf(buf, "Hijo sale del pause\n");
write(1,buf,strlen(buf));
} else {
if (delay) {
»   alarm(5);
»   //pause();
»   sigfillset(&mask);
      » sigdelset(&mask, SIGALRM);
      »   sigdelset(&mask, SIGINT);
      »   sigsuspend(&mask);
}
sprintf(buf, "Padre manda signal SIGUSR1\n");
write(1,buf,strlen(buf));
if (kill (pid_h, SIGUSR1) < 0) error_y_exit("kill", 1);
waitpid(-1, NULL, 0);
sprintf(buf, "Padre sale del waitpid\n");
write(1,buf,strlen(buf));
```

SIGNALS

sigset_t → Màscara de signals

sigfillset(&set) → Afegeix totes les signals a set

sigemptyset(&set) → Treu totes les signals de set

sigaddset(&set, SIGUSR1) → Afegeix SIGUSR1 a set

sigdelset(&set, SIGUSR1) → Treu SIGUSR1 de set

sigprocmask(how, &set, NULL);

- SIG_BLOCK → Bloqueja les senyals de set
- SIG_UNBLOCK → Desbloqueja "
- SIG_SETMASK → Bloqueja només les senyals de set

sigaction(signal, &sa, NULL);

↳ Canvia el tractament de signal

struct sigaction sa;

- sa_handler → funció que s'executa al rebre el signal
- sa_mask → màscara de senyals bloquejades mentre s'executa sa_handler
- sa_flags → opcions extra (SA_RESTART)

sigsuspend(&set);

↳ Suspen el procés fins que arribi un signal que NO estigui a set

execlp

↳ Perdem:
- Signal handlers (sigaction)

Mantenim:
- Signals bloquejades (sigprocmask)
- Alarmes pendents (alarm)

sa_handler

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define DEFAULT_ITERATIONS 3
#define REGION_SIZE    4096

char globalArray[REGION_SIZE];

int
main (int argc, char *argv[])
{
  char buff[80];
  int niterations = DEFAULT_ITERATIONS;
  int i;
  char localArray[REGION_SIZE];
  char *p;

  if (argc > 2)
    {
      sprintf (buff, "Usage: mem1 [numMallocs]\n");
      write(2,buff,strlen(buff));
      exit (1);
    }

  if (argc == 2)
    {
      niterations = atoi (argv[1]);
    }

  sprintf (buff, "Addresses:\n");
  write(1,buff,strlen(buff));
  sprintf (buff, "\tglobalArray: %p\n", globalArray);
  write(1,buff,strlen(buff));
  sprintf (buff, "\tlocalArray: %p\n", localArray);
  write(1,buff,strlen(buff));
  sprintf (buff, "\tp: %p\n", &p);
  write(1,buff,strlen(buff));


  for (i = 0; i < niterations; i++)
    {
      p = malloc (REGION_SIZE);
      sprintf (buff, "\tp: %p, region %d: %p\n", &p, i, p);
      write(1,buff,strlen(buff));
      free(p);
    }

  while (1);
```

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

#define REGION_SIZE    4096

int *p;

void func(int signal)
{
    char buff[256];
        sprintf( buff, "\tProgram code -- p address: %p, p value: %p, p heap value: %p\n", &p, p, sbrk(0));
    write(1, buff, strlen(buff));
    exit(0);
}

int main(int argc, char *argv[])
{
    struct sigaction sa;
    sa.sa_handler = &func;
    sa.sa_flags = SA_RESTART;
    sigfillset(&sa.sa_mask);
    sigaction(SIGSEGV, &sa, NULL);

    int i = 0;
    char buff[256];

    sprintf( buff, "Addresses:\n");
    write(1, buff, strlen(buff));
    sprintf( buff, "\tp: %p\n", &p);
    write(1, buff, strlen(buff));

    p = malloc(sizeof(int));

    if (p == NULL) {
        sprintf(buff, "ERROR en el malloc\n");
        write(2,buff,strlen(buff));
    }

    while (1) {
        *p = i;
        sprintf( buff, "\tProgram code -- p address: %p, p value: %p, *p: %d\n",
                &p, p, *p);
        write(1, buff, strlen(buff));
        p++;
        i++;
    }

}
```

```c
for (i = 0; i < niterations; i++)
  {
    p = sbrk(REGION_SIZE);
    sprintf (buff, "\tp: %p, region %d: %p\n", &p, i, p);
    write(1,buff,strlen(buff));
  }
```

```c
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

int main(int argc, char **argv){
    char buff[80];
    int fd = open("./pipe",O_WRONLY|O_NONBLOCK);
    if ((fd < 0) && (errno == ENXIO)) perror("Esperando lector");
    fd = open("./pipe",O_WRONLY);
    int ret;
    while((ret = read(fd,buff,ret)) > 0){
        write(fd,buff,ret);
    }
}
```

```c
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <string.h>

void func(int s) {
    char buff[128] = "recibido\n";
    write(1, &buff, strlen(buff));
}

int main(){
    struct sigaction sa;

    sa.sa_handler = &func;
    //sa.sa_flags = SA_RESTART;
    sigfillset(&sa.sa_mask);
    sigaction(SIGINT, &sa, NULL);


    char buf[256];
    char c;

    int ret = read(0, &c, sizeof(char));


    if (ret >= 0) {
        sprintf(buf, "read correcto\n");
        write(1,buf,strlen(buf));
    } else {
        if (errno == EINTR) {
            sprintf(buf, "read interrumpido por signal\n");
            write(1,buf,strlen(buf));
        } else {
            sprintf(buf, "read con error\n");
            write(1,buf,strlen(buf));
        }
    }
```

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

main (int argc, char *argv[])
{
    char buff[80];
    int p[2];
    pipe(p);
    int pid = fork();
    if (pid == 0){
        //close(0);
        dup2(p[0],0);
        close(p[0]);
        close(p[1]);
        execlp("cat","cat",(char*)0);
    }
    else if(pid > 0){
        close(p[0]);
        sprintf(buff,"Inicio\n");
        write(p[1],buff,strlen(buff));
        close(p[1]);
        waitpid(-1,NULL,0);
        sprintf(buff,"Fin\n");
        write(1,buff,strlen(buff));
    }
}
```

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

void halt(char c[]) {
    perror(c);
    exit(0);
}

void Usage(void) {
    char buff[] = "Usage: ,/append [arg0]";
    write(1, buff, strlen(buff));
}

int main(int argc, char **argv) {
    if (argc < 2) Usage();
    else {
        int fd, ini, fi;
        char buff[128];
        if ((fd = open(argv[1], O_RDWR)) < 0) halt("open");

        ini = 0;
        if ((fi = lseek(fd, 0, SEEK_END)) < 0) halt("lseek");
        //sprintf(buff, "%d\n", fi);
        //write(1, buff, strlen(buff));
        char c;
        lseek(fd, ini, SEEK_SET);
        while ((read(fd, &c, sizeof(char) > 0)
            && ini < fi)) {
            lseek(fd, 0, SEEK_END);
            write(fd, &c, sizeof(char));
            ++ini;
            lseek(fd, ini, SEEK_SET);
        }
    }
}
```

```c
int main(int argc, char **argv) {
    if (argc < 2) Usage();
    else {
        int fd1, fd2, ini;
        char buff[128];
        if ((fd1 = open(argv[1], O_RDONLY)) < 0) halt("open");
        if (lseek(fd1, 0, SEEK_END) < 0) halt("lseek");

        if ((fd2 = creat("out", 0600)) < 0) halt("creat");
```

```c
main(){
char read_buffer[256]="";
char *buf="fin ejecución\n";
char buffer[1024];
int ret;
    // USO
    sprintf(buffer,"................................................\n");
    write(2,buffer,strlen(buffer));
    sprintf(buffer,"Este programa escribe por la salida std todo lo que lee de la entrada std. Si no has redirigido la salida, lo que escribas en el teclado saldra por la pantalla\n");
    write(2,buffer,strlen(buffer));
    sprintf(buffer,"Para acabar CtrlD\n");
    write(2,buffer,strlen(buffer));
    sprintf(buffer,"................................................\n");
    write(2,buffer,strlen(buffer));


    // Leemos del canal 0 (entrada std), 1 bye
    ret=read(0,&read_buffer,sizeof(read_buffer));
    // Cuando el read devuelve 0 significa que se ha acabado la
    // entrada de datos --> acabamos el bucle de lectura
    while(ret>0){
        // Escribimos en el canal 1 (salida std) 1 byte
        write(1,&read_buffer,sizeof(read_buffer));
        ret=read(0,&read_buffer,sizeof(read_buffer));
    }
    write(1,buf,strlen(buf));
```

```c
int main(int argc, char **argv) {
    if (argc < 2) Usage();
    else {
        int pos = atoi(argv[2]);
        char c;
        int fd1, ini;
        char buff[128];
        if ((fd1 = open(argv[1], O_RDWR)) < 0) halt("open");
        if (lseek(fd1, 0, SEEK_END) < 0) halt("lseek");

        //sprintf(buff, "%d\n", fi);
        //write(1, buff, strlen(buff));
```

```c
lseek(fd1, -1, SEEK_END);
ini = -2;
while (read(fd1, &c, sizeof(char) > 0)) {
    write(fd1, &c, sizeof(char));
    if (lseek(fd1, ini, SEEK_END) < 0) return;
    --ini;
    if (lseek(fd1, 0, SEEK_CUR) == pos) {
        write(fd1, 'X', sizeof(char));
        return;
    }
```