

---

---

---

---

---



# Ejercicios : T1

3. Explique como trabajan los punteros

```
#include <stdio.h>
void main(void) {
    int edad = 20;
    int *pEdad = &edad;
    printf("%d\n", edad); //20
    printf("%p\n", &edad); //dirección de memoria
    printf("%p\n", pEdad); //dirección de memoria
    printf("%d\n", *pEdad); //20

    /* En la variable edad guardamos el valor 20, en la *pEdad guardamos
     la dirección donde se guardó edad, luego si imprimimos la dirección de edad
     esta guardada tanto en pEdad como en &edad. Si queremos el propio valor podemos verlo en edad
     o con el puntero que nos indica el contenido de la dirección de pEdad*/
```

## /\* Ejercicio paso por referencia y paso por valor \*/

```
#include <stdio.h>

/*hay que llamar a las funciones antes */
void funct1 (int a, int b);
void funct2 (int *pa, int *pb);

main(){
    int a = 2;
    int b = 3;
    printf("-----FUNC1-----\n");
    printf("numeros antes: a=%d, b=%d \n", a, b);
    funct1(a,b);
    printf("\n numeros despues: a=%d, b=%d ", a, b);

    printf("-----FUNC2-----\n");
    printf("numeros antes: a=%d, b=%d \n", a, b);
    funct2(&a,&b);
    printf("\n numeros despues: a=%d, b=%d ", a, b);

}

void funct1 (int a, int b){ //paso por valor (COPIA VALOR)
    a = a + 3;
    b = b + 3;
    printf( "Dentro de la f1: a=%d, b=%d", a, b);
}

void funct2 (int *pa, int *pb){ //paso por referencia (ALTERA VALOR)
    *pa = 1;
    *pb = 1;
    printf("Dentro de la f2: a= %d, b=%d", *pa, *pb);
}
```

5. Haz un programa que espera la introducción de una letra y lo imprime por pantalla.

```
#include <stdio.h>

void main(void) {
    char letra;
    printf("Introduzca una letra: ");
    letra = getchar();
    printf("\n su letra es: %c", putchar(letra));
}
```

6. Crea un programa que pida un nombre y un número al usuario (con scanf y 2 llamadas) y lo imprime.

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

void cogerInfo(char nombre[20], int *pnumero){ //hay q pasarle un puntero (paso por referencia)
    printf("Introduzca su nombre: \n");
    scanf("%s", nombre);
    printf("\n");
    printf("Introduzca su número de usuario: \n");
    scanf("%d", pnumero); //hay q pasarle directamente numero
}

void imprimirInfo(char nom2[20], int num2){
    printf("%s %d \n", nom2, num2);
}

void main(void){
    char nombre2[20];
    int numero2;
    cogerInfo(nombre2, &numero2);
    imprimirInfo(nombre2, numero2);
}
```

# Ejercicios : T2

```

1 /*Llamada al sistema*/
2
3 #include <stdio.h>
4 #include <fcntl.h>
5
6 char name[]="archivo.txt";
7 int main(){
8     int fd;
9     fd = creat(name,0666);
10    return 0;
11 }
12
13 /*
14 ¿Qué valor devuelve? ¿Por qué?
15 Devuelve un cero puesto que se crea el fileDescriptor correctamente y hace un return de 0
16
17 Revisa los permisos del archivo que se ha creado, ¿son los esperados?
18 No, por ello hemos utilizado el comando umask 0000 para cambiarlos*/
19

3 #include<stdio.h>
4 #include <fcntl.h>
5 #include <sys/types.h>
6 #include <sys/stat.h>
7 #include <string.h>
8 #include <unistd.h>
9
10
11 char name[] = "open_write.txt";
12 char buffer[1024];
13 int fileDescriptor1, fileDescriptor2;
14 int numbytes;
15
16 int main(){
17
18     //Open
19     fileDescriptor1 = open("open_write.txt", O_CREAT|O_RDWR, 0666);
20
21     //Write: escribir una frase
22     char frase[] = "Quiero imprimir esto!";
23     write(fileDescriptor1, frase, strlen(frase)); //Puede devolver -1(fallo) o 3 (éxito)
24
25     //Read
26     char c;
27     lseek(fileDescriptor1, 0, SEEK_SET); //para mover el puntero al inicio del fichero
28     while(read(fileDescriptor1, &c,1)>0){
29         printf("%c\n", c); //write(1, &c, 1 )
30         sleep(1);
31     }
32
33     /*Lseek:
34         off_t lseek(int fd, off_t offset, int whence);
35         --> offset:
36             --> directiva whence: SEEK_SET, SEEK_CUR, SEEK_END
37                 SEEK_SET: El puntero se coloca a offset bytes
38
39     */
40 }

```

2. Realiza un programa que cree un fichero con `open`, abre un `fd` en él con `write` y lee el contenido con `read`  
(+ retardo de 1 seg)

Ejercicio 3. Obtener un mensaje asociado a la variable `errno`.

- En este programa se invoca a la llamada al sistema `read` para leer un archivo con identificador de archivo igual a 20. Esta llamada al sistema va a fallar, ya que no se ha creado previamente un fichero al que se le haya asociado dicho descriptor. En consecuencia, la función de liberaría asociada a la llamada al sistema devolverá el valor -1 y colocará en la variable `errno` el identificador número del error cometido. ¿Cuál es el error que se muestra por pantalla?

```
1 /*Error errno*/
2 #include <errno.h>
3 #include <stdio.h>
4 #include <fcntl.h>
5 #include <unistd.h>
6 #include <string.h>

7 void main()
8 {
9     char buffer[100];
10    int iden=20;
11    if(read(iden,buffer,100)==-1);
12    printf("%d: %s\n",errno,strerror(errno));
13 }
```

ubuntu@ubuntu:~/Desktop/SistOp/Tema\$ gedit ej3cuaderno.c  
ubuntu@ubuntu:~/Desktop/SistOp/Tema\$ gcc ej3cuaderno.c -o ej3  
ubuntu@ubuntu:~/Desktop/SistOp/Tema\$ ./ej3

Ejercicio 4. Realiza un programa en C que simule el funcionamiento de copia de dos archivos.

- Al ejecutarlo se deben introducir los archivos origen y destino tal y como funciona el comando copy: ./cp origen.txt destino.txt

El archivo origen ya estará creado, mientras que el destino deberás crearlo con el nombre que haya introducido el usuario en la ejecución del programa.

Utiliza las llamadas al sistema **open**, **read** y **write** para ello.

\* Crear comando copy ./cp origen.txt destino.txt \*/

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

int fdOrigen, fdDestino;
char c;

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

    //read: origen.txt --while--> write: destino.txt
    fdOrigen = open(argv[1], O_CREAT|O_RDWR, 0666);
    fdDestino = open(argv[2], O_CREAT|O_RDWR, 0666);
    lseek(fdOrigen, 0, SEEK_SET);
    while(read(fdOrigen, &c, 1)>0){
        write(fdDestino, &c, 1 );
    }
}
```

# Ejercicios : T3

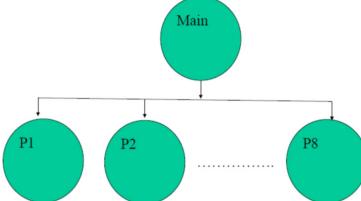
Ejercicio 2. Escriba un programa que cree un fichero de texto y un proceso hijo. Si se trata del proceso hijo escriba un carácter en el fichero (p. ej. '1') y si se trata del proceso padre escriba otro carácter (p. ej. '2'). Permita que se alternen al escribir en el fichero y escriba un total de 20 caracteres. -NUEVO

```
1 /*Creación de procesos: fork()*/
2
3 #include <stdio.h>
4 #include <string.h>
5 #include <sys/types.h>
6 #include <fcntl.h>
7 #include <sys/stat.h>
8 #include <string.h>
9 #include <unistd.h>
10 #include <wait.h>
11
12
13 void main(void){
14     int fd1;
15     char padre[7] = "padre\n";
16     char hijo[6] = "hijo\n";
17
18
19     fd1 = open("letras.txt", O_CREAT|O_RDWR, 0666);
20     pid_t pid = fork();
21     for(int i=0; i< 50; i++){
22         if(pid != 0){
23             write(fd1, padre, strlen(padre));
24             fflush(stdout); //podemos vaciar y volcar el buffer donde corresponde
25
26
27         }else{
28             write(fd1, hijo, strlen(hijo));
29             fflush(stdout);
30         }
31     }
32     while(wait(NULL)>0);
33 }
```

```
hijo
padre
hijo
hijo
hijo
hijo
hijo
hijo
hijo
padre
hijo
```

Ejercicio 3. Programa la siguiente jerarquía horizontal de procesos.

Ilustración 1: jerarquía horizontal de procesos



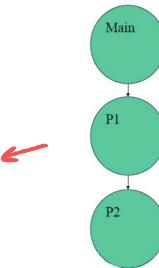
```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(void) {
    int i;
    pid_t pid;
    char letra = 'A';
    pid = fork();

    for (i = 0; i < 8; i++) {
        if(pid != 0) pid = fork(); //padre
        else{
            printf("Soy el hijo %d con PID %d y PPID %d \n", i + 1, getpid(), getppid()); //crea hijo
            exit(0); //matas hijo
        }
    }
    while(wait(NULL)>0); //cond para el padre, mientras uno abierto no cierres
}
```

Ejercicio 4. Programa la siguiente jerarquía vertical de procesos.

Ilustración 2: jerarquía vertical de procesos



# Ejercicios : T3

Ejercicio 6. Modifica el ejercicio 4 para que el hijo ejecute el comando ls y el nieto el comando ps.

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4
5
6 void main(){
7
8     //Abuelo
9     printf("Soy el proceso Abuelo con PID %d \n", getpid()); //esto lo imprime el padre
10
11    if(fork() == 0){ //Padre
12        execl("./bin/ls", "ls", "-l", (char*)0);
13        if(fork() == 0){ //hijo
14            execl("./bin/ps", "ps", "-l", (char*)0);
15        }
16    }
17
18    while(wait(NULL) > 0);
19
20 }
```

Ejercicio 7. Modifica el siguiente código para que el proceso termine al recibir la señal SIGINT (o al pulsar Ctrl + C en el teclado) y se muestre el mensaje: "Saliendo del programa al recibir la señal SIGINT".

```
1 /** poner $kill -sigint pid_del_proceso ***/
2 #include<sys/types.h>
3 #include<unistd.h>
4 #include<stdlib.h>
5 #include<signal.h>
6 #include <stdio.h>
7
8     # define VUELTAS 10000000000
9 void Salir (int sig){
10     switch (sig){
11         case(SIGTERM):
12             printf("Saliendo del programa al recibir la señal SIGTERM\n");
13             exit(EXIT_SUCCESS);
14             break;
15
16         case(SIGINT):
17             printf("Saliendo del programa al recibir la señal SIGINT\n");
18             exit(EXIT_SUCCESS);
19             break;
20     }
21 }
22
23 int main (void){
24     int i;
25     signal(SIGTERM, Salir);
26     signal(SIGINT, Salir);
27     for (i=0; i<VUELTAS; i++){
28         printf ("Fin del programa sin recibir la señal SIGTERM\n");
29         exit(EXIT_SUCCESS);
30     }
31 }
```

```
ubuntu@ubuntu:~/Desktop/SistOp/Tema3$ gcc ej7cuad.c -o ej7cuad
ubuntu@ubuntu:~/Desktop/SistOp/Tema3$ ./ej7cuad
^CSaliendo del programa al recibir la señal SIGINT
ubuntu@ubuntu:~/Desktop/SistOp/Tema3$
```

# Ejercicios : T4

Ejercicio 1. Modifique el código siguiente para que el proceso hijo ejecute el comando ps -l, al recibir una señal reservada para el usuario (SIGUSR1).

Nota: para poder ejecutar el comando ps -l antes del cambio de imagen, no se olvide de ejecutar el programa en segundo plano (./nombrePrograma &).

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <sys/types.h>
4 #include <unistd.h>
5 #include <sys/time.h>
6 #include <stdlib.h>
7 #include <signal.h> //IMPORTANTE
8
9 void tratamiento(int sig){
10     switch(sig){
11         case(SIGUSR1):
12             fflush(stdout);
13             execvp("ps", "ps", "-l", NULL);
14             break;
15     }
16 }
17
18 void main (void){
19     int pid=fork();
20     if (pid==0){
21         printf("Haz el primer ps\n");
22         fflush(stdout);
23         sleep(10);
24         printf("Ahora lanzar la señal\n");
25         fflush(stdout);
26         printf("%$d\n", "El PID del proceso hijo es ", getpid()); //se queda aqui
27         fflush(stdout);
28         printf("=====\n");
29         signal(SIGUSR1, tratamiento);
30         sleep(15);
31         printf(".....");
32         exit(0);
33     }
34
35 while(wait(NULL)>0);
36 }
```

Ejercicio 2. Cree un programa B que imprima "Soy B". Posteriormente, cree un programa A para que ejecute el programa B al reemplazar la imagen del proceso actual (exec).

Nota: tendrá que compilar primero el programa B y A para que la ejecución de A sea correcta.

```
1 #include <stdio.h>
2
3 void main(void) {
4     printf("Soy B\n");
5 }
```

A  
B

```
ubuntu@ubuntu:~/Desktop/SistOp/Tema4/ejercicios$ gcc ej2procesoA.c -o a
[2]+  Done                      gedit ej2procesoA.c
ubuntu@ubuntu:~/Desktop/SistOp/Tema4/ejercicios$ ./a
Soy B
ubuntu@ubuntu:~/Desktop/SistOp/Tema4/ejercicios$ gcc ej2procesoA.c -o a
ubuntu@ubuntu:~/Desktop/SistOp/Tema4/ejercicios$ ./a
Soy B
```

Ejercicio 3. Explique los parámetros de la función pthread\_create() y cuál es el resultado del programa.

```
1 #include <stddef.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 void * process(void * args){
7     printf("%s", (char *)args);
8     fflush(stdout);
9     pthread_exit(0);
10 }
11
12 int main(void){
13     pthread_t th_a,th_b; // Variable tipo thread
14
15     pthread_create(&th_a, NULL, process, "Hello");
16     pthread_create(&th_b, NULL, process, "World");
17     sleep(1);
18 }
19
20 /**
21  - &th_a: Es un puntero a la estructura pthread_t, pthread_create() lo llenará con información sobre el hilo que crea.
22  - NULL: puntero a la estructura pthread_attr_t
23  - Process: función que se ejecutará en el hilo
24  - "Hello" argumento que pasa a process()
25 */
26
```

Ejercicio 4. Realiza un programa que cree 8 hilos de ejecución y cada uno de ellos imprima: "Soy el hilo X".

```
1 #include<stddef.h>
2 #include<pthread.h>
3 #include<stdio.h>
4
5 void * process(void * args){
6     int id = (int) args; // Convertir el argumento a entero
7     printf("Soy el hilo %d\n", id); // Imprimir el mensaje
8     fflush(stdout);
9     pthread_exit(0);
10 }
11
12 int main(){
13     pthread_t th[8];
14     for (int i = 0; i<8; i++){
15         pthread_create(&th[i], NULL, process, (void *) i);
16     }
17     sleep(1);
18 }
```

```
ubuntu@ubuntu:~/Desktop/SistOp/Tema4/ejercicios$ ./ej4
Soy el hilo 0
Soy el hilo 6
Soy el hilo 5
Soy el hilo 7
Soy el hilo 4
Soy el hilo 3
Soy el hilo 1
Soy el hilo 2
```

# Ejercicios : T4

Ejercicio 5. Realiza un programa que cree 3 hilos de ejecución y cada uno de ellos imprima una letra 3 veces (A, B y C respectivamente).



```
1 #include<stddef.h>
2 #include<pthread.h>
3 #include<stdio.h>
4
5 void *process(void * args){
6     char letra = (char) args; // process --> devuelve char
7     printf("%d \n", letra); // Imprimir el mensaje
8     printf("%d \n", letra);
9     printf("%d \n", letra); //3 veces --> la letra
10    fflush(stdout);
11    pthread_exit(0);
12 }
13
14 int main(){
15
16     char* a = "A";
17     pthread_t th[3];
18     for (int i = 0; i<3; i++){
19         pthread_create(&th[i], NULL, process, (void *) a);
20         a++;
21     }
22     sleep(1);
23 }
```

Ejercicio 6. Explique el resultado del siguiente programa que utiliza hilos de ejecución.



```
1 #include <stddef.h>
2 #include<pthread.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 void * process(void * args){
7     int num = *(int *)args; //coge el num q aparece en el proceso
8     printf("%d", num); //imprime el numero
9     pthread_exit(0); //termina el hilo
10 }
11
12 int main (void){
13     pthread_t th[8];
14     for (int i=0; i<8; i++){
15         pthread_create(&th[i], NULL, process, (void *) &i);
16         fflush(stdout); //fuerza al sistema a imprimir lo que salga sin q el sist lo corrija
17         /** Cuando hay algo q no es correcto porq puede retrasarlo --> el sist lo corrige solo */
18     }
19 }
20 }
```