

S.O.-PRACTICAS.pdf



charlieangel



Sistemas Operativos



2º Grado en Ingeniería Informática



Escuela Politécnica Superior de Córdoba
Universidad de Córdoba



[Accede al documento original](#)



Escuela de
Organización
Industrial

Contigo que evoluciones.
Contigo que lideras. Contigo que transformas.

**Esto es EOI.
Mismo propósito,
nueva energía.**



Descubre más aquí



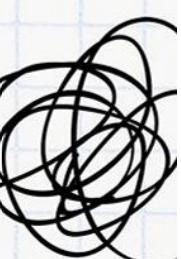
EOI Escuela de
Organización
Industrial

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah

PRÁCTICA 1:

DEMO1:

```
#include <sys/types.h> getppid() / getpid() / fork()
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> getpid() / getpid() / fork()
#include <errno.h>
#include <sys/wait.h> wait()

int main(void)
{
    pid_t hijo_pid, childpid;
    int status;

    hijo_pid = fork(); → copia del proceso actual

    if (hijo_pid == -1) → no se ha creado * error *
    {
        perror("fork error");
        printf("errno value= %d\n", errno);
        exit(EXIT_FAILURE); //exit(-1); //Necesaria la librería <stdlib.h>
    }
    else if (hijo_pid == 0) → creado con éxito
    {
        printf("Soy el hijo, mi pid es: %d\n", getpid()); → función
        para obtener ID del proceso

        exit(EXIT_SUCCESS); //exit(0);
    }
    else /* padre */
    {
        printf("Padre con pid: %d, el pid de mi hijo es: %d\n", getpid(), hijo_pid);

        childpid = wait(&status); → bloquea
        el p. padre
        hasta que
        acabe p. hijo
        if (childpid > 0) → si el wait() es positivo ⇒ ID_hijo → TERMINADO / LISTO
        {
            if (WIFEXITED(status)) → ≠ 0
            {
                if (WIFEXITED(status)) → ≠ 0
                {
                    hijo termina con normalidad
                }
            }
        }
    }
}
```

WIFEXITED(int) ≠ 0 → hijo termina forma normal
↳ WEXITSTATUS(int) → imprime el estado
(código de salida del hijo)

WIFSIGNALED (int) ≠ 0 → hijo termina x señal NO CONTROLADA
↳ WTERMSIG(int) → que señal lo causó
SIGSEGV SIGKILL

WIFSTOPPED (int) ≠ 0 → hijo detenido
(depuración - control de trabajos)
↳ WSTOPSIG(int) → señal que lo detiene
SIGSTOP SIGSTOP

WIFCONTINUED (int) → hijo continua después
de haberse detenido

MACROS WAIT()

```

printf("Soy el padre, hijo %d recogido, status=%d\n", childpid, WEXITSTATUS(status));
}

else if (WIFSIGNALED(status)) → ( ≠ 0 ) ↗ Hijo terminó
    ↗ x señal NO CONTROLADA (interrumpido
    ↗ abruptamente)

{
    printf("Soy el padre, hijo %d matado (signal %d)\n", childpid, WTERMSIG(status));
    ↗ devuelve nº señal
    ↗ causa la terminación
    ↗ del hijo

else → childpid < 0

{
    printf("Error en la invocación de wait o la llamada ha sido interrumpida por una señal.\n");
    exit(EXIT_FAILURE);
}

exit(EXIT_SUCCESS); //exit 0; return 0;
}
}

```

TERMINAL:

```

[REDACTED]@ubuntu: ~/Desktop/S0/P1/DEMOS$ gcc demo1.c -o demo1
[REDACTED]@ubuntu: ~/Desktop/S0/P1/DEMOS$ ./demo1
Padre con pid: 4004, el pid de mi hijo es: 4005
Soy el hijo, mi pid es: 4005
Soy el padre, hijo 4005 recogido, status=0
[REDACTED]@ubuntu: ~/Desktop/S0/P1/DEMOS$ 

```

DEMO2:

```

#include <sys/types.h> //Para tipo pid_t
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> //Para fork()
#include <errno.h>
#include <sys/wait.h>
#include <string.h> //Para la función strerror(), que permite describir el valor de errno como cadena.

int main(void)
{
    pid_t pid, childpid;
    int status;

    pid = fork();
    switch(pid)
    {
        case -1:
            perror("fork error\n");
            printf("errno value= %d\n", errno);
            exit(EXIT_FAILURE);
        //exit (-1);

        case 0: /* proceso hijo */
            printf("Soy el Hijo, mi PID es %d y el PPID de mi padre es %d \n", getpid(), getppid());
            exit(EXIT_SUCCESS);
    }
}

```

```

default: /* padre */
printf("Soy el Padre, mi PID es %d y el PID de mi hijo es %d, el PPID de mi
padre es %d\n", getpid(), pid, getppid());
/*Averigüe quien es el padre del proceso padre*/

//Se espera al hijo
//Uso de wait(). No permite detectar la parada y
reanudación de procesos hijos:

WIFSTOPPED(status), WIFCONTINUED(status)

    while ((childpid=wait(&status)) > 0)
    {
        if (WIFEXITED(status))
        {
            printf("Proceso padre %d, hijo con PID
%d finalizado, status = %d\n", getpid(), (long int)childpid, WEXITSTATUS(status));
        }

        else if (WIFSIGNALED(status)) //Para señales
como las de finalizar o matar
        {
            printf("Proceso padre %d, hijo con PID
%d finalizado al recibir la señal %d\n", getpid(), (long int)childpid,
WTERMSIG(status));
        }

        if (childpid==(pid_t)-1 && errno==ECHILD) //Entra cuando
vuelve al while y no hay más hijos que esperar
        {
            printf("Proceso padre %d, no hay más hijos que
esperar. Valor de errno = %d, definido como: %s\n", getpid(), errno, strerror(errno));
        }
        else
        {
            printf("Error en la invocación de wait o waitpid.
Valor de errno = %d, definido como: %s\n", errno, strerror(errno));
            exit(EXIT_FAILURE);
        }
    }

    exit(EXIT_SUCCESS); //return 0; exit(0);
}
}

```

TERMINAL:

```

p@ubuntu:~/Desktop/S0/P1/DEMOS$ ./demo2
Soy el Padre, mi PID es 4173 y el PID de mi hijo es 4174, el PPID de mi padre es 3984
Soy el Hijo, mi PID es 4174 y el PPID de mi padre es 4173
Proceso padre 4173, hijo con PID 4174 finalizado, status = 0
Proceso padre 4173, no hay mas hijos que esperar. Valor de errno = 10, definido como: No child processes

```

DEMO3:

```

#include <sys/types.h> fork() y pid_t
#include <unistd.h>
#include <stdio.h> operaciones E/S
#include <stdlib.h> exit()
#include <errno.h> maneja errores
#include <sys/wait.h> MACROS_WAIT()
#include <string.h> //Para la función strerror(), que permite describir el valor de errno como cadena.

int main ()
{
    int i=0,j, status;
    pid_t rf, childpid;           pid_t fork();

    rf = fork();
    switch (rf) → control x switch
    {
        case -1:
            perror("fork error");
            printf("errno value= %d\n", errno);
            exit(EXIT_FAILURE);

        case 0:
            ↳ proceso hijo
    }
}

```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah

```

printf("Soy el hijo, mi PID es %d y mi variable i (initialmente a %d) es:\n", getpid(), i);
for (j=0; j<5; j++)
{
    i++;
    //sleep(5); → si se activa, el hijo dormiría durante 5 seg en cada ciclo
    i++;
    printf ("Soy el hijo, mi variable i es %d\n", i);
}
break;

default: >0 → es el PADRE

printf("Soy el padre, mi PID es %d y mi variable i (initialmente a %d) es:\n", getpid(), i);
for (j=0; j<5; j++)
{
    i++;
    i++;
    printf ("Soy el padre, mi variable i es %d\n", i);

//Se espera al hijo
/*Espera del padre a los hijos*/
while ((childpid=waitpid(-1,&status,WUNTRACED | WCONTINUED)) > 0) → mientras el padre espera al hijo
{
    if (WIFEXITED(status))
    {
        printf("Proceso Padre %d, hijo con PID %ld finalizado, status = %d\n",
getpid(), (long int)childpid, WEXITSTATUS(status));
    }
    else if (WIFSIGNALED(status))
    {
        printf("Proceso Padre %d, hijo con PID %ld finalizado al recibir la señal
%d\n", getpid(), (long int)childpid, WTERMSIG(status));
    }
    else if (WIFSTOPPED(status)) //Para cuando se para un proceso. Con wait() no nos serviría.
    {
        printf("Proceso Padre %d, hijo con PID %ld parado al recibir la señal %d\n",
getpid(), (long int)childpid, WSTOPSIG(status));
    }
    else if (WIFCONTINUED(status)) //Para cuando se reanuda un proceso
parado. Con wait() no nos serviría.
    {
        printf("Proceso Padre %d, hijo con PID %ld reanudado\n", getpid(), (long int)
childpid);
    }
}//while

if (childpid==(pid_t)-1 && errno==ECHILD) → VERIFICA SI NO + HIJOS QUE ESPERAR
{
    printf("Proceso Padre %d, no hay más hijos que esperar. Valor de errno = %d,
definido como: %s\n", getpid(), errno, strerror(errno));
}
else
{
    printf("Error en la invocación de wait o waitpid. Valor de errno = %d, definido
como: %s\n", errno, strerror(errno));
    exit(EXIT_FAILURE);
} //switch

//Esta línea la ejecuta tanto el padre como el hijo
printf ("Final de ejecución...\n");
exit(EXIT_SUCCESS);
}

```

TERMINAL:

wuolah

```

root@ubuntu:~/Desktop/SO/P1/DEMOS$ ./demo3
Soy el padre, mi PID es 4279 y mi variable i (inicialmente a 0) es:
Soy el padre, mi variable i es 2
Soy el padre, mi variable i es 4
Soy el padre, mi variable i es 6
Soy el hijo, mi PID es 4280 y mi variable i (inicialmente a 0) es:
Soy el hijo, mi variable i es 2
Soy el padre, mi variable i es 8
Soy el padre, mi variable i es 10
Soy el hijo, mi variable i es 4
Soy el hijo, mi variable i es 6
Soy el hijo, mi variable i es 8
Soy el hijo, mi variable i es 10
Final de ejecucion...
Proceso Padre 4279, hijo con PID 4280 finalizado, status = 0
Proceso Padre 4279, no hay mas hijos que esperar. Valor de errno = 10, definido como: No child processes
Final de ejecucion...

```

DEMO4:

```

#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h> //Para la funcion strerror(), que permite describir el valor de errno como cadena.

int main()
{
    pid_t idProceso, childpid;
    int status;
    /* Variable para comprobar que se copia inicialmente en cada proceso y
       que luego puede cambiarse independientemente en cada uno de ellos. */
    int variable = 1;

    idProceso = fork();
    if (idProceso == -1)
    {
        perror("fork error");
        printf("errno value= %d\n", errno);
        exit(EXIT_FAILURE);
    }
    else if (idProceso == 0)
    {
        /* El hijo escribe su pid en pantalla y el valor de variable */
        printf ("Hijo: Mi pid es %d. El pid de mi padre es %d\n", getpid(), getppid());
    }
}

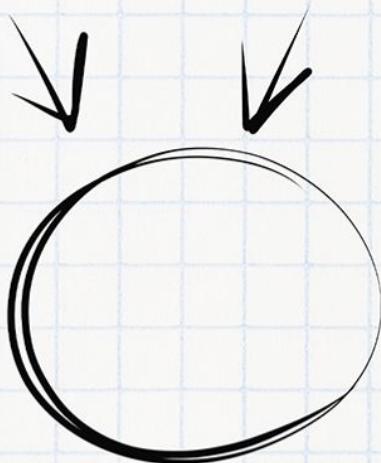
```

Imagínate aprobando el examen

Necesitas tiempo y concentración

Planes	PLAN TURBO	PLAN PRO	PLAN PRO+
diamond Descargas sin publi al mes	10 🟡	40 🟡	80 🟡
clock Elimina el video entre descargas	✓	✓	✓
folder Descarga carpetas	✗	✓	✓
download Descarga archivos grandes	✗	✓	✓
circle Visualiza apuntes online sin publi	✗	✓	✓
glasses Elimina toda la publi web	✗	✗	✓
€ Precios	Anual <input type="checkbox"/>	0,99 € / mes	3,99 € / mes
			7,99 € / mes

Ahora que puedes conseguirlo,
¿Qué nota vas a sacar?



WUOLAH

```

/* Escribe valor de variable y la cambia */

printf ("Hijo: valor de variable = %d. Cambiadola a un valor de 2...\n",
variable);

variable = 2;

/* Espera 5 segundos, saca en pantalla el valor de variable y sale */

/*OJO, ESTOS SLEEPS SON A MODO DIVULGATIVO. NO USAR SLEEPS
PARA SINCRONIZAR SUS PROCESOS*/
sleep (5); → NO USAR

printf ("Hijo: valor de variable = %d y termino devolviendo codigo 33.\n",
variable);

exit (33);

}

else //Padre

{

/* Espera un segundo (para dar tiempo al hijo a hacer sus cosas y no
entremezclar salida en la pantalla) y escribe su pid y el de su hijo */

/*OJO, ESTOS SLEEPS SON A MODO DIVULGATIVO. NO USAR SLEEPS
PARA SINCRONIZAR SUS PROCESOS*/
sleep (1);

printf ("Padre: Mi pid es %d. El pid de mi hijo es %d\n",getpid(), idProceso);

/*Espera del padre a los hijos*/

while ((childpid=waitpid(-1,&status,WUNTRACED | WCONTINUED)) > 0)

{

if (WIFEXITED(status))

{

printf("Proceso Padre %d, hijo con PID %ld finalizado, status = %d\n", getpid(), (long
int)childpid, WEXITSTATUS(status));

}

else if (WIFSIGNALED(status))

{

printf("Proceso Padre %d, hijo con PID %ld finalizado al recibir la señal %d\n",
getpid(), (long int)childpid, WTERMSIG(status));

}

else if (WIFSTOPPED(status)) //Para cuando se para un proceso. Con wait() no nos
serviría.

{

printf("Proceso Padre %d, hijo con PID %ld parado al recibir la señal %d\n",
getpid(), (long int)childpid,WSTOPSIG(status));

}
}

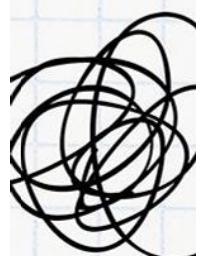
```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah

```
else if (WIFCONTINUED(status)) //Para cuando se reanuda un proceso parado. Con
wait() no nos serviría.

{
    printf("Proceso Padre %d, hijo con PID %ld reanudado\n", getpid(), (long int)
childpid);
}

}//while

if (childpid==(pid_t)-1 && errno==ECHILD)

{
    printf("Proceso Padre %d, no hay mas hijos que esperar. Valor de errno = %d,
definido como: %s\n", getpid(), errno, strerror(errno));
}

else

{
    printf("Error en la invocacion de wait o waitpid. Valor de errno = %d, definido como:
%s\n", errno, strerror(errno));

    exit(EXIT_FAILURE);
}

printf("Proceso Padre %d, valor de variable = %d\n", getpid(), variable);

}

exit(EXIT_SUCCESS);
}
```

TERMINAL:

```
lp@ubuntu:~/Desktop/S0/P1/DEMOS$ ./demo4
Hijo: Mi pid es 4368. El pid de mi padre es 4367
Hijo: valor de variable = 1. Cambiadola a un valor de 2...
Padre: Mi pid es 4367. El pid de mi hijo es 4368
Hijo: valor de variable = 2 y termino devolviendo codigo 33.
Proceso Padre 4367, hijo con PID 4368 finalizado, status = 33
Proceso Padre 4367, no hay mas hijos que esperar. Valor de errno = 10, definido como: No child processes
Proceso Padre 4367, valor de variable = 1
```

DEMO5:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h> //Para la funcion strerror(), que permite describir el valor de errno como cadena.

void main(void)
{
    pid_t pid, childpid;
    int status;

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

wuolah

```

{
    case -1: /* error del fork() */
    perror("fork error");
    printf("errno value= %d\n", errno);
    exit(EXIT_FAILURE);

    case 0: /* proceso hijo */
    printf("Proceso hijo con PID %d; mi padre con PPID %d\n", getpid(), getppid());

    if(execlp("ls","ls","-l",NULL)==-1) → si entra, A error
    {
        perror("Falla en la ejecucion exec de ls -l");
        printf("errno value= %d\n", errno);
        exit(EXIT_FAILURE);
    }

    default: /* padre */
    printf("Proceso padre con pid %d\n", getpid());

    //Se espera al hijo
    //Uso de wait(). No permite detectar la parada y reanudacion
    de procesos hijos: WIFSTOPPED(status), WIFCONTINUED(status)
    while ( (childpid=wait(&status)) > 0 )
    {
        if (WIFEXITED(status))
        {
            printf("Proceso padre %d, hijo con PID %ld finalizado, status = %d\n", getpid(), (long int)childpid, WEXITSTATUS(status));
        }
        else if (WIFSIGNALED(status)) //Para seniales como las de
        finalizar o matar
        {
            printf("Proceso padre %d, hijo con PID %ld finalizado al recibir la señal %d\n", getpid(), (long int)childpid, WTERMSIG(status));
        }
    }

    if (childpid==(pid_t)-1 && errno==ECHILD) //Entra cuando vuelve
    al while y no hay más hijos que esperar
    {
        printf("Proceso padre %d, no hay mas hijos que esperar.
        Valor de errno = %d, definido como: %s\n", getpid(), errno,
        strerror(errno));
    }
    else
    {
        printf("Error en la invocacion de wait o waitpid. Valor de
        errno = %d, definido como: %s\n", errno, strerror(errno));
        exit(EXIT_FAILURE);
    }
} //switch

//La ejecutará en este caso solo el padre, el hijo tiene una
llamada a exec()
printf("Fin del programa. PID del que ejecuta esta linea: %d\n",
getpid());
exit(EXIT_SUCCESS);
}

```

TERMINAL:

```

Proceso padre con pid 4390
Proceso hijo con PID 4391; mi padre con PPID 4390
total 116
-rwxrwxr-x 1 i32lovip i32lovip 16256 Nov  6 19:27 demo1
-rw-rw-r-- 1 i32lovip i32lovip 1256 Jul 16 11:25 demo1.c
-rwxrwxr-x 1 i32lovip i32lovip 16304 Nov  6 19:43 demo2
-rw-rw-r-- 1 i32lovip i32lovip 1996 Jul 16 11:25 demo2.c
-rwxrwxr-x 1 i32lovip i32lovip 16304 Nov  6 19:50 demo3
-rw-rw-r-- 1 i32lovip i32lovip 2983 Jul 16 11:25 demo3.c
-rwxrwxr-x 1 i32lovip i32lovip 16352 Nov  6 19:53 demo4
-rw-rw-r-- 1 i32lovip i32lovip 3175 Jul 16 11:25 demo4.c
-rwxrwxr-x 1 i32lovip i32lovip 16352 Nov  6 19:57 demo5
-rw-rw-r-- 1 i32lovip i32lovip 2314 Jul 16 11:25 demo5.c
-rw-rw-r-- 1 i32lovip i32lovip 2358 Jul 16 11:25 demo6.c
-rw-rw-r-- 1 i32lovip i32lovip 1425 Jul 16 11:25 demo7.c
-rw-rw-r-- 1 i32lovip i32lovip 1244 Jul 16 11:25 demo8.c
-rw-rw-r-- 1 i32lovip i32lovip 3298 Jul 16 11:25 while-wait-waitpid.c
Proceso padre 4390, hijo con PID 4391 finalizado, status = 0
Proceso padre 4390, no hay mas hijos que esperar. Valor de errno = 10, definido como: No child processes
Fin del programa. PID del que ejecuta esta linea: 4390

```

DEMO6:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h> //Para la funcion strerror(), que permite describir el valor de errno como cadena.
```

/*Pruebe por ejemplo:

```
./a.out tar -cf prueba.tar demo1 demo2 demoN
```

donde file1 hasta fileN son ficheros que desea comprimir en el fichero prueba.tar
-Pruébelo también sin usar nada en la linea de argumentos.
-Por ultimo pruébelo usando un "file1" que no exista.

```
*/
```

```
int main(int argc, char **argv)
{
    pid_t pid, childpid;
    int status;

    pid = fork();
    switch(pid)
    {
        case -1: /* error del fork() */
            perror("fork error");
            printf("errno value= %d\n", errno);
            exit(EXIT_FAILURE);

        case 0: /* proceso hijo */
            printf("Proceso hijo con PID %d; mi padre con PPID %d\n", getpid(), getppid());

            if (execvp(argv[1], &argv[1])<0) //Exec devuelve -1 en caso de error
            {
                perror("exec");
                printf("errno value= %d\n", errno);
                exit(EXIT_FAILURE);
            }

            default: /* padre */
            printf("Padre con pid: %ld\n", (long)getpid());

            //Se espera al hijo
            //Uso de wait(). No permite detectar la parada y reanudacion de procesos
            hijos: WIFSTOPPED(status), WIFCONTINUED(status)
            while ((childpid=wait(&status)) > 0 )
            {
                if (WIFEXITED(status))
                {
                    printf("Proceso padre %d, hijo con PID %ld finalizado, status = %d\n", getpid(),
                    (long int)childpid, WEXITSTATUS(status));
                }
                else if (WIFSIGNALED(status)) //Para seniales como las de finalizar o morir
                {
                    printf("Proceso padre %d, hijo con PID %ld finalizado al recibir la señal %d\n",
                    getpid(), (long int)childpid, WTERMSIG(status));
                }
            }

            if (childpid==(pid_t)-1 && errno==ECHILD) //Entra cuando vuelve al while y no
            hay más hijos que esperar
            {
                printf("Proceso padre %d, no hay mas hijos que esperar. Valor de errno = %d,
                definido como: %s\n", getpid(), errno, strerror(errno));
            }
            else
            {
                printf("Error en la invocacion de wait o waitpid. Valor de errno = %d, definido
                como: %s\n", errno, strerror(errno));
                exit(EXIT_FAILURE);
            }
        }
        printf("Programa finalizado\n");
        exit(EXIT_SUCCESS);
}
```

#include <unistd.h>

execl (const char* path, const char* arg 0, NULL)
ruta que se desea ejecutar
debe terminar en NULL (= fin de argumentos)

execv (const char* path, char * const argv[])
array de punteros (cada entrada es un argumento)
↳ el ult = NULL

TERMINAL:

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah

```
ntu:~/Desktop/S0/P1/DEMOS$ ./demo6
Mi proceso con pid: 4484
Proceso hijo con PID 4485; mi padre con PPID 4484
Proceso padre 4484, hijo con PID 4485 finalizado al recibir la señal 11
Proceso padre 4484, no hay mas hijos que esperar. Valor de errno = 10, definido como: No child processes
Programa finalizado
```

DEMO7:

```
#include <signal.h> → USO DE SEÑALES
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>

void sig_alarmp(int signo)
{
    printf("Dentro de la funcion sig_alarmp!!\n");
    printf("Valor del entero recibido como parametro:%d\n",signo);
    return; /* nothing to do, just return to wake up the pause */
}

configura
alarma
nsecs
despues,
despues
se suspende
hasta
senal
SIGALRM

unsigned int f1(int nsecs)
{
    alarm(nsecs);           /* start the timer */
    pause();                 /* pause - suspend the thread until a signal
is received */

int main ()
{
    /* Si la solicitud de tratamiento de senial se puede llevar a cabo, la
funcion signal() devolverá
el nombre de la función (* void) que la tratará, en caso contrario se
devuelve el valor de la macro
SIG_ERR y se pone errno a un valor positivo*/
    if (signal(SIGALRM, sig_alarmp) == SIG_ERR)
    {
        perror("Signal error");
        printf("errno value= %d\n", errno);
        exit(EXIT_FAILURE);
    }

    /* Código alternativo al anterior

//https://stackoverflow.com/questions/35828466/sig-ign-sig-dfl-sig-err-s-defini
tion
//Puntero a funcion que recibe un entero
void (* value)(int);
value = signal(SIGINT, sig_alarmp);
printf("Valor de value:%p\n",value);

if(value == SIG_ERR)
{
    perror("Signal error");
    printf("errno value= %d\n", errno);
    exit(EXIT_FAILURE);
}
*/
    printf ("Una alarma en 3 segundos....\n");
    f1(3);
    printf ("Una alarma en 2 segundos...\n");
    f1(2);
    printf ("Fin del programa\n");
}
```

TERMINAL:

```

Ubuntu:~/Desktop/SO/P1/DEMOS$ ./demo7
Una alarma en 3 segundos...
Dentro de la funcion sig_alarm!!!
Valor del entero recibido como parametro:14
Una alarma en 2 segundos...
Dentro de la funcion sig_alarm!!!
Valor del entero recibido como parametro:14
Fin del programa

```

DEMO8:

```

/*
 * Programa de demostración de signal()
 */
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

/*
 * Controlador para la señal de ctrl-c
 */
void controlador (int);

/*
 * Programa principal.
 * Cambia el controlador para la señal de ctrl-c y se mete en un bucle
 * infinito de espera
 */

int main(void)
{
    /* Se cambia el controlador para ctrl-c, escribiendo error en
pantalla
     * si lo hay.*/
    if (signal (SIGINT, controlador) == SIG_ERR)
    {
        perror("Signal error");
        printf("errno value= %d\n", errno);
        exit(EXIT_FAILURE);
    }

    /* Bucle infinito de espera. pause() deja el proceso dormido hasta
que
     * llegue una señal.*/
    while (!)
    {
        pause ();
    }
}

/*
 * Controlador para ctrl-c.
 * Indica en pantalla que no se quiere salir y pone el controlador por
 * defecto para ctrl-c. La segunda vez que se pulse ctrl-c, el programa
 * se saldrá normalmente.
 */
void controlador (int pepito)
{
    printf ("Pues no me salgo \n");
    printf ("Intentalo otra vez\n");
    printf ("Valor de la señal pepito:%d\n",pepito);

    /* Se pone controlador por defecto para ctrl-c */
    //signal (SIGINT, SIG_DFL); //Anula la recepción anterior de SIGINT
por esta otra. Pruebe a comentar esta linea.
}

```

kill (pid -> pid, int sig)

TERMINAL:

WHILE-WAIT-WAITPID:

```

/***********************
Esquema de espera de hijos usando wait()
***** */

/* Espera del padre a los hijos */
while ((flag = wait(&valor)) > 0)
{
    if (WIFEXITED(valor))
    {
        printf("Hijo ID:%ld finalizado, estado=%d\n", (long int) flag, WEXITSTATUS(valor));
    }
    else if (WIFSIGNALED(valor)) //Para seniales como las de finalizar o matar
    {
        printf("Hijo ID:%ld finalizado al recibir la señal %d\n", (long int) flag, WTERMSIG(valor));
    }
}//while

if (flag == -1 && errno == ECHILD)
{
    printf("Proceso padre %d, no hay mas hijos que esperar. Valor de errno = %d, definido como: %s\n", getpid(), errno, strerror(errno));
    //perror("Valor de errno usando perror"); //Descomente esta linea y compruebe su salida.
}
else
{
    printf("Error en la invocacion de wait o waitpid. Valor de errno = %d, definido como: %s\n", errno, strerror(errno));
    exit(EXIT_FAILURE);
}

/***********************
Esquema de espera de hijos usando waitpid()
***** */

while ((flag = waitpid(-1, &status)) > 0)
{
    if (WIFEXITED(status))
    {
        printf("Proceso padre %d, hijo con PID %ld finalizado, status = %d\n", getpid(), (long int) flag, WEXITSTATUS(status));
    }
}

```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah

```
        }
        else if (WIFSIGNALED(status)) //Para seniales como las de
        finalizar o matar
        {
            printf("Proceso padre %d, hijo con PID %ld finalizado al
        recibir la señal %d\n", getpid(), (long int)flag, WTERMSIG(status));
        }
    }
    if (flag==(pid_t)-1 && errno==ECHILD) //Entra cuando vuelve al while y no
    hay más hijos que esperar
    {
        printf("Proceso padre %d, no hay mas hijos que esperar. Valor
        de errno = %d, definido como: %s\n", getpid(), errno, strerror(errno));
    }
    else
    {
        printf("Error en la invocacion de wait o waitpid. Valor de errno =
        %d, definido como: %s\n", errno, strerror(errno));
        exit(EXIT_FAILURE);
    }

    /**************************************************************************
    Esquema de espera de hijos usando waitpid() con la posibilidad de
    capturar que un hijo sea parado y reanudado
    **************************************************************************

/*Espera del padre a los hijos*/
while ( (flag=waitpid(-1,&status,WUNTRACED | WCONTINUED)) > 0 )
{
    if (WIFEXITED(status))
    {
        printf("Proceso Padre %d, hijo con PID %ld finalizado, status =
        %d\n", getpid(), (long int)flag, WEXITSTATUS(status));
    }
    else if (WIFSIGNALED(status))
    {
        printf("Proceso Padre %d, hijo con PID %ld finalizado al
        recibir la señal %d\n", getpid(), (long int)flag, WTERMSIG(status));
    }
    else if (WIFSTOPPED(status)) //Para cuando se para un proceso. Con
    wait() no nos serviría.
    {
        printf("Proceso Padre %d, hijo con PID %ld parado al recibir
        la señal %d\n", getpid(), (long int)flag, WSTOPSIG(status));
    }
    else if (WIFCONTINUED(status)) //Para cuando se reanuda un
    proceso parado. Con wait() no nos serviría.
    {
        printf("Proceso Padre %d, hijo con PID %ld reanudado\n",
        getpid(), (long int) flag);
    }
}/while

if (flag==(pid_t)-1 && errno==ECHILD)
{
    printf("Proceso Padre %d, no hay mas hijos que esperar. Valor de
    errno = %d, definido como: %s\n", getpid(), errno, strerror(errno));
}
else
{
    printf("Error en la invocacion de wait o waitpid. Valor de errno =
    %d, definido como: %s\n", errno, strerror(errno));
    exit(EXIT_FAILURE);
}
```

PRÁCTICA 2:

DEMO1:

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

wuolah

```

void * mifucion(void *arg)
{
    int *argu;
    printf("Hilo hijo: Ejecutando...\n");
    argu = (int *) arg; //Casting a entero del parámetro de entrada.
    printf("Hilo hijo: Valores recibidos: arg0= %d arg1= %d\n", argu[0], argu[1]);
    printf("Hilo hijo: Finalizando....\n");
    /* Esta función no devuelve nada, por tanto no se podrá recoger nada con un join().
     Por defecto, si no se incluye se hace implícitamente un pthread_exit(NULL); */
    pthread_exit(NULL);
    //exit(0);
}

int main ()
{
    pthread_t tid;
    //Vector de enteros que vamos a pasar como parámetro a una hebra haciendo casting a (void *)
    int misargs[2];
    misargs[0] = -5;
    misargs[1] = -6;

    printf("Hilo principal: Se va a crear un hilo...\n");
    if( pthread_create(&tid, NULL, (void *) mifucion, (void *) misargs) )
    {
        fprintf(stderr, "Error creating thread\n");
        exit(EXIT_FAILURE);
    }

    /*Si comentamos el join(), puede que a la hebra no le de tiempo a ejecutarse ni siquiera parcialmente,
     de manera que el main() terminará y con el todas las hebras asociadas a este proceso*/
    printf("Hilo principal: Hilo creado, esperando su finalización desde el main()\n");
    if( pthread_join(tid, NULL) )
    {
        fprintf(stderr, "Error joining thread\n");
        exit(EXIT_FAILURE);
    }
}

```

```

}

printf("Hilo principal: Hilo finalizado y recogido con pthread_join()\n");
exit(EXIT_SUCCESS);
}

```

TERMINAL:

```

p@ubuntu:~/Desktop/S0/P2/DEMOS$ ./demo1
Hilo principal: Se va a crear un hilo...
Hilo principal: Hilo creado, esperando su finalización desde el main()...
Hilo hijo: Ejecutando...
Hilo hijo: Valores recibidos: arg0= -5 arg1= -6
Hilo hijo: Finalizando....
Hilo principal: Hilo finalizado y recogido con pthread_join()...

```

DEMO2:

```

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

void * hebra(void * puntero_x)
{
    int *mi_puntero = (int *) puntero_x;

    printf("Hebra hija: valor de x=%d, incrementandolo 100 veces.\n", *mi_puntero);

    //incrementar x hasta 100
    while(*mi_puntero < 100)
    {
        *mi_puntero=*mi_puntero+1;
    }

    printf("Hebra hija: incremento de x finalizado.\n");

    pthread_exit(NULL);
}

int main()
{
    int x = 0, y = 0;

```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



```
//Declaración de una hebra
pthread_t hebra_incr;

printf("Hebra principal o main(): valor de variable x: %d\n", x);
printf("Hebra principal o main(): valor de variable y: %d\n", y);

//Creación de la hebra
if(pthread_create(&hebra_incr, NULL, (void *) hebra, (void *) &x))
{
    fprintf(stderr, "Error creating thread\n");
    exit(EXIT_FAILURE);
}

printf("Hebra principal o main(): incrementando y 100 veces.\n");
//Incrementar y hasta 100
while(y < 100)
{
    y=y+1;
}
printf("Hebra principal o main(): incremento de y finalizado.\n");

//Espera de la hebra
if(pthread_join(hebra_incr, NULL))
{
    fprintf(stderr, "Error joining thread\n");
    exit(EXIT_FAILURE);
}

printf("Hebra principal o main(): valor de x: %d\n",x);
printf("Hebra principal o main(): valor de y: %d\n",y);

exit(EXIT_SUCCESS);
}
```

TERMINAL:

ali ali oooh
esto con 1 coin me
lo quito yo...

wuolah

wuolah

```

p@ubuntu:~/Desktop/S0/P2/DEMOS$ ./demo2
Hebra principal o main(): valor de variable x: 0
Hebra principal o main(): valor de variable y: 0
Hebra principal o main(): incrementando y 100 veces.
Hebra principal o main(): incremento de y finalizado.
Hebra hija: valor de x=0, incrementandolo 100 veces.
Hebra hija: incremento de x finalizado.
Hebra principal o main(): valor de x: 100
Hebra principal o main(): valor de y: 100

```

DEMO3:

```

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

//Declaración de una estructura
struct param
{
    char frase[30];
    int numero;
};

/*Función que se asignará a los hilos que se creen. Recibe un puntero a estructura */
void *hiloMensaje (void * mensa)
{
    struct param *aux = (struct param *) mensa;
    printf("%s %d\n", aux->frase, aux->numero);

    //Las lineas anteriores se pueden sustituir por esta otra
    //printf("%s %d\n", ((struct param *) mensa)->frase, ((struct param *) mensa)->numero);

    pthread_exit(NULL); //Fin de la hebra sin devolver nada
}

int main()
{
    //Declaración de dos hebras, hilos o procesos ligeros. NO CREACION
    pthread_t thd1, thd2;

```

```

// Inicialización de 2 estructuras de tipo "struct param"
struct param param1 = {"Soy el hilo ", 1};
struct param param2 = {"Soy el hilo ", 2};

/*Creamos dos hilos. La función la pasaremos como (void *) nombreFuncion.
Es decir, hacemos un casting a (void *), aunque por defecto no es necesario, ya que el
nombre de una función es su dirección de memoria. También es importante realizar
esto con la dirección de memoria de la variable que contiene los parámetros que se le
pasan a la función */
if( pthread_create (&thd1, NULL, (void *) hiloMensaje, (void *) &param1) )

{
    fprintf(stderr, "Error creating thread\n");
    exit(EXIT_FAILURE);
}

if( pthread_create (&thd2, NULL, (void *) hiloMensaje, (void *) &param2) )

{
    fprintf(stderr, "Error creating thread\n");
    exit(EXIT_FAILURE);
}

```

/ Como EJERCICIO, rehaga este programa de forma que cree las hebras y las
recoja mediante bucles for() */*

/ Esperamos la finalización de los hilos. Si la función devolviera algo habría
que recogerlo con el segundo argumento, que en este caso está a NULL. Cuando el
segundo argumento no es NULL, se recogen los resultados que vienen de
pthread_exit(), que se explicará en las siguientes demos.*/*

```

if( pthread_join(thd1, NULL) )

{
    fprintf(stderr, "Error joining thread\n");
    exit(EXIT_FAILURE);
}

if( pthread_join(thd2, NULL) )

{
    fprintf(stderr, "Error joining thread\n");
    exit(EXIT_FAILURE);
}

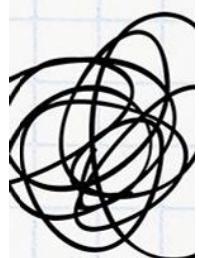
```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah

/*Si no se ponen estos join() en el programa principal y simplemente lanzamos los dos hilos y finalizamos, lo más probable es que los hilos no lleguen a ejecutarse completamente o incluso que no lleguen ni a empezar antes de que el programa principal termine.*

```
    printf("Hilo principal: han finalizado las threads.\n");
    exit(EXIT_SUCCESS);
}
```

TERMINAL:

```
[...]:~$ ip@ubuntu:~/Desktop/S0/P2/DEMOS$ ./demo3
Soy el hilo 1
Soy el hilo 2
Hilo principal: han finalizado las threads.
```

DEMO4:

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

//Prototipos
void * mifuncion(void *);
void miFuncion2();

//Hilo principal
int main (void)
{
    pthread_t tid;
    char micadena[11] = "Hola mundo";

    printf("Hilo principal. Se va a crear un hilo...\n");
    if( pthread_create(&tid, NULL, mifuncion, (void *) micadena) )
    {
        fprintf(stderr, "Error creating thread\n");
        exit(EXIT_FAILURE);
    }
}
```

/*Si comentamos la siguiente linea, puede que a la hebra no le de tiempo a ejecutarse

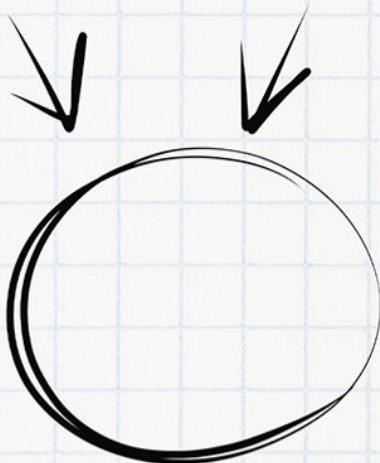
wuolah

Imagínate aprobando el examen

Necesitas tiempo y concentración

Planes	PLAN TURBO	PLAN PRO	PLAN PRO+
diamond Descargas sin publi al mes	10 🟡	40 🟡	80 🟡
clock Elimina el video entre descargas	✓	✓	✓
folder Descarga carpetas	✗	✓	✓
download Descarga archivos grandes	✗	✓	✓
circle Visualiza apuntes online sin publi	✗	✓	✓
glasses Elimina toda la publi web	✗	✗	✓
€ Precios	Anual <input type="checkbox"/>	0,99 € / mes	3,99 € / mes
			7,99 € / mes

Ahora que puedes conseguirlo,
¿Qué nota vas a sacar?



WUOLAH

```

ni siquiera parcialmente, de manera que el main() terminará y con el todas las hebras
asociadas a este proceso*/
printf("Hilo principal. Esperando finalizacion de hilo creado...\n");
if( pthread_join(tid, NULL) )
{
    fprintf(stderr, "Error joining thread\n");
    exit(EXIT_FAILURE);
}

printf("Hilo principal finalizado...\n");
exit(EXIT_SUCCESS);
}

//Función asignada a hilo
void *mifuncion(void *arg)
{
    char *argu;
    printf("Hilo hijo %lu ejecutando...\n", pthread_self());
    argu = (char *) arg; //Casting a char del parámetro de entrada.
    printf("Hilo hijo, valor recibido: %s\n", argu);
    printf("Hilo hijo invocando a subrutina....\n");
    miFuncion2();
    /*No se llegará a ejecutar de aquí hasta el final de esta función*/
    printf("Soy el hilo. He salido de la llamada a subrutina\n");
    pthread_exit(NULL);
}

//Esta funcion no es un hilo, es una subrutina
void miFuncion2()
{
    printf("Soy el hilo hijo creado y estoy dentro de una subrutina para invocar a pthread_exit()...\n");
    pthread_exit(NULL);
}

```

TERMINAL:

```

p@ubuntu:~/Desktop/S0/P2/DEMOS$ ./demo4
Hilo principal. Se va a crear un hilo...
Hilo principal. Esperando finalizacion de hilo creado...
Hilo hijo 134896762750656 ejecutando...
Hilo hijo, valor recibido: Hola mundo
Hilo hijo invocando a subrutina....
Soy el hilo hijo creado y estoy dentro de una subrutina para invocar a pthread_exit()...
Hilo principal finalizado...

```

DEMO5:

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

void * th_function (void *arg)
{
    printf("Soy la hebra %lu\n", (unsigned long) pthread_self());
    printf("Soy la hebra %lu\n", pthread_self());

    printf("Soy la hebra %ld\n", (long int) pthread_self());
    printf("Soy la hebra %ld\n", pthread_self());

    // Es conveniente usar largos con los identificadores de hebra, concretamente
    pthead_t es un unsigned long int (%lu)

    // y esta definido en el fichero de cabecera pthreadtypes.h de la distribucion de linux
    // concreta, por ejemplo en

    // /usr/include/x86_64-linux-gnu/bits/pthreadtypes.h
    printf("Soy la hebra %u\n", (unsigned int) pthread_self());
    printf("Soy la hebra %d\n", (int) pthread_self());

    long * y = malloc(sizeof(long));
    *y = 10;

    pthread_exit((void *)y);
}

int main(int argc, char ** argv)
{
    int join_value;
    long rvalue;
    long *ret;
    pthread_t thread;

    if( pthread_create(&thread, NULL, th_function, NULL) )
    {
        fprintf(stderr, "Error creating thread\n");
    }
```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah

```
exit(EXIT_FAILURE);
}

/* A pthread_join() le estamos pasando por referencia como segundo parámetro la
dirección

de un puntero a entero. Al pasar la dirección de memoria de un puntero por
referencia, el contenido

de ese puntero (a donde apunta) puede ser modificado. ¿Con qué se modifica? Con
la dirección de

memoria de un entero que se devuelve en "pthread_exit((void *)y)". Por tanto, "long *
ret" ahora

apunta a donde apuntaba "long * y", solo que está casteado a void.*/
join_value = pthread_join(thread,(void**) &ret);

if(join_value!=0)
{
    perror("Fallo en pthread_join()\n");
    exit(EXIT_FAILURE);
}

rvalue = *ret;
printf("Proceso o hilo principal, el valor devuelto por la hebra es: %ld\n", rvalue);

exit(EXIT_SUCCESS);
}
```

TERMINAL:

```
[root@ip:~]# ./demo5
Soy la hebra 128107008755392
Soy la hebra 128107008755392
Soy la hebra 128107008755392
Soy la hebra 128107008755392
Soy la hebra 1019217600
Soy la hebra 1019217600
Proceso o hilo principal, el valor devuelto por la hebra es: 10
```

DEMO6:

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

wuolah

```

#include <unistd.h>

void manejador(int sig)
{
    printf("Hilo %lu recibió señal %d\n", pthread_self(), sig);
    pthread_exit(NULL);
}

void * hilo(void * arg)
{
    printf("Soy el hilo %lu\n", pthread_self());
    if (signal(SIGTERM, manejador) == SIG_ERR)
    {
        fprintf(stderr, "Error en signal\n");
        exit(EXIT_FAILURE);
    }
    while (1)
    {
        printf("Hilo %lu en ejecución\n", pthread_self());
        sleep(1);
    }
}

int main(int argc, char const *argv[])
{
    // Se podría establecer el manejador de señal aquí
    // en lugar de hacerlo en el hilo
    // if (signal(SIGTERM, manejador) == SIG_ERR)
    // {
    //     fprintf(stderr, "Error en signal\n");
    //     exit(EXIT_FAILURE);
    // }

    pthread_t thread;
    if (pthread_create(&thread, NULL, hilo, NULL))
    {
        fprintf(stderr, "Error creando hilo\n");
        exit(EXIT_FAILURE);
    }
}

```

```

}

sleep(5);

if (pthread_kill(thread, SIGTERM))
{
    fprintf(stderr, "Error matando hilo\n");
    exit(EXIT_FAILURE);
}

if (pthread_join(thread, NULL))
{
    fprintf(stderr, "Error en pthread_join\n");
    exit(EXIT_FAILURE);
}

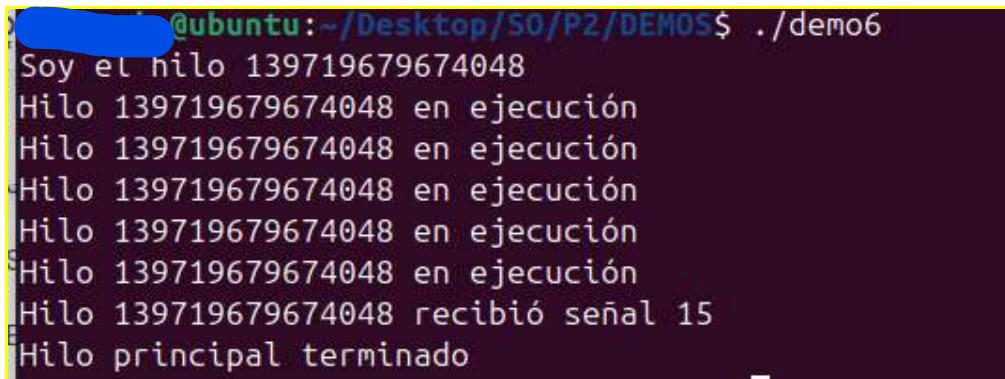
sleep(3);

printf("Hilo principal terminado\n");

return 0;
}

```

TERMINAL:



```

@ubuntu:~/Desktop/SO/P2/DEMOS$ ./demo6
Soy el hilo 139719679674048
Hilo 139719679674048 en ejecución
Hilo 139719679674048 recibió señal 15
Hilo principal terminado

```

HELLO_ARG_BAD_PARAMETER:

```

*****
* FILE: hello_arg_bad_parameter.c
* DESCRIPTION:
* This "hello world" Pthreads program demonstrates an unsafe (incorrect)
* way to pass thread arguments at thread creation. In this case, the
* argument variable is changed by the main thread as it creates new threads.

```

This example performs argument passing incorrectly. It passes the address of variable t, which is shared memory space and visible to all threads. As the loop iterates,

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah

the value of this memory location changes, possibly before the created threads can access it.

* AUTHOR: Blaise Barney
* Modified by Juan Carlos Fernandez Caballero

******/

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> //Para el sleep a modo divulgativo
#define NUM_THREADS 4
```

```
void * PrintHello(void * threadid)
{
    int * taskid;

    //sleep(1);
    taskid = (int *)threadid;

    printf("Hola desde la hebra %d...\n", *taskid);

    pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc, t, i, join_value;

    for(t=0;t<NUM_THREADS;t++)
    {
        printf("Hilo principal: creando hebra %d...\n", t);
        rc = pthread_create(&threads[t], NULL, (void *) PrintHello, (void *) &t);

        if (rc)
        {
            printf("ERROR; El codigo de error en pthread_create() es %d\n",
            rc);
            exit(EXIT_FAILURE);
        }
    }
}
```

wuolah

/*Pruebe a descomentar el sleep(), posiblemente ya si obtenga la salida que usted esperaba...*/

pero es una mala programacion para intentar sincronizar sus hilos.*/

```
//sleep();
```

/*Con el sleep() comentado, posiblemente las hebras impriman todas que su tarea es la 2, 3 o la 4, ya que al proceso principal le dará

tiempo a crear bastantes o incluso todas las hebras antes de que una de ellas empiece a ejecutar.*/

```
}
```

```
for(i=0;i<NUM_THREADS;i++)
```

```
{
```

```
    join_value = pthread_join(threads[i], NULL);
```

```
    if(join_value!=0)
```

```
{
```

```
    perror("Fallo en pthread_join()\n");
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```
}
```

```
}
```

TERMINAL:

```
123 ip@ubuntu:~/Desktop/S0/P2/DEMOS$ ./hello_arg_bad_parameter
Hilo principal: creando hebra 0...
Hilo principal: creando hebra 1...
Hilo principal: creando hebra 2...
Hilo principal: creando hebra 3...
Hola desde la hebra 4...
```

SAMPLE_FAIL:

```
/*
```

Programa que imprime un supuesto valor devuelto por una hebra y asignado a un long en el hijo principal.

Observe la salida y saque las conclusiones oportunas.

```
*/
```

```
#include <pthread.h>
```

```

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

void * th_function (void *arg)
{
    /* OJO, no estamos malloc() para reservar memoria en el monticulo de lo
     que se desea devolver */

    long y = 10;

    /*Tenemos que devolver un puntero, es decir, una direccion de memoria a un tipo de dato,
     y castearla a void ** */

    pthread_exit((void *)&y);
}

int main(int argc, char ** argv)
{
    int join_value;
    long rvalue;
    long *ret;
    pthread_t thread;

    if( pthread_create(&thread, NULL, th_function, NULL) )
    {
        fprintf(stderr, "Error creating thread\n");
        exit(EXIT_FAILURE);
    }

    //Recogemos el resultado devuelto por pthread_exit() en ret.
    join_value = pthread_join(thread,(void**) &ret);

    if(join_value!=0)
    {
        perror("Fallo en pthread_join()\n");
        exit(EXIT_FAILURE);
    }

    rvalue = *ret;
    printf("Proceso o hilo principal, el valor devuelto por la hebra es (basura): %ld\n", rvalue);
}

```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah

```
    exit(EXIT_SUCCESS);
}
```

TERMINAL:

```
[1001] lop@ubuntu:~/Desktop/S0/P2/DEMOS$ ./sample_FAIL
Proceso o hilo principal, el valor devuelto por la hebra es (basura): 130209206501376
```

SAMPLE_OK_ALTERNATIVA1:

```
/*
Alternativa al fichero sample_OK.c
```

```
Varía en los tipos de datos utilizados, fíjese en ellos  
y razoné el uso de los mismos.
```

```
/*
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
void * th_function (void *arg)
{
    long * y = malloc(sizeof(long));
    *y = 10;
    /*
```

A continuación estamos devolviendo un puntero a entero al que se le hace un casting a void.

Esto significa que estaremos devolviendo la dirección de memoria de un entero cuyo contenido es 10. Este entero se ha reservado en el montículo del proceso o hilo principal, que es accesible a todos los hilos creados.

```
    pthread_exit((void *)y);
}
```

```
int main(int argc, char ** argv)
{
    int join_value;
    long rvalue;
    void *ret;
    pthread_t thread;
    if( pthread_create(&thread, NULL, th_function, NULL) )
    {
        fprintf(stderr, "Error creating thread\n");
        exit(EXIT_FAILURE);
    }
```

wuolah

```

/* A pthread_join() le estamos pasando por referencia como segundo parámetro la dirección
de un puntero a entero. Al pasar la dirección de memoria de un puntero por referencia, el contenido
de ese puntero (a donde apunta) puede ser modificado. ¿Con qué se modifica? Con la dirección de
memoria de un entero que se devuelve en "pthread_exit((void *)y)". Por tanto, "long * ret" ahora
apunta a donde apuntaba "long * y", solo que está casteado a void. */
join_value = pthread_join(thread,&ret);

if(join_value!=0)
{
    perror("Fallo en pthread_join()\n");
    exit(EXIT_FAILURE);
}

rvalue = *(long *)ret; //Necesitamos hacer el correspondiente casting.
printf("Proceso o hilo principal, el valor devuelto por la hebra es: %ld\n", rvalue);

exit(EXIT_SUCCESS);
}

```

TERMINAL:

```

@ubuntu:~/Desktop/S0/P2/DEMOS$ ./sample_OK_alternativa1
Proceso o hilo principal, el valor devuelto por la hebra es: 10

```

SAMPLE_OK_ALTERNATIVA2:

```

/*
Alternativa al fichero sample.OK.c

Varía en los tipos de datos utilizados, fíjese en ellos
y razoné el uso de los mismos.

*/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

long * th_function (long *arg)
{
    long * y = malloc(sizeof(long));
    *y = 10;
    /*

```

A continuación estamos devolviendo un puntero a entero al que se le hace un casting a void.

Esto significa que estaremos devolviendo la dirección de memoria de un entero cuyo contenido

es 10. Este entero se ha reservado en el montículo del proceso o hilo principal, que es accesible

a todas los hilos creados.

```
*/  
pthread_exit(y);  
}
```

```
int main(int argc, char ** argv)  
{  
    int join_value;  
    long rvalue;  
    long *ret;  
    pthread_t thread;  
  
    if( pthread_create(&thread, NULL, (void *) th_function, NULL) )  
    {  
        fprintf(stderr, "Error creating thread\n");  
        exit(EXIT_FAILURE);  
    }
```

/* A pthread_join() le estamos pasando por referencia como segundo parámetro la dirección de un puntero a entero. Al pasar la dirección de memoria de un puntero por referencia, el contenido de ese puntero (a donde apunta) puede ser modificado. ¿Con qué se modifica? Con la dirección de memoria de un entero que se devuelve en "pthread_exit((void *)y)". Por tanto, "long * ret" ahora apunta a donde apuntaba "long * y", solo que está casteado a void.*/

```
    join_value = pthread_join(thread,(void **) &ret);  
  
    if(join_value!=0)  
    {  
        perror("Fallo en pthread_join()\n");  
        exit(EXIT_FAILURE);  
    }  
  
    rvalue = *ret;  
    printf("Proceso o hilo principal, el valor devuelto por la hebra es: %ld\n", rvalue);  
  
    exit(EXIT_SUCCESS);  
}
```

TERMINAL:

```
p@ubuntu:~/Desktop/S0/P2/DEMOS$ ./sample_OK_alternativa2  
Proceso o hilo principal, el valor devuelto por la hebra es: 10
```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah

SAMPLE_OK_ALTERNATIVA3:

```
/*  
Alternativa al fichero sample_OK.c.
```

Varía en los tipos de datos utilizados, fíjese en ellos
y razoné el uso de los mismos.

```
*/  
  
#include <pthread.h>  
  
#include <stdio.h>  
  
#include <stdlib.h>
```

```
long * th_function (long *arg)
```

```
{  
  
    long * y = malloc(sizeof(long));  
  
    *y = 10;  
  
    /*
```

A continuación estamos devolviendo un puntero a entero al que se le hace un casting a void.

Esto significa que estaremos devolviendo la dirección de memoria de un entero cuyo contenido es 10. Este entero se ha reservado en el montículo del proceso o hilo principal, que es accesible a todas los hilos creados.

```
*/  
  
    pthread_exit(y);  
}
```

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

```
{  
  
    int join_value;  
  
    long rvalue;  
  
    void *ret;  
  
    pthread_t thread;
```

```
    if( pthread_create(&thread, NULL, (void *) th_function, NULL) )
```

```
{  
  
    fprintf(stderr, "Error creating thread\n");  
  
    exit(EXIT_FAILURE);  
}
```

/* A pthread_join() le estamos pasando por referencia como segundo parámetro la dirección

de un puntero a entero. Al pasar la dirección de memoria de un puntero por referencia, el contenido

wuolah

de ese puntero (a donde apunta) puede ser modificado. ¿Con qué se modifica? Con la dirección de memoria de un entero que se devuelve en "pthread_exit((void *)y)". Por tanto, "long * ret" ahora apunta a donde apuntaba "long * y", solo que está casteado a void. */

```

join_value = pthread_join(thread,&ret);

if(join_value!=0)
{
    perror("Fallo en pthread_join()\n");
    exit(EXIT_FAILURE);
}

rvalue = *(long *)ret; //Necesitamos hacer el correspondiente casting.
printf("Proceso o hilo principal, el valor devuelto por la hebra es: %ld\n", rvalue);

exit(EXIT_SUCCESS);
}

```

TERMINAL:

```
p@ubuntu:~/Desktop/SO/P2/DEMOS$ ./sample_OK_alternativa3
Proceso o hilo principal, el valor devuelto por la hebra es: 10
```

SAMPLE_OK_INAPROPIADO1:

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

*****
Este ejercicio "aparentemente" funciona, pero lo que se está haciendo no va con la
filosofía y funcionamiento de los hilos con la biblioteca pthread. Por tanto, no debemos
utilizar esta metodología!!!.
```

Observe esto, es en lo que se basa este ejercicio:

```
ptr = (void *)5;
```

Now ptr points at the memory address 0x5

```
******/
```

```
void * th_function (void *arg)
{
    long y = 10;
    /* A continuación estamos devolviendo la dirección de memoria 0X10. ¿Por qué?
```

Porque le estamos haciendo un casting a puntero genérico a un tipo de dato que es un entero (no es la dirección de memoria de un entero). Ese casting automáticamente convierte el entero 10 a la dirección de memoria 0X10 */

```

pthread_exit((void *)y);
}

int main(int argc, char ** argv)
{
    int join_value;
    long rvalue;
    void *ret;
    pthread_t thread;
    pthread_create(&thread, NULL, th_function, NULL);

    /* Con este pthread_join() estamos asignando a ret una dirección de memoria. ¿Cuál?
    Pues la que se devolvió con pthread_exit((void *)y), es decir, 0X10. */

    join_value=pthread_join(thread,&ret);
    if(join_value!=0)
    {
        perror("Fallo en el join...\n");
        exit(EXIT_FAILURE);
    }

    /*Aquí viene la segunda "chapuza". Estamos haciendo un casting a long de una dirección de memoria,
    es decir, estamos convirtiendo 0X10 a 10. Por eso se muestra el resultado correcto, pero
    la metodología que hemos seguido no es correcta. Si en vez de enteros se usasen floats, la salida
    ya ni se mostraría falsamente correcta */

    rvalue = (long)ret;
    printf("Proceso o hilo principal, el valor devuelto por la hebra es: %ld\n", rvalue);

    exit(EXIT_SUCCESS);
}

```

TERMINAL:

```

[REDACTED]@ubuntu:~/Desktop/S0/P2/DEMOS$ ./sample_OK_inapropiado1
Proceso o hilo principal, el valor devuelto por la hebra es: 10

```

SAMPLE_OK_INAPROPIADO2:

```

#include <pthread.h>
#include <stdio.h>

```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



Necesito
concentración

ali ali oooh
esto con 1 coin me
lo quito yo...

wuolah

```
#include <stdlib.h>

/***********************/

Por lo mismo que se ha explicado en el ejercicio "sample_OK2_inapropiado.c" este ejemplo muestra la salida
correcta pero su implementación no es la apropiada. Por tanto no debemos programar con hebras
de esta manera. Con datos de tipo float ya no obtendría la salida aparentemente correcta.

Comparelo con "sample_OK2_inapropiado.c".

/***********************/

void * th_function (void *arg)
{
    long y = 10;

    pthread_exit((void *)y);
}

int main(int argc, char ** argv)
{
    int join_value;
    long rvalue;
    long *ret;
    pthread_t thread;

    pthread_create(&thread, NULL, th_function, NULL);

    join_value = pthread_join(thread,(void **) &ret);

    if(join_value!=0)
    {
        perror("Fallo en el join...\n");
        exit(EXIT_FAILURE);
    }

    rvalue = (long)ret;
    printf("Proceso o hilo principal, el valor devuelto por la hebra es: %ld\n", rvalue);
    exit(EXIT_SUCCESS);
}
```

wuolah

TERMINAL:

```
[user@ubuntu:~/Desktop/SO/P2/DEMOS$ ./sample_OK_inapropiado2
Proceso o hilo principal, el valor devuelto por la hebra es: 10
```

SAMPLE_OK:

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

void * th_function (void *arg)
{
    /* OJO: Siempre que vayamos a devolver algo desde una hebra usar malloc()
       para reservar memoria en el monticulo */

    long * y = malloc(sizeof(long));
    *y = 10;

    /* A continuación estamos devolviendo un puntero a entero al que se le hace un casting a void.
       Esto significa que estaremos devolviendo la dirección de memoria de un entero cuyo contenido
       es 10. Este entero se ha reservado en el montículo del proceso o hilo principal, que es accesible
       a todos los hilos creados.

    */
    pthread_exit((void *) y);
}

int main(int argc, char ** argv)
{
    int join_value;
    long rvalue;
    long *ret;
    pthread_t thread;

    if( pthread_create(&thread, NULL, (void *) th_function, NULL) )
    {
        fprintf(stderr, "Error creating thread\n");
        exit(EXIT_FAILURE);
    }

    /* A pthread_join() le estamos pasando por referencia como segundo parámetro la dirección
       de un puntero a entero. Al pasar la dirección de memoria de un puntero por referencia, el contenido
```

de ese puntero (a donde apunta) puede ser modificado. ¿Con qué se modifica? Con la dirección de memoria de un entero largo que se devuelve en "pthread_exit((void *)y)".

Por tanto, "long * ret" ahora apunta a donde apuntaba "long * y", solo que se ha casteado a void** */

```
join_value = pthread_join(thread,(void**) &ret);
```

```
if(join_value!=0)
{
    perror("Fallo en pthread_join()\n");
    exit(EXIT_FAILURE);
}

rvalue = *ret;
printf("Proceso o hilo principal, el valor devuelto por la hebra es: %ld\n", rvalue);

exit(EXIT_SUCCESS);
}
```

TERMINAL:

```
p@ubuntu:~/Desktop/SO/P2/DEMOS$ ./sample_OK
Proceso o hilo principal, el valor devuelto por la hebra es: 10
```

Importante

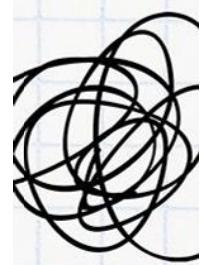
Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo
espacio



(=)



Necesito
concentración

ali ali ooooh
esto con 1 coin me
lo quito yo...

wuolah
~~wuolah~~

wuolah