EJERCICIOS REPASO PROCESOS

1.- Ejecuta el siguiente programa y analiza el resultado:

- Salida1:

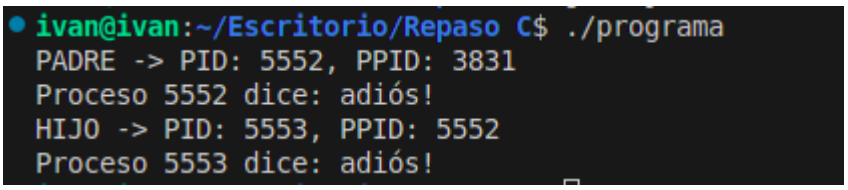
PID: 6389, PPID: 6126, valor devuelto por fork(): 6390
PID: 6390, PPID: 6389, valor devuelto por fork(): 0

- Explicación:

La primera línea es el padre porque su pid es mayor que 0 y la segunda línea es el hijo porque el pid es 0.

2.- Ejecuta el siguiente programa y analiza el resultado:

- Salida:



- Explicación:

El padre escribe su pid y el pid de su padre y continúa el programa despidiéndose; después el hijo escribe su pid y el pid del padre continúa el programa y se despide.

Ejercicio 3.- En el ejercicio anterior, evita que el hijo imprima el mensaje final.

En el código del hijo hay que añadir “exit(0);” de la siguiente manera:

```c
int main() {
pid_t pid = fork(); // Crea un nuevo proceso

if (pid < 0) {
perror("Error en fork");
return 1;
}
if (pid == 0) {
// PROCESO HIJO
 printf("HIJO -> PID: %d, PPID: %d\n", getpid(), getppid());
 exit(0);
} else {
// PROCESO PADRE
printf("PADRE -> PID: %d, PPID: %d\n", getpid(), getppid());
}
// ESTA PARTE LA EJECUTAN **AMBOS** PROCESOS
printf("Proceso %d dice: adiós!\n", getpid());
return 0;
```

```
}
```

-Salida:

```
ivan@ivan:~/Escritorio/Repaso C$ ./programa
PADRE -> PID: 5395, PPID: 3831
Proceso 5395 dice: adiós!
HIJO -> PID: 5396, PPID: 5395
```

Ejercicio 4.- Crea un programa que: Use fork() y el padre imprima los números del 1 al 5, el hijo imprima los números del 6 al 10. Observa y explica por qué el orden de salida puede variar en cada ejecución.

Este programa crea dos procesos usando fork(): el padre y el hijo.
El proceso padre imprime los números del 1 al 5 y el hijo imprime del 6 al 10.
Ambos terminan ejecutando el return 0, pero cada uno hace su bucle correspondiente.

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

int main() {
pid_t pid = fork(); // Crea un nuevo proceso

if (pid < 0) {
perror("Error en fork");
return 1;
}
if (pid == 0) {
// PROCESO HIJO
   for (int i = 6; i <= 10; i++){
       printf("HIJO -> %d\n", i);
   }
} else {
   // PROCESO PADRE
   for (int i = 1; i <= 5; i++){
       printf("PADRE -> %d\n", i);
   }
}
```

```
// ESTA PARTE LA EJECUTAN **AMBOS** PROCESOS
return 0;
}
```

- Salida:

```
ivan@ivan:~/Escritorio/Repaso C$ ./programa
PADRE -> 1
PADRE -> 2
PADRE -> 3
PADRE -> 4
PADRE -> 5
HIJO -> 6
HIJO -> 7
HIJO -> 8
HIJO -> 9
HIJO -> 10
```

Ejercicio5.- Crea un programa con un bucle en el que se ejecute un fork() y cree cuatro procesos hijos que impriman un mensaje. Debes evitar que un proceso hijo vuelva a duplicarse. El proceso padre debe esperar a la finalización de todos los hijos e imprimir un mensaje de salida. Comprueba que ocurre si los hijos no finalizan su ejecución (un bucle infinito).

El bucle crea 4 procesos hijo usando fork().
Cada hijo imprime el mensaje y termina inmediatamente con _exit(0) para no seguir creando más procesos.
El padre espera a que todos los hijos acaben con wait() y luego muestra “Programa finalizado”.

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

int main() {

       for(int i=0; i<4; i++){
           pid_t pid = fork();

           if (pid < 0) {
           perror("Error en fork");
           return 1;
```

```
            }

            if (pid == 0) {
            // PROCESO HIJO
                printf("HIJO -> %d\n", pid);
                _exit(0);
        }
    }
    for(int i=0; i<=4; i++){
        wait(NULL);
    }
    printf("Programa finalizado\n");
    return 0;

}
```

- Salida:

```
ivan@ivan:~/Escritorio/Repaso C$ ./programa
HIJO -> 0
HIJO -> 0
HIJO -> 0
HIJO -> 0
Programa finalizado
```

Ejercicio 6.- Usando IA genera un resumen de como comunicar procesos usando un PIPE. Puntos a tener en cuenta.: Crear Pipe Cerrar descriptores no usados. Como leer información en el pipe. Diferenciar escribir una cadena y un numero. Cómo escribir información en un pipe.

📌 ¿Qué es un PIPE?

Un pipe es un mecanismo de comunicación entre procesos que permite que la salida de un proceso sea la entrada de otro, funcionando como un canal unidireccional de datos.

🔧 1. Crear un pipe

Se crea con la llamada al sistema:

int fd[2];
pipe(fd);

fd[0] → extremo de lectura

fd[1] → extremo de escritura

Debe crearse antes de hacer fork() para que ambos procesos lo compartan.

🚪 2. Cerrar descriptores no usados

Es fundamental cerrar los extremos que no se usan, para evitar bloqueos y fugas de recursos.

Proceso padre (escribe):

close(fd[0]); // No lee

Proceso hijo (lee):

close(fd[1]); // No escribe

✔️ Si no se cierran correctamente:

El proceso lector puede quedarse esperando indefinidamente.

El sistema no detecta correctamente el fin de datos (EOF).

📥 3. Cómo leer información de un pipe

La lectura se hace con read():

read(fd[0], buffer, tamaño);

🔹 Leer una cadena

Se usa un buffer de caracteres.

Es importante asegurar el carácter nulo \0 para tratarla como string.

```
char buffer[50];
read(fd[0], buffer, sizeof(buffer));
```

🔹 Leer un número

Se lee directamente en una variable del tipo adecuado.

```
int numero;
read(fd[0], &numero, sizeof(int));
```

📌 Diferencia clave:

Cadenas → array de char

Números → variable del tipo (int, float, etc.)

📤 4. Cómo escribir información en un pipe

La escritura se realiza con write():

write(fd[1], dato, tamaño);

🔹 Escribir una cadena

Se envía el contenido del array.

Normalmente se incluye el \0.

```
char mensaje[] = "Hola proceso";
write(fd[1], mensaje, sizeof(mensaje));
```

🔹 Escribir un número

Se pasa la dirección de la variable.

```
int numero = 10;
write(fd[1], &numero, sizeof(int));
```

⚠️ Puntos importantes a tener en cuenta

Un pipe es unidireccional.

El orden de lectura debe coincidir con el orden de escritura.

read() se bloquea si no hay datos.

Siempre cerrar los descriptores cuando ya no se usen.

Pipes solo funcionan entre procesos relacionados (padre-hijo).

✅ Resumen rápido

✔ Crear pipe antes de fork()

✔ Cerrar extremos no usados

✔ Usar read() y write()

✔ Diferenciar entre cadenas y tipos numéricos

✔ Mantener orden y tamaños correctos

Ejercicio 7.- Crea un programa que realice la siguiente tarea.:

- **El padre envíe un número entero al hijo mediante un pipe.**
- **El hijo calcule el cuadrado del número.**
- **El hijo muestre el resultado por pantalla.**

Este programa usa un pipe para comunicar un padre con un hijo.
El padre genera un número aleatorio entre 1 y 10 y se lo envía al hijo.
El hijo lo recibe, calcula su cuadrado y lo muestra por pantalla.

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

int main() {
    int fd[2];
    int numero;
    pid_t pid;

    // Inicializa srand para los números aleatorios
    srand(time(NULL));

    // Crear el pipe
    if (pipe(fd) == -1) {
        perror("Error al crear el pipe");
        return 1;
    }

    pid = fork();

    if (pid > 0) {
        //PROCESO PADRE
        close(fd[0]); // Cierra lectura

        numero = rand() % 10 + 1; // Número aleatorio entre 1 y
10
        printf("PADRE -> Numero generado: %d\n", numero);

        write(fd[1], &numero, sizeof(int));
```

```
            close(fd[1]); // Cierra escritura
        }
        else if (pid == 0) {
            //PROCESO HIJO
            close(fd[1]); // Cierra escritura

            read(fd[0], &numero, sizeof(int));

            int cuadrado = numero * numero;
            printf("HIJO -> El cuadrado del numero es: %d\n",
cuadrado);

            close(fd[0]); // Cierra lectura
        }

        return 0;
    }
```

- Salida:

```
ivan@ivan:~/Escritorio/Repaso C$ ./programa
PADRE -> Numero generado: 9
HIJO -> El cuadrado del numero es: 81
```

Ejercicio 8.- Usa dos pipes para comunicación bidireccional.

- **El padre envíe un número al hijo.**
- **El hijo devuelva el doble del número al padre.**
- **El padre muestre el resultado recibido.**

Este programa crea dos pipes para que el padre y el hijo se comuniquen en ambos sentidos.
El padre envía un número aleatorio al hijo y el hijo lo duplica y se lo devuelve.
El padre recibe el resultado, lo muestra por pantalla y espera a que el hijo termine.

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

int main() {

    int pipe_ph[2]; // Padre -> Hijo
    int pipe_hp[2]; // Hijo -> Padre
    int numero, resultado;

    // Inicializar srand para los números aleatorios
    srand(time(NULL));

    // Crear los pipes
    if (pipe(pipe_ph) == -1 || pipe(pipe_hp) == -1) {
        perror("Error al crear los pipes");
        return 1;
    }

    for (int i = 0; i < 1; i++) {
        pid_t pid = fork();

        if (pid < 0) {
            perror("Error en fork");
            return 1;
        }

        if (pid == 0) {
            // ===== PROCESO HIJO =====
```

```
                close(pipe_ph[1]); // No escribe al padre
                close(pipe_hp[0]); // No lee del padre

                read(pipe_ph[0], &numero, sizeof(int));

                resultado = numero * 2;

                write(pipe_hp[1], &resultado, sizeof(int));

                close(pipe_ph[0]);
                close(pipe_hp[1]);

                _exit(0);
            }
            else {
                // ===== PROCESO PADRE =====
                close(pipe_ph[0]); // No lee del hijo
                close(pipe_hp[1]); // No escribe al hijo

                numero = rand() % 10 + 1; // Número aleatorio entre 1
y 10
                printf("PADRE -> Numero generado: %d\n", numero);

                write(pipe_ph[1], &numero, sizeof(int));

                read(pipe_hp[0], &resultado, sizeof(int));
                printf("PADRE -> El doble recibido es: %d\n",
resultado);

                close(pipe_ph[1]);
                close(pipe_hp[0]);
            }
        }

        for (int i = 0; i < 1; i++) {
            wait(NULL);
        }

        printf("Programa finalizado\n");
        return 0;
    }
```

- Salida:

```
ivan@ivan:~/Escritorio/Repaso C$ ./programa
PADRE -> Numero generado: 1
PADRE -> El doble recibido es: 2
Programa finalizado
```

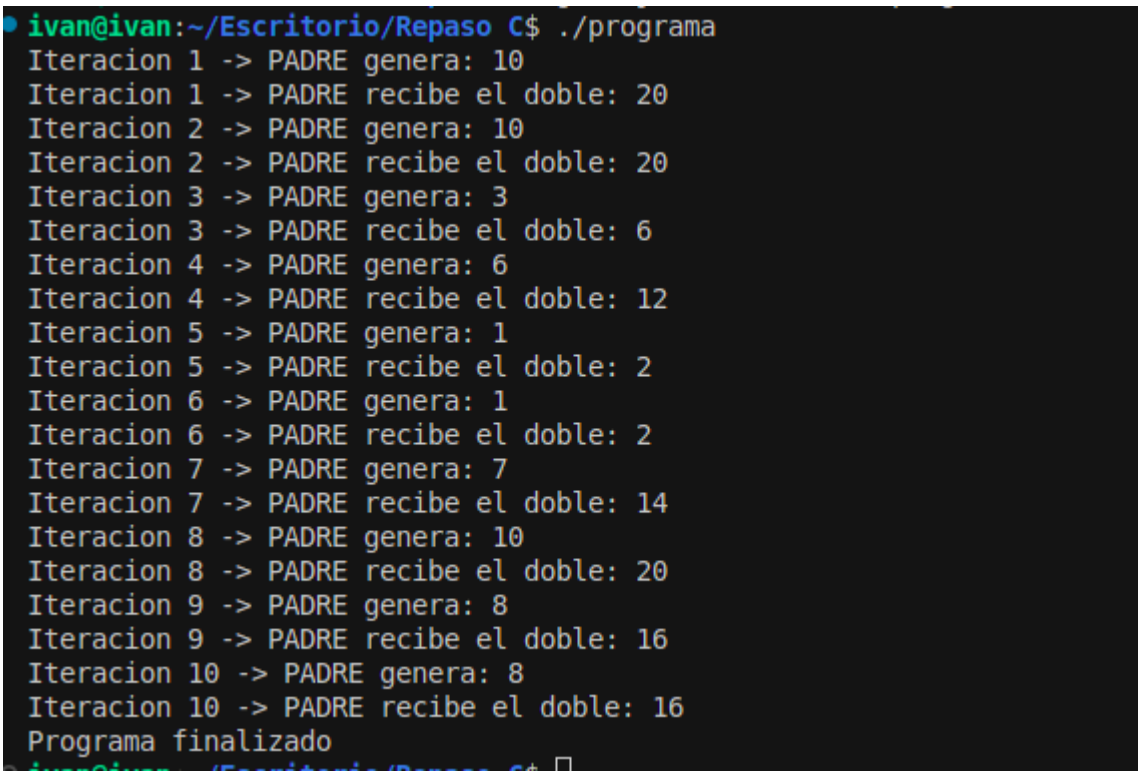
Ejercicio 9.- Un programa que repita el proceso anterior 10 veces.

Este programa usa dos pipes para comunicarse varias veces entre padre e hijo.
El padre envía 10 números aleatorios y el hijo devuelve el doble de cada uno.
Tras completar las 10 iteraciones, ambos cierran los pipes y el programa finaliza.

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

int main() {

    int pipe_ph[2]; // Padre -> Hijo
    int pipe_hp[2]; // Hijo -> Padre
    int numero, resultado;

    // Inicializar semilla de números aleatorios
    srand(time(NULL));

    // Crear los pipes
    if (pipe(pipe_ph) == -1 || pipe(pipe_hp) == -1) {
        perror("Error al crear los pipes");
        return 1;
    }

    pid_t pid = fork();

    if (pid < 0) {
        perror("Error en fork");
        return 1;
    }

    if (pid == 0) {
```

```
        // ===== PROCESO HIJO =====
        close(pipe_ph[1]); // No escribe al padre
        close(pipe_hp[0]); // No lee del padre

        for (int i = 0; i < 10; i++) {
            read(pipe_ph[0], &numero, sizeof(int));

            resultado = numero * 2;

            write(pipe_hp[1], &resultado, sizeof(int));
        }

        close(pipe_ph[0]);
        close(pipe_hp[1]);
        _exit(0);
    }
    else {
        // ===== PROCESO PADRE =====
        close(pipe_ph[0]); // No lee del hijo
        close(pipe_hp[1]); // No escribe al hijo

        for (int i = 1; i <= 10; i++) {
            numero = rand() % 10 + 1;
            printf("Iteracion %d -> PADRE genera: %d\n", i, numero);

            write(pipe_ph[1], &numero, sizeof(int));

            read(pipe_hp[0], &resultado, sizeof(int));
            printf("Iteracion %d -> PADRE recibe el doble: %d\n", i,
resultado);
        }

        close(pipe_ph[1]);
        close(pipe_hp[0]);
        wait(NULL);
    }

    printf("Programa finalizado\n");
    return 0;
}
```

- Salida:


ivan@ivan:~/Escritorio/Repaso C$ ./programa
Iteracion 1 -> PADRE genera: 10
Iteracion 1 -> PADRE recibe el doble: 20
Iteracion 2 -> PADRE genera: 10
Iteracion 2 -> PADRE recibe el doble: 20
Iteracion 3 -> PADRE genera: 3
Iteracion 3 -> PADRE recibe el doble: 6
Iteracion 4 -> PADRE genera: 6
Iteracion 4 -> PADRE recibe el doble: 12
Iteracion 5 -> PADRE genera: 1
Iteracion 5 -> PADRE recibe el doble: 2
Iteracion 6 -> PADRE genera: 1
Iteracion 6 -> PADRE recibe el doble: 2
Iteracion 7 -> PADRE genera: 7
Iteracion 7 -> PADRE recibe el doble: 14
Iteracion 8 -> PADRE genera: 10
Iteracion 8 -> PADRE recibe el doble: 20
Iteracion 9 -> PADRE genera: 8
Iteracion 9 -> PADRE recibe el doble: 16
Iteracion 10 -> PADRE genera: 8
Iteracion 10 -> PADRE recibe el doble: 16
Programa finalizado

Ejercicio 10.- Crea un programa en el que • El padre envíe una cadena al hijo y el hijo la escriba al revés.

Este programa usa un pipe para enviar una cadena de texto del padre al hijo.
El padre manda la frase y el hijo la invierte carácter a carácter.
Finalmente, el hijo muestra la cadena al revés y el programa termina.

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

int main() {

    int fd[2];
    char cadena[100];

    // Crear el pipe
    if (pipe(fd) == -1) {
        perror("Error al crear el pipe");
        return 1;
    }

    pid_t pid = fork();

    if (pid < 0) {
        perror("Error en fork");
        return 1;
    }

    if (pid == 0) {
        // ===== PROCESO HIJO =====
        close(fd[1]); // No escribe

        read(fd[0], cadena, sizeof(cadena));

        int len = strlen(cadena);
        for (int i = 0; i < len / 2; i++) {
            char tmp = cadena[i];
            cadena[i] = cadena[len - 1 - i];
```

```
            cadena[len - 1 - i] = tmp;
        }

        printf("HIJO -> Cadena al reves: %s\n", cadena);

        close(fd[0]);
        _exit(0);
    }
    else {
        // ===== PROCESO PADRE =====
        close(fd[0]); // No lee

        strcpy(cadena, "Programacion de procesos");
        write(fd[1], cadena, strlen(cadena) + 1);
        printf("PADRE -> Cadena: %s\n", cadena);

        close(fd[1]);
        wait(NULL);
    }

    printf("Programa finalizado\n");
    return 0;
}
```

- Salida:

```
ivan@ivan:~/Escritorio/Repaso C$ ./programa
PADRE -> Cadena: Programacion de procesos
HIJO -> Cadena al reves: sosecorp ed noicamargorP
Programa finalizado
```

Ejercicio 11.- Crea un programa en el que

- **El padre esté pidiendo números por teclado de forma continua.**
- **cuando se introduzca un número negativo el padre mandará una señal sigusr1 al hijo**
- **El hijo capturará la señal e imprimirá un mensaje por pantalla.**

Este programa usa señales para comunicar a padre e hijo.
El padre pide números y, cuando se introduce uno negativo, envía una señal SIGUSR1 al hijo.
El hijo recibe la señal utilizando el manejador, muestra un mensaje y termina el programa.

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

// Manejador de la señal en el hijo
void manejador(int sig) {
    if (sig == SIGUSR1) {
        printf("HIJO -> Señal SIGUSR1 recibida. Numero negativo
detectado.\n");
        exit(0);
    }
}

int main() {

    pid_t pid;
    int numero;

    pid = fork();

    if (pid < 0) {
        perror("Error en fork");
        return 1;
    }

    if (pid == 0) {
        // ===== PROCESO HIJO =====
        signal(SIGUSR1, manejador);

        // El hijo espera señales
```

```
        while (1) {
            pause();
        }
    }
    else {
        // ===== PROCESO PADRE =====
        while (1) {
            printf("Introduce un numero: ");
            scanf("%d", &numero);

            if (numero < 0) {
                kill(pid, SIGUSR1);
                break;
            }
        }

        wait(NULL);
        printf("Programa finalizado\n");
    }

    return 0;
}
```

- Salida:

```
ivan@ivan:~/Escritorio/Repaso C$ ./programa
PADRE -> Numero generado: -2
Programa finalizado
```

Ejercicio 12.- Modifica el programa anterior para que en el mensaje inscrito por el hijo aparezca el numero negativo introducido por el padre.

Este programa combina pipes y señales para comunicar a padre e hijo.
El padre pide números y los envía utilizando el pipe al hijo, cuando se introduce uno negativo,manda una señal al hijo.
El hijo recibe la señal utilizando el manejador, lee el número negativo del pipe, lo muestra y termina el programa.

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

int fd[2];      // Pipe global
int numero;     // Variable global para el hijo

// Manejador de la señal en el hijo
void manejador(int sig) {
    if (sig == SIGUSR1) {
        read(fd[0], &numero, sizeof(int));
        printf("HIJO -> Señal SIGUSR1 recibida. Numero negativo: %d\n",
numero);
        exit(0);
    }
}

int main() {

    pid_t pid;

    // Crear el pipe
    if (pipe(fd) == -1) {
        perror("Error al crear el pipe");
        return 1;
    }

    pid = fork();

    if (pid < 0) {
        perror("Error en fork");
        return 1;
```

```c
    }

    if (pid == 0) {
        // ===== PROCESO HIJO =====
        close(fd[1]); // No escribe
        signal(SIGUSR1, manejador);

        while (1) {
            pause(); // Espera señales
        }
    }
    else {
        // ===== PROCESO PADRE =====
        close(fd[0]); // No lee

        while (1) {
            printf("Introduce un numero: ");
            scanf("%d", &numero);

            if (numero < 0) {
                write(fd[1], &numero, sizeof(int));
                kill(pid, SIGUSR1);
                break;
            }
        }

        close(fd[1]);
        wait(NULL);
        printf("Programa finalizado\n");
    }

    return 0;
}
```

- Salida:

```
ivan@ivan:~/Escritorio/Repaso C$ ./programa
Introduce un numero: 2
Introduce un numero: 2
Introduce un numero: -2
HIJO -> Señal SIGUSR1 recibida. Numero negativo: -2
Programa finalizado
```

Ejercicio 13.- Crea un script alarma.sh, basta con que imprima un mensaje por pantalla. #!/bin/bash echo "ALARMA: se ha recibido un numero negativo: $1" no olvides darle permisos de ejecución chmod +x alarma.sh Volvamos al programa anterior..

- **El padre pide números de forma continua.**
- **Cuando recibe uno negativo envía una señal al hijo.**
- **Este la captura y ejecuta el script. Investiga que pasa con el proceso hijo y el padre al ejecutar el script. ¿Qué ocurre con los procesos padre e hijo al ejecutar con exec( ) el script?**

- El archivo alarma.sh queda así:

```
#!/bin/bash
echo "ALARMA: se ha recibido un numero negativo: $1"
```

Este programa usa pipes y señales para pasar un número del padre al hijo.
Cuando al padre le llega un número lo envía utilizando el pipe al hijo, el cual imprime por pantalla ese mismo número, pero cuando se introduce un número negativo, el padre envía la señal SIGUSR1 utilizando kill.
El hijo recibe la señal utilizando un manejador, lee el número y ejecuta el script alarma.sh usando execl.

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

int fd[2];      // Pipe global
int numero;     // Variable global para el hijo

// Manejador de la señal en el hijo
void manejador(int sig) {
    if (sig == SIGUSR1) {
        read(fd[0], &numero, sizeof(int));
        execl("./alarma.sh", "alarma.sh", numero, NULL);
        exit(0);
    }
}

int main() {
```

```
pid_t pid;

// Crear el pipe
if (pipe(fd) == -1) {
    perror("Error al crear el pipe");
    return 1;
}

pid = fork();

if (pid < 0) {
    perror("Error en fork");
    return 1;
}

if (pid == 0) {
    // ===== PROCESO HIJO =====
    close(fd[1]); // No escribe
    signal(SIGUSR1, manejador);

    while (1) {
        pause(); // Espera señales
    }
}
else {
    // ===== PROCESO PADRE =====
    close(fd[0]); // No lee

    while (1) {
        printf("Introduce un numero: ");
        scanf("%d", &numero);

        if (numero < 0) {
            write(fd[1], &numero, sizeof(int));
            kill(pid, SIGUSR1);
        }
    }

    close(fd[1]);
    wait(NULL);
    printf("Programa finalizado\n");
}
```

```
    return 0;
}
```

- Salida:

```
ivan@ivan:~/Escritorio/Repaso C$ ./programa
Introduce un numero: -4
ALARMA: se ha recibido un numero negativo: -4
Programa finalizado
```

```
ivan@ivan:~/Escritorio/Repaso C$ ./programa
Introduce un numero: 3
Introduce un numero: HIJO -> Mi padre me ha mandado un: 3
3
Introduce un numero: HIJO -> Mi padre me ha mandado un: 3
3
Introduce un numero: HIJO -> Mi padre me ha mandado un: 3
-3
ALARMA: se ha recibido un numero negativo: -3
Programa finalizado
```