

Ing. de software

Tema

Práctica 5

Docente: Maribel Molina Barriga

Curso: Sistemas Operativos

INTEGRANTES

Castro Mamani Sebastian Adriano

Huamaní Vasquez Juan José

Yabar Carazas Melvin

Zela Flores Gabriel Frank

Semestre: V

Arequipa - Perú

04 de junio de 2025

EJERCICIOS PROPUESTOS

1. Analice y describa la actividad que realiza el siguiente código. Explique cómo sucede el proceso de envío de información del proceso padre al proceso hijo.

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

int main() {
    pid_t idProceso;
    int estadoHijo;
    int descriptorTuberia[2];
    char buffer[100];

    if (pipe (descriptorTuberia) == - 1) {
        perror ("No se puede crear Tuberia");
        exit( -1);
    }
    idProceso = fork();

    if (idProceso == - 1) {
        perror ("No se puede crear proceso");
        exit( -1);
    }

    if (idProceso == 0) {
        close (descriptorTuberia[1]);
        read (descriptorTuberia[0], buffer, 9);
        printf ("Hijo : Recibido \"%s\\n\"", buffer);
        exit(0);
    }

    if (idProceso > 0) {
        close (descriptorTuberia[0]);
        printf ("Padre : Envio \"Sistemas\\n\"");
        strcpy (buffer, "Sistemas");
        write (descriptorTuberia[1], buffer, strlen(buffer)+1);

        wait(&estadoHijo);
    }
}
```

```

        exit(0);
    }
    return(1);
}

```

La comunicación se efectúa de la siguiente manera:

Los primeros dos `if` son verificaciones para saber si la tubería y el proceso hijo se crearon correctamente, luego en `if (idProceso == 0)` está la ejecución del proceso hijo y en `if (idProceso > 0)` está el proceso padre.

Flujo de Comunicación:

Primero el padre cierra la tubería por donde recibe información (`close(descriptorTuberia[0])`), luego copia en el **buffer** la palabra "Sistemas" y finalmente escribe el mensaje en la parte de la tubería donde manda la información `write(descriptorTuberia[1], buffer, strlen(buffer)+1)`.

Después el proceso hijo cierra la tubería por donde manda la información ya que no lo hará `close(descriptorTuberia[1])`, luego, espera y recibe la información del padre `read(descriptorTuberia[0], buffer, 9)`, finalmente se lanzan mensajes comprobando el flujo de la información.

```

Padre : Envio "Sistemas"
Hijo : Recibido "Sistemas"

...Program finished with exit code 0
Press ENTER to exit console.

```

Link para probar el código: <https://onlinegdb.com/wEjTuo2ly>

2. Analice y describa la actividad que realiza el siguiente código. Explique como sucede el proceso de envío de información del proceso padre al proceso hijo

```

#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

#define LEER 0
#define ESCRIBIR 1

char *frase = "Envia esto a traves de un tubo o pipe";
extern int errno;

int main(){
    int fd[2], bytesLeidos;
    char mensaje[100];

```

```

int control;

// se crea la tubería
control = pipe(fd);
if ( control != 0 ) {
    perror("pipe:");
    exit(errno);
}

// se crea el proceso hijo
control = fork() ;
switch(control) {
    case - 1 :
        perror("fork:");
        exit(errno);
    case 0 :
        close(fd[LEER]);
        write(fd[ESCRIBIR], frase, strlen(frase) + 1);
        close(fd[ESCRIBIR]);
        exit(0);
    default :
        close(fd[ESCRIBIR]);
        bytesLeidos = read(fd[LEER], mensaje, 100);
        printf("Leídos %d bytes : %s\n", bytesLeidos, mensaje);
        close(fd[LEER]);
}
exit(0);
}

```

Errores corregidos:

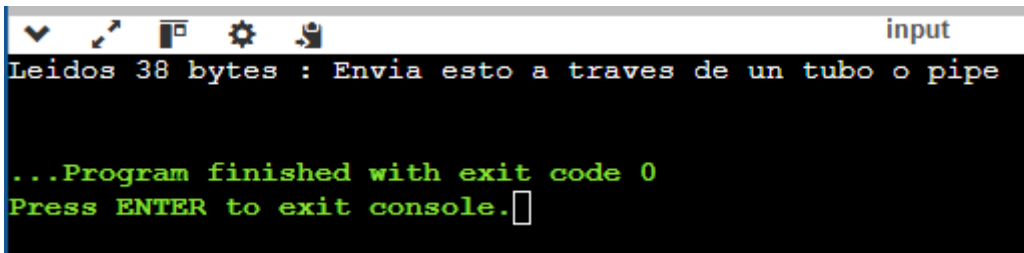
Se cambió la línea 10, de **char *frase** a **const char *frase** para que el código se ejecute correctamente

Flujo de Comunicación:

Primero se crea la tubería **control = pipe(fd)**, luego se crea el proceso hijo **control = fork()**, justo después vemos como se hace un switch con los casos (0,-1,default) esto viene en remplazo de múltiples if del caso anterior. El flujo de comunicación sigue así:

El proceso padre cierra el extremo de escritura **close(fd[ESCRIBIR])** y luego espera el mensaje.

El proceso hijo cierra el extremo de lectura **close(fd[LEER])** y luego escribe el mensaje **write(fd[ESCRIBIR], frase, strlen(frase) + 1)**, finalmente cierra el extremo de escritura y volvemos al proceso padre, el cuál recibe el mensaje y lo imprime.



```

input
Leídos 38 bytes : Envía esto a través de un tubo o pipe

...Program finished with exit code 0
Press ENTER to exit console.

```

Código para probar: <https://onlinegdb.com/ojFXfM1fh>

3. Elabore un programa propio que emplee la comunicación entre procesos (padre e hijo) utilizando pipes.

// GITHUB

4. Investigue cómo se pueden enviar datos de un proceso padre a un proceso hijo y viceversa.

En los sistemas operativos existen diversas formas de lograr la comunicación entre procesos, conocido como IPC, incluyendo la comunicación entre procesos padre e hijo. La elección del método de comunicación siempre va a depender de las funcionalidades o necesidades que se priorice, como el número de datos, persistencia de comunicación, etc. Los siguientes ejemplos son formas en las que nosotros, como programadores, podemos comunicar procesos entre sí.

1. Pipes Anónimos (Unnamed Pipes)

- a. Son la forma más sencilla y común de comunicación unidireccional entre procesos relacionados (es decir, aquellos que comparten un ancestro común, como padre e hijo después de un **fork**). Los datos fluyen de un extremo a otro, como una tubería.
- b. **Ventajas:** Simples de usar, eficientes para datos pequeños o flujos de datos.
- c. **Desventajas:** Unidireccionales, solo para procesos relacionados, no persistentes.

2. Pipes con Nombre (Named Pipes o FIFOs)

- a. Son similares a los pipes anónimos, pero tienen un nombre en el sistema de archivos (se representan como archivos especiales). Esto permite que procesos no relacionados (que no comparten un ancestro común) se comuniquen.
- b. **Ventajas:** Permiten la comunicación entre procesos no relacionados, persisten en el sistema de archivos hasta que son eliminados.
- c. **Desventajas:** Siguen siendo unidireccionales .

3. Señales

- a. Es un mecanismo ligero para notificar a un proceso sobre un evento. No transmiten datos complejos, solo una notificación.
- b. **Ventajas:** Simples y eficientes para notificaciones de eventos.

- c. **Desventajas:** No aptas para la transmisión de datos, solo para eventos.

4. Memoria Compartida (Shared Memory)

- a. Este método nos permite que varios procesos accedan a la misma región de memoria RAM. Es la forma más rápida de IPC, ya que los datos no necesitan ser copiados entre el espacio de usuario y el kernel.
- b. **Ventajas:** Muy alta velocidad de comunicación, ideal para grandes volúmenes de datos.
- c. **Desventajas:** Requiere mecanismos de sincronización explícitos (como semáforos o) para evitar condiciones de carrera al acceder a la memoria compartida. Si no se sincroniza adecuadamente, los datos pueden corromperse.

IV

BIBLIOGRAFÍA

<https://www.programacion.com.py/escritorio/c/pipes-en-c-linux>

<https://www.geeksforgeeks.org/inter-process-communication-ipc/>

<https://linasm.sourceforge.net/docs/syscalls/index.php>