

UNIDAD 1. PROGRAMACIÓN MULTIPROCESO

EJERCICIOS GUIADOS DE LA UNIDAD

EJERCICIO 1. Escribe el siguiente programa en C en el editor gedit. Nombralo como ejemploSystem.c. Compila y ejecuta el programa en Linux y comprueba el resultado obtenido. Añade una captura de pantalla del resultado.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    printf("Ejemplo de uso de system():");
    printf("\n\tListado del directorio actual y envío a un fichero:");
    printf("%d",system("ls > ficsalida"));
    printf("\n\tAbrimos con el gedit el fichero...");

    printf("%d",system("gedit ficsalida"));
    printf("\n\tEste comando es erróneo: %d",system("ged"));
    printf("\nFin de programa....\n");
}
```

EJERCICIO 2. Escribe el siguiente programa en C en el editor gedit. Nombralo como ejemploExec.c. Compila y ejecuta el programa en Linux y comprueba el resultado obtenido. Añade una captura de pantalla del resultado.

```
#include <unistd.h>
#include <stdio.h>
void main()
{
    printf("Ejemplo de uso de exec():");
    printf("Los archivos en el directorio son:\n");
    execl("/bin/ls", "ls", "-l", (char *)NULL);
    printf("!!! Esto no se ejecuta !!!\n");
}
```

EJERCICIO 3. Escribe el siguiente programa en C en el editor gedit. Nombralo como ejemploPadres.c. Compila y ejecuta el programa en Linux y comprueba el resultado obtenido. Añade una captura de pantalla del resultado.

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

void main(void)
{
    pid_t id_pactual, id_padre;

    id_pactual = getpid();
    id_padre = getppid();

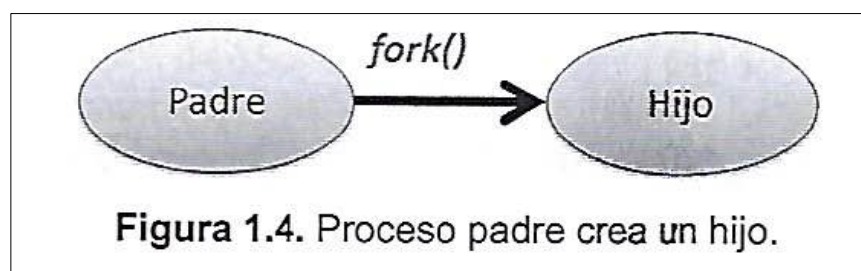
    printf("PID de este proceso: %d\n", id_pactual);
    printf("PID del proceso padre: %d\n", id_padre);
}
```

Creación y ejecución de procesos

EJERCICIO 4. Escribe el siguiente programa en C en el editor gedit. Nombralo como ejemploFork.c. Compila y ejecuta el programa en Linux y comprueba el resultado obtenido. Añade una captura de pantalla del resultado.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
void main() {
    pid_t pid, Hijo_pid;
    pid = fork();

    if (pid == -1 ) //Ha ocurrido un error
    {
        printf("No se ha podido crear el proceso hijo...");
        exit(-1);
    }
    if (pid == 0 ) //Nos encontramos en Proceso hijo
    {
        printf("Soy el proceso hijo \n\t
            Mi PID es %d, El PID de mi padre es: %d.\n",
            getpid(), getppid() );
    }
    else //Nos encontramos en Proceso padre
    {
        Hijo_pid = wait(NULL); //espera la finalización del proceso hijo
        printf("Soy el proceso padre:\n\t
            Mi PID es %d, El PID de mi padre es: %d.\n\t
            Mi hijo: %d terminó.\n",
            getpid(), getppid(), pid);
    }
    exit(0);
}
```



Comunicación entre procesos

EJERCICIO 5. Escribe el siguiente programa en C en el editor gedit. Nombralo como **ejemploWriteRead.c**. Compila y ejecuta el programa en Linux y comprueba el resultado obtenido. Añade una captura de pantalla del resultado.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main(void)
{
    char saludo[] = "Un saludo!!!\n";
    char buffer[10];
    int fd, bytesleidos;

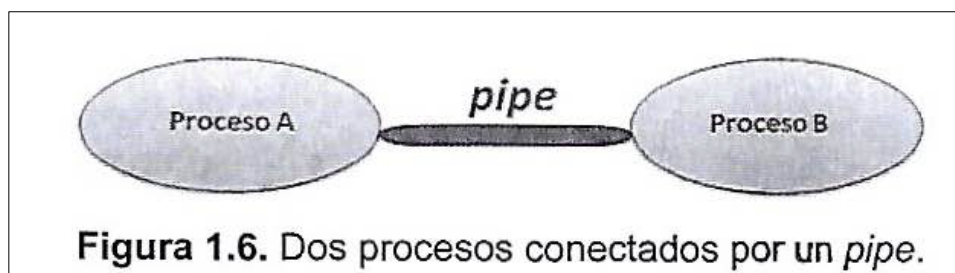
    fd=open("texto.txt",1);//fichero se abre solo para escritura

    if( fd == -1 )
    {
        printf("ERROR AL ABRIR EL FICHERO...\n");
        exit(-1);
    }

    printf("Escribo el saludo...\n");
    write(fd,saludo, strlen(saludo));
    close(fd); //cierro el fichero

    fd=open("texto.txt",0);//el fichero se abre solo para lectura
    printf("Contenido del Fichero: \n");

    //leo bytes de uno en uno y lo guardo en buffer
    bytesleidos= read(fd, buffer, 1);
    while (bytesleidos!=0){
        printf("%lc", buffer[0]); //pinto el byte leído
        bytesleidos= read(fd, buffer, 1);//leo otro byte
    }
    close(fd);
}
```



EJERCICIO 6. Escribe el siguiente programa en C en el editor gedit. Nombralo como ejemploPipe.c. Compila y ejecuta el programa en Linux y comprueba el resultado obtenido. Añade una captura de pantalla del resultado.

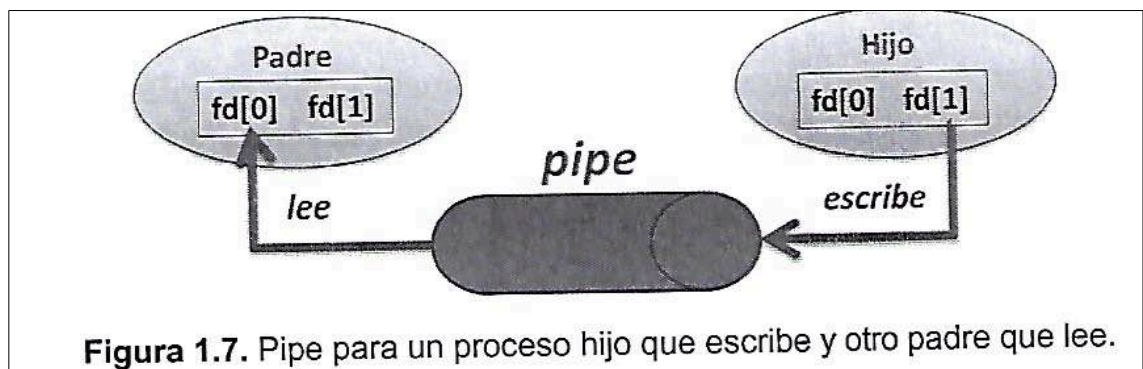
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()

{
    int fd[2];
    char buffer[30];
    pid_t pid;

    pipe(fd); //se crea el pipe

    pid=fork(); //se crea el proceso hijo

    switch(pid) {
        case -1 : //ERROR
            printf("NO SE HA PODIDO CREAR HIJO...");
            exit(-1);
            break;
        case 0 : //HIJO
            printf("El HIJO escribe en el pipe...\n");
            write(fd[1], "Hola papi", 10);
            break;
        default : //PADRE
            wait(NULL); //espera que finalice proceso hijo
            printf("El PADRE lee del pipe...\n");
            read(fd[0], buffer, 10);
            printf("\tMensaje leído: %s\n",buffer);
            break;
    }
}
```



Cuando el flujo de información va del padre hacia el hijo:

- El padre debe cerrar el descriptor de lectura *fd[0]*.
- El hijo debe cerrar el descriptor de escritura *fd[1]*.

Cuando el flujo de información va del hijo hacia padre ocurre lo contrario:

- El padre debe cerrar el descriptor de escritura *fd[1]*.
- El hijo debe cerrar el descriptor de lectura *fd[0]*.

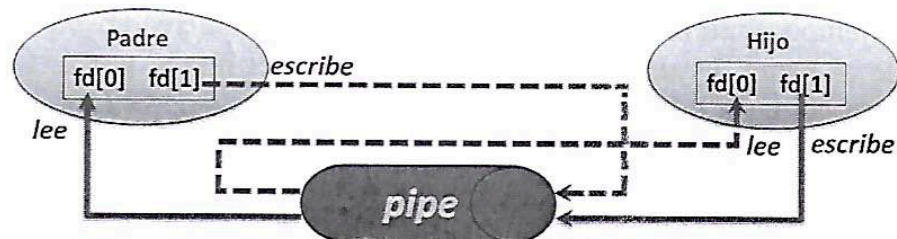


Figura 1.8. Pipe para un proceso padre e hijo que se envían datos.

EJERCICIO 7. Escribe el siguiente programa en C en el editor gedit. Nombralo como ejemploPipe2.c. Compila y ejecuta el programa en Linux y comprueba el resultado obtenido. Añade una captura de pantalla del resultado.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
int main(void)
{
    int fd[2];
    pid_t pid;
    char saludoPadre[]="Buenos dias hijo.\0";
    char buffer[80];

    pipe(fd); //creo pipe
    pid=fork(); //creo proceso

    switch(pid) {
        case -1 : //ERROR
            printf("NO SE HA PODIDO CREAR HIJO...");
            exit(-1);

        case 0 : //HIJO RECIBE
            close(fd[1]); //cierra el descriptor de entrada
            read(fd[0], buffer, sizeof(buffer)); //leo el pipe
            printf("\tEl HIJO recibe algo del pipe: %s\n",buffer);
            break;

        default : //PADRE ENVIA
            close(fd[0]);
            write(fd[1],saludoPadre,strlen(saludoPadre)); //escribo en pipe
            printf("El PADRE ENVIA MENSAJE AL HIJO...\n");
            wait(NULL); //espero al proceso hijo
            break;
    }

    return 0;
}
```


Sincronización entre procesos

EJERCICIO 8. Escribe el siguiente programa en C en el editor gedit. Nombralo como sincronizar.c. Compila y ejecuta el programa en Linux y comprueba el resultado obtenido. Añade una captura de pantalla del resultado.

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <fcntl.h>
/*-----*/
/* gestión de señales en proceso HIJO */
void manejador( int signal )
{
    printf("Hijo recibe señal..%d\n", signal);
}

/*-----*/
int main()
{
    int pid_hijo;
    pid_hijo = fork(); //creamos hijo

    switch(pid_hijo)
    {
        case -1:
            printf( "Error al crear el proceso hijo...\n");
            exit( -1 );
        case 0: //HIJO
            signal(SIGUSR1, manejador); //MANEJADOR DE SEÑAL EN HIJO
            while(1) {
            };
            break;
        default: //PADRE envia 2 señales
            sleep(1);
            kill(pid_hijo, SIGUSR1); //ENVIA SEÑAL AL HIJO
            sleep(1);
            kill(pid_hijo, SIGUSR1); //ENVIA SEÑAL AL HIJO
            sleep(1);
            break;
    }
    return 0;
}
```