PRÁCTICA 5

COMUNICACIÓN ENTRE PROCESOS USANDO LENGUAJE C PARA LINUX

Curso: Sistemas Operativos

Integrantes:

Esther Chunga Pacheco

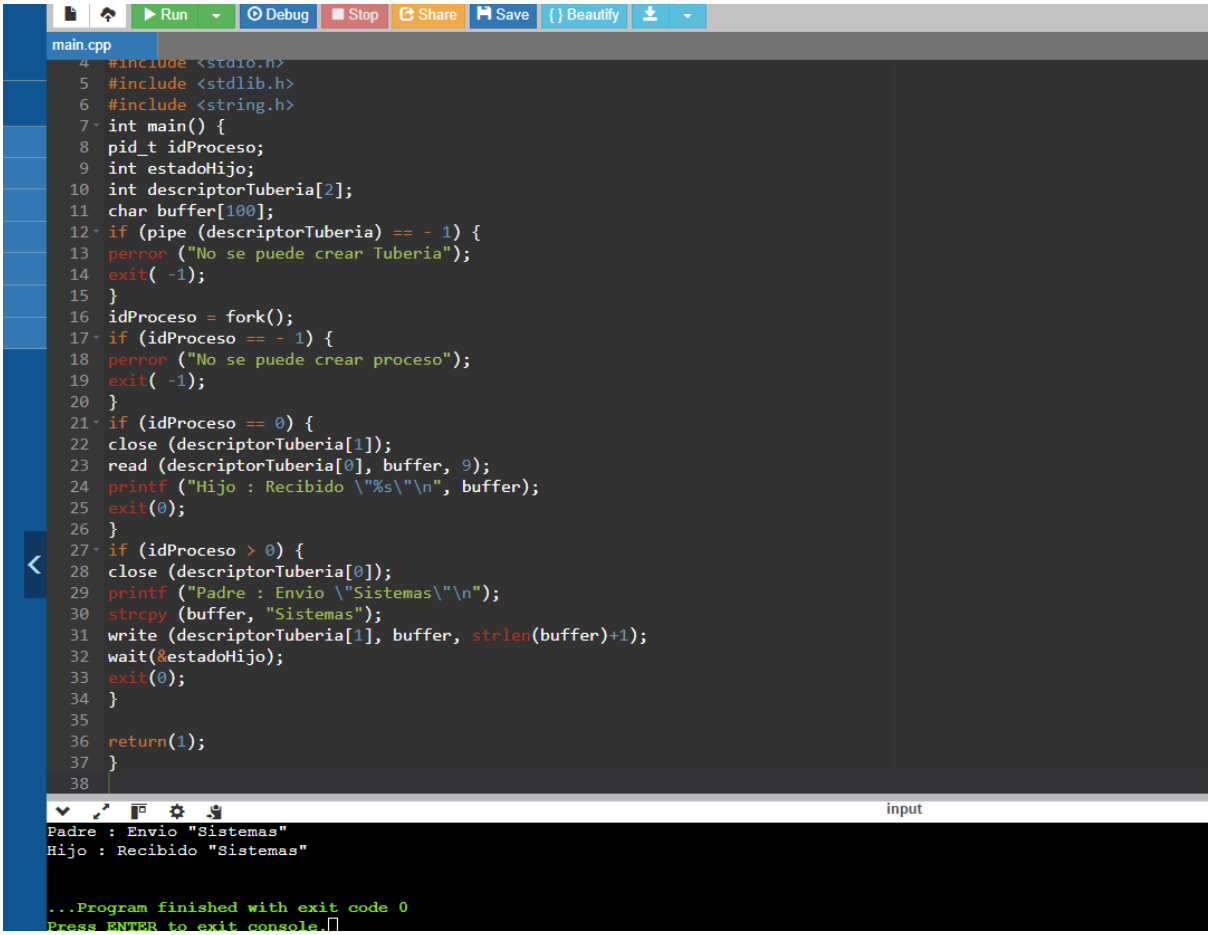
Hugo Diaz Chavez

Ronaldo Estrada Quicaño

Miguel Ocharan Coaquira

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);
}
```



Respuesta:

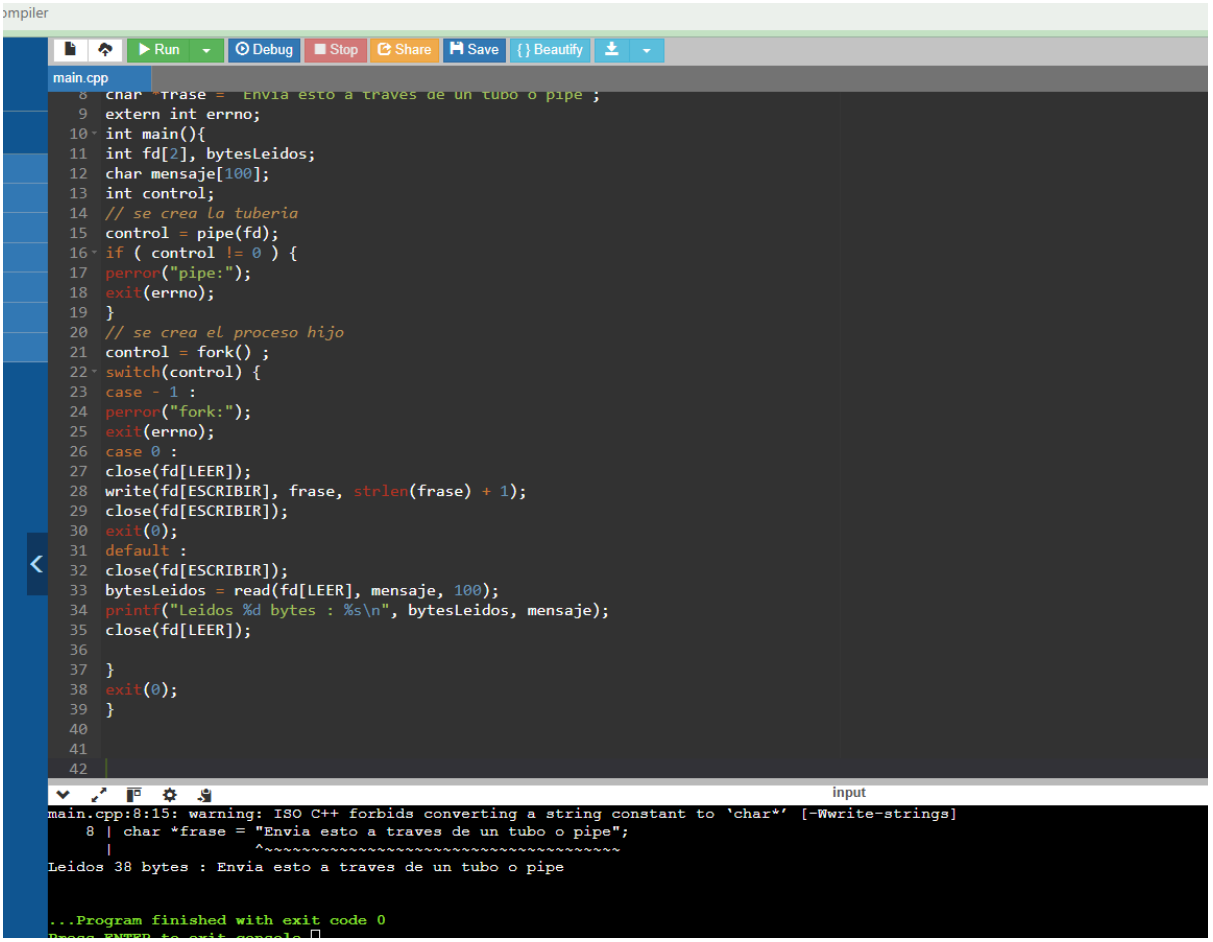
- crea un pipe con pipe(descriptorTuberia), eso genera 2 descriptores de archivo; uno para leer y otro para escribir. (descriptorTuberia[0]) para leer y (descriptorTuberia[1]) para escribir.
- Luego hace un fork que crea el proceso hijo. Desde ese punto hay 2 procesos: el padre y el hijo.
- Si está en el proceso hijo, se cierra el descriptor de estructura (close (descriptorTuberia[1])) porque solo va a leer.
- Usa read() para leer el pipe y guarda lo que le mandó el padre.
- imprime el mensaje que recibió el hijo.
- Si está en el proceso padre, cierra el descriptor de lectura porque solo va a escribir. Copia la palabra “sistemas” y la escribe en el pipe con write(). Espera que el hijo termine con wait().

2. Analice y describa la actividad que realiza el siguiente código. Explique como sucede el proceso de envió 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 tuberia
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("Leidos %d bytes : %s\n", bytesLeidos, mensaje);
close(fd[LEER]);

}
exit(0);
}
```

Respuesta:

- Crea el pipe(fd), luego hace el fork().
- Si está en el proceso hijo, cierra el descriptor de lectura (close()) para escribir. Escribe el mensaje al pipe, luego cierra el descriptor de lectura.
- Si está en el proceso padre, cierra el descriptor de escritura (close()) para leer. Lee el mensaje y lo imprime.

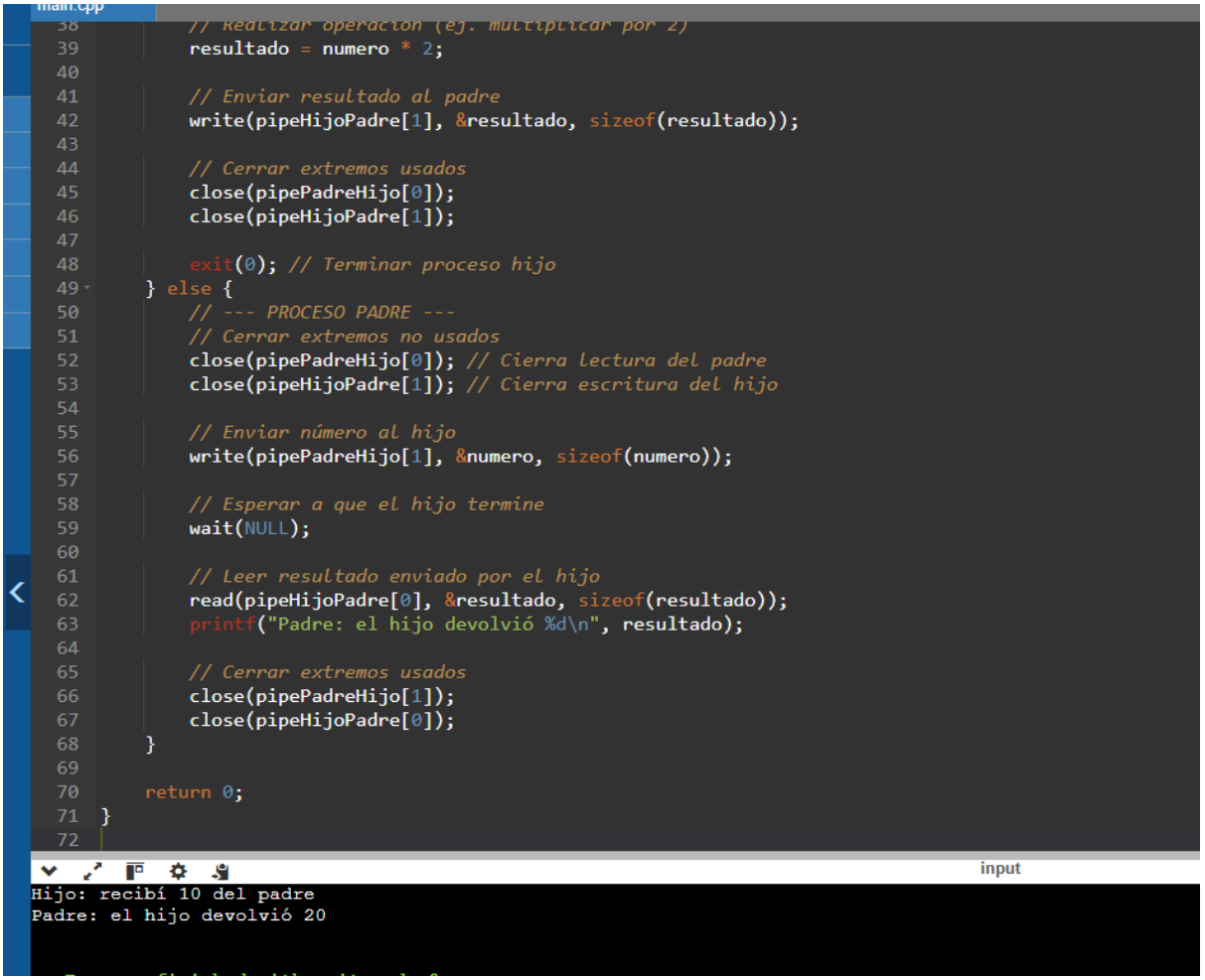

ompiler
Run
Debug
Stop
Share
Save
{ } Beautify
main.cpp
char *frase = "Envia esto a traves de un tubo o pipe";
extern int errno;
int main(){
int fd[2], bytesLeidos;
char mensaje[100];
int control;
// se crea la tuberia
control = pipe(fd);
if (control != 0) {
perror("pipe:");
exit(errno);
}
// se crea 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("Leidos %d bytes : %s\n", bytesLeidos, mensaje);
close(fd[LEER]);
}
exit(0);
}
input
main.cpp:8:15: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
8 | char *frase = "Envia esto a traves de un tubo o pipe";
Leidos 38 bytes : Envia esto a traves de un tubo o pipe
...Program finished with exit code 0

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

Respuesta:

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

int main() {
    int pipePadreHijo[2];  // Pipe para enviar datos del padre al hijo
    int pipeHijoPadre[2];  // Pipe para enviar datos del hijo al padre
    pid_t pid;
    int numero = 10;
    int resultado;

    // Crear los pipes
    if (pipe(pipePadreHijo) == -1 || pipe(pipeHijoPadre) == -1) {
        perror("Error al crear los pipes");
        exit(EXIT_FAILURE);
    }

    // Crear el proceso hijo
    pid = fork();

    if (pid == -1) {
        perror("Error al crear el proceso hijo");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        // --- PROCESO HIJO ---
        // Cerrar extremos no usados
        close(pipePadreHijo[1]); // Cierra escritura del padre
        close(pipeHijoPadre[0]); // Cierra lectura del padre

        // Leer número enviado por el padre
        read(pipePadreHijo[0], &numero, sizeof(numero));
        printf("Hijo: recibí %d del padre\n", numero);

        // Realizar operación (ej. multiplicar por 2)
        resultado = numero * 2;

        // Enviar resultado al padre
        write(pipeHijoPadre[1], &resultado, sizeof(resultado));

        // Cerrar extremos usados
```

```
        close(pipePadreHijo[0]);
        close(pipeHijoPadre[1]);

        exit(0); // Terminar proceso hijo
    } else {
        // --- PROCESO PADRE ---
        // Cerrar extremos no usados
        close(pipePadreHijo[0]); // Cierra lectura del padre
        close(pipeHijoPadre[1]); // Cierra escritura del hijo

        // Enviar número al hijo
        write(pipePadreHijo[1], &numero, sizeof(numero));

        // Esperar a que el hijo termine
        wait(NULL);

        // Leer resultado enviado por el hijo
        read(pipeHijoPadre[0], &resultado, sizeof(resultado));
        printf("Padre: el hijo devolvió %d\n", resultado);

        // Cerrar extremos usados
        close(pipePadreHijo[1]);
        close(pipeHijoPadre[0]);
    }

    return 0;
}
```



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

Si un proceso padre y su proceso hijo quieren comunicarse entre sí de forma bidireccional en Linux, necesitan usar dos tuberías. Una sirve para que el padre le envíe información al hijo, y la otra para que el hijo le responda al padre. Cada tubería tiene un extremo para leer y otro para escribir, entonces hay que tener cuidado de cerrar los extremos que no se usan, porque si no pueden ocurrir errores o la comunicación puede fallar. Cuando se usa fork(), los descriptores de las tuberías se copian al hijo, y eso permite que ambos procesos puedan usar las tuberías para mandarse datos. Es una forma sencilla pero efectiva de lograr que dos procesos se hablen entre ellos.