

**Universidad la**

**Salle**

# **Trabajo Productor - Consumidor**

**INTEGRANTES**

**Joshua David Ortiz Rosas**

**Frederick Dicarlo Mares Graos**

**Dylan Edward Davila Grau**

**Abimael Ernesto Frontado Fajardo**

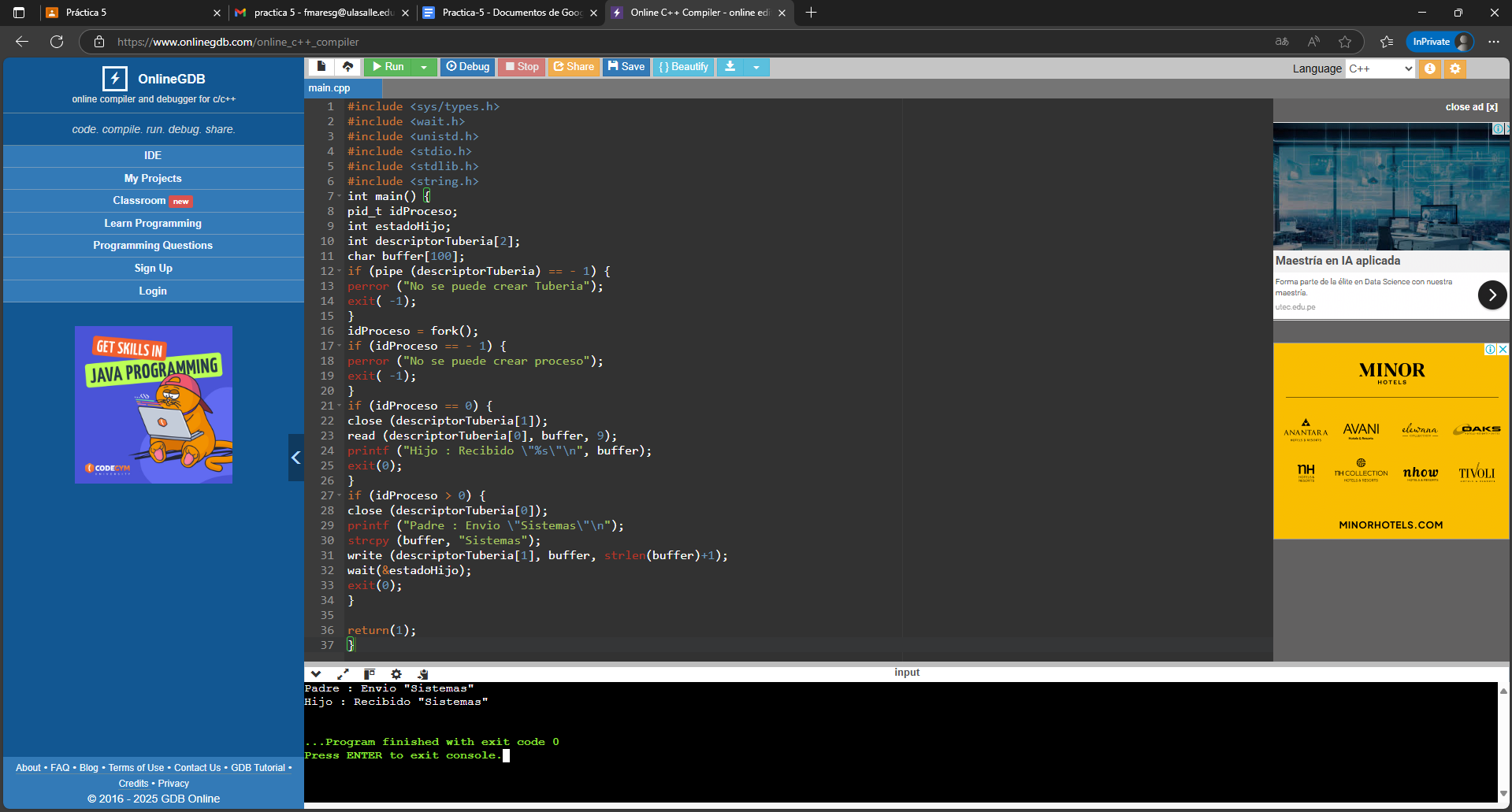
**INFORME**

| **Informe** | **Duración** |
| --- | --- |
| 03 | 05 horas |

| **Semestre académico** | **Fecha de inicio** | **Fecha de entrega** |
| --- | --- | --- |
| 2025 | 04/06/25 | 04/06/25 |

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.



El programa en C muestra cómo comunicar un proceso padre con su hijo usando una tubería (pipe). La tubería permite enviar datos en una sola dirección: del padre al hijo.

Primero, se crea la tubería con pipe(). Luego se crea un nuevo proceso con fork(). Si todo funciona correctamente, ambos procesos (padre e hijo) siguen caminos distintos.

El padre cierra el extremo de lectura de la tubería, escribe el mensaje "Sistemas" en el extremo de escritura y espera a que el hijo termine.

El hijo cierra el extremo de escritura de la tubería, lee 9 bytes del mensaje que el padre envió, y lo muestra en pantalla con printf.

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





### El programa comienza definiendo una tubería mediante pipe(fd), lo que genera dos descriptores de archivo: uno para lectura (fd[0]) y otro para escritura (fd[1]). Luego, crea un nuevo proceso utilizando fork().

### **Proceso de envío de información:**

1. **Creación de la tubería:** Se inicializa un arreglo fd de dos posiciones. Con la llamada pipe(fd), el sistema operativo establece un canal unidireccional de comunicación entre procesos. Si esta operación falla, se muestra un mensaje de error.
2. **Creación del proceso hijo:** Se invoca fork() para crear un nuevo proceso. A partir de este punto, el programa se ejecuta en dos procesos independientes: el padre y el hijo. La variable control determina en cuál de los dos se encuentra cada uno.
3. **Proceso hijo (cuando control == 0):** El hijo se encarga de **escribir un mensaje en la tubería**. Primero cierra el extremo de lectura (close(fd[LEER])) porque no lo necesita. Luego utiliza write() para enviar la cadena "Envia esto a traves de un tubo o pipe" al padre, escribiendo a través del descriptor fd[ESCRIBIR]. Finalmente, cierra ese descriptor para liberar recursos y termina.
4. **Proceso padre (caso default):** El padre, por su parte, **lee el mensaje enviado por el hijo**. Cierra el extremo de escritura (close(fd[ESCRIBIR])), ya que solo le interesa recibir información. Usa read() para capturar hasta 100 bytes del extremo de lectura y almacena lo recibido en la variable mensaje. Luego muestra en pantalla cuántos bytes recibió y cuál fue el contenido. Finalmente, cierra el descriptor de lectura.

### **Resultado en la consola:**

Cuando se ejecuta el programa, la salida será:

Leidos 38 bytes : Envia esto a traves de un tubo o pipe

Esto indica que el proceso padre recibió correctamente los 38 bytes enviados por el hijo, correspondientes a la cadena completa incluyendo el carácter nulo (\0) que marca el final de la cadena en C.

### **Conclusión:**

Este código muestra de forma sencilla cómo los procesos pueden intercambiar datos mediante una tubería. El proceso hijo actúa como emisor y el padre como receptor. La correcta gestión de los extremos de la tubería (cerrando los que no se usan) es esencial para que la comunicación funcione sin errores.

**3. Elabore un programa propio que emplee la comunicación entre procesos (padre e**

**hijo) utilizando pipes.**

import os

from multiprocessing import Process, Pipe

def proceso\_hijo(conn\_padre\_hijo, conn\_hijo\_padre):

# Cerrar el extremo que no usaremos (escritura del pipe de entrada)

conn\_padre\_hijo[1].close()

# 1. Recibir mensaje del padre

mensaje\_recibido = conn\_padre\_hijo[0].recv()

print(f"[HIJO] Mensaje recibido: {mensaje\_recibido}")

# 2. Enviar respuesta al padre

respuesta = "Hola padre, estoy bien gracias!"

print("[HIJO] Enviando respuesta...")

conn\_hijo\_padre[1].send(respuesta)

# Cerrar conexiones

conn\_padre\_hijo[0].close()

conn\_hijo\_padre[1].close()

print("[HIJO] Proceso terminado.")

# Crear los pipes:

conn\_padre\_hijo, conn\_hijo\_padre = Pipe(), Pipe()

# Crear proceso hijo

hijo = Process(target=proceso\_hijo, args=(conn\_padre\_hijo, conn\_hijo\_padre))

hijo.start()

# PROCESO PADRE

conn\_padre\_hijo[0].close()

# 1. Enviar mensaje al hijo

mensaje = "Hola hijo, ¿cómo estás?"

print("[PADRE] Enviando mensaje...")

conn\_padre\_hijo[1].send(mensaje)

# 2. Recibir respuesta del hijo

respuesta = conn\_hijo\_padre[0].recv()

print(f"[PADRE] Mensaje recibido: {respuesta}")

# Cerrar conexiones

conn\_padre\_hijo[1].close()

conn\_hijo\_padre[0].close()

# Esperar a que el hijo termine

hijo.join()

print("[PADRE] Proceso terminado.")

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

Para que haya una comunicación entre padre e hijo hay varios métodos o formas a tomar en cuenta, las siguientes son algunas de ellas:  
  
a. Pipes: Las tuberías o pipes sirven para comunicar dos procesos, de tal forma que uno de ellos escribe en una tubería, mientras que el otro lee de ella

b. Paso de mensajes: El paso de mensajes es un método de comunicación entre procesos en el sistema operativo. Implica el intercambio de mensajes entre procesos, donde cada uno envía y recibe mensajes para coordinar sus actividades e intercambiar datos con otros procesos.

c. Memoria compartida: La comunicación entre procesos es un concepto fundamental en los sistemas operativos que permite que múltiples procesos se comuniquen y sincronicen sus acciones. La comunicación entre procesos a través de memoria compartida es un concepto donde dos o más procesos pueden acceder a la memoria común y la comunicación se realiza a través de esta memoria compartida donde los cambios realizados por un proceso pueden ser vistos por otro proceso.

Bibliografía:

GeeksforGeeks. (2023). *IPC through shared memory*..<https://www.geeksforgeeks.org/ipc-shared-memory/>

Garcia, R.PROCESOS Y COMUNICACIÓN ENTRE PROCESOS. <https://dspace.ceu.es/server/api/core/bitstreams/8af432e0-0d41-38b8-e053-0100007fe1f5/content>