Pontificia Universidad Javeriana

Facultad de Ingeniería

Sistemas Operativos

Proyecto

Diego Andrés Martínez Cuervo

Julián Camilo Pérez Gómez

Jose David Ontiveros Gutiérrez

Daniel Galvis

25 de mayo de 2025

Introducción 2
Objetivos 3
2.1 Objetivo General 3
2.2 Objetivos específicos 3
Descripción del sistema 4
Arquitectura de procesos 4
4.1 Componentes 4
4.2 Comunicación y sincronización 5
Diseño y desarrollo 6
5.1 Estructuras de datos 6
5.2 Descripción del código fuente 7
5.2.1 procesoSolicitante.c 7
5.2.2 procesoReceptor.c 7
5.3 Mecanismos POSIX utilizados 7
Plan de pruebas 8
Herramientas y entorno 9
- Sistema Operativo Linux. 9
Capturas de ejecución 15
Conclusiones 18
Referencias 18

Introducción

En el desarrollo de sistemas operativos modernos, la implementación de procesos concurrentes y mecanismos de comunicación interprocesos representa una competencia fundamental para garantizar el rendimiento, la escalabilidad y la eficiencia en múltiples aplicaciones del mundo real. Bajo este contexto, el presente proyecto tiene como propósito diseñar e implementar un sistema funcional para la gestión de préstamos de libros, utilizando conceptos clave como creación de procesos, uso de hilos POSIX, comunicación mediante pipes y sincronización con semáforos y exclusión mutua.

El sistema simula el entorno de una biblioteca donde múltiples usuarios (procesos solicitantes) pueden interactuar con un sistema central (receptor de peticiones) para realizar operaciones como préstamo, devolución y renovación de ejemplares. Estas operaciones deben gestionarse de manera concurrente, garantizando que los datos se mantengan consistentes en todo momento, incluso bajo acceso simultáneo a los recursos compartidos.

Además de responder a los requerimientos técnicos del curso de Sistemas Operativos, el desarrollo de este sistema permite aplicar en la práctica estructuras de datos, técnicas de sincronización y arquitecturas orientadas a eventos, fortaleciendo así las habilidades de programación concurrente del estudiante. El documento que se presenta a continuación describe en detalle los objetivos, diseño, arquitectura, funcionamiento, pruebas realizadas y conclusiones obtenidas tras la construcción del sistema.

Objetivos

2.1 Objetivo General

Desarrollar un sistema concurrente para la gestión de préstamos de libros en una biblioteca, implementado en lenguaje C, utilizando procesos, hilos POSIX y mecanismos de comunicación y sincronización, con el fin de simular un entorno real de múltiples usuarios accediendo simultáneamente a un recurso compartido.

2.2 Objetivos específicos

- Implementar procesos solicitantes que generen solicitudes de préstamo, devolución y renovación de libros mediante menú interactivo o archivo de entrada.
- Diseñar un proceso receptor que administre las solicitudes entrantes y coordine las acciones requeridas en la base de datos mediante hilos auxiliares.
- Utilizar pipes como mecanismo de comunicación entre procesos, y semáforos junto con mutex para garantizar la sincronización y consistencia en el acceso concurrente a los datos.
- Registrar los eventos realizados (préstamos, devoluciones, renovaciones) para permitir auditorías del sistema mediante comandos específicos desde consola.
- Validar el funcionamiento del sistema mediante un plan de pruebas que verifique el comportamiento esperado en distintos escenarios operacionales.

Descripción del sistema

El sistema implementado simula un entorno de biblioteca en el que múltiples usuarios pueden interactuar para realizar operaciones sobre un catálogo de libros. Esta simulación se estructura a través de procesos y hilos que comunican y sincronizan sus acciones para gestionar correctamente el acceso a los recursos compartidos (libros y ejemplares).

Existen dos tipos principales de componentes en el sistema:

- Procesos Solicitantes (PS): representan a los usuarios que realizan operaciones sobre los libros. Cada solicitante puede generar peticiones de préstamo (P), renovación (R), devolución (D) o salida del sistema (Q). Estas solicitudes pueden ser ingresadas de forma manual (mediante menú interactivo) o automática (desde un archivo de texto preformateado). Las solicitudes son enviadas al proceso receptor a través de un pipe nombrado.
- Proceso Receptor de Peticiones (RP): es el componente central del sistema, responsable de recibir y procesar las solicitudes de los PS. El receptor atiende directamente los préstamos, mientras que las solicitudes de devolución y renovación son gestionadas mediante un hilo auxiliar, siguiendo el patrón de diseño productor-consumidor. Adicionalmente, el receptor cuenta con un segundo hilo auxiliar que permite al operador ingresar comandos desde la consola, como s para finalizar el sistema o r para generar un reporte de transacciones.

Las operaciones afectan a una base de datos simulada mediante un archivo de texto, donde se almacenan los libros, sus ejemplares, estados (disponible o prestado) y fechas asociadas. La concurrencia y el acceso sincronizado a esta base de datos se manejan utilizando mutex y semaforización, garantizando la integridad de la información incluso cuando múltiples procesos están activos simultáneamente.

El sistema ha sido diseñado para ser modular, extensible y robusto, haciendo uso de las herramientas de programación concurrente disponibles en el estándar POSIX y cumpliendo con los requerimientos técnicos del curso.

Arquitectura de procesos

El sistema implementado está basado en una arquitectura concurrente que emplea procesos, hilos y mecanismos de sincronización para permitir la gestión simultánea de múltiples solicitudes de usuarios sobre una base de datos común. Esta arquitectura sigue un enfoque modular y está compuesta por dos niveles principales: procesos externos y hilos internos, organizados de la siguiente manera:

4.1 Componentes

.Procesos Solicitantes (PS):

- Son procesos independientes que simulan a los usuarios de la biblioteca.

- Pueden ejecutarse de forma interactiva (menú) o desde archivos de entrada.
- Envían solicitudes de tipo Préstamo (P), Renovación (R), Devolución (D) o Salida (Q) a través de un pipe nombrado al proceso receptor.

Proceso Receptor (RP):

- Es el componente central del sistema.
- Recibe todas las solicitudes a través del *pipe*.
- Procesa directamente las solicitudes de préstamo.
- Crea dos **hilos auxiliares**:
  - r: genera un reporte en consola de todos los eventos registrados.
  - s: detiene la ejecución del sistema de forma segura.
- Base de datos:
  - Actúa como almacenamiento persistente para el estado de los libros.
  - Es modificada por el receptor o el hilo consumidor según la operación.

4.2 Comunicación y sincronización

La comunicación entre procesos y hilos se implementa de la siguiente forma:

Entre PS y RP:

- Se usa un pipe nombrado (pipeBiblioteca) para enviar mensajes estructurados como:
  P, Nombre del Libro, ISBN

Entre RP e hilo consumidor:

- Se usa un buffer circular compartido de tamaño fijo (TAM_BUFFER), con variables in y out para indicar la posición de escritura y lectura respectivamente.
- El acceso a este buffer se sincroniza usando:
  - sem_t hay_espacio: semáforo que indica si hay espacio para insertar una nueva solicitud.
  - sem_t hay_datos: semáforo que indica si hay datos disponibles para consumir.
  - pthread_mutex_t mutex: garantiza exclusión mutua al modificar el buffer.

Acceso a la base de datos:

- Todas las modificaciones sobre el archivo de base de datos están protegidas con mutex, asegurando que no haya condiciones de carrera entre el hilo consumidor y el receptor.

Eventos del sistema:

- Cada acción (préstamo, devolución o renovación) es registrada como un evento en una estructura global (eventos[]) que puede ser consultada mediante el comando r en consola.

Diseño y desarrollo

El desarrollo del sistema se llevó a cabo siguiendo una estructura modular y organizada, que permite separar claramente las responsabilidades entre los distintos componentes del programa. Esta sección describe las estructuras de datos utilizadas, la organización del código fuente y los mecanismos del sistema POSIX empleados para garantizar la correcta ejecución concurrente.

5.1 Estructuras de datos

Para representar los libros y las operaciones realizadas sobre ellos, se definieron las siguientes estructuras:

- Ejemplar
  Representa un ejemplar físico de un libro.

```c
typedef struct {
    int numero;
    char estado;        // 'D' disponible, 'P' prestado
    char fecha[20];     // Fecha del último cambio
} Ejemplar;
```

- Libro
  Contiene la información general del libro, incluyendo su nombre, ISBN y arreglo de ejemplares.

```c
typedef struct {
    char nombre[100];
    int isbn;
    int cantidad;
    Ejemplar ejemplares[MAX_EJEMPLARES];
} Libro;
```

- Solicitud
  Utilizada para representar las peticiones enviadas por los PS al RP.

```c
typedef struct {
    char tipo;              // 'P', 'D' o 'R'
    char nombreLibro[100];
    int isbn;
} Solicitud;
```

- Evento
  Permite registrar y mostrar todas las operaciones realizadas.

```c
typedef struct {
    char status[10];        // Prestado, Renovado, Devuelto
    char nombreLibro[100];
    int isbn;
    int ejemplar;
    char fecha[20];
} Evento;
```

Estas estructuras permiten un manejo organizado y eficiente de la información, además de facilitar la implementación de los algoritmos de búsqueda y modificación en la base de datos.

5.2 Descripción del código fuente

5.2.1 procesoSolicitante.c

Este archivo implementa el proceso solicitante. Su función es tomar solicitudes de préstamo, devolución o renovación de libros e ingresarlas al sistema. Puede ejecutarse en dos modos:

- Interactivo: ofrece un menú al usuario para ingresar manualmente los datos.
- Automático: lee solicitudes desde un archivo de texto con formato predefinido.

Las solicitudes se escriben en el pipe nombrado especificado por el usuario mediante el flag -p.

5.2.2 procesoReceptor.c

Este archivo implementa el proceso receptor, el cual actúa como núcleo del sistema. Sus responsabilidades incluyen:

- Recibir solicitudes desde el pipe.
- Procesar directamente los préstamos.
- Delegar solicitudes de devolución o renovación al hilo consumidor.
- Crear un hilo adicional para recibir comandos desde consola (s y r).
- Actualizar la base de datos y registrar eventos.
- Guardar el estado final en un archivo si se usa el flag -s.

El archivo también incluye funciones auxiliares para cargar y guardar la base de datos, registrar eventos y obtener la fecha actual.

5.3 Mecanismos POSIX utilizados

Durante el desarrollo se utilizaron varias herramientas del estándar POSIX para manejar concurrencia y sincronización:

- Pipes (FIFO): permiten la comunicación entre procesos mediante archivos especiales creados con mkfifo.
- Hilos (pthread): se usaron para ejecutar tareas en paralelo, como el manejo de solicitudes y comandos.
- Semáforos (sem_t): empleados para controlar el acceso al buffer circular en el patrón productor-consumidor.
- Mutex (pthread_mutex_t): usados para garantizar exclusión mutua al acceder a estructuras de datos compartidas, como el buffer y el registro de eventos.

Estos mecanismos fueron fundamentales para garantizar la integridad de los datos y evitar condiciones de carrera entre los diferentes componentes concurrentes del sistema.

Plan de pruebas

Verificar que el sistema de gestión de libros funcione correctamente bajo distintos escenarios de operación, garantizando:

- La integridad de los datos compartidos.
- El correcto funcionamiento de los procesos y hilos.
- La sincronización efectiva usando semáforos y Mutex.
- La correcta recepción, procesamiento y registro de las solicitudes.

Tipos de Pruebas

Tipo de prueba	**Descripción**
Pruebas Funcionales	Verifican que cada función del sistema (préstamo, devolución, renovación, reporte, salida) se ejecute como se espera.
Pruebas de Concurrencia	Evalúan cómo el sistema maneja múltiples procesos solicitantes simultáneamente.
Pruebas de Sincronización	Se aseguran de que los semáforos y mutex eviten condiciones de carrera.
Pruebas de Comunicación Interprocesos	Comprueban la transmisión correcta de mensajes por pipe y su formato.
Pruebas de Comandos del Operador	Evalúan si los comandos r y s funcionan correctamente en cualquier momento.
Pruebas de Estrés	Miden el desempeño del sistema con alta

	carga de solicitudes.
Pruebas de Persistencia	Verifican que la base de datos y los eventos se actualicen y almacenen correctamente.

Herramientas y entorno

- Sistema Operativo Linux.
- Compilador GCC
- Terminal de ejecución y comandos.

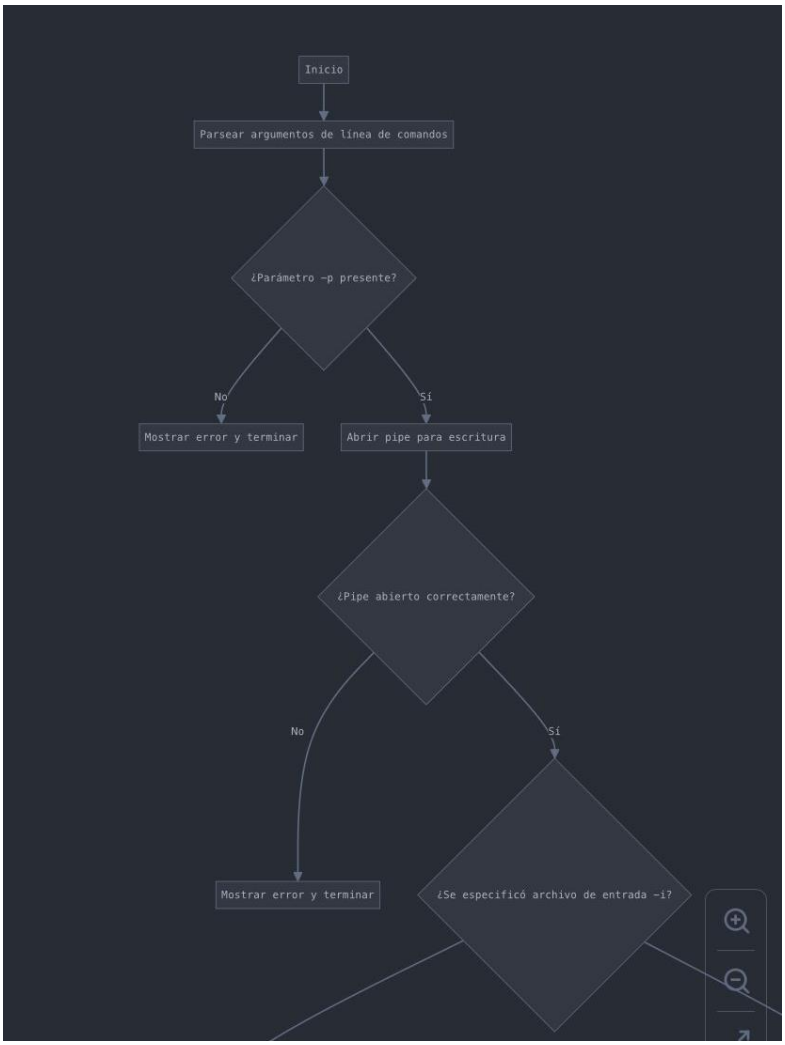
Criterios de Éxito

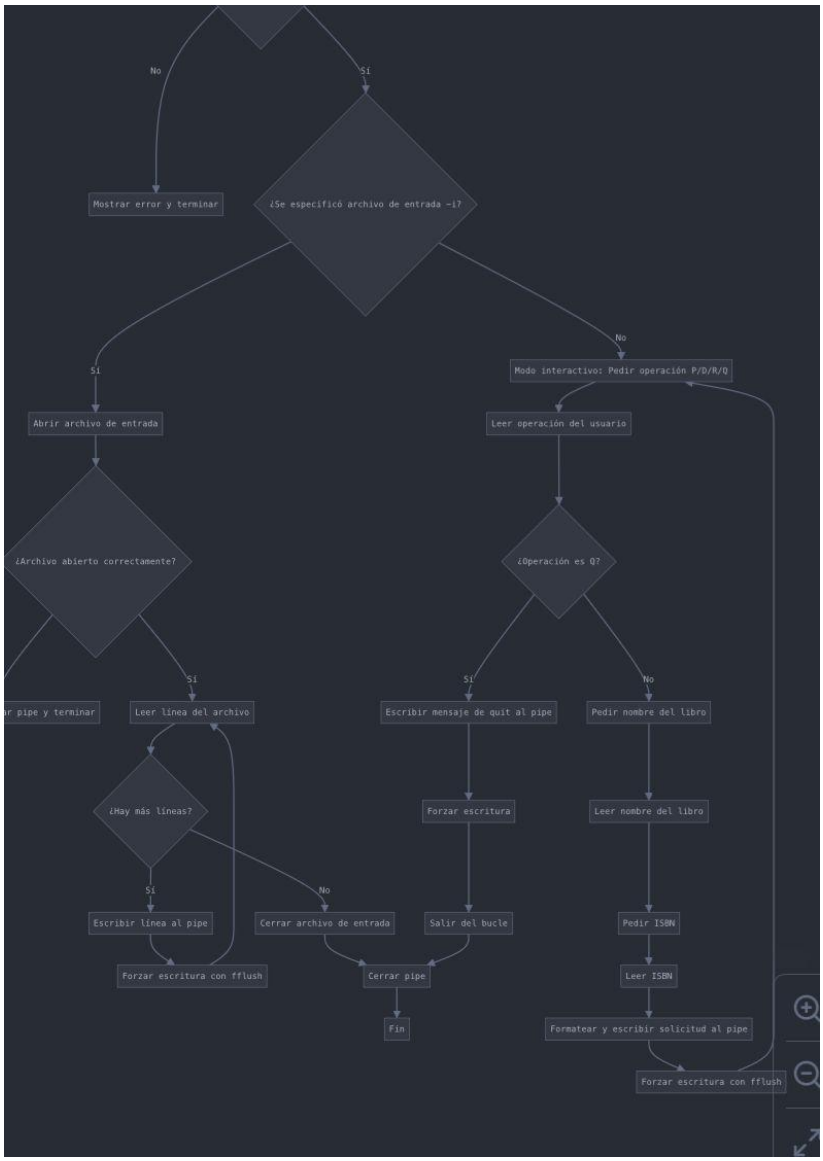
- Todas las solicitudes válidas son procesadas correctamente.
- No hay corrupción de datos en la base de datos.
- No hay condiciones de carrera.
- Todos los eventos son registrados y reportados correctamente.
- El sistema responde adecuadamente ante entradas válidas.

Diagrama de flujo código

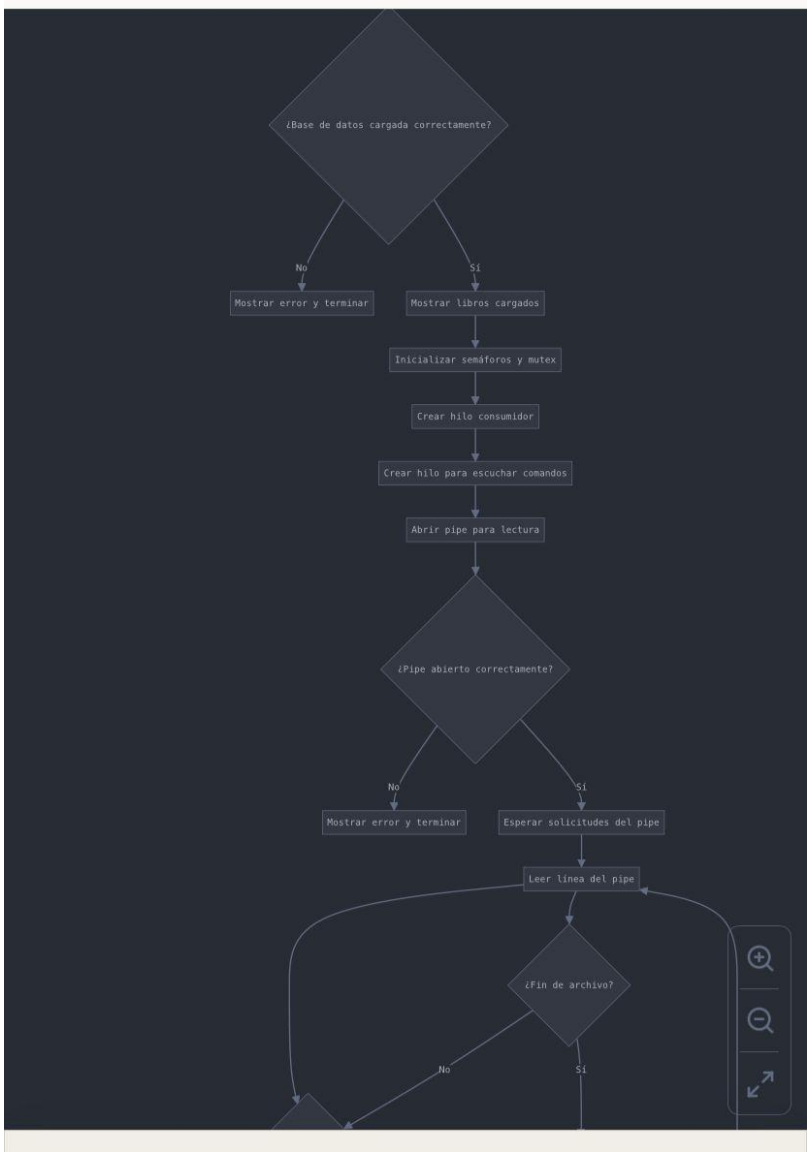
A continuación, se mostrará por medio de un diagrama la ejecución del código, recordando que se divide en Proceso Solicitante y Proceso Receptor

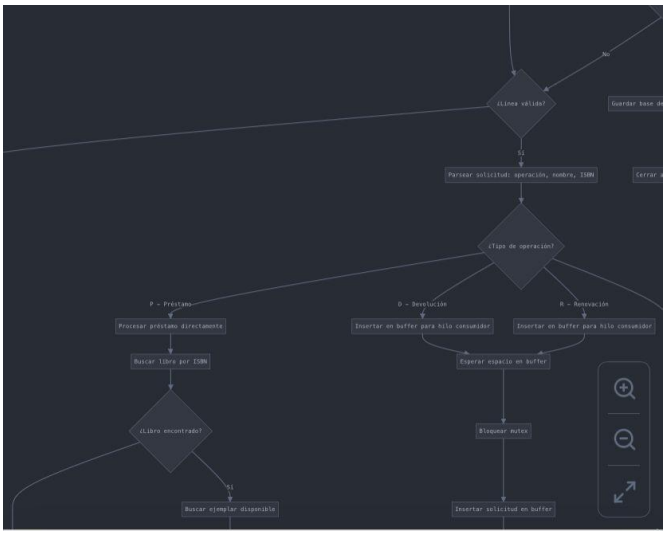
Proceso Solicitante.

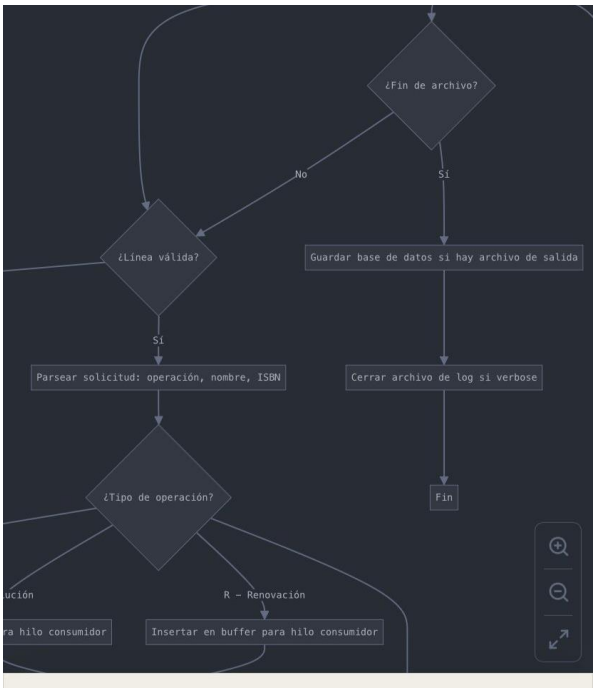

Inicio
Parsear argumentos de línea de comandos
¿Parámetro -p presente?
No
Sí
Mostrar error y terminar
Abrir pipe para escritura
¿Pipe abierto correctamente?
No
Sí
Mostrar error y terminar
¿Se especificó archivo de entrada -i?

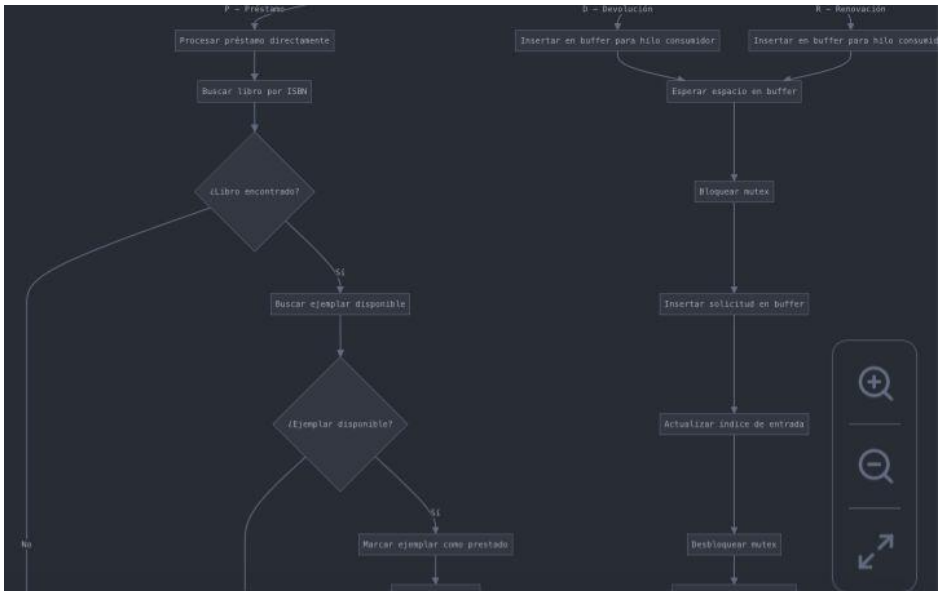


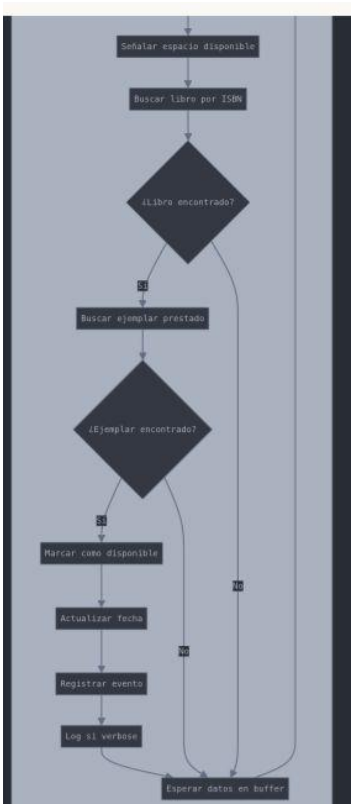
Proceso Receptor


¿Base de datos cargada correctamente?
No
Sí
Mostrar error y terminar
Mostrar libros cargados
Inicializar semáforos y mutex
Crear hilo consumidor
Crear hilo para escuchar comandos
Abrir pipe para lectura
¿Pipe abierto correctamente?
No
Sí
Mostrar error y terminar
Esperar solicitudes del pipe
Leer linea del pipe
¿Fin de archivo?
No
Sí

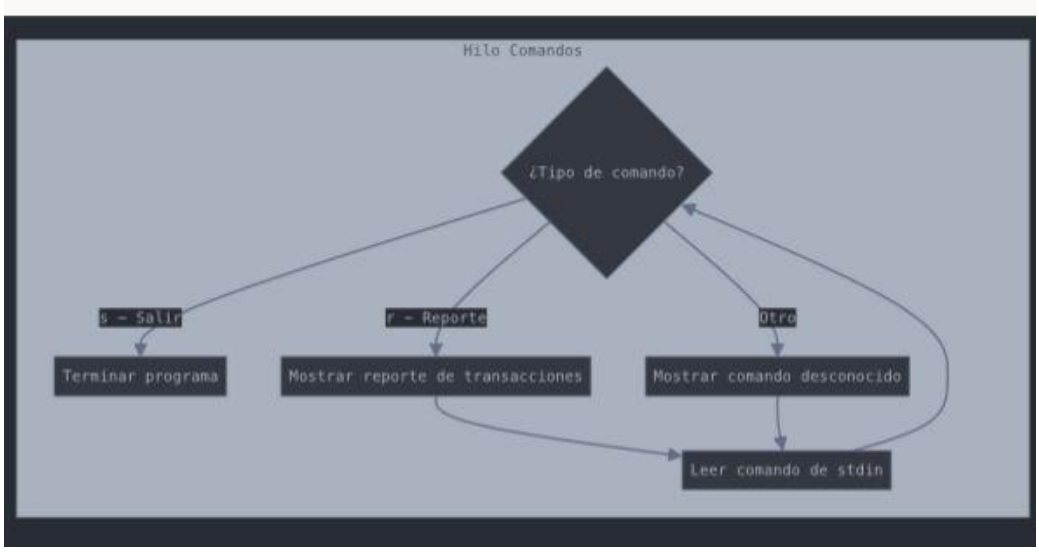

No
¿Línea válida?
Guardar base de
Sí
Parsear solicitud: operación, nombre, ISBN
Cerrar
¿Tipo de operación?
P – Préstamo
D – Devolución
R – Renovación
Procesar préstamo directamente
Insertar en buffer para hilo consumidor
Insertar en buffer para hilo consumidor
Buscar libro por ISBN
Esperar espacio en buffer
¿Libro encontrado?
Bloquear mutex
Sí
Buscar ejemplar disponible
Insertar solicitud en buffer


¿Fin de archivo?
No
Sí
¿Línea válida?
Guardar base de datos si hay archivo de salida
Sí
Parsear solicitud: operación, nombre, ISBN
Cerrar archivo de log si verbose
¿Tipo de operación?
Fin
ución
R – Renovación
ra hilo consumidor
Insertar en buffer para hilo consumidor


P – Prestamo
D – Devolucion
R – Renovación
Procesar préstamo directamente
Insertar en buffer para hilo consumidor
Insertar en buffer para hilo consumid
Buscar libro por ISBN
Esperar espacio en buffer
¿Libro encontrado?
Bloquear mutex
Si
Buscar ejemplar disponible
Insertar solicitud en buffer
¿Ejemplar disponible?
Actualizar índice de entrada
Si
No
Marcar ejemplar como prestado
Desbloquear mutex


Señalar espacio disponible
Buscar libro por ISBN
¿Libro encontrado?
Si
Buscar ejemplar prestado
¿Ejemplar encontrado?
Si
Marcar como disponible
No
Actualizar fecha
No
Registrar evento
Log si verbose
Esperar datos en buffer

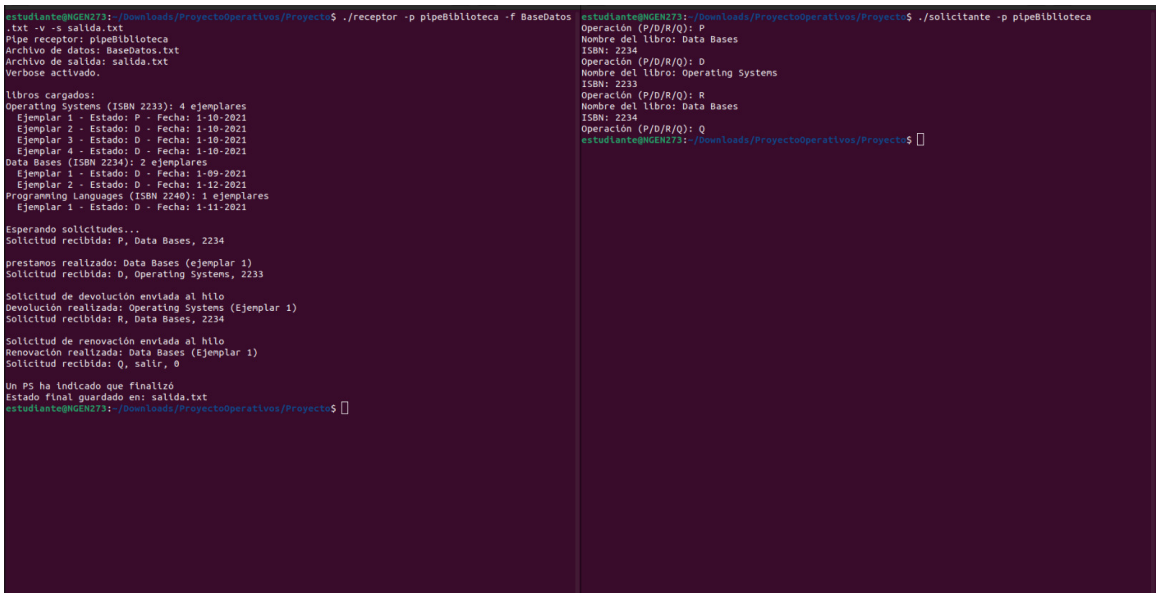
Tipos de comando



Capturas de ejecución

A continuación, se presentan las capturas de pantalla que evidencian la correcta ejecución del programa desarrollado. En ellas se puede observar el flujo de las operaciones realizadas, incluyendo la carga del archivo de registros, el procesamiento de los datos y la generación del reporte final con el estado actualizado de los libros en la biblioteca.

Estas imágenes demuestran el cumplimiento de los requisitos del enunciado y validan el funcionamiento del sistema en distintos escenarios de prueba.



La captura muestra la ejecución simultánea del proceso Receptor (RP) y el proceso Solicitante (PS) del sistema de préstamo de libros. En la parte superior se observa el RP inicializándose, cargando los datos de libros desde el archivo BaseBatos.txt y configurando el pipe biblioteca para comunicación. El receptor muestra el estado inicial de los libros disponibles, incluyendo títulos como "Operativo Systemic" y "Data Bases" con sus respectivos ISBNs y ejemplares. A medida que recibe solicitudes a través del pipe, procesa operaciones de préstamo para "Data Bases" y "Operativa Systemic", así como una renovación, delegando algunas tareas al hilo auxiliar como se especifica en el diseño. Cuando recibe la señal Q de terminación, guarda el estado final en saltda.txt. En la parte derecha aparece la interfaz del PS con su menú interactivo donde el usuario puede ingresar operaciones de préstamo (P), renovación (R) o salida (Q), junto con los datos del libro como nombre e ISBN. Se ejemplifica con la solicitud de préstamo para "Data Bases" y posterior renovación del mismo libro. Esta captura evidencia el correcto funcionamiento de la comunicación entre procesos mediante pipes, el uso de hilos auxiliares para procesamiento concurrente, y el manejo de las operaciones básicas del sistema manteniendo la persistencia de datos en archivos.

salida.txt

```
Operating Systems, 2233, 4
1, D, 26-05-2025
2, D, 1-10-2021
3, D, 1-10-2021
4, D, 1-10-2021

Data Bases, 2234, 2
1, D, 26-05-2025
2, D, 1-12-2021

Programming Languages, 2240, 1
1, D, 1-11-2021

```

La captura evidencia la actualización y/o generación del archivo de salida.txt con los cambios realizados en la captura anterior.

Esta captura muestra el uso clave de los comandos 'r' (reporte) y 's' (salida) en el proceso Receptor (RP). Tras procesar una devolución del libro "Operating Systems", se ejecuta el comando 'r' desde la terminal del RP, generando un reporte en pantalla con el formato especificado. Posteriormente, el comando **'s'** finaliza la ejecución ordenada del RP.

```
P, Operating Systems, 2233, 1, 1-10-2021
P, Calculo Integral, 2211, 1, 1-10-2021
```

En esta captura se evidencia el contenido del archivo solicitudes.txt, con el formato especifico.

Esta imagen demuestra dos funcionalidades clave del sistema: primero, la capacidad de procesar solicitudes desde un archivo de texto (solicitudes.txt) mediante el parámetro -i, donde el PS lee y envía automáticamente las peticiones al RP a través del pipe biblioteca. Segundo, el manejo robusto de errores cuando un libro no existe en la base de datos.

Conclusiones

El desarrollo de gestión de préstamos de libros permitió aplicar de forma integral los conceptos fundamentales de los sistemas operativos, en particular aquellos relacionados con concurrencia, sincronización y comunicación interprocesos. A través del uso de procesos, hilos POSIX, semáforos, mutex y pipes, se logró implementar una solución capaz de manejar múltiples solicitudes sin comprometer la integridad de los datos.

Durante el proceso de diseño y desarrollo se evidenció la importancia de una arquitectura modular que facilitara la distribución de responsabilidades entre procesos y hilos, así como la correcta implementación del patrón productor-consumidor para gestionar eficientemente las solicitudes concurrentes. Las pruebas realizadas confirmaron el funcionamiento adecuado del sistema en distintos escenarios operativos, incluyendo condiciones de carga y estrés.

El proyecto cumplió los objetivos y brindó experiencia práctica en programación concurrente, destacando el valor de los mecanismos POSIX para desarrollar aplicaciones eficientes y seguras en entornos multitarea.

Referencias

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2020). *Operating System Concepts* (10th ed.). Wiley.
2. Kerrisk, M. (2010). *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press.
3. IEEE. (2008). *POSIX.1-2008 - IEEE Std 1003.1*. Disponible en: https://pubs.opengroup.org/onlinepubs/9699919799/
4. GNU Project. (n.d.). *Using Pipes in C Programming*. Disponible en: https://www.gnu.org/software/libc/manual/html_node/Pipes.html

5. Linux Programmer's Manual. (n.d.).