

Proyecto: Sistema para el Préstamo de Libros

Carlos Daniel Güiza

24 de Mayo de 2025

1 Introducción

El presente documento describe el desarrollo e implementación de un sistema de gestión de préstamos de libros con un enfoque en hilos pthreads. El programa se enfoca en simular una interacción entre procesos cliente-servidor a través del uso de named pipes, donde uno o más procesos solicitantes envían operaciones al proceso receptor, encargado de procesarlas y mantener el estado actualizado de una base de datos en memoria, la cual es cargada desde un archivo de texto.

Este proyecto tiene como propósito aplicar y consolidar conocimientos fundamentales en concurrencia, comunicación entre procesos, sincronización mediante mutex y condiciones POSIX, así como la organización modular del código utilizando estructuras y archivos de encabezado en lenguaje C.

También al final, se abordará un caso de uso en el cual los usuarios pueden realizar operaciones como préstamos, devoluciones y renovaciones de ejemplares, las cuales son gestionadas en paralelo mediante hilos auxiliares y estructuras de datos compartidas protegidas para garantizar la consistencia. A lo largo del documento se describen las decisiones de diseño, la implementación detallada y las pruebas realizadas para validar el correcto funcionamiento del sistema.

2 Objetivos

El objetivo del proyecto es implementar un sistema de gestión de préstamos de libros utilizando comunicación mediante named pipes. Este informe detallará la arquitectura de procesos e hilos que se utilizaron para implementar con éxito las indicaciones del proyecto; a su vez, presenta diagramas de secuencia para el caso de uso específico mencionado anteriormente y evalúa las funcionalidades implementadas, viendo si se logró implementar todo o no, dando de esa manera unas conclusiones finales. Además, se busca garantizar la consistencia de datos mediante sincronización, soportar múltiples procesos solicitantes sin condiciones de carrera y validar todas las operaciones mediante un caso de uso específico que incluye préstamo, renovación, devolución y terminación.

3 Diseño y Arquitectura de Procesos e Hilos

3.1 Descripción general

El sistema consta de dos programas principales:

- **receptor.c:** Proceso principal que gestiona la base de datos de libros y procesa las operaciones enviadas por el solicitante.
- **solicitante.c:** Proceso que envía solicitudes al receptor y recibe respuestas.

También contiene sus respectivos archivos de encabezado y están documentados con su membrete respectivo.

Por otro lado, tenemos los hilos adicionales utilizados por el receptor, siendo los siguientes:

- **Hilo auxiliar1:** Procesa operaciones de devolución (D) y renovación (R) en segundo plano.
- **Hilo auxiliar2:** Maneja comandos interactivos como salir (s) y generar reportes (r).

3.2 Comunicación y sincronización

Para la comunicación de solicitante a receptor se utiliza un named pipe que se crea a partir del nombre que el usuario digite, enviando todas las operaciones a partir de este pipe. Al llegar al receptor, estas se dividen dependiendo de si son devolución, renovación o préstamo. En caso de ser alguna de las primeras dos, estas se guardan en un buffer, que está protegido gracias a un mutex, el cual evita corrupción en los datos y evita una condición de carrera, dado que este buffer es manejado en un hilo auxiliar. Por otro lado, también notifica a los hilos con `pthread_cond_wait` en caso de sobrepasar el límite establecido para el buffer y haciéndolo esperar a que se vacíe, asegurando completa sincronización con estas señales. De igual manera, se asegura que al ejecutar el programa, el pipe no exista desde antes, pues el receptor al terminar de ejecutarse usa `unlink` para remover el archivo de la carpeta.

Listing 1: Buffer con mutex

```

1 void anadirBuffer(struct Operaciones *op) {
2     pthread_mutex_lock(&mutex);
3     while (bufferCont >= BUFFER_TAM) {
4         pthread_cond_wait(&cond_no_lleno, &mutex);
5     }
6     buffer[bufferCont++] = *op;
7     pthread_cond_signal(&cond_no_vacio);
8     pthread_mutex_unlock(&mutex);
9 }

```

Por otro lado, está el named pipe que manda respuestas desde receptor hasta solicitante, dejando al tanto a solicitante de todas las actualizaciones posibles. Este pipe a diferencia del otro, se crea a partir del pid del proceso solicitante, ya que al tener la posibilidad de múltiples procesos solicitantes, de esta manera aseguramos que la respuesta se envíe al proceso que envió la operación en primer lugar. De igual manera que con el otro named pipe, solicitante se asegura de que no exista desde antes el pipe que se está creando, y con `unlink` elimina el pipe de la carpeta al terminar de ejecutar el programa. Además, se manejan errores al abrir los pipes de tal manera que se dan ciertos intentos al abrir el pipe, asegurando de dar oportunidad a que el pipe se abra de manera efectiva.

Listing 2: Creación pipe de respuesta

```

1  pid_t pid = getpid();
2  char pipeRecibe[20];
3  snprintf(pipeRecibe, sizeof(pipeRecibe), "pipe_%d", pid);
4
5  if (mkfifo(pipeRecibe, 0666) == -1 && errno != EEXIST) {
6      printf("Error al crear el pipe de respuesta %s\n",
7             pipeRecibe);
8      close(fd);
9      exit(1);
10 }
11
12 int fdResp = open(pipeRecibe, O_RDWR); //abierto en ambos
13        sentidos para evitar problemas
14 if (fdResp < 0) {
15     printf("Error al abrir el pipe de respuesta %s\n",
16            pipeRecibe);
17     close(fd);
18     unlink(pipeRecibe);
19     exit(1);
20 }

```

Listing 3: Ejemplo reintentos de pipe

```

1  while (intentos-- > 0) {
2      int bytes = read(fdResp, respuesta + total_bytes,
3                      sizeof(respuesta) - 1 - total_bytes);
4      if (bytes > 0) {
5          total_bytes += bytes;
6          if (respuesta[total_bytes - 1] == '\0') { // Mensaje
7              completo recibido
8              printf("Respuesta del receptor para operaci n
9                     %c, ISBN %d: %s\n", tipo, isbn, respuesta);
10             return;
11         }
12     } else if (bytes == 0) {
13         // Fin (pipe cerrado por el otro extremo)
14         printf("El pipe de respuesta %s fue cerrado por el
15                receptor\n", pipeRecibe);
16         return;
17     } else {
18         printf("Error al leer el pipe de respuesta \n");
19         return;
20     }
21     usleep(100000); // Esperar 100ms antes de reintentar
22 }

```

3.3 Estructuras de Datos y Archivos

- **Estructura Ejemplar:** Almacena el número, estado (D para disponible, P para prestado) y fecha de un ejemplar en formato dd-mm-aaaa.

Listing 4: Estructura ejemplar

```

1 struct Ejemplar {
2     int numero;
3     char status;
4     char fecha[11];
5 };

```

- **Estructura Libros:** Contiene el ISBN, nombre y arreglo de ejemplares de cada libro, que tiene un límite de 10 ejemplares por libro. Se tiene un límite de 100 libros para el proceso Receptor.

Listing 5: Estructura Libros

```

1 struct Libros {
2     int isbn;
3     char nombre[250];
4     int numEj;
5     struct Ejemplar ejemplares[MAX_EJEMPLAR];
6 };

```

- **Estructura Operaciones:** Representa las operaciones enviadas (tipo, nombre, ISBN, y PID, que será de utilidad frente a las respuestas del receptor). Se utiliza tanto en el proceso receptor como en el proceso solicitante.

Listing 6: Estructura Operaciones

```

1 struct Operaciones {
2     char tipo;
3     char nombre[250];
4     int isbn;
5     int pid;
6 };

```

- **Buffer:** Arreglo de **Operaciones** con espacio para hasta 10 operaciones, protegido por un mutex y condiciones de sincronización al momento de escribir y leer en él. Se estableció un límite de 10 operaciones dentro del buffer.

Listing 7: Buffer

```

1 struct Operaciones buffer[BUFFER_TAM];

```

- **Archivos de comunicación:**
 - **pipeReceptor:** Pipe principal para recibir solicitudes del solicitante. El nombre de este pipe esta a elección del usuario al momento de ejecutar el programa, siempre y cuando sea el mismo para tanto receptor como para solicitante.
 - **pipe_PID:** Pipes de respuesta generados dinámicamente por el PID del solicitante para recibir respuestas del receptor acerca de las operaciones realizadas y si fueron un éxito o no.
- **Archivos de datos:**

- `filedatos` (ej. `datos.txt`): Archivo de entrada con la base de datos inicial.
- `filesalida` (ej. `salida.txt`): Archivo de salida con el estado final de los libros tras todas las operaciones realizadas en el programa, queda a elección del usuario si quiere crear este archivo o no.
- `file` (ej. `operaciones.txt`): Archivo opcional dentro del proceso solicitante en donde se podrán colocar todas las operaciones que el usuario desee hacer.

3.4 Flujo de Procesos

El receptor lee la base de datos desde `filedatos` y crea los pipes necesarios. El solicitante envía operaciones a través de `pipeReceptor` que pueden ser enviadas a través de un menú o de un archivo de texto y recibe respuestas vía `pipe_PID`. Las operaciones de préstamo se procesan directamente en el hilo principal, mientras que devoluciones y renovaciones se manejan en el buffer para auxiliar1. Mientras se ejecutan estas tareas, el usuario podrá manejar reportes o salir de la ejecución del proceso mediante `auxiliar2`, y se puede generar una base de datos actualizada si el usuario lo desea al momento de ejecutar. Para salir, el solicitante tiene que detener el envío de operaciones mandando una operación de salida para luego escribir 's' en la consola y así cerrando los recursos, mientras que en el proceso receptor el usuario puede digitar 's' en cualquier momento para salir; sin embargo, la ejecución solo se detendrá una vez el proceso solicitante se haya detenido, de lo contrario, se quedará esperando a que esto ocurra.

4 Plan de Pruebas

Para mostrar el funcionamiento correcto del proyecto, se utilizará el caso de uso planteado por el enunciado, en donde un solicitante solicita un préstamo, una renovación y hace una devolución, terminando su ejecución con el comando Q. Además de esto, se mostrarán casos en donde se digite el nombre y el ISBN del libro erróneamente, y en donde no se encuentren más ejemplares disponibles para préstamo o no hayan prestado para hacer una renovación. Se guardaran en un archivo de salida los resultados para comparar los cambios con la base de datos dada al proceso receptor en un inicio. En la siguiente tabla, se detalla resumidamente los casos de prueba a utilizar. Por último, trataremos estos mismos casos, pero a partir de múltiples procesos solicitantes, demostrando la correcta funcionalidad en estos casos. Las pruebas también validan la sincronización entre hilos y procesos, asegurando que no haya condiciones de carrera ni corrupción de datos, especialmente en escenarios con múltiples solicitantes.

Caso	Tipo de operación	Entrada	Resultado Esperado
1	Devolución válida	D, Operating Systems, 2233	Estado cambiado a 'D' para el ejemplar prestado.
2	Renovación válida	R, Operating Systems, 2233	Fecha extendida 7 días (ej. de 01-10-2021 a 08-10-2021).
3	Préstamo válido	P, Programming Languages, 2240	Ejemplar prestado exitosamente con nueva fecha de devolución.

Continúa en la siguiente página

Caso	Tipo de operación	Entrada	Resultado Esperado
4	ISBN no existente	P, Libro Z, 999	Error: ISBN no encontrado.
5	Sin ejemplares disponibles	P, Data Bases, 2234	Error: No se encontró un ejemplar disponible para ISBN 2234.
6	Formato inválido	Q, Data Bases, 2234	Operación inválida. Debe ser P, D, R o Q.
7	Terminación	Q, Salir, 0	Receptor envía "Solicitante X ha terminado", ambos procesos terminan.

4.1 Diagrama de secuencia

A continuación, se presenta el diagrama de secuencia del caso de uso previamente explicado, mostrando la dinámica que sigue el programa para comprender mejor su funcionamiento.

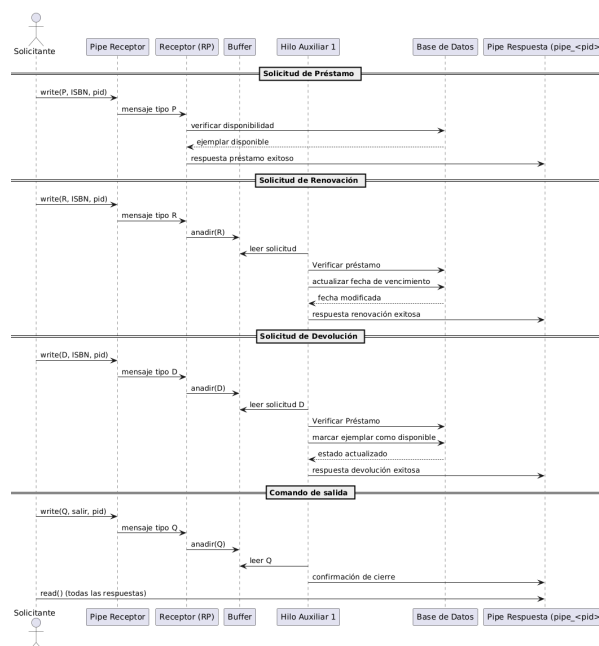


Figure 1: Diagrama de secuencia del caso de uso. Elaboración propia

4.2 Capturas de ejecución

Para la prueba, se usará la siguiente base de datos:

```
witzhid04@DESKTOP-2J8MCRP:~/ProyOp$ make
gcc -Wall -pthread -o receptor receptor.c
gcc -Wall -pthread -o solicitante solicitante.c
```

Figure 4: Ejecución con make

```
GNU nano 7.2                                basedatos.txt
Operating Systems, 2233, 4
1, D, 1-10-2021
2, D, 1-10-2021
3, P, 1-10-2021
4, P, 1-10-2021
Data Bases, 2234, 2
1, P, 1-10-2021
2, P, 1-10-2021
Programming Languages, 2240, 1
1, D, 1-10-2021
```

Figure 2: Base de datos del plan de pruebas

Para compilar, se utiliza el siguiente makefile y se ejecuta escribiendo make dentro de la consola de comandos, tal y como se muestra en las figuras. Todo el código es compilado y ejecutado en una máquina Linux con sistema operativo Ubuntu.

```
GNU nano 7.2                                makefile
# Compilador y banderas
CC = gcc
CFLAGS = -Wall -pthread

# Archivos fuente y encabezado
RECEPTOR = receptor
SOLICITANTE = solicitante

# Regla principal
all: receptor solicitante

# Compilar receptor
receptor: receptor.c receptor.h
    $(CC) $(CFLAGS) -o $(RECEPTOR) receptor.c

# Compilar solicitante
solicitante: solicitante.c solicitante.h
    $(CC) $(CFLAGS) -o $(SOLICITANTE) solicitante.c

# Limpiar ejecutables y pipes
clean:
    rm -f receptor solicitante pipe.* pipeReceptor
```

Figure 3: archivo makefile

Para mostrar el caso de uso, se usará primero el menú realizado para mostrar uno por uno las operaciones previamente mencionadas, para luego utilizar un archivo de texto con las mismas operaciones, mostrando que queden iguales y demostrando la correcta funcionalidad de ambos métodos. El proceso receptor se ejecutará primero, haciendo uso de verbose para mostrar las operaciones en consola y de la salida para mostrar los cambios con respecto a la base de datos inicial.

```
witzhid04@DESKTOP-2J8MCRP:~/ProyOp$ ./receptor
Use: ./receptor -p pipeReceptor -f filedatos [-v] [-s filesalida]
witzhid04@DESKTOP-2J8MCRP:~/ProyOp$ ./receptor -p pipeReceptor -v -s salida.txt
```

Figure 5: Ejecución receptor

Por otro lado, para ejecutar solicitante, cómo se explicó con anterioridad, lo ejecutaremos de las dos maneras posibles, sin el archivo de operaciones y con este para mostrar el correcto funcionamiento de los procesos.

```
witzhid04@DESKTOP-2J8MCRP:~/ProyOp$ ./solicitante
Use: ./solicitante [-i file] -p pipeReceptor
witzhid04@DESKTOP-2J8MCRP:~/ProyOp$ |
```

Figure 6: Ejecución solicitante

A continuación, se muestra un ejemplo de solicitud de préstamo.

```

witezid04@DESKTOP-2J8MCRP:~/ProyOp$ ./solicitante -p pipeReceptor
Operación (D/R/P): D
Nombre del libro: Operating Systems
ISBN: 2233
Respuesta del receptor para operación D, ISBN 2233: Devolución exitosa: ISBN 2233, Ejemplar 3
Introduzca si desea enviar otra operación [0 para sí, 1 para no]
:

```

Figure 7: Devolución libro con solicitante

```

witezid04@DESKTOP-2J8MCRP:~/ProyOp$ ./receptor -p pipeReceptor -f basedatos.txt -v -s salida.txt
Libro leído: Operating Systems, ISBN: 2233, NumEj: 4
Ejemplar leído: Num: 1, Status: D, Fecha: 01-10-2021
Ejemplar leído: Num: 2, Status: D, Fecha: 01-10-2021
Ejemplar leído: Num: 3, Status: P, Fecha: 01-10-2021
Ejemplar leído: Num: 4, Status: P, Fecha: 01-10-2021
Libro leído: Data Bases, ISBN: 2234, NumEj: 2
Ejemplar leído: Num: 1, Status: P, Fecha: 01-10-2021
Ejemplar leído: Num: 2, Status: P, Fecha: 01-10-2021
Libro leído: Programming Languages, ISBN: 2240, NumEj: 1
Ejemplar leído: Num: 1, Status: D, Fecha: 01-10-2021
Recibido: tipo = D, nombre = Operating Systems, isbn = 2233, pid = 4143
Devolución realizada del libro: ISBN 2233, Ejemplar 3

```

Figure 8: Devolución libro con receptor

Como se puede observar, el receptor recibe los datos que solicitante le manda, y menciona que el libro fue exitosamente devuelto. De igual manera, este mismo mensaje aparece en el proceso solicitante, siendo esta la respuesta del receptor.

Ahora, revisaremos un ejemplo en donde se renueva el préstamo de un libro.

```

Operación (D/R/P): R
Nombre del libro: Operating Systems
ISBN: 2233
Respuesta del receptor para operación R, ISBN 2233: Renovación exitosa: ISBN 2233, Ejemplar 4
Introduzca si desea enviar otra operación [0 para sí, 1 para no]
:

```

Figure 9: Renovación libro solicitante

```

Recibido: tipo = R, nombre = Operating Systems, isbn = 2233, pid = 4143
Renovación procesada: ISBN 2233, Ejemplar 4, Nueva fecha: 08-10-2021

```

Figure 10: Renovación libro receptor

Como se puede observar, el libro se renueva con éxito, pasando la fecha del primero de octubre de 2021 al 8 de octubre de ese mismo año. De igual manera, se puede observar cómo el pipe de respuesta aún funciona de manera exitosa, mostrando la renovación exitosa también dentro del proceso solicitante.

Hecho esto, ahora revisaremos un ejemplo de préstamo de libro.

```

Operación (D/R/P): P
Nombre del libro: Programming Languages
ISBN: 2240
Respuesta del receptor para operación P, ISBN 2240: Préstamo exitoso: ISBN 2240, Ejemplar 1
Introduzca si desea enviar otra operación [0 para sí, 1 para no]
:

```

Figure 11: Préstamo en solicitante


```

Recibido: tipo = P, nombre = Programming Languages, isbn = 2240,
pid = 4143
Préstamo realizado del libro: ISBN 2240, Ejemplar 1

```

Figure 12: Préstamo en receptor

Tal y como se puede observar, se hace el préstamo con éxito, mandando el mensaje de respuesta al solicitante y guardando la información. Por lo que visto ya cada una de las funcionalidades implementadas, revisaremos los casos donde pueden mandar un error, tal y como se explicó antes con la tabla de casos de uso.

```

Operación (D/R/P): P
Nombre del libro: Libro Z
ISBN: 999
Respuesta del receptor para operación P, ISBN 999: Error: ISBN 9
99 no encontrado o nombre erróneo
Introduzca si desea enviar otra operación [0 para sí, 1 para no]
:

```

Figure 13: ISBN no existente en solicitante

```

Operación (D/R/P): P
Nombre del libro: Data Bases
ISBN: 2234
Respuesta del receptor para operación P, ISBN 2234: Error: No se
encontró un ejemplar disponible para ISBN 2234
Introduzca si desea enviar otra operación [0 para sí, 1 para no]
:

```

Figure 14: Sin ejemplares en solicitante

```

Reporte:
D, Operating Systems, 2233, 1, 01-10-2021
D, Operating Systems, 2233, 2, 01-10-2021
D, Operating Systems, 2233, 3, 01-10-2021
P, Operating Systems, 2233, 4, 08-10-2021
P, Data Bases, 2234, 1, 01-10-2021
P, Data Bases, 2234, 2, 01-10-2021
P, Programming Languages, 2240, 1, 08-10-2021
Recibido: tipo = P, nombre = Data Bases, isbn = 2234, pid = 6618
No se encontró un ejemplar disponible para ISBN 2234

```

Figure 15: Sin ejemplares en receptor

```

Operación (D/R/P): Q
Nombre del libro: Data Bases
ISBN: 2234
Operación inválida. Debe ser D, R o P.
Operación (D/R/P):

```

Figure 16: Formato incorrecto en solicitante

Ya revisados todos los casos de uso, incluyendo en los que el programa manda un mensaje de error, podemos revisar cómo quedó la base de datos en comparación a la original.

GNU nano 7.2	basedatos.txt	GNU nano 7.2	salida.txt
	Operating Systems, 2233, 4		Operating Systems, 2233, 4
	1, D, 1-10-2021		1, D, 01-10-2021
	2, D, 1-10-2021		2, D, 01-10-2021
	3, P, 1-10-2021		3, D, 01-10-2021
	4, P, 1-10-2021		4, P, 08-10-2021
	Data Bases, 2234, 2		Data Bases, 2234, 2
	1, P, 1-10-2021		1, P, 01-10-2021
	2, P, 1-10-2021		2, P, 01-10-2021
	Programming Languages, 2240, 1		Programming Languages, 2240, 1
	1, D, 1-10-2021		1, P, 08-10-2021

Figure 17: Comparación bases de datos

Se pueden notar varias diferencias a lo largo de ambas, gracias a las operaciones realizadas desde el programa, por ejemplo, solo queda un ejemplar prestado en Operating Systems, el cual está renovado y tiene plazo hasta el 8 de octubre, tal y como hicimos en el programa. Se puede observar cómo el único ejemplar de Programming Languages cambió a estar prestado y con un plazo nuevo para ser devuelto, siendo de igual manera, el 8 de octubre. Esto muestra que efectivamente los cambios que hicimos dentro del programa se vieron reflejados en la base de datos de salida.

Por último, haremos estas mismas operaciones con un archivo de texto en el proceso solicitante, mostrando de esta manera la completa funcionalidad de todo el programa.

```
GNU nano 7.2 operaciones.txt
D, Operating Systems, 2233
R, Operating Systems, 2233
P, Programming Languages, 2240
P, Libro Z, 999
P, Data Bases, 2234
Q, Data Bases, 2234
Q, Salir, 0
```

Figure 18: Archivo con operaciones a realizar

```

# Solicitante
./solicitante -i operaciones
Resposta del receptor para operación D, ISBN 2233: Devolución e
citos: ISBN 2233, Ejemplar 3
Resposta del receptor para operación R, ISBN 2233: Renovación e
citos: ISBN 2233, Ejemplar 4
Resposta del receptor para operación P, ISBN 2240: Préstamo exi
to: ISBN 2240, Ejemplar 1
Resposta del receptor para operación P, ISBN 999: Error: ISBN 9
99 no encontrado o nombre erróneo
Resposta del receptor para operación P, ISBN 2234: Error: No se
encontró un ejemplar disponible para ISBN 2234
Egreso "x" para salir:

# Receptor
./receptor -p pipeReceptor -
f basedatos.txt -v -s salida.txt
Libro leído: Operating Systems, ISBN: 2233, NumJ: 4
Ejemplar leído: Num: 1, Status: S, Fecha: 01-10-2021
Ejemplar leído: Num: 2, Status: D, Fecha: 01-10-2021
Ejemplar leído: Num: 3, Status: S, Fecha: 01-10-2021
Ejemplar leído: Num: 4, Status: P, Fecha: 01-10-2021
Libro leído: Data Bases, ISBN: 2234, NumJ: 2
Ejemplar leído: Num: 1, Status: P, Fecha: 01-10-2021
Ejemplar leído: Num: 2, Status: P, Fecha: 01-10-2021
Libro leído: Programming Languages, ISBN: 2240, NumJ: 1
Ejemplar leído: Num: 1, Status: D, Fecha: 01-10-2021
Recibido: tipo = D, nombre = Operating Systems, isbn = 2233, pid
= 6702
Devolución realizada del libro: ISBN 2233, Ejemplar 3
Recibido: tipo = R, nombre = Operating Systems, isbn = 2233, pid
= 6702
Renovación procesada: ISBN 2233, Ejemplar 4, Nueva fecha: 08-10-
2021
Recibido: tipo = P, nombre = Programming Languages, isbn = 2240,
pid = 6702
Préstamo realizado del libro: ISBN 2240, Ejemplar 1
Recibido: tipo = P, nombre = Libro Z, isbn = 999, pid = 6702
ISBN 999 no encontrado
Recibido: tipo = P, nombre = Data Bases, isbn = 2234, pid = 6702
No se encontró un ejemplar disponible para ISBN 2234
Recibido: tipo = Q, nombre = Salir, isbn = 0, pid = 6702

```

Figure 19: Resultados con archivo de texto

Tal y como se observa en la consola, los resultados son los esperados y coinciden con los realizados anteriormente con la opción del menú, mostrando la completa y correcta funcionalidad del programa que hace las tareas necesarias frente al sistema para préstamo de libros.

4.3 Múltiples procesos solicitantes

Para manejar esto, vamos a utilizar 3 procesos solicitantes, donde 2 de ellos serán ejecutados a partir del siguiente archivo de texto.

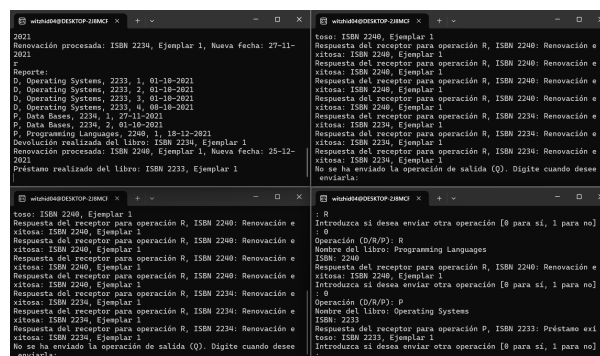
```

D, Operating Systems, 2233
R, Operating Systems, 2233
P, Programming Languages, 2240
P, Libro Z, 999
P, Data Bases, 2234
R, Data Bases, 2234
P, Programming Languages, 2240
P, Programming Languages, 2240
P, Programming Languages, 2240
R, Programming Languages, 2240
R, Programming Languages, 2240
R, Programming Languages, 2240
R, Programming Languages, 2240

```

Figure 20: Archivo de texto múltiples procesos

De esta forma, también se pondrá a prueba el uso de la sincronización, demostrando que hace buen uso de la condición `cond_wait` usada cuando se sobrepasaba el límite del buffer. El último proceso que se ejecute será con la opción del menú, realizando las operaciones de devolución, renovación, y préstamo según como quede la base de datos tras los dos procesos con archivos de texto. Al igual que antes, vamos a usar el archivo de salida para comparar las bases de datos sin la operación salir para no cerrar ningún pipe. No se utilizará `verbose` para mayor facilidad al verificar los resultados con los procesos solicitantes.

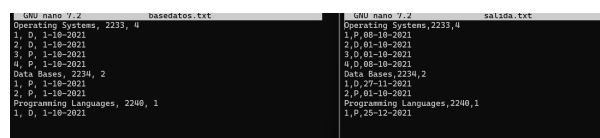


```

2021
Renovación procesada: ISBN 2234, Ejemplar 1, Nueva fecha: 27-11-2021
Reporte:
D, Operating Systems, 2233, 1, 01-10-2021
D, Operating Systems, 2233, 2, 01-10-2021
D, Operating Systems, 2233, 3, 01-10-2021
D, Operating Systems, 2233, 4, 01-10-2021
P, Data Bases, 2234, 1, 27-11-2021
P, Data Bases, 2234, 2, 01-10-2021
P, Programming Languages, 2240, 1, 18-12-2021
Renovación realizada del libro: ISBN 2234, Ejemplar 1
Préstamo procesado: ISBN 2240, Ejemplar 1, Nueva fecha: 25-12-2021
Préstamo realizado del libro: ISBN 2233, Ejemplar 1

```

Figure 21: Múltiples procesos solicitante



```

GNU nano 7.2 basedatos.txt
Operating Systems, 2233, 4
1, D, 1-10-2021
2, D, 1-10-2021
3, P, 1-10-2021
4, P, 1-10-2021
Data Bases, 2234, 2
1, P, 1-10-2021
2, P, 1-10-2021
Programming Languages, 2240, 1
1, D, 1-10-2021

```

```

GNU nano 7.2 salida.txt
Operating Systems, 2233, 4
1, P, 08-10-2021
2, D, 01-10-2021
3, D, 01-10-2021
4, D, 08-10-2021
Data Bases, 2234, 2
1, D, 27-11-2021
2, P, 01-10-2021
Programming Languages, 2240, 1
1, P, 25-12-2021

```

Figure 22: Salida múltiples procesos

Como se ve en las previas figuras, el proceso receptor efectivamente es capaz de recibir varias operaciones de múltiples procesos solicitantes y guardarlas en la base de datos (por ejemplo, las múltiples renovaciones en programming languages por parte de ambos archivos de texto que dieron lugar a que se tenga el préstamo hasta el 25 de diciembre) enviando respuestas solo al proceso que haya enviado la operación y también evitando cualquier condición de carrera manteniendo la información sin corrupción alguna, mostrando que el buffer gestiona correctamente el límite de 10 operaciones, utilizando `pthread_cond_wait` para pausar el envío cuando está lleno.

5 Evaluación de Funcionalidades

5.1 Funcionalidades Implementadas

Según el enunciado, las funcionalidades requeridas son:

- **Préstamo (P):** Implementado y funcionando correctamente. Permite prestar un ejemplar disponible, actualizando su estado a P y la fecha de devolución.
- **Devolución (D):** Implementado y funcionando correctamente. Cambia el estado de un ejemplar de P a D y se hace dentro del hilo auxiliar.
- **Renovación (R):** Implementado y funcionando correctamente. Extiende la fecha de devolución en 7 días y se hace dentro del hilo auxiliar.
- **Reporte (r):** Implementado y funcionando correctamente. Muestra el estado de todos los ejemplares.
- **Salida (s y Q):** Implementado y funcionando adecuadamente siempre y cuando el solicitante termine su ejecución con el comando Q, de esa manera, podrán salir ambos procesos con éxito.
- **Respuestas a solicitante:** Implementado y funcionando correctamente. El proceso receptor envía una respuesta a cada operación que envíe solicitante, definiendo si la operación fue un éxito o no, mostrando la razón de fracaso en caso de ocurrir.
- **Múltiples procesos solicitantes:** Implementado y funcionando correctamente siempre y cuando los pipes nunca se cierren, es decir, ninguno de los procesos solicitante se puede salir hasta que los demás terminen de enviar sus operaciones. Dicho esto, se pudo observar que esta funcionalidad fue implementada adecuadamente, recibiendo las operaciones y mandando las respuestas al proceso que se encargó de enviar dicha operación, manteniendo un rendimiento óptimo con varios procesos a la vez.

5.2 Funcionalidades no Implementadas

No se identificaron funcionalidades del enunciado que no estén implementadas o funcionen inadecuadamente. Todas las operaciones requeridas están cubiertas y han sido validadas con casos de prueba. Hay limitaciones, como la mencionada anteriormente de que los procesos solicitantes deben completar sus operaciones antes de salir y que el receptor no podrá terminar de ejecutarse hasta que el solicitante lo haga, o el hecho de que el buffer tiene un tamaño fijo de 10 operaciones, lo que podría limitar el rendimiento con un número muy alto de solicitantes simultáneos. Pero estas limitaciones son detalles más que funcionalidades no implementadas.

6 Conclusiones

El sistema implementado cumple con los requisitos del enunciado, utilizando una arquitectura basada en procesos y hilos con comunicación mediante named pipes. La sincronización entre el receptor y el solicitante se gestiona efectivamente con un buffer compartido y condiciones de espera para los casos de devolución y renovación de libros, ya

que el préstamo de estos se realiza aparte. El diagrama de secuencia confirma el flujo correcto de las operaciones, el cual es demostrado con las capturas de ejecución y las breves explicaciones de estas. El proyecto también está abierto a futuras mejoras, haciendo que pueda tener nuevas implementaciones y mejoras a futuro.

7 Repositorio

Adjunto enlace al repositorio donde se encuentran todos los archivos del proyecto. <https://github.com/W>

References

- [1] GeeksforGeeks, “Named Pipe or FIFO with example in C,” 2018, [Online]. Available: <https://www.geeksforgeeks.org/named-pipe-fifo-examplec-program/>.
- [2] The Linux Programming Interface, “POSIX Threads (pthreads),” [Online]. Available: <https://man7.org/linux/man-pages/man7/pthreads.7.html>.
- [3] GNU Project, “GNU Make Manual,” 2023, [Online]. Available: <https://www.gnu.org/software/make/manual/make.html>.
- [4] IEEE Std 1003.1-2017, “POSIX.1-2017: The Open Group Technical Standard for POSIX Threads,” 2017.