Proyecto Sistema para el Préstamo de Libros

Viviana Gómez, Roberth Méndez, Luz Salazar, Guden Silva

Pontificia Universidad Javeriana

Sistemas Operativos

Profesor: John Jairo Corredor Franco

26 de Mayo de 2025

1. Introducción

En el presente informe se documenta el desarrollo de un sistema para el préstamo de libros en una biblioteca, implementado en lenguaje ANSI C. El sistema permite realizar operaciones de préstamo, renovación y devolución de libros mediante el uso de múltiples procesos e hilos POSIX. Para procesar ciertas solicitudes, se implementa el esquema productor-consumidor a través de un buffer circular. Para la comunicación entre procesos se hace uso de named pipes. Además, se emplean mecanismos de sincronización con mutex, así como archivos de texto que funcionan como base de datos para almacenar la información de los libros de la biblioteca.

2. Diseño e Implementación del Sistema

En esta sección se describe la arquitectura desarrollada, detallando el comportamiento de cada componente, los mecanismos utilizados para la comunicación entre procesos, la sincronización entre hilos, el parseo de argumentos y la forma en que se gestionan las distintas operaciones del sistema de biblioteca.

2.1 Componentes Principales

El sistema para el Préstamo de Libros está compuesto por dos tipos principales de procesos: los Procesos Solicitantes (PS) y el Proceso Receptor (RP). Los PS se encargan de recibir las solicitudes del usuario y enviarlas al RP, quien es el encargado de ejecutar las operaciones correspondientes: préstamo, renovación y devolución de libros. De esta forma, los procesos solicitantes gestionan la interacción con el usuario, mientras que el Proceso Receptor se encarga del procesamiento interno y la actualización de la base de datos.

Cada uno de estos procesos recibe diferentes argumentos según lo requerido al momento de su ejecución. En el caso de los PS, las solicitudes del usuario pueden ingresarse por consola o desde un archivo de texto. Esto se define mediante las opciones -i para especificar el archivo de entrada y -p para indicar el named pipe a través del cual se envían las solicitudes.

Por otro lado, el RP acepta las opciones -p para el named pipe de recepción, -f para el archivo de la base de datos inicial, -v para mostrar las operaciones en consola, y -s para establecer el archivo de salida donde se almacenará la base de datos actualizada al finalizar la ejecución.

Para gestionar estos argumentos, se utilizó la librería *getopt.h*, que permite procesar opciones de línea de comandos, usando la función getopt() con la variable global optarg para facilitar la lectura de los valores asociados a cada opción (por ejemplo, el nombre del archivo tras -i). La asignación de estos valores a variables locales se realiza mediante un bloque condicional switch, como se muestra en el siguiente bloque de código correspondiente al PS.

```c
//Parseo de argumentos
while ((opt = getopt(argc, argv, "i:p:")) != -1) {
    switch (opt) {
        case 'i':
            archivoEntrada = optarg; //Guarda el nombre de archivo de entrada
            break;
        case 'p':
            snprintf(pipeSolicitudes, sizeof(pipeSolicitudes), "/tmp/%s", optarg); //Concatena el nombre del pipe con /tmp/
            break;
        default:
            printf("Uso: %s [-i archivo_entrada] -p named pipe\n", argv[0]);
            exit(1);
    }
}
```

Imagen 1 - Parseo de Argumentos

2.1.1 Comunicación entre Procesos

Para facilitar la comunicación entre los procesos solicitantes y el Proceso Receptor se definió una estructura de datos llamada *Solicitud*, que almacena la información necesaria para representar una operación de préstamo, renovación, devolución, salida u otros tipos de solicitudes que se crearon para mejorar el funcionamiento del sistema y que se explicarán más adelante.

```c
typedef struct {
    char operacion; // 'D' (devolver), 'R' (renovar), 'P' (prestar), 'Q' (salir), 'I' (inicio), 'S' (Finalizar)
    char libro[100]; // Nombre del libro
    int isbn; // ISBN del libro
    char pipe[100]; // Nombre del named pipe por el que recibirá la respuesta
} Solicitud;
```

Imagen 2 - Definición de la Estructura Solicitud

Estas solicitudes se envían a través de un named pipe, cuyo nombre se especifica como argumento al ejecutar tanto los PS como el RP. Mediante este pipe, todos los PS podrán transmitir sus operaciones hacia el RP. Para una mayor facilidad, denominaremos a este named pipe como el pipe de solicitudes.

Por otro lado, cada Proceso Solicitante crea su propio named pipe para recibir las respuestas del Proceso Receptor. El nombre de este pipe es único y se construye utilizando el ID del Proceso Solicitante. Este identificador también se incluye en la estructura *Solicitud*, para que el RP sepa por cuál pipe debe enviar la respuesta correspondiente a cada solicitud. A estos named pipes los denominaremos pipes de respuestas.

La recepción de respuestas en los PS se maneja con ayuda de un *hilo lector*, el cual permanece a la espera de mensajes en su pipe correspondiente mientras que este exista. La siguiente imagen muestra la lógica empleada por este hilo.

```c
void *leer(void* arg){

    int idPipeRespuestas = *(int *)arg;
    char mensaje[MAX_LINE];

    while (read(idPipeRespuestas, mensaje, sizeof(mensaje)) > 0)
        printf("Mensaje recibido: %s\n", mensaje);

    return NULL;

}
```

Imagen 3 - Función usada por el Hilo Lector

2.1.2 Procesos Solicitantes

Cada vez que se crea un nuevo Proceso Solicitante, este se conecta inmediatamente al pipe de solicitudes, cuyo nombre fue ingresado como argumento (el mismo que se ingresó al ejecutar el RP). Luego, el PS crea su propio pipe de respuestas, utilizando su PID para asegurar un nombre único.

Con ambos canales de comunicación establecidos, el PS envía una solicitud inicial al RP con la operación *I*, indicando que un nuevo Proceso Solicitante está activo. Posteriormente, el PS inicia el *hilo lector* que permanece escuchando el pipe de respuestas para recibir notificaciones del RP.

Dependiendo del modo de ejecución, el Proceso Solicitante puede funcionar en dos casos distintos. Si se ejecuta en modo lectura de un archivo, leerá línea por línea un archivo de texto, interpretando cada entrada como una solicitud, hasta encontrar una operación con el carácter *Q*, que indica la finalización del proceso. En caso de ejecutarse con una lectura por consola, se le mostrará al usuario un menú con las operaciones disponibles (préstamo, renovación o devolución) o con la opción de salida.

Finalmente, cada solicitud generada por el usuario será enviada por el pipe de solicitudes al Proceso Receptor. Una vez procesada la operación, el RP responde a través del pipe de respuestas, y el hilo lector imprime en consola un mensaje indicando si la operación fue realizada con éxito o no.

2.1.3 Proceso Receptor

Al iniciar el Proceso Receptor, se crea inmediatamente el pipe de solicitudes, utilizando el nombre de pipe indicado con la opción -p. De igual forma, se carga la base de datos de libros con el nombre del archivo de texto especificado con la opción -f. Esta base de datos se almacena en un arreglo de estructuras llamado *bd*, donde cada elemento representa un libro mediante la estructura de datos *Libro*. Esta estructura contiene el nombre del libro, el ISBN, el estado de cada ejemplar y una fecha asociada (fecha actual).

Después de cargar la base de datos, el RP genera dos hilos auxiliares. El hilo auxiliar 1 se encarga de procesar las operaciones de renovación y devolución de libros, mientras que el hilo auxiliar 2 recibe los comandos que ingrese el usuario desde la consola. Los posibles comandos para el RP son: *s*, que finaliza el programa, y *r* que genera un reporte con las operaciones realizadas hasta el momento. Por otro lado, las operaciones de préstamos son gestionadas por el Proceso Receptor.

Una vez creados los hilos, el Proceso Receptor comienza a recibir las peticiones enviadas por los PS. Para llevar el control de los procesos solicitantes activos, se implementó un contador que incrementa cuando se recibe una solicitud de inicio *I* y disminuye al recibir una solicitud de salida *Q*. Cuando este contador llega a cero, se considera que todos los PS han finalizado, lo cual indica que se tiene que terminar el RP. Para ello, se modifica la operación de la última solicitud por una operación especial *S*, que le indica al RP que debe cerrar el pipe de solicitudes y esperar a que finalicen los hilos auxiliares antes de finalizar el proceso.

Cuando se indica la finalización del programa, ya sea desde el Proceso Receptor o desde el hilo auxiliar 2, uno de los dos establece el contador de solicitudes del buffer en -1. Este valor funciona como una señal para el hilo auxiliar 1, indicándole que debe finalizar su ejecución. Asimismo, el

hilo auxiliar 2 también verifica periódicamente este contador a través de un ciclo, y al detectar que ha sido establecido en -1, procede a cerrar su ejecución.

Cada vez que se recibe una solicitud, si el programa fue ejecutado con el flag -v (verbose), el contenido de la solicitud se muestra en consola. Una vez procesada la solicitud, se registra en un arreglo de estructuras *Libro* llamado *reporte*, que almacena las operaciones que va realizando el RP, para posteriormente mostrarlas al usuario mediante el comando *r*.

Por último, la distribución de las solicitudes de devolución y renovación entre el Proceso Receptor y el hilo auxiliar 1, se realiza siguiendo el esquema productor-consumidor a través de un buffer circular. Este buffer tiene una capacidad máxima de diez solicitudes y es compartido entre el RP (productor) y el hilo auxiliar 1 (consumidor). Las solicitudes de devolución o renovación se agregan al buffer, mientras que el hilo auxiliar 1 las extrae y ejecuta las operaciones correspondientes en la base de datos. Para gestionar este flujo, se tiene un contador de la cantidad de solicitudes actuales en el buffer, así como dos índices: in, que indica la posición donde el RP insertará la siguiente solicitud, y out, que señala cual será la próxima solicitud a procesar por el hilo auxiliar 1. Ambos índices se reinician a cero cuando alcanzan el final del buffer, implementando así el comportamiento circular.

2.2 Sincronización entre Hilos

Dado que el Proceso Receptor opera de forma concurrente con dos hilos auxiliares, se nos presenta un problema común dentro de los sistemas concurrentes: la posibilidad de que múltiples hilos intenten acceder simultáneamente a los mismos recursos compartidos, lo que puede generar condiciones de carrera. Para evitar esto, se implementaron mecanismos de sincronización de hilos utilizando mutex y variables de condición.

En total, se definieron tres mutex distintos para proteger los recursos compartidos: uno para el buffer circular, otro para el arreglo de estructuras que representa la base de datos de libros *bd*, y un tercero para el arreglo de estructuras *reporte*.

Dentro del buffer circular, el hilo auxiliar 1 es el encargado de leer las solicitudes de devolución y renovación almacenadas en este. Por ello, al momento de acceder al buffer, este hilo bloquea el mutex del buffer garantizando acceso exclusivo a esta variable. Por otro lado, el RP es el encargado de agregar solicitudes al buffer circular cuando se recibe una devolución o renovación, por lo que también debe bloquear el mutex antes de insertar una nueva solicitud. De este modo, se asegura que en ningún momento ambos hilos accedan simultáneamente al buffer.

Además, se emplearon variables de condición para manejar los casos en los que el buffer está vacío o lleno. Si el buffer está vacío, el hilo auxiliar 1 se bloquea hasta recibir una señal del RP en la que indique que hay una nueva solicitud disponible. Esta señal se envía cuando el RP inserta una nueva solicitud en el buffer. Por otro lado, si el buffer está lleno, el RP también se bloquea y solo continúa cuando el hilo auxiliar 1 procese alguna solicitud y envíe una señal indicando que ya hay espacio disponible. Durante estas esperas, los mutex se liberan temporalmente. Una vez recibida la señal se vuelven a bloquear automáticamente, lo que mejora la sincronización de los hilos.

En las siguientes imágenes se muestra el código con el uso del mutex para el buffer circular, junto con la lógica del esquema productor-consumidor.

```c
void *auxiliar1_(void *arg) {

    while(1) {

        pthread_mutex_lock(&mutexBuffer);

        while (contador == 0)
            pthread_cond_wait(&noVacio, &mutexBuffer);

        if(contador == -1) //Verifica si el hilo auxiliar2 le indicó que se cierre
            break;

        // Procesar solicitud
        Solicitud soli = buffer[out]; // Lee una solicitud del búfer
        out = (out + 1) % BUFFER_SIZE; // Cambia la posición para sacar la próxima solicitud
        contador--;

        pthread_cond_signal(&noLleno); // Envía señal para que el hilo principal sepa que el búfer no está lleno
        pthread_mutex_unlock(&mutexBuffer);
```

Imagen 4 - Mutex Buffer en Hilo Auxiliar 1

```c
void devolverRenovar(Solicitud solicitud){

    // Agregar al búfer para el hilo auxiliar
    pthread_mutex_lock(&mutexBuffer);

    while (contador == BUFFER_SIZE)
        pthread_cond_wait(&noLleno, &mutexBuffer);

    buffer[in] = solicitud;
    in = (in + 1) % BUFFER_SIZE;
    contador++;

    pthread_cond_signal(&noVacio);
    pthread_mutex_unlock(&mutexBuffer);

}
```

Imagen 5 - Mutex Buffer en RP

En cuanto al arreglo *reporte*, este es accedido por los tres componentes del RP: el proceso principal (para registrar solicitudes de préstamo), el hilo auxiliar 1 (para registrar devoluciones y renovaciones), y el hilo auxiliar 2 (que imprime el reporte en consola cuando el usuario lo solicita). Para evitar accesos simultáneos, se protege con un mutex exclusivo, que se bloquea antes de leer o modificar el arreglo y se libera inmediatamente después. En la siguiente imagen se muestra un ejemplo de uso del mutex de reporte.

```c
//Asegura exclusividad para modificar el reporte con un candado
pthread_mutex_lock(&lock);

// Guarda una copia del libro devuelto en el arreglo de reportes
for(int i = 0; i < 100; i++)
    if(reporte[i].isbn == 0){
        reporte[i] = *devolucion;
        break;
    }
//Libera el candado
pthread_mutex_unlock(&lock);
```

Imagen 6 - Ejemplo de Uso del Mutex Reporte

De forma similar, se protege el acceso a la base de datos *bd*, ya que tanto el RP como el hilo auxiliar 1 modifican su contenido. El RP actualiza la base de datos cuando se procesa una solicitud de préstamo, mientras que el hilo auxiliar 1 lo hace para devoluciones y renovaciones. El mutex de la base de datos garantiza que solo un hilo pueda modificar su contenido a la vez, evitando inconsistencias y manteniendo la integridad de los datos. En la siguiente imagen se muestra un ejemplo de uso del mutex de la base de datos.

```
//Buscar el libro de la operación
pthread_mutex_lock(&mutexBD); // Mutex que asegura que solo un hilo acceda a la base de datos

if(!buscar_libro(soli.isbn, soli.libro)){
    snprintf(mensaje, sizeof(mensaje), "Libro no encontrado: %s, %d", soli.libro, soli.isbn);
    write(fifoRespuesta, mensaje, strlen(mensaje) + 1);
    pthread_mutex_unlock(&mutexBD); // Desbloquea el mutex de la Base de Datos
    continue;
}

// Aquí se actualizaría la BD según la operación
soli.operacion == 'D' ? devolver(soli, fifoRespuesta) : renovar(soli, fifoRespuesta);
pthread_mutex_unlock(&mutexBD); // Desbloquea el mutex de la Base de Datos
```

Imagen 7 - Ejemplo de Uso del Mutex BD

2.3 Operaciones de la Biblioteca

El sistema de la biblioteca permite realizar tres operaciones fundamentales sobre los libros: préstamo, devolución y renovación. En todos los casos, cuando se completa la operación, se actualiza la información del libro correspondiente en la estructura *bd*, y los detalles de la operación se registran en el arreglo *reporte*. Esta información puede ser consultada posteriormente mediante el comando *r*, o escrita en un archivo de texto si se indicó la opción -s al ejecutar el programa. Igualmente, las respuestas que envía ya sea el Proceso Receptor o el hilo auxiliar 1, son mediante pipe de respuestas respectivo a cada PS

Préstamo de un libro:
Cuando se recibe una solicitud de préstamo, el Proceso Receptor busca en la base de datos un libro que coincida con el nombre y el ISBN registrados por el usuario. Si se encuentra y hay al menos un ejemplar disponible, se actualiza su estado a “prestado” y se asigna una fecha de devolución a siete días desde la fecha actual. Luego, el RP envía un mensaje al PS respectivo indicando que el préstamo fue realizado con éxito. En caso contrario, si el libro no existe o no hay ejemplares disponibles, se envía un mensaje informando que no es posible realizar el préstamo.

Devolución de un libro:
Las solicitudes de devolución son gestionadas por el hilo auxiliar 1. Este verifica si el libro indicado existe en la base de datos y si se encuentra actualmente en estado “prestado”. Si ambas condiciones se cumplen, se cambia su estado a “disponible” y se actualiza la fecha con la fecha actual del sistema. Posteriormente, se envía una confirmación al PS correspondiente indicando que la devolución fue exitosa. En caso contrario, si el libro no existe o no estaba prestado, se responde con un mensaje explicando la razón por la que no se pudo realizar la devolución.

Renovación de un libro:
La renovación también es gestionada por el hilo auxiliar 1. Se verifica si el libro existe y si el

ejemplar se encuentra en estado “prestado”. Si es así, se extiende la fecha de devolución actual sumándole siete días y se informa al PS respectivo que la renovación se realizó correctamente. Si no se cumplen las condiciones, se notifica al PS con el motivo del fallo.

3. Caso de Uso

A continuación, se ilustrará con un diagrama de secuencia, el flujo de nuestra implementación para un caso de uso en el que un Proceso Solicitante solicite un préstamo, una renovación, una devolución y acabe su ejecución con la operación *Q*.

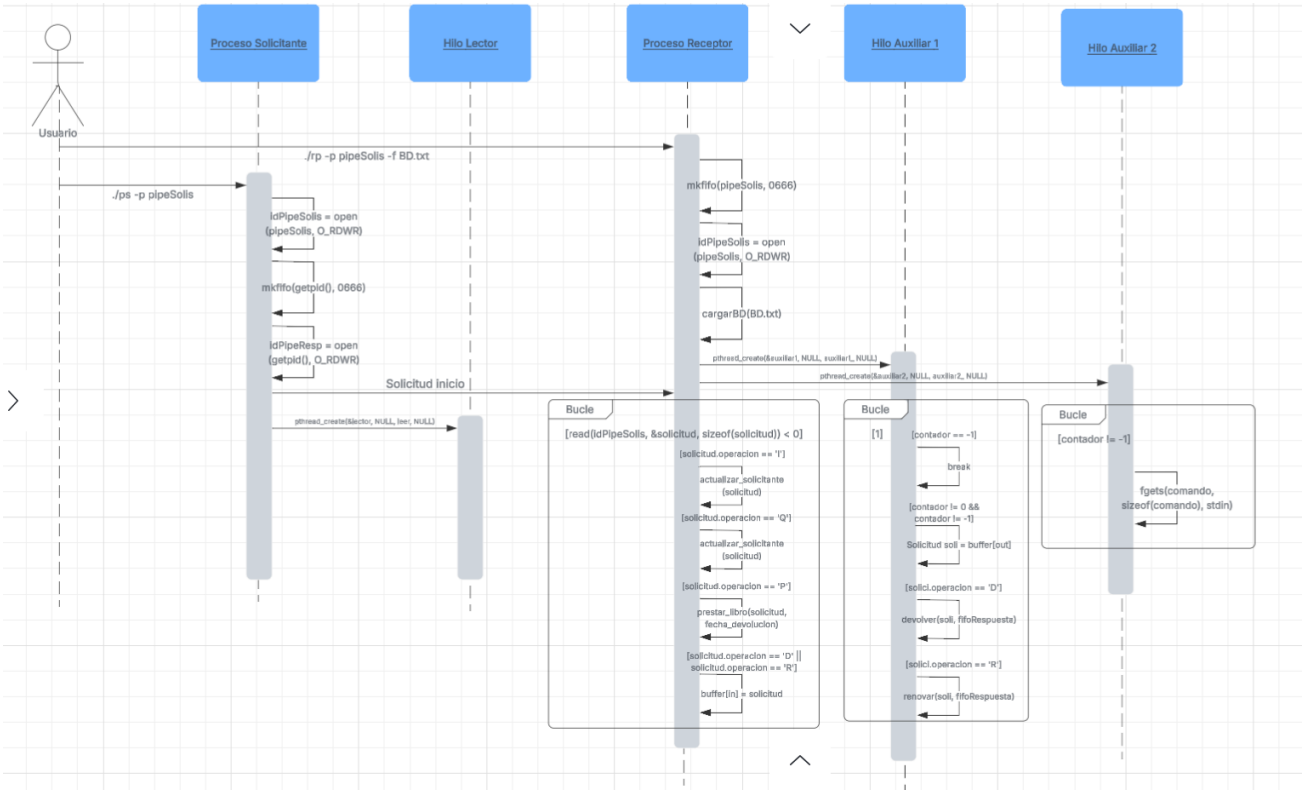

Imagen 8 *- Diagrama de Secuencia*

4. Conclusiones y Recomendaciones

En primera instancia, se concluye que es de gran importancia conocer las herramientas proporcionadas por los sistemas operativos, tales como los procesos, hilos, pipes y mecanismos que permiten la concurrencia, con el objetivo de maximizar el rendimiento y uso de los recursos de la máquina. De igual manera, con base a los lineamientos del problema y a la identificación del problema Productor – Consumidor, la separación entre los procesos solicitantes y el proceso receptor, permitió estructurar una arquitectura distribuida y concurrente, en la cual cada componente cumple un rol especifico y se comunica de manera eficiente con sus otras partes. Esta estructura no solo facilito la implementación modular del sistema, sino que además permitió hacer uso de las ventajas del paralelismo y la comunicación entre procesos.

De esta manera, el uso del named pipes fue fundamental para la comunicación entre los procesos solicitantes y el proceso receptor. A través de estos canales, se creó un sistema asíncrono en donde múltiples usuarios pueden enviar solicitudes simultáneamente, recibiendo respuestas personalizadas por medio de su propio pipe de retorno. Por otro lado, el uso de hilos dentro del proceso receptor, en especial a través del modelo Productor – Consumidor con el buffer circular, fue de suma importancia para dividir la carga de trabajo y optimizar el procesamiento de solicitudes permitiendo una administración efectiva de estos.

Así, mientras que el hilo principal del RP gestionaba prestamos directamente, el hilo auxiliar se encargaba de las devoluciones y renovaciones. Esta distribución del trabajo permitió procesar múltiples tareas y clientes de forma paralela, mejorando el rendimiento del sistema y evitando bloqueos innecesarios. Sin embargo, esta concurrencia requiere la necesidad de implementar mecanismos de sincronización. De esta manera, el uso del mutex fue fundamental para garantizar la integridad de los datos compartidos, tal como la base de datos y el arreglo de reportes. Estos mecanismos permitieron evitar condiciones de carrera y garantizar que un solo hilo accediera a un recurso compartido al mismo tiempo.

Con base en estas conclusiones y reflexiones acerca de las herramientas aplicadas para el desarrollo de este proyecto, se halla de gran importancia la recomendación del uso de estas para sistemas distribuidos y de modelo Productor – Consumidor, ya que permiten la comunicación eficiente entre procesos simultáneos, una mejor distribución de la carga de trabajo y un control seguro sobre los recursos compartidos. La correcta implementación de pipes, hilos y mecanismos de sincronización como mutex y variables de condición no solo mejora el rendimiento del sistema, sino que también asegura su estabilidad y coherencia en entornos concurrentes. Por tanto, se recomienda ampliamente su aplicación en el diseño de soluciones que requieran procesamiento paralelo, interacción entre múltiples entidades y una alta disponibilidad de servicios.