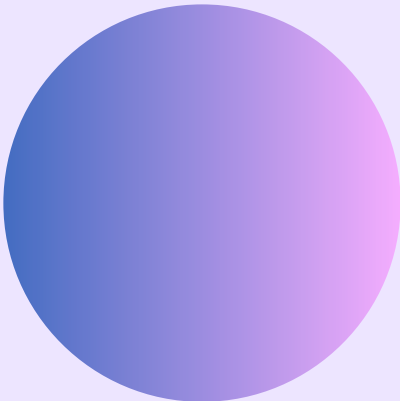


Proyecto Préstamo de libros

Viviana Gómez, Roberth Méndez, Adriana Salazar, Roberth Méndez

Introducción



Sistema de gestión de solicitudes de préstamo de libros mediante procesos solicitantes y procesos receptores, haciendo uso de hilos auxiliares y mecanismos de sincronización como mutexes y variables de condición.

- **Solicitante**

Envían solicitudes al receptor para devolver, renovar o solicitar prestado un libro.

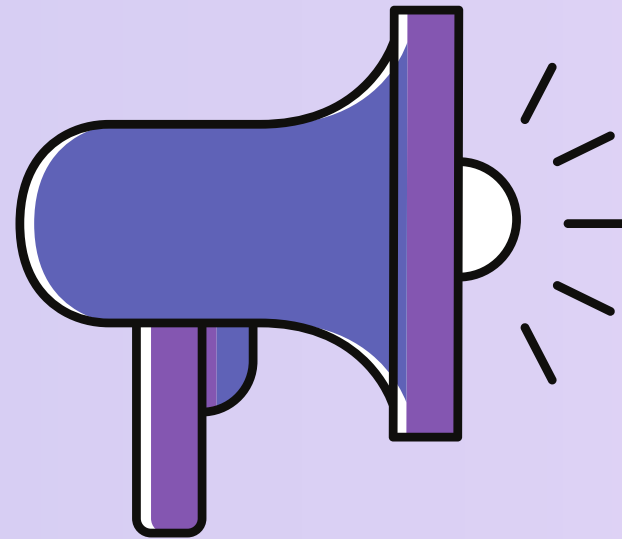
- **Receptor**

Procesa y responde las solicitudes enviadas por los diferentes solicitantes.



Named Pipes

El sistema utiliza named pipes (FIFOs) para la comunicación entre **procesos no relacionados**.



- **Named Pipe de Solicitudes**

Es creado por el proceso receptor y los PS envían una estructura que contiene los datos de la solicitud a realizar.

- **Named Pipe de Respuestas**

Cada proceso solicitante crea su propio pipe usando su PID y a través de este recibe las respuestas del receptor.

Proceso Solicitante

Envío de Mensaje de Inicio y Fin

● Mensaje de inicio ('I')

Al iniciar, el PS envía una solicitud de tipo 'I' al RP con su pipe de respuesta.

● Mensaje de fin ('Q')

Al finalizar, envía una solicitud 'Q' para indicar que no enviará más operaciones.

● MODOS

Modo interactivo:

El usuario elige cuándo salir

Modo archivo:

Al encontrar la operación 'Q' en el archivo, se da por finalizado el PS.

Proceso Solicitante

● Creación del hilo lector

- Se crea un hilo (pthread_t lector) que ejecuta la función leer().
- Este hilo se asocia al pipe de respuestas del PS y se mantiene activo.

● Funcionalidad del hilo

- El hilo espera continuamente mensajes del RP.
- Al recibir una respuesta, la muestra en consola indicando si la operación fue exitosa.

Hilo de Lectura de Respuestas

```
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;  
}
```

Proceso Receptor

Cargar la Base de Datos

Se leen los datos del archivo y la información de cada libro se guarda en un arreglo de estructuras de “Libro”.



```
void cargarBD(char* archivoBD){

    char line[MAX_LINE]; // Variable para leer cada linea de la base de datos
    char nombre[100]; // Variable para guardar el nombre del libro
    int isbn; // Variable para guardar el isbn del libro
    int copias; // Número de copias de cada libro

    FILE *archivo_bd = stdin;
    if (archivoBD != NULL) {
        archivo_bd = fopen(archivoBD, "r");
        if (archivo_bd == NULL) {
            perror("fopen");
            exit(EXIT_FAILURE);
        }
    }

    while (fgets(line, MAX_LINE, archivo_bd) != NULL) { // Ciclo hasta terminar de leer el archivo
        sscanf(line, "%[^,], %d, %d", nombre, &isbn, &copias); //Lee una línea y guarda los datos
        Libro libro;
        strcpy(libro.nombre, nombre);
        libro.isbn = isbn;
        for(int i=0; i < copias; i++){

            fgets(line, MAX_LINE, archivo_bd); // Lee los datos de cada copia
            char fecha[100];
            sscanf(line, "%d, %c, %[^,]", &libro.ejemplar, &libro.estado, fecha); // Guarda los datos de cada copia

            strptime(fecha, "%d-%m-%Y", &libro.fecha);
            bd[totalLibros] = libro; //Agrega el libro a la base de datos
            totalLibros++; // Aumenta el contador de libros
        }
    }

    if (archivoBD != NULL) {
        fclose(archivo_bd);
    }

    return;
}
```

```

while (read(idPipeSolicitudes, &solicitud, sizeof(Solicitud)) > 0) {
    if(verbose && solicitud.operacion != 'S' && solicitud.operacion != 'I')
        printf("\nSolicitud recibida: %c, %s, %d\n", solicitud.operacion, solicitud.libro, solicitud.isbn);

    if(solicitud.operacion == 'I') {
        // Solicitud de inicio para registrar el pipe del solicitante
        actualizar_solicitante(solicitud); // Actualiza la cantidad de solicitantes activos
        continue;
    }
    if(solicitud.operacion == 'Q'){
        if(verbose == true)
            printf("Solicitud de salida recibida\n");
        actualizar_solicitante(solicitud); // Actualiza la cantidad de solicitantes activos
    }

    if(solicitantesActivos == 0){
        contador = -1; // Indica que no hay clientes activos
        pthread_cond_signal(&noVacio); // Desbloquea el hilo auxiliar1
        solicitud.operacion = 'S'; // Indica que no hay clientes activos
        printf("\n!!!!!!Todos los procesos solicitantes han terminado!!!!!!\n");
        printf("Presione ENTER para salir\n");
    }

    // Procesar la solicitud
    if (solicitud.operacion == 'P'){
        pthread_mutex_lock(&mutexBD); // Mutex que asegura que solo un hilo acceda a la base de datos
        prestamo(solicitud);
        pthread_mutex_unlock(&mutexBD); //Desbloquea el mutex de la base de datos
    }
    else if (solicitud.operacion == 'D' || solicitud.operacion == 'R'){
        devolverRenovar(solicitud);
    }
    else if (solicitud.operacion == 'S') {
        actualizar_bd(archivoBD); // Actualiza la base de datos
        if(reporte != NULL)
            escribir_reporte(reporte); // Escribir reporte de la base de datos
        close(idPipeSolicitudes);
        break;
    }
}

```

Proceso Receptor

● Operaciones 'I' y S'

Se implementaron operaciones de inicio para cada solicitud y finalización en caso que no haya ningún PS activo.

● Contador para los PS activos

Se usa un contador para llevar registro de cuantos procesos solicitantes están usando el pipe de solicitudes.


```

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);

    int fifoRespuesta = open(soli.pipe, O_WRONLY); //Abre el descriptor del pipe de respuestas
    char mensaje[MAX_LINE];
    pthread_mutex_lock(&mutexBD); // Mutex que asegura que solo un hilo acceda a la base de datos
    //Buscar el libro de la operación
    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
}

```

Hilo Auxiliar 1

- **Operaciones ‘D’ y ‘R’**

Mediante este hilo se ejecutan las operaciones de devolución y renovación.

- **Contador para la terminación del hilo**

Se usa el contador de la cantidad de PS activos como señal para indicarle al auxiliar 1 cuándo debe acabar.



Esquema Productor Consumidor

- **Productor (RP)**

Ingresa las solicitudes recibidas dentro del buffer circular.
Índice in para indicar la posición de la siguiente solicitud.

- **Consumidor (PS)**

Extrae las solicitudes del buffer del buffer circular.
Índice out que señala cual será la próxima solicitud a procesar.

- **Buffer Circular**

Ambos índices se reinician a cero cuando alcanzan el final del buffer, implementando así el comportamiento circular.

```
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);  
  
}
```

```
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--;
```

```

printf("\n===== MENÚ RECEPTOR =====\n");
printf("Ingrese s para salir\n");
printf("Ingrese r para ver reporte\n");
printf("=====\n");

while (contador != -1) {

    if (fgets(comando, sizeof(comando), stdin) != NULL) {
        if (comando[0] == 's') {
            printf("Finalizando programa...\n");

            // Envía señal para cerrar el hilo auxiliar1
            contador = -1;
            pthread_cond_signal(&noVacio); // Desbloquea el hilo auxiliar1

            // Envía solicitud de salida para desbloquear el read() en main
            Solicitud fin = {0};
            fin.operacion = 'S'; // 'S' para salir
            write(idPipeSolicitudes, &fin, sizeof(Solicitud));
            break;

        } else if (comando[0] == 'r') {
            // Mostrar reporte de solicitudes
            pthread_mutex_lock(&lock);
            printf("\n==== Reporte de Operaciones Realizadas ==== \n");
            for (int i = 0; i < 100; i++) {
                if (reporte[i].isbn != 0) {
                    char fecha[20];
                    strftime(fecha, sizeof(fecha), "%d-%m-%Y", &reporte[i].fecha);
                    printf("Estado: %c, Libro: %s, ISBN: %d, Ejemplar: %d, Fecha: %s\n",
                        reporte[i].estado, reporte[i].libro, reporte[i].isbn,
                        reporte[i].ejemplar, fecha);
                }
            }
            pthread_mutex_unlock(&lock);
        }
    }
}
}

```

Hilo Auxiliar 2

● Comando 's'

En caso de acabar el RP, se envía una señal al auxiliar 1 para que se acabe. Para el hilo principal del RP, se envía una solicitud con la operación S para finalizar.

● Comando 'r'

Se imprime en consola todo lo que almacenado en el arreglo de reporte hasta el momento.

MUTEXES

- **Problema**

El Proceso Receptor (RP) trabaja concurrentemente con dos hilos auxiliares, lo que puede generar condiciones de carrera al acceder recursos compartidos.

- **Solución**

Se usaron mutex y variables de condición para sincronizar el acceso a:

- Buffer circular
- Base de datos bd
- Arreglo reporte

- **En Buffer Circular**

Se usan variables de condición para:

- Bloquear el consumidor si el buffer está vacío.
- Bloquear el productor si el buffer está lleno.
- Liberar mutex temporalmente durante esperas.

```
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);  
  
}
```

```
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);  
  
    }
```



MUTEXES

- **Reporte**

Usa Mutex exclusivo para evitar accesos simultáneos.

Accedido por:

- Proceso principal (préstamos)
- Hilo Auxiliar 1 (devoluciones/renovaciones)
- Hilo Auxiliar 2 (impresión del reporte)

- **Base de Datos**

Se protege con mutex para evitar inconsistencias

Accedida por:

- RP (préstamos)
- Hilo Auxiliar 1 (devoluciones/renovaciones)

```
//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);
```

```
//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
```



Conclusiones y Recomendaciones

- **Importancia de las herramientas del S.O**

Procesos, hilos, pipes y sincronización permiten aprovechar al máximo los recursos del sistema.

- **Arquitectura distribuida y concurrente**

La separación entre PS y RP facilitó un diseño modular, paralelo y eficiente.

- **Uso de named pipes y hilos**

Permitieron comunicación asíncrona y procesamiento simultáneo de solicitudes.



Conclusiones y Recomendaciones

- **Modelo Productor – Consumidor**

Procesos, hilos, pipes y sincronización permiten aprovechar al máximo los recursos del sistema.

- **Sincronización con mutex y condiciones**

La separación entre PS y RP facilitó un diseño modular, paralelo y eficiente.

- **Recomendación final**

Estas herramientas (pipes, mutex, hilos) son clave para sistemas concurrentes, distribuidos y de alta demanda.



The background is a solid light purple. In the center, there are two overlapping circles. The left circle is a gradient from light blue at the top to light pink at the bottom. The right circle is a gradient from light pink at the top to light blue at the bottom. At the bottom of the image, there is a horizontal bar with a gradient from light blue on the left to light purple on the right.

Thank
You.