Práctica 3 - Sincronización de procesos con hilos

Informe Entregable: Resolver el ejercicio con hilos , mutexes y variables de condición

Lucía Pérez González, Adrián Quiroga Linares

Sistemas Operativos II, Grupo 04
lucia.perez.gonzalez1@rai.usc.es, adrian.quiroga@rai.usc.es

I. Introducción

Este programa implementa el **problema productor-consumidor** usando:

- **Múltiples hilos productores y consumidores**
- **Un buffer compartido (cola de tamaño fijo)**
- **Mecanismos de sincronización con mutexes y condiciones**
- **Barrera para sincronizar el inicio de los hilos**

II. Ejecución

El código se compila con `gcc` y se ejecuta desde la terminal con los siguientes pasos:
1. Compilación:

```
gcc -o prod_cons prod_cons.c -lpthread
```

2. Ejecución:

```
./prod_cons <numero_de_productores> <numero_de_consumidores>
```

Ejemplo:

```
./prod_cons 2 2
```

Durante la ejecución, el programa solicita por consola los parámetros necesarios para cada hilo. A cada productor se le pide el nombre del fichero de entrada desde el que se leerán los caracteres, así como un valor de tiempo de espera T. Por otro lado, a cada consumidor solo se le solicita este valor T. Los consumidores guardan lo que consumen en un fichero generado automáticamente, cuyo nombre sigue el formato `fileCons_<id>.txt`.

III. Funcionamiento del código

El programa implementa un sistema de múltiples *productores* y *consumidores* que interactúan a través de un **buffer circular compartido**. Cada hilo productor lee datos desde un fichero específico, mientras que los consumidores retiran esos datos y los almacenan en sus propios ficheros. La coordinación entre los hilos se maneja mediante mecanismos de sincronización proporcionados por `POSIX Threads (pthreads)`, asegurando la integridad de los datos y evitando condiciones de carrera.

A. Productores

Cada hilo productor recibe el nombre de un fichero de entrada y un tiempo de espera (`sleep`) que simula el tiempo de producción. A partir de ahí, lee carácter por carácter del fichero, y filtra para aceptar únicamente caracteres alfanuméricos o el asterisco `*`. Este último tiene un significado especial: indica el final de la producción para ese hilo.

Cuando el carácter leído es válido, el productor intenta insertarlo en el buffer circular. Si el buffer está lleno, el hilo entra en espera hasta que haya espacio disponible. Para evitar conflictos al acceder al buffer compartido, se utiliza un `mutex` que asegura el acceso exclusivo, y una `variable de condición` (`condp`) que permite esperar a que el consumidor libere espacio. Una vez insertado un carácter, se notifica a los consumidores mediante otra condición (`condc`). El ciclo de producción continúa hasta que se encuentra un `*`, el cual también se inserta en el buffer para notificar a los consumidores el fin de producción.

B. Consumidores

Los hilos consumidores también reciben un tiempo de espera y escriben en su propio fichero de salida (por ejemplo, `fileCons_1.txt`). Después de sincronizar su inicio con una `barrera` común, los consumidores retiran caracteres del buffer circular. Si el buffer está vacío, el hilo entra en espera utilizando una condición (`condc`) y espera a que un productor inserte un nuevo carácter.

Cuando el consumidor extrae un carácter, lo escribe en su fichero correspondiente, salvo que se trate de un `*`. En ese caso, interpreta que un hilo productor ha terminado, y se incrementa una variable global (`contProd`) que lleva el conteo de cuántos productores han finalizado. Los consumidores terminan su ejecución una vez que todos los productores han finalizado, es decir, cuando `contProd == numProd`.

C. Sincronización

El programa hace un uso intensivo de mecanismos de sincronización de `pthreads`. Para evitar condiciones de carrera en el acceso al buffer compartido, se emplea un `mutex` (`pthread_mutex_t`). Este mutex asegura que solo un hilo pueda acceder y modificar el buffer a la vez. Además, se utilizan dos variables de condición, `condp` y `condc`, para gestionar la espera de los productores (cuando el buffer está lleno) y los consumidores (cuando el buffer está vacío), respectivamente.

Por otro lado, para asegurar que todos los hilos empiecen al mismo tiempo, tanto productores como consumidores esperan sobre una `barrera de sincronización` (`pthread_barrier_t`). Esto garantiza un inicio simultáneo de la operación, haciendo que el comportamiento concurrente sea más ordenado y predecible.

IV. Conclusiones obtenidas

La implementación desarrollada demuestra un uso correcto y eficiente del patrón *productor-consumidor* mediante el uso de múltiples hilos. Cada componente del sistema ha sido diseñado para trabajar de manera concurrente, interactuando a través de un buffer compartido con un control riguroso de acceso.

Uno de los aspectos más destacables es la adecuada gestión de la concurrencia. Gracias al uso de `mutexes` y `variables de condición`, se evita el acceso simultáneo al buffer circular, previniendo condiciones de carrera y garantizando la integridad de los datos. Esta estrategia asegura que los hilos productores y consumidores puedan operar de forma sincronizada sin interferencias.

En cuanto a la lógica de finalización, el sistema está diseñado de manera que los hilos consumidores no concluyen su ejecución hasta haber detectado la finalización de todos los productores. Esta condición se verifica a través de la recepción de un carácter especial (`'*'`) por cada productor, lo que aporta claridad y robustez al control del flujo del programa.

El diseño del programa también destaca por su modularidad y escalabilidad. La estructura permite fácilmente modificar el número de hilos productores y consumidores, facilitando la

experimentación con distintos escenarios y adaptaciones futuras. Además, el uso de `barreras de sincronización` asegura que todos los hilos inicien su ejecución al mismo tiempo, mejorando la coherencia y predictibilidad del comportamiento concurrente.

En conjunto, el sistema demuestra ser una solución bien estructurada, que combina eficiencia, seguridad y claridad en la ejecución de tareas concurrentes en un entorno multihilo.

```
❯ ./prod_cons 2 1
Productor 1) Fichero a emplear: file1.txt
             Tiempo a dormir el hilo: 1
Productor 2) Fichero a emplear: file2.txt
             Tiempo a dormir el hilo: 2
Consumidor 1) Tiempo a dormir el hilo: 1

                PRODUCTORES             |          CONSUMIDORES
----------------------------------------|----------------------------------------
 Prod 1: Introdujo a en la posición 0 |
                                        | Cons 1: Retiró a de la posición 0
 Prod 1: Introdujo b en la posición 1 |
 Prod 1: Introdujo c en la posición 2 |
                                        | Cons 1: Retiró b de la posición 1
                                        | Cons 1: Retiró c de la posición 2
 Prod 1: Introdujo d en la posición 3 |
                                        | Cons 1: Retiró d de la posición 3
 Prod 1: Introdujo e en la posición 4 |
                                        | Cons 1: Retiró e de la posición 4
 Prod 1: Introdujo 1 en la posición 5 |
                                        | Cons 1: Retiró 1 de la posición 5
 Prod 1: Introdujo 2 en la posición 6 |
                                        | Cons 1: Retiró 2 de la posición 6
 Prod 1: Introdujo F en la posición 7 |
                                        | Cons 1: Retiró F de la posición 7
 Prod 1: Introdujo G en la posición 0 |
                                        | Cons 1: Retiró G de la posición 0
 Prod 1: Introdujo * en la posición 1 |
                                        | Cons 1: Retiró * de la posición 1
 Prod 2: Introdujo 0 en la posición 2 |
 Prod 2: Introdujo 1 en la posición 3 |
                                        | Cons 1: Retiró 0 de la posición 2
 Prod 2: Introdujo m en la posición 4 |
                                        | Cons 1: Retiró 1 de la posición 3
                                        | Cons 1: Retiró m de la posición 4
 Prod 2: Introdujo n en la posición 5 |
                                        | Cons 1: Retiró n de la posición 5
 Prod 2: Introdujo 8 en la posición 6 |
                                        | Cons 1: Retiró 8 de la posición 6
 Prod 2: Introdujo 7 en la posición 7 |
                                        | Cons 1: Retiró 7 de la posición 7
```

Figura 1: *Ejecución del programa ./prod_cons 2 1*