Práctica 3 - Sincronización de procesos con mutexes y variables de condición

Informe optativo 2: Resolver el ejercicio solamente con mutexes, sin 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 ejercicio implementa el **problema del productor-consumidor** usando:

- Múltiples hilos productor y consumidor
- Mecanismos de sincronización con mutexes
- Un buffer compartido entre productores y consumidores
- Una estructura para cada hilo que contiene variables necesarias para su ejecución: nombre del fichero a utilizar, tiempo para el sleep(), número de hilo

II. Ejecución

El programa se compila con gcc:
gcc -o opt2_prod_cons opt2_prod_cons.c -lpthread

y se ejecuta desde la terminal, pasándole la cantidad de procesos productor y consumidor que se desean crear:
./opt2_prod_cons <número_de_productores><número_de_consumidores>

- Si no se incluyen los parámetros requeridos por línea de comandos o si se insertan valores que no son enteros o mayores que 0, el programa informará al usuario del error y finalizará.

Al inicio de la ejecución se solicitará al usuario que inserte: para cada proceso productor hijo, el nombre del fichero de lectura del que obtener los items y el tiempo a emplear en su sleep(); y para cada proceso consumidor hijo, el tiempo a emplear en su sleep().

III. Funcionalidad del código

El ejercicio es prácticamente igual al ejercicio 1 obligatorio, ya que, tal y como se comentó en la Introducción, implementa el **problema del productor-consumidor** empleando hilos y mecanismos de sincronización como mutexes. Cabe ser destacado, que en este caso no se emplearán variables de condición. Esto provocará que el main sea prácticamente el mismo:

- Se inicia comprobando los parámetros introducidos por línea de comandos, pasando a continuación a inicializar los mutexes, cola compartida y barrera. Se declaran y crean los hilos productores y consumidores, solicitando la introducción por teclado de: fichero de lectura y tiempo para el sleep() para los hilos productores, y tiempo para el sleep() para hilos consumidores. Mientras no se introduzcan los datos para todos los hilos, éstos

esperarán en la barrera hasta que se introduzca el tiempo para el último consumidor, y una vez sincronizados, iniciarán su producción/consumo. Mientras los hilos están ejecución, el programa padre estará esperando a que finalicen, para poder proceder a finalizar la ejecución destruyendo mutexes y barrera.

A. Hilo productor: void *productor(void *arg)

Una vez iniciado el hilo, se abre en modo lectura el fichero del que cada uno extraerá los items y se espera en la barrera. Una vez llegados todos los hilos a ella, es decir, están sincronizados, empezarán a producir.

Los productores solamente leerán caracteres alfanuméricos del fichero. Tras generar un item del caracter leído, intenta obtener el mutex para acceder a la región crítica, si lo obtiene comprueba la cantidad de elementos de la cola, y si está llena, libera el mutex, cede su CPU para que trabaje otro hilo y vuelve a intentar obtener el mutex. Mientras espera a que se libere espacio por parte de un consumidor, el productor hará los 3 pasos anteriores continuamente. Si hay espacio libre, inserta el item al final de la cola y libera el mutex. Si el hijo llega al final del fichero de lectura, producirá un asterisco como item. Este asterisco indicará el fin de la producción del hijo correspondiente, por lo que será añadido a la cola compartida y finalizará, cerrando el fichero de lectura y liberando la memoria de su propia estructura de datos antes de terminar.

B. Hilo consumidor: void *consumidor(void *arg)

Una vez iniciado el hilo, se abre en modo escritura un fichero propio sobre el que escribirán los caracteres recibidos y se espera en la barrera. Una vez llegados todos los hilos a ella, es decir, están sincronizados, empezarán a consumir.

Se inicia el bucle de consumo intentando obtener el mutex para acceder a la región crítica, si lo obtiene comprueba la cantidad de elementos de la cola, y si está vacía, realiza varios pasos: primero comprueba si el número de productores finalizados contProd iguala al número de hilos productores creados numProd, en caso afirmativo finalizará su ejecución liberando el mutex, cerrando el fichero de escritura y liberando la memoria de su propia estructura de datos; y a continuación libera el mutex, cede su CPU para que trabaje otro hilo e intentará obtener de nuevo el mutex. Mientras espera a que un productor inserte items, el hilo consumidor repetirá la comprobación y 3 pasos anteriores continuamente. Si hay items que consumir, extrae el primer item de la cola y comprueba si este es un asterisco: si no lo es, escribe el item en su fichero de escritura; si lo es, incrementa el contador de productores finalizados contProd. Finaliza el ciclo liberando el mutex. El bucle finalizará cuando el buffer compartido se encuentre vacío y el número de productores finalizados contProd iguale al número de hilos productores creados numProd.

IV. Conclusiones obtenidas

La implementación desarrollada resuelve correctamente el **problema del productor-consumidor** empleando hilos y solamente mutexes como mecanismo de sincronización. A pesar de no utilizar variables de condición, se logra mantener la integridad del buffer compartido y una coordinación segura y eficaz entre hilos productores e hilos consumidores. El aspecto clave de esta versión es el uso de espera activa mediante un *sched_yield()* para ceder el control de la CPU cuando un hilo no puede acceder al buffer (por estar lleno (hilos productores) o vacío (hilos consumidores)).

Por otra parte, como se puede comprobar en la Figura 1, para una ejecución del programa para 2 hilos productores y 3 hilos consumidores, las barreras de sincronización entre procesos funcionan correctamente (no se inicia ni la producción ni el consumo hasta que el usuario termine de insertar los datos) y que la sincronización mediante mutexes y espera activa gestionan correctamente los accesos concurrentes a la memoria compartida de los distintos hilos.

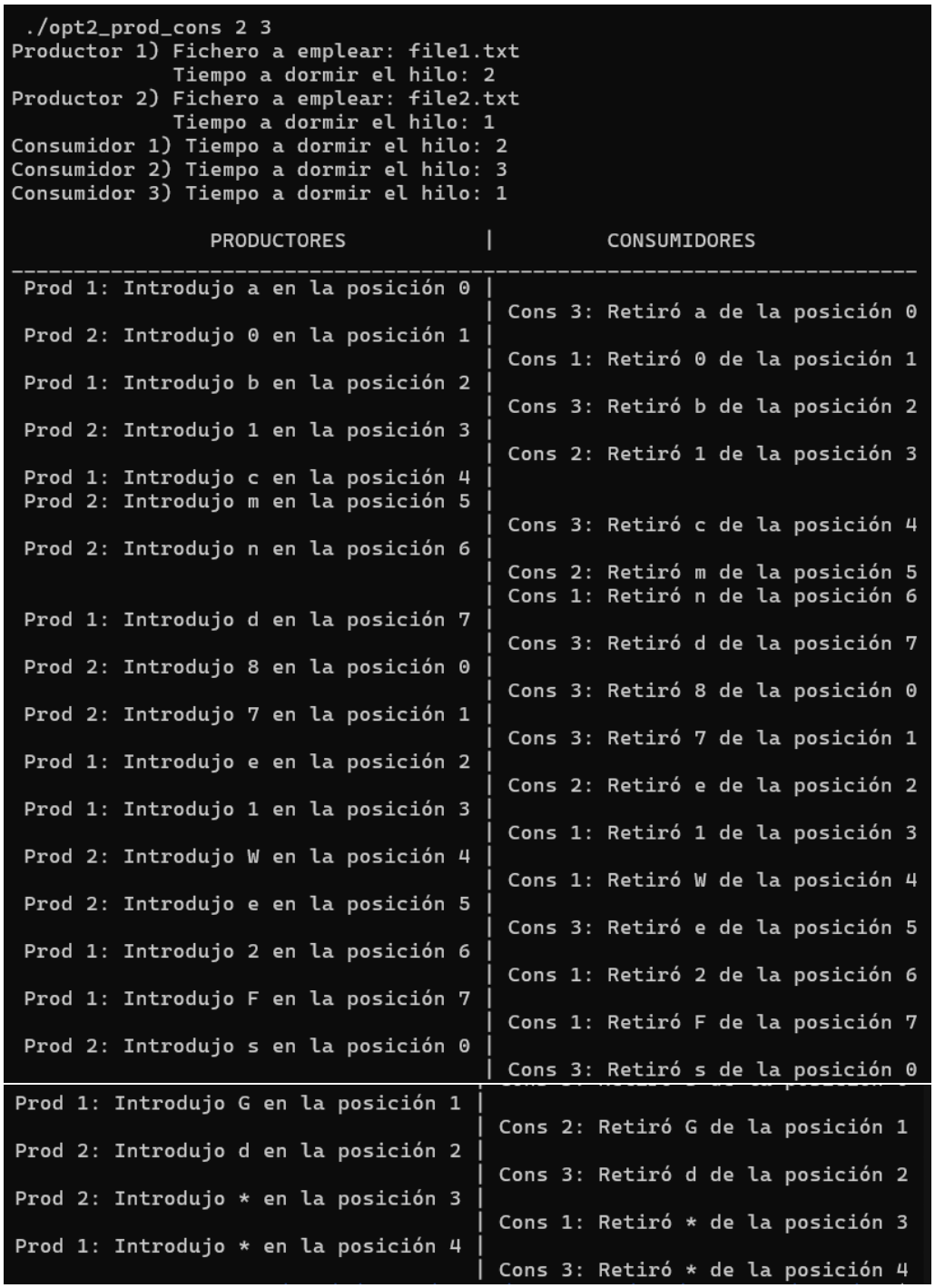

Figura 1: *Ejecución de opt2_prod_cons.c para 2 hilos productores y 3 hilos consumidores*