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

Informe optativo 1: Resolver el ejercicio con procesos en vez de hilos

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 procesos productor y consumidor, los cuales son programas separados: opt1_prod.c y opt1_cons.c
- Mecanismos de sincronización con mutexes y variables de condición
- Una estructura compartida entre productor y consumidor que contiene estos mecanismos y la cola compartida
- Una estructura para cada programa que será compartida entre los múltiples procesos que contiene una barrera para sincronizar el inicio de la ejecución y variables necesarias

II. Ejecución

Los programas se compilan con gcc:
gcc -o opt1_prod opt1_prod.c -lpthread
gcc -o opt1_cons opt1_cons.c -lpthread

y se ejecutan desde la terminal, pasándole al programa correspondiente la cantidad de procesos productor o consumidor que se desean crear:
./opt1_prod <número_de_productores>
./opt1_cons <número_de_consumidores>

- Si no se incluye el parámetro requerido por línea de comandos o si se inserta un valor que no sea entero o mayor que 0, el programa informará al usuario del error y finalizará.

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

IMPORTANTE: Se debe de ejecutar primero el programa productor opt1_prod.c, ya que este crea tanto la memoria compartida como los mutexes y variables de condición a emplear.

III. Funcionalidad del código

A. opt1_prod.c

Programa correspondiente al generador de productores. Tal y como arriba se indicó, después de comprobar la entrada por línea de comandos, se crea y mapea dos tipos de memoria compartida: una compartida por productores y consumidores, que contiene la cola de items, el número de

procesos productor a crear (número de asteriscos que el consumidor debe de recibir), los mutexes y las variables de condición; y otra compartida entre los distintos procesos productor hijo, la cual contendrá la barrera de sincronización. A continuación se inicializan los mutexes, variables de condición y la barrera, para que éstos puedan funcionar entre procesos. Una vez realizados estos pasos de perparación, se inicia el proceso de producción: se inicia un bucle for que irá pidiendo al usuario el fichero de lectura y tiempo para el sleep() para cada proceso hijo a crear, el cual se crea a continuación con un *fork()* y se espera en la barrera a que se introduzcan estos datos para cada proceso hijo. Una vez todos los procesos productores hijo han llegado a la barrera, es decir, están sincronizados, abrirán en modo lectura el fichero del que cada uno extraerá los items y 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, espera a que se libere espacio por parte de un consumidor. Si hay espacio libre, inserta el item al final de la cola, despierta a un proceso consumidor y libera el mutex. Si el hijo llega al final del fichero de lectura, producirá un asterico como item. Este asterisco indicará el fin de la producción del hijo correspondiente, por lo que será añadido a la cola compartida y finalizará.

Fuera de la ejecución de los procesos hijos, el proceso padre espera a que estos finalicen. Una vez todos hayan finalizado, se libera el mapeo de las dos memorias compartidas, se cierra el fichero de memoria compartida, y se destruye la barrera. A pesar de que ha sido el productor quien inicializó los mutexes y variables de condición, el consumidor que continúa en ejecución aún requiere de su uso, por lo que será este último el que se encargará de destruírlos al finalizar.

B. opt1_cons.c

Programa correspondiente al generador de consumidores. De la misma forma que en el programa productor, después de comprobar la entrada por línea de comandos, se abre y mapea los dos tipos de memoria compartida: la compartida por productores y consumidores, que contiene la cola de items, el número de procesos productor creados (número de asteriscos que el consumidor debe de recibir), los mutexes y las variables de condición; y la compartida entre los distintos procesos consumidor hijo, la cual contiene la barrera de sincronización y el contador de productores finalizados (contador de asteriscos recibidos). A continuación, se inicializa la barrera de sincronización de consumidores. En este caso, no es necesario inicializar los mutexes y variables de condición, porque de esto ya se encargó el productor. Una vez realizados estos pasos de perparación, se inicia el proceso de consumo: se inicia un bucle for que irá pidiendo al usuario el tiempo para el sleep() para cada proceso hijo a crear, el cual se crea a continuación con un *fork()* y se espera en la barrera a que se introduzcan estos datos para cada proceso hijo. Una vez todos los procesos consumidores hijo han llegado a la barrera, es decir, están sincronizados, abrirán en modo escritura un fichero propio sobre el que escribirán los caracteres recibidos y 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, espera a un productor inserte items. 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. A continuación, despierta a un proceso productor y libera el mutex. La condición del bucle de consumo es que el contador de productores finalizador contProd (número de asteriscos recibidos) no sea igual al número de productores creados (número de astericos a recibir). En caso de que coincidan estas variables, significará que todos los productores han finalizado, por lo que no hay más items que consumir. Se finaliza el bucle y se finaliza el proceso hijo.

Fuera de la ejecución de los procesos hijos, el proceso padre espera a que estos finalicen. Una vez todos hayan finalizado, se destruyen los mutexes, variables de condición y barrera; se libera el mapeo de las dos memorias compartidas y se cierra el fichero de memoria compartida.

IV. Conclusiones obtenidas

Lo fundamental para la implementación requerida en este ejercicio para el **problema del productor-consumidor** empleando procesos en vez de hilos y mecanismos de sincronización con mutexes y variables de condición, ha sido el aprender a crear, inicializar y utilizar correctamente los atributos de los mutexes, variables de condición y barrera para que estos pudiesen funcionar entre procesos distintos y programas distintos (PTHREAD_PROCESS_SHARED). Ésto junto con extrar los mutexes y variables de condición como memoria compartida entre programas han sido los aspectos clave para lograr una coordinación segura y eficaz entre productores y consumidores.

Por otra parte, como se puede comprobar en las Figuras 1a y 1b que las barreras de sincronización entre procesos funcionan correctamente (no se inicia ni la producción ni la consumición hasta que el usuario termine de insertar los datos) y que los mutexes y variables de condición gestionan correctamente los accesos concurrentes a la memoria compartida de los distintos procesos.

```
/SOII/p03$ ./opt1_prod 2
Productor 1) Fichero a emplear: file1.txt
             Tiempo a dormir el productor: 2
Productor 2) Fichero a emplear: file2.txt
             Tiempo a dormir el productor: 2
 Prod 1: Introdujo a en la posición 0
 Prod 1: Introdujo b en la posición 1
 Prod 1: Introdujo c en la posición 2
 Prod 1: Introdujo d en la posición 3
 Prod 2: Introdujo 0 en la posición 4
 Prod 1: Introdujo e en la posición 5
 Prod 1: Introdujo 1 en la posición 6
 Prod 2: Introdujo 1 en la posición 7
 Prod 2: Introdujo m en la posición 0
 Prod 2: Introdujo n en la posición 1
 Prod 1: Introdujo 2 en la posición 2
 Prod 1: Introdujo F en la posición 3
 Prod 1: Introdujo G en la posición 4
 Prod 2: Introdujo 8 en la posición 5
 Prod 2: Introdujo 7 en la posición 6
 Prod 2: Introdujo W en la posición 7
 Prod 2: Introdujo e en la posición 0
 Prod 1: Introdujo * en la posición 1
 Prod 2: Introdujo s en la posición 2
 Prod 2: Introdujo d en la posición 3
 Prod 2: Introdujo * en la posición 4
```

(a) *Ejecución de opt1_prod.c para 2 procesos productor*

```
/SOII/p03$ ./opt1_cons 3
Consumidor 1) Tiempo a dormir el consumidor: 1
Consumidor 2) Tiempo a dormir el consumidor: 2
Consumidor 3) Tiempo a dormir el consumidor: 1
Cons 1: Retiró a de la posición 0
Cons 1: Retiró b de la posición 1
Cons 1: Retiró c de la posición 2
Cons 3: Retiró d de la posición 3
Cons 1: Retiró 0 de la posición 4
Cons 3: Retiró e de la posición 5
Cons 1: Retiró 1 de la posición 6
Cons 3: Retiró 1 de la posición 7
Cons 1: Retiró m de la posición 0
Cons 3: Retiró n de la posición 1
Cons 1: Retiró 2 de la posición 2
Cons 3: Retiró F de la posición 3
Cons 1: Retiró G de la posición 4
Cons 3: Retiró 8 de la posición 5
Cons 3: Retiró 7 de la posición 6
Cons 1: Retiró W de la posición 7
Cons 2: Retiró e de la posición 0
Cons 3: Retiró * de la posición 1
Cons 2: Retiró s de la posición 2
Cons 1: Retiró d de la posición 3
Cons 3: Retiró * de la posición 4
```

(b) *Ejecución de opt1_cons.c para 3 procesos consumidor*

Figura 1: *Ejecución paralela de los programas productor y consumidor*