Sistemas Operativos, Práctica 3

Escuela Politécnica Superior, UAM, 2022-2023

Elena Balseiro García & Paula Beltrán Marianini
Grupo 2291, Pareja 3

Ejercicio 7: Sistema de monitoreo multiproceso.

a) Programa de monitoreo:

Requisitos	Satisfecho
El programa se debe ejecutar con un parámetro	X
Se deberán lanzar dos procesos: Comprobador y Monitor	X

El funcionamiento de nuestro programa principal *monitor.c* es el siguiente:

Se leen y comprueban los parámetros de entrada, a continuación, se inicializan los semáforos.

Inicialmente, el programa comprueba si existe un segmento de memoria compartida.

Si no existe, se ejecuta la parte del código correspondiente al *comprobador*, quien crea el segmento de memoria compartida y ejecuta su funcionalidad.

Si el segmento de memoria ya existe, se ejecuta el código correspondiente al *monitor*, encargado de mostrar por pantalla los bloques aceptados.

En caso de error, se liberan los recursos y se finaliza el programa.

monitor():

El monitor se encarga de acceder a la memoria compartida y leer los bloques que le envía el comprobador. En concreto, la variable *accepted* de estos.
Si el bloque es aceptado o no, se imprime el mensaje correspondiente por pantalla.

El acceso a esta región se gestiona con los tres semáforos sin nombre alojados en memoria compartida atendiendo al esquema productor-consumidor.

b) Comunicación desde Comprobador a Monitor:

Requisitos	Satisfecho
C: crea e inicializa el segmento de memoria compartida	X
C: Recibe bloques a través de la cola de mensajes y le añade una bandera indicando si es correcto o no	X
C: Tras comprobar el bloque, lo introducirá en memoria compartida para que lo lea el monitor	X
C: Realizará una espera de <LAG> milisegundos	X
C: Repetirá el proceso recepción/comprobación/escritura hasta que se reciba un bloque especial de finalización	X
C: Al recibir el bloque de finalización, lo introducirá en memoria compartida para notificar al monitor, liberará recursos y terminará	X

Requisitos	Satisfecho
M: Abrirá el segmento de memoria compartida	X
M: Extraerá un bloque	X
M: Mostrará por pantalla la sintaxis correspondiente dependiendo de si es correcto o no	X
M: Realizará una espera de <LAG> milisegundos	X
M: Repetirá el proceso extracción/muestra hasta que se reciba un bloque especial de finalización	X
M: Al recibir el bloque de finalización, liberará recursos y terminará	X

Requisitos	Satisfecho
La introducción de bloques en memoria compartida seguirá el esquema de productor-consumidor	X
Se garantizará que no se pierde ningún bloque con independencia del retraso que se introduzca en cada proceso	X
Se usará un buffer circular de 6 bloques en el segmento de memoria compartida.	X
Se alojarán en el segmento de memoria compartida tres semáforos sin nombre para implementar el algoritmo de productor-consumidor.	X

comprobador():

Al comienzo del programa, se mapea la región de memoria compartida, se inicializan los semáforos y se intenta abrir la cola. Esto falla cuando se ejecuta el monitor antes que el minero, que es quien crea la cola.

El comprobador hace wait del semáforo *semReady* para asegurar que no se pierdan bloques en la memoria compartida. Este semáforo se desbloquea desde el monitor cuando está listo para leerla.

El comprobador lee los bloques introducidos en la cola y realiza la función hash con la solución propuesta. En caso de que la solución coincida con el objetivo, el comprobador pone a 1 la bandera *accepted* del bloque y a continuación lo introduce en la memoria compartida para que monitor lo muestre. Esta introducción se gestiona con los semáforos sin nombre mencionados anteriormente.

La lectura de los bloques se hace en un buffer circular donde se va actualizando el id del último elemento añadido. Dado que el tamaño del buffer es 6, al meter el séptimo elemento, se escribiría en la primera posición.

c) Programa de minería:

Requisitos	Satisfecho
El programa se debe ejecutar con dos parámetros	X
Creará una cola de mensajes con capacidad para 7 mensajes	X
Establecerá un objetivo inicial fijo para la POW	X
Para cada ronda, resolverá la POW	X
Enviará un mensaje por la cola que contenga, al menos, el objetivo y la solución hallada	X
Realizará una espera de <LAG> milisegundos	X
Una vez terminadas las rondas, enviará un bloque especial que permita saber a *Comprobador* que el sistema está finalizando	X
Liberará los recursos y terminará	X

El funcionamiento de nuestro programa *miner.c* es el siguiente:

Se leen y comprueban los argumentos.

A continuación se ejecuta el bucle de rondas donde en cada ronda se itera buscando la solución al objetivo *target* haciendo pow_hash.
Cuando se encuentra una solución, se escribe en el bloque junto con el objetivo y se envía a la cola para que *Comprobador* la compruebe.

El minero se queda esperando hasta que introduzca todos los bloques y luego finaliza.

Al acabar las rondas, se envía el bloque especial (cuyos valores son todos -1) para indicar la finalización del sistema.

Se liberan los recursos y se finaliza el programa.

d) En base al funcionamiento del sistema anterior:

-¿Es necesario un sistema tipo productor-consumidor para garantizar que el sistema funciona correctamente si el retraso de *Comprobador* es mucho mayor que el de Minero? ¿Por qué? ¿Y si el retraso de *Comprobador* es muy inferior al de Minero? ¿Por qué?

En general, si el programa más rápido espera a que el programa más lento termine su operación antes de continuar, entonces no habrá problemas de sincronización en la memoria compartida. Sin embargo, si el programa más rápido no espera al programa más lento y escribe en la memoria compartida antes de que el programa lento haya terminado de leer los datos anteriores, entonces podría haber problemas de sincronización.

-¿Se simplificaría la estructura global del sistema si entre *Comprobador* y *Monitor* hubiera también una cola de mensajes? ¿Por qué?

Sí, la estructura global del sistema podría simplificarse ya que la cola proporciona una interfaz de comunicación de más alto nivel que la memoria compartida, lo que hace que la sincronización y comunicación entre procesos sea más sencilla y segura. Se asegura de que los mensajes se entreguen en el orden correcto mediante una comunicación asíncrona que permite que un programa pueda enviar mensajes y continuar su ejecución sin tener que esperar a que el otro procese dichos mensajes.

ANEXO.

Análisis de la ejecución, pruebas y conclusiones:

Para probar el correcto funcionamiento de nuestro programa, hemos realizado varias pruebas.

A la hora de ejecutar el programa, es imprescindible que se lance primero el minero, ya que es el que crea la cola. Si esto no fuese así, el comprobador daría un error y finalizaría el programa ya que este solo intenta abrirla y no la crea en caso de no existir.

Con el minero corriendo, ejecutamos ./monitor en dos terminales diferentes. La primera de ellas realizará la función correspondiente al comprobador y la segunda al monitor.

Durante la implementación de la comunicación entre el comprobador y el monitor, nos dimos cuenta de que si había un gran retraso entre el lanzamiento de dichos procesos, se perdían bloques, ya que cuando el monitor comenzaba a leer el contenido del buffer, obtenía un bloque erróneo.

Esto lo solucionamos mediante la implementación de un semáforo con nombre *semReady*, que hace que el comprobador no envíe bloques hasta que el monitor se haya lanzado. Así, conseguimos que la recepción de bloques sea la esperada independientemente del retraso en que se introduzca cada proceso.

Según nuestra implementación del buffer circular, cuando extraemos elementos realizamos un pop-front, por tanto, cada vez que sacamos un bloque de la cabeza, movemos los siguientes a una posición anterior y decrementamos el índice, que indica donde está el último elemento.
Por esto tuvimos que proteger el caso en que solo haya un elemento, para que cuando ocurra, no quede apuntando a una posición negativa.

En la pruebas, hemos comprobado que cuando se lanza minero con más bloques de los que caben en la cola y todavía no se ha ejecutado ./monitor, minero no termina su finalización hasta que se lance el proceso comprobador que vaya extrayendo elementos de la cola y este pueda introducir los bloques pendientes. No se pierde ningún bloque en la conexión minero-comprobador.

Nuestro sistema cumple con los requisitos satisfactoriamente.