PRÁCTICA 3: MEMORIA COMPARTIDA Y COLAS DE MENSAJES

Alexis Canales Molina y Luis Miguel Nucifora Izquierdo

17/04/2023

2.º INGENIERÍA INFORMÁTICA, SISTEMAS OPERATIVOS

INTRODUCCIÓN

Ésta se trata de una documentación para el ejercicio 7 a entregar, presente en el enunciado de la práctica.

Cuenta con los requisitos del mismo, las respuestas a los apartados teóricos y aquellas funcionalidades que no se han logrado implementar, así como con las pruebas realizadas durante el proceso de codificación.

ENUNCIADO Y REQUISITOS

Ejercicio 7: Sistema de monitoreo multiproceso. Escribir dos programas en C para implementar un sistema de monitoreo de bloques del sistema Blockchain. El ejercicio se divide en varias partes diferenciadas:

a) Programa de monitoreo:

- El programa se debe ejecutar con un parámetro:

```
./monitor <LAG>
```

  donde `monitor` es el nombre del ejecutable y `<LAG>` es el retraso en milisegundos entre cada monitorización.
- Se deberán lanzar dos procesos correspondientes a este programa:
  - El primer proceso detectará que la memoria compartida no está creada y se convertirá en el proceso Comprobador, encargado de verificar los bloques

recibidos.
- El segundo proceso detectará que la memoria compartida está creada y se convertirá en el proceso Monitor, encargado de mostrar los resultados por pantalla.

b) Comunicación desde Comprobador a Monitor usando productor–consumidor:
- El proceso Comprobador:
  - Creará e inicializará un segmento de memoria compartida.
  - A continuación, recibirá un bloque (a través de la cola de mensajes, ver Apartado c)) y lo comprobará, añadiéndole una bandera que indique si es correcto o no.
  - Una vez comprobado, lo introducirá en memoria compartida para que lo lea Monitor.
  - Realizará una espera de `<LAG>` milisegundos.
  - Repetirá el proceso de recepción/comprobación/escritura hasta que reciba un bloque especial que indique la finalización del sistema.
  - Cuando reciba el bloque de finalización, lo introducirá en memoria compartida para notificar al Monitor de la finalización del sistema, liberará los recursos y terminará.
- El proceso Monitor:
  - Abrirá el segmento de memoria compartida.
  - Extraerá un bloque.
  - Lo mostrará por pantalla, con la sintaxis `"Solution accepted: %08ld --> %08ld"` (el primer número es el objetivo y el segundo la solución) para los bloques correctos, y `"Solution rejected: %08ld !-> %08ld"` para los incorrectos.
  - Realizará una espera de `<LAG>` milisegundos.
  - Repetirá el ciclo de extracción/muestra hasta que reciba un bloque especial que indique la finalización del sistema.
  - Cuando reciba el bloque de finalización, liberará los recursos y terminará.
- La introducción/extracción de bloques en memoria compartida seguirá el esquema de productor–consumidor:

```
Productor {
    Down(sem_empty);
    Down(sem_mutex);
    AñadirElemento();
    Up(sem_mutex);
    Up(sem_fill);
}

Consumidor {
    Down(sem_fill);
    Down(sem_mutex);
    ExtraerElemento();
    Up(sem_mutex);
    Up(sem_empty);
}
```

  - Se garantizará que no se pierde ningún bloque con independencia del retraso que se introduzca en cada proceso.
  - Se usará un buffer circular de 6 bloques en el segmento de memoria compartida.

- Se alojarán en el segmento de memoria compartida tres semáforos sin nombre para implementar el algoritmo de productor–consumidor.

Nota. *Se puede comprobar que este apartado es correcto enviando bloques aleatorios generados por el proceso* Comprobador.

c) Programa de minería:
   - El programa se debe ejecutar con dos parámetros:

     ```
     ./miner <ROUNDS> <LAG>
     ```

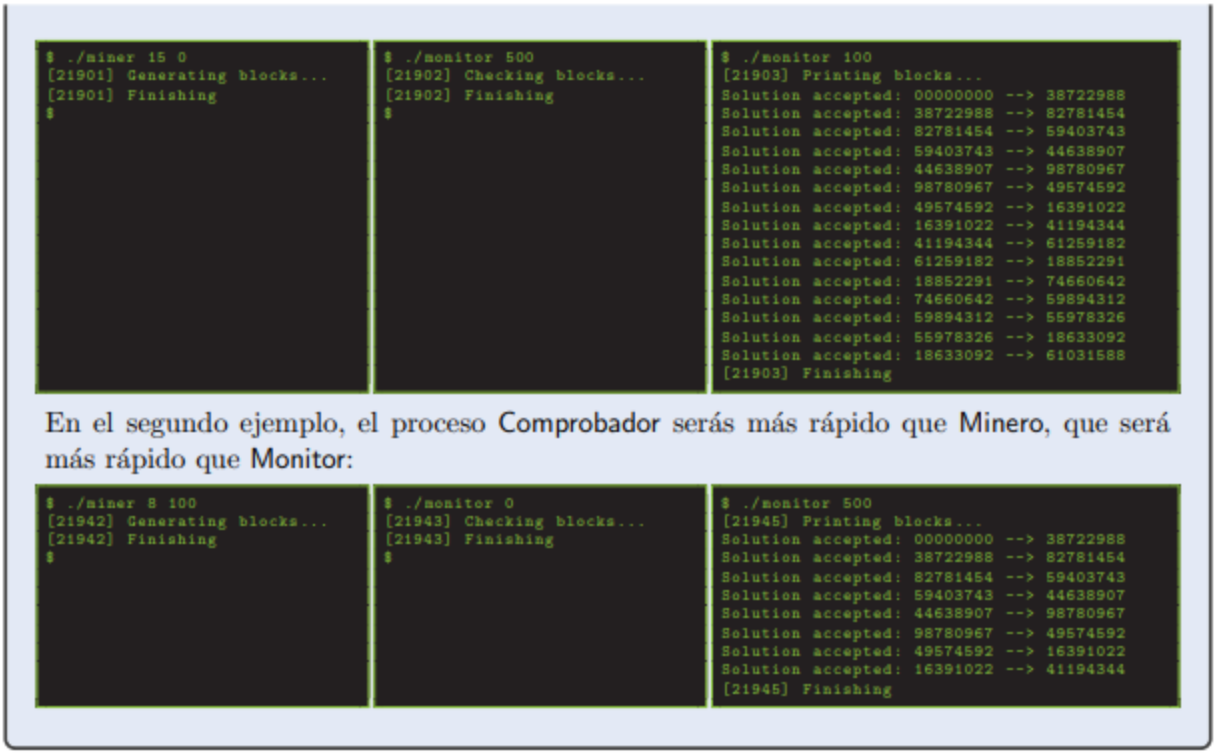
     donde `miner` es el nombre del ejecutable, `<ROUNDS>` el número de rondas que se va a realizar y `<LAG>` es el retraso en milisegundos entre cada ronda.
   - El proceso resultante de ejecutar este programa, Minero:
     - Creará una cola de mensajes con capacidad para 7 mensajes.
     - Establecerá un objetivo inicial fijo para la POW (por ejemplo, 0).
     - Para cada ronda, resolverá la POW; en el caso de que no se tenga un minero funcional capaz de generar una secuencia de bloques encadenados, se puede utilizar el fichero con la secuencia de 200 rondas proporcionado para «falsear» el minero.
     - Enviará un mensaje por la cola de mensajes que contenga, al menos, el objetivo y la solución hallada.
     - Realizará una espera de `<LAG>` milisegundos.
     - Establecerá como siguiente objetivo la solución anterior, y repetirá la ronda.
     - Una vez terminadas las rondas, enviará un bloque especial con algún código que permita saber al proceso Comprobador que el sistema está finalizando.
     - Liberará los recursos y terminará.

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

El esquema global del programa se resume en la Figura 2.

Se deberán tener en cuenta los siguientes aspectos: (a) entrega de la documentación en formato `.pdf` requerida en la normativa de prácticas y con las respuestas a las cuestiones planteadas, (b) control de errores, (c) eliminación de la cola, (d) eliminación del segmento de memoria compartida, y (e) garantizar que no se producen bloqueos ni se pierden mensajes con independencia del retraso introducido en cada proceso.

A continuación se muestran dos ejemplos de ejecución del sistema final. En el primer ejemplo, el proceso Minero serás más rápido que Monitor, que será más rápido que Comprobador:



En el segundo ejemplo, el proceso Comprobador serás más rápido que Minero, que será más rápido que Monitor:

CUESTIONES TEÓRICAS

Las cuestiones teóricas de la práctica se plantean en el apartado D.

¿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 respuesta a las 2 primeras cuestiones propuestas, **sí es necesario utilizar semáforos**, pues garantizan la exclusión mutua, cosa que no se puede saber a ciencia cierta de otra manera, además de que **no es correcto basarse en suposiciones de tiempo relativo entre procesos para garantizar la protección de las zonas críticas**.

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

En cuanto a la última cuestión, **sí se simplificaría la estructura del programa**, ya que las **colas de mensajes están implementadas como colas circulares**, por no hablar de que **nos ahorraríamos el mapeo** de la zona de memoria compartida.

REQUISITOS SATISFECHOS

Se ha podido implementar toda la funcionalidad solicitada.

PRUEBAS REALIZADAS

Se han realizado comprobaciones mediante printf y fprintf, así como probando con distintos valores de entrada y pasando el valgrind.

FUNCIONALIDADES PENDIENTES DE IMPLEMENTACIÓN

No ha habido ninguna funcionalidad pendiente de implementación.