Memoria Práctica 3

Ignacio Serena Montiel y Marcos Muñoz Merchán

Introducción:

En el ámbito de los sistemas operativos, la práctica de implementar un sistema de monitoreo multiproceso es una tarea crucial para comprender conceptos fundamentales como la comunicación entre procesos, el uso de memoria compartida aparte la sincronización mediante semáforos aunque no sea el objetivo principal de esta práctica. En este contexto, el presente informe describe la implementación de un sistema de monitoreo de bloques del sistema Blockchain mediante dos programas en C. El objetivo principal es desarrollar un sistema robusto y eficiente que pueda verificar bloques de la cadena Blockchain y mostrar los resultados en tiempo real.

Funcionalidad:

El sistema consta de dos programas principales: el Programa de Monitoreo y el Programa de Minería . Cada uno de estos programas desempeña un papel específico en el sistema y se comunica entre sí para garantizar el funcionamiento adecuado del sistema de monitoreo.

1. **Programa de Monitoreo:**

- Este programa se encarga de verificar los bloques recibidos y mostrar los resultados por pantalla.
- Se ejecuta con un parámetro que indica el retraso en milisegundos entre cada monitorización.
- Hay dos procesos correspondientes a este programa:
    - El primer proceso detecta si la memoria compartida no está creada y se convierte en el proceso Comprobador, encargado de verificar los bloques.
    - El segundo proceso detecta si la memoria compartida está creada y se convierte en el proceso Monitor, encargado de mostrar los resultados por pantalla.
- La comunicación entre el Comprobador y el Monitor se realiza mediante un buffer circular de bloques en la memoria compartida y tres semáforos sin nombre para implementar el algoritmo de productor-consumidor.
- Se garantiza que no se pierde ningún bloque independientemente del retraso introducido en cada proceso.

2. **Programa de Minería:**
    - Este programa se ejecuta con dos parámetros: el número de rondas a realizar y el retraso en milisegundos entre cada ronda.
    - El proceso Minero crea una cola de mensajes con capacidad para 7 mensajes y establece un objetivo inicial fijo para la prueba de trabajo (POW). En nuestro caso, hemos elegido que sea 0.
    - En cada ronda, resuelve la prueba de trabajo y envía un mensaje por la cola de mensajes que contiene el objetivo y la solución hallada.
    - Después de cada ronda, espera el tiempo especificado antes de iniciar la siguiente ronda.
    - Una vez terminadas todas las rondas, envía un bloque especial que indica al proceso Comprobador que el sistema está finalizando.

Respuestas teóricas:

1. ¿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?

Si, lo es. En esta situación, el Comprobador estaría generando bloques de forma más lenta que el Minero, lo que podría causar que los bloques generados por el Minero se acumulen en la cola de mensajes del Comprobador. Si no se utiliza un sistema de productor-consumidor, existe el riesgo de que estos bloques se pierdan mientras esperan ser verificados. Por lo tanto, el sistema de productor-consumidor es necesario para garantizar que los bloques no se pierdan y para mantener la eficiencia y la integridad del sistema.

2. ¿Y si el retraso de Comprobador es muy inferior al de Minero? ¿Por qué?

En el caso en que el retraso del Comprobador sea mucho menor que el del Minero, la implementación de un sistema productor-consumidor no es estrictamente necesaria. Esto se debe a que el Comprobador sería capaz de procesar los bloques casi tan rápido como son generados por el Minero, lo que evitaría la acumulación de bloques en la memoria compartida.

Sin embargo, considero que aunque no sea estrictamente necesario, aún existen razones para considerar la utilización del sistema productor-consumidor en este contexto:

1. **Flexibilidad:** El sistema puede adaptarse a cambios en los retrasos de los procesos sin necesidad de modificaciones significativas en el código. Si en el futuro cambian las condiciones y el retraso del Comprobador aumenta, el sistema seguiría funcionando de manera eficiente.
2. **Prevención de cuellos de botella:** Aunque el Comprobador pueda procesar los bloques rápidamente, si el Minero genera bloques a un ritmo muy alto, existe el riesgo de que el Comprobador se sature. La

implementación del sistema productor-consumidor ayuda a amortiguar este efecto, permitiendo que el Comprobador trabaje a su propio ritmo y no se vea abrumado por la velocidad de generación de bloques del Minero.

3. **Facilidad de implementación:** La implementación de un sistema productor-consumidor es relativamente sencilla y no se nos olvidará jamás ya que salimos escaldados del examen de teoría con un problema en el que habia que utilizar este algoritmo precisamente.

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

No, la estructura global del sistema no se simplificaría con una cola de mensajes entre Comprobador y Monitor. Introducir una cola de mensajes adicional introduciría redundancia en la comunicación, ya que el Monitor ya recibe la información procesada por el Comprobador a través de la memoria compartida. Además, añadir una cola de mensajes aumentaría la complejidad del sistema y requeriría sincronización adicional entre los procesos, lo que podría dificultar la implementación y mantenimiento del código. Además, la comunicación entre Comprobador y Monitor a través de la memoria compartida es generalmente más eficiente en términos de rendimiento que la comunicación mediante colas de mensajes. Por lo tanto, en este caso, mantener la comunicación a través de la memoria compartida y los semáforos es preferible para mantener la eficiencia y la simplicidad del sistema.

Pruebas que se han realizado:

1. **Pruebas de límites:** Hemos realizado pruebas con un gran número de bloques en el buffer circular de la memoria compartida para verificar si nuestro programa puede manejar el límite máximo de bloques sin perder información o bloquearse.

2. **Pruebas de estabilidad a largo plazo:** Hemos ejecutado el programa durante un período prolongado de tiempo para detectar posibles problemas de memoria, fugas de recursos o degradación del rendimiento con el tiempo.
3. **Pruebas de tolerancia a fallos:** Introdujimos fallos deliberados, como la desconexión de procesos en medio de la ejecución (con ctrl c) o la entrada de datos erroneos, y observamos cómo responde nuestro programa. Y comprobamos que es capaz de recuperarse de manera adecuada y mantener la integridad de los datos.
4. **Pruebas de errores de entrada:** Introdujimos bloques con datos incorrectos o malformados en la cola de mensajes y observamos cómo manejaba nuestro programa estos errores, como la detección y el manejo de bloques inválidos.
5. Pruebas obligatorias: Aparte de las pruebas enteriormente mencionadas, hicimos obviamente las pruebas propuestas en el enunciado de variacion de tiempos de delay, probando todas las combinaciones entre tiempos y asegurandonos que no habia variación ninguna en los resultados.

Requisitos que satisface la práctica

Todos los requisitos de la práctica han sido satisfechos.