UNIVERSIDAD AUTÓNOMA DE MADRID

Grado en Ingeniería Informática

Sistemas Operativos

Práctica 3

Enmanuel De Abreu Gil

Jorge Álvarez Fernández

Grupo 2262

1. Introducción.

Esta práctica se inspira en el concepto de minería en blockchain y consta de tres componentes principales: un proceso Minero que lleva a cabo una Prueba de Trabajo (Proof of Work, POW), un proceso Comprobador encargado de verificar y compartir la información generada por el minero, y un proceso Monitor que muestra los resultados de cada iteración de la minería a través de la terminal. El objetivo principal de esta práctica es profundizar en la comunicación inter-proceso a través de memoria compartida y colas de mensajes.

2. Proceso Minero.

Este programa recibe dos parámetros por línea de comando: 'lag' y 'rounds', donde 'lag' representa el tiempo de espera en milisegundos entre cada iteración de minería, y 'rounds' indica la cantidad de ciclos de minería que se llevarán a cabo. Para manejar la información de cada bloque minado, declaramos una estructura llamada Message, que será reutilizada en cada iteración para almacenar los datos obtenidos de la Prueba de Trabajo (POW).

Para establecer una comunicación eficiente con el proceso Comprobador, creamos una cola de mensajes POSIX con capacidad para almacenar hasta 7 mensajes. A continuación, realizamos un bucle que va desde 0 hasta 'rounds', en el que la estructura Message se rellena con la información resultante de la POW. Después de cada iteración, enviamos el mensaje a través de la cola de mensajes y esperamos 'lag' milisegundos antes de continuar con la siguiente ronda.

Una vez que se han completado todas las rondas de minería, levantamos una bandera adjunta al mensaje para indicar al proceso Comprobador que el minero ha finalizado su trabajo. Esta señal permite que el Comprobador sepa cuándo debe detener su proceso de verificación y compartir los datos con el proceso Monitor.

3. Proceso Comprobador.

El Comprobador recibe el parámetro 'lag' por línea de comando, que determina el tiempo de espera en milisegundos entre cada iteración de comprobación. Para comenzar, abre y mapea un segmento de memoria compartida que contiene un array circular de bloques. Cada bloque está representado por una bandera (aceptado o denegado), un entero 'target' y un entero 'solution'. La memoria compartida también proporciona acceso a semáforos sin nombre que protegen las zonas críticas del proceso.

El Comprobador entra en un bucle infinito, en el que se bloquea a la espera de la entrada de mensajes en la cola declarada por el proceso minero. Cuando recibe un mensaje, lo parsea, verifica su validez y actualiza la información en la memoria compartida, añadiendo un bloque al array circular. Después de cada actualización, el Comprobador duerme 'lag' milisegundos antes de continuar con la siguiente comprobación.

Cuando llega el mensaje que indica la finalización del proceso minero, el Comprobador sube a la memoria compartida el último paquete, que contiene la bandera especial

levantada por el minero. Tras esta acción, el Comprobador libera los recursos utilizados y se cierra.

4. Proceso Monitor.

El Monitor se encarga de revisar y mostrar los bloques conforme son validados y añadidos al array circular. Primero abre el segmento de memoria compartida, que se presupone ya creado por el proceso Comprobador. De esta forma, el Monitor tiene acceso a los bloques validados y registrados en la memoria compartida. Luego entra en un bucle infinito, en el que lee de manera secuencial los bloques que van llegando a la memoria compartida desde el proceso Comprobador.

Por último, imprime por pantalla la información del bloque, incluyendo la bandera (aceptado o denegado), el 'target' y 'solution'. De esta manera, el usuario puede visualizar el resultado de cada iteración de minería y su respectiva verificación.

En cuanto el Monitor lee la bandera especial que indica la finalización del proceso minero, cierra el segmento de memoria compartida y libera el resto de recursos asociados al proceso antes de salir.

5. Funcionalidad

El sistema cumple con toda la funcionalidad propuesta en el enunciado. Genera tres procesos Comprobador, Monitor y Minero, de los cuales los dos primeros corren el mismo script de C. Se abren y manejan correctamente tanto el segmento de memoria compartida como la cola de mensajes. Se estructura la memoria compartida como un array circular de elementos y se utilizan semáforos POSIX sin nombre para proteger el acceso concurrente al segmento.

Adjuntamos un script runv.sh que ejecuta *valgrind –show-leaks=all* para cada uno de los procesos en tres terminales separadas. Hemos logrado que el proyecto no tenga leaks de memoria ni errores reportados por Valgrind.

6. Preguntas

1. En nuestro caso no es necesario, especialmente entre Minero y Comprobador ya que la comunicación entre esos dos procesas ocurre a través de una cola de mensajes. Si Comprobador consume mensajes a un ritmo mucho menor que aquel con el que se generan en Minero, la cola se llenará. No obstante esto no supone pérdida de datos por mq_send() bloquea el proceso hasta que la cola tenga hueco de nuevo.

2. De nuevo, no habría problemas porque mq_receive() también bloquea el proceso en caso de que no haya mensajes a leer en la cola.

3. Si queremos mantener la estructura circular del array de bloques, no. Para mantener actualizado el array a través de mensajes, habría que mandar en cada mensaje el array completo, gastando recursos innecesarios. Es mejor mantenerlo estático en un segmento compartido con acceso para ambos procesos y proteger la concurrencia con semáforos.

[FINAL DEL DOCUMENTO]