MEMORIA PRÁCTICA 3

FECHA DE ENTREGA: DEL 17 DE ABRIL AL 21 DE ABRIL

DANIEL DENEB MECHA MARTÍN Y PEDRO JOSÉ ALEMAÑ QUILES

En esta práctica hemos tenido que orquestar la comunicación entre tres procesos: minero, comprobador y monitor. Se ejecutan en tal orden de modo que el proceso minero produce bloques y los envía por cola de mensajes a un 'buzón' con capacidad para siete que consecuentemente va abriendo comprobador. A su vez este otro establece una relación con monitor de productor-consumidor a través de una cola circular de bloques (ya comprobados y señalizados con una bandera por comprobador) localizada en un segmento de memoria compartida accesible por ambos.

La dificultad recae en la robustez de la comunicación entre las partes involucradas. Algunas de las herramientas utilizadas, como la cola de mensajes implementan ya su propio sistema de esperas de lectura y liberaciones de espera que solo han precisado de una inicialización debida. Para organizar otras herramientas, como la cola de bloques, hemos implementado en el propio struct MemoryQueue unos semáforos sin nombre compartidos por comprobador y monitor (los pertinentes para facilitar la dinámica productor-consumidor). Para cerciorarnos de la integridad de la cadena de secuencia, hemos realizado pruebas modificando el número de rondas y los valores de lag y en todas ellas hemos obtenido buenos resultados.

Esta vez no hemos tenido demasiados problemas para realizar el ejercicio. En todo caso, hacia el inicio sobre todo, no sabíamos muy bien cómo diseñar la ejecución de monitor y comprobador: si uno tenía que llamar al otro con un fork, si habían de comunicarse con señales además de la memoria compartida y otra serie de posibilidades, todas ellas más complejas (quiźas innecesariamente) que la implementación final.

Hemos decido responder a las preguntas del apartado d) aparte del cuerpo principal de la memoria con tal de mantener su integridad:

¿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?

Posiblemente existan otras soluciones a los problemas presentados por la discordancia entre los valores de lag, pero esta funciona y se darían dos tipos de problemas de no ser implementada: que comprobador sobre escribiera entradas de la cola circular que todavía no hubieran sido leídas por monitor o que monitor leyera entradas todavía no escritas. Este protocolo garantiza que no puede suceder tal cosa. Gracias al uso de dos semáforos se evitan estos problemas y gracias a un tercero (un mutex) se evita que se den condiciones de carrera en el segmento de memoria compartida, que solo se verían acrecentados por las disparidades en los lag de no ser por esto.

Esto evita problemas de comunicación entre comprobador y monitor, pero lo que soluciona los problemas entre minero y comprobador no es eso. Si minero fuera mucho más rápido que comprobador, los mensajes, podrían desbordar la cola, pero, gracias a que la capacidad de esta es menor que la máxima que nos permite el sistema operativo, tal cosa no puede suceder y minero se debe quedar esperando a que se reciban mensajes para poder enviar más.

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

Este problema es la cara opuesta del presentado anteriormente y la respuesta es análoga. la llamada a mq_receive(); nos sirve del mismo modo que en la anterior mq_send(); , es deci, que comprobador esperará a tener algún mensaje antes de proceder con el resto de su ejecución. De modo que aunque haya una gran diferencia en el retraso, no se darán problemas de ejecución.

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

Sí. Se podría evitar el uso de semáforos de este modo y la robustez del sistema sería idéntica, pero debería establecerse un modo de terminación de la comunicación. Algo así como una señal de final de conversación el bloque final de comprobador o indicación inicial del número de bloques esperados, de modo que monitor no se quedara esperando indefinidamente una cantidad infinita de bloques. Por otra parte cabe discutir si es mejor idea comunicar dos procesos que ya comparten un segmento de memoria a través de este o de una cola de mensajes. Esto podría depender de los costes de memoria y de cpu de una u otra implementación si es que no son equivalentes. Pero todo esto requiere de un conocimiento del funcionamiento interno de linux más profundo del que tenemos.