SISTEMAS OPERATIVOS: PROYECTO FINAL

Por Oscar Riquelme y Javier Simancas

Objetivos cumplidos del proyecto:

Minero:

1. Lanza una cierta cantidad de hilos determinada en su primer parámetro de entrada. El hilo principal del programa no realiza la tarea de minado (buscar la solución del reto matemático), delegando la misma de forma exclusiva a los hilos creados.

El minero es capaz de lanzar una serie de hilos que ejecutan la función Worker. Esta función divide el intervalo a minar entre el número de trabajadores especificados y logra que cada hilo mine un subintervalo gracias a un identificador pasado por argumento. Se puede comprobar que el minado tiende a ser más eficiente a mayor número de trabajadores.

2. Acepta dos parámetros de entrada que deberán ser los siguientes, respetando el orden:

El minero es capaz de aceptar dos argumentos como parametros de entrada y actuar en consencuencia de los valores pasados. Si no se especifican dichos parámetros el minero falla.

3. El reto matemático viene definido en el archivo miner.c (proporcionado con el enunciado) mediante la función simple_hash. Esta es la función hash que debe utilizarse en el proyecto.

El minero lanza hilos que resuelven la función hash propuesta.

4. Cuando un trabajador (hilo) encuentre una solución al problema deberá notificarlo a todos los trabajadores, tanto del propio minero como al resto de mineros. La comunicación a los trabajadores del propio minero (compañeros) se podrá implementar como se desee, pudiendo utilizarse una variable global. Para comunicar al resto de mineros que se ha encontrado una solución se deberá usar la señal SIGUSR2.

Cuando un trabajador encuentra la solución, manda SIGUSR2 al resto de mineros y envia SIGUSR1 a si mismo para saber que el es el ganador, ademas de setear en memoria compartida las variables pertinentes.

5. Cuando un minero recibe la señal SIGUSR2, sus trabajadores deberán detener su tarea de minado. El minero verifica si la solución encontrada es correcta y vota a favor o en contra de la misma de manera consecuente.

Una vez que se le ha notificado SIGUSR2 a los mineros perdedores, estos entran en espera no activa hasta que el ganador les señaliza que esta preparado para recibir sus votos. Cada minero envia su voto y el ganador se queda a la espera de que todos voten.

6. 7. 8.

Para compartir la solución con el resto de mineros, se utiliza memoria compartida. Se establecerá un mecanismo de sincronización de forma que el minero ganador sea el único que escriba la solución en memoria compartida durante una determinada ronda de minado. El resto de mineros accederá a la misma una vez la solución se haya establecido. La tarea de los mineros perdedores es verificar si la solución es correcta para el objetivo fijado y votar en consecuencia.

La votación se realiza una vez por ronda de minado, después de que un ganador haya enviado la señal SIGUSR2. Para votar, los mineros pueden utilizar un array de voto guardado en la memoria compartida (en la estructura sugerida se denomina voting_poll). Si el minero considera que el voto es positivo su posición del array tendrá el valor 1, si considerá que la solución no es válida el valor será 0. Se podrá usar un valor distinto para marcar que aún no se ha registrado ese voto.

Para saber si hay mayoría de votos, el minero ganador de la ronda previa comprueba el quorum de votos, es decir la cantidad de mineros que van a participar en la votación. Se considera que un minero está activo si el envío de una señal al PID del minero tiene éxito. Para conocer el quorum el minero ganador envía la señal SIGUSR1, y analiza el resultado de la llamada a kill en cada caso. La cantidad de mineros activos se comparte en la red mediante la actualización de memoria compartida. Se sugiere el uso del campo total_miners.

A traves de la función vote, los mineros acceden a memoria compartida y dejan en su posición correspondiente una constante VALID_VOTE o INVALID_VOTE que señaliza si la solución es válida o no. El minero ganador logra decidir si la solución propuesta es válida o no de acorde a lo votado. Para ello, consigue el número de mineros activos y sus posiciones y hace un recuento de voto y comprueba si dicho recuento alcanza la mayoría. Se ha decidido que total_miners guarda el número de mineros totales en vez del número de mineros activos pues solo el ganador necesita acceder a dicho número. Según se decide quién es el ganador, con la función notifyAllMiners, se obtienen los valores ya mencionados. Más adelante, a través de esta función se irá comunicando el ganador con los mineros perdedores.

9. Con el quorum actualizado, el nuevo ganador env´ıa la se˜nal SIGUSR2 para iniciar el proceso de voto

En vez de SIGUSR2, se decidió utilizar SIGUSR1 para reutilizar la función notifyAllMiners. Dicha función notifica a todos los míneros que estaban en espera no activa de que un evento ocurrió y pueden seguir adelante.

10. El minero ganador cuenta los votos para saber si su propuesta ha sido aceptada. En caso de serlo da valor 1 al campo is_valid del bloque compartido en memoria. Esta variable compartida se usa de manera que los perdedores sepan si la votación ha sido positiva o negativa. Es el minero ganador quien debe modificarla.

El minero ganador es capaz de setear el campo is_valid a 1 o a 0 en función de los resultados obtenidos del referendum.

11. En caso de que el bloque sea aceptado (válido) todos los mineros lo incorporan a su cadena de bloques propia.

Los mineros tienen un array de bloques el cual va siendo rellenado con los bloques válidos según avanza sus rondas.

12. Una vez que todos los mineros han incorporado el bloque (si ha sido aceptado), el minero ganador prepara un bloque nuevo que tendrá como objetivo (target) la solución del anterior (último bloque añadido a la cadena de bloques). En el caso del primer minero de la red, actuará como el ganador de la ronda anterior, y será el encargado de preparar la siguiente ronda.

El minero ganador es capaz de añadir el bloque a la blockchain de memoria compartida a traves de la funcion addToBlockchain. Esta borra el mapeado de memoria del anterior bloque, trunca el segmento de memoria compartida de tamaño x a x + sizeof(Block) y retorna el nuevo segmento mapeado. Una vez hecho esto, cada minero perdedor hace lo propio con la función updateBlock y así el nuevo bloque queda preparado con los datos necesarios para la siguiente ronda.

13. En caso de que haya un proceso monitor activo, los mineros intentan enviar a través de una cola de mensajes el nuevo bloque creado por cada uno al programa monitor. Si es un minero ganador la prioridad será 2, si es un minero perdedor la prioridad será 1.

Los mineros son capaces de mandar bloques a traves de una cola en caso de que monitor_pid sea distinto de la constante NO_MONITOR con la prioridad especificada en cada caso.

14. Se debe implementar una función que imprima en un fichero identificado por su PID la cadena de bloques propia del minero antes de acabar su ejecución.

Se ha implementado la función printBlockchainAtExit la cual es capaz de imprimir el blockchain propio del minero antes de salir. Dicha función construye el path como ./blockchains/pid por lo que es necesaria la existencia de la carpeta blockchains. El path se genera juntando la string ./blockchains/ con lo devuelto por la función implementada intToAscii. (A dicha función se le pasa el pid del minero).

15. El minero acabará su ejecución al recibir la señal SIGINT, o cuando haya realizado el número de rondas indicado en el tercer parámetro de entrada.

El minero es capaz de terminar su ejecución tras recibir SIGINT o cuando llega al máximo de rondas especificadas

Trabajador

1. Cada hilo trabajador recibe del hilo principal una estructura con al menos el objetivo de la POW.

Se ha implementado una estructura WorkerInfo con toda la información necesaria para que el trabajador realice la POW.

2. Esa misma estructura se puede utilizar para devolver datos al hilo principal.

Se ha decidido pasar en la estructura los punteros a memoria compartida para que sea el trabajador el que escribe y no el hilo al principal.

3. Cada ronda de minado tendrá un conjunto de trabajadores nuevo. Por tanto, al acabar el trabajo los hilos deben finalizar.

El minero lanza para cada ronda un conjunto de trabajadores que son esperados con pthread_join.

Funcionamiento de la Red

La red se compone de un conjunto de procesos minero y opcionalmente un proceso monitor (se debe implementar este programa, pero no es necesario para que la red funcione con normalidad).
El monitor no es necesario para el funcionamiento de la red, pero tanto el monitor como los mineros son capaces de incorporarse a la red a la perfección en cualquier momento.

1. Estos procesos se pueden lanzar desde una misma shell, o desde shells diferentes.

Es posible lanzar estos procesos desde una misma Shell (con el operador &) o desde Shells distintas.

2. Una vez se ha lanzado el primer minero, cualquier otro minero lanzado se unirá a la red.

En cualquier momento cualquier minero se puede unir a la red, haya mineros en ella o no.

3. Se considera que un minero se ha unido a la red cuando se junta (attach) a los segmentos de memoria compartida con la información de la red y del bloque compartidos.

Cuando se lanza un minero a través de la función signUp el minero comprueba si hay una ronda en curso en cuyo caso se apunta a una lista de espera de la cual saldrá cuando termine la ronda. En caso contrario, el minero es capaz de registrarse en la red e incorporarse a las rondas de minado en cualquier momento.

4. En la información de red se pueden compartir elementos de sincronización como semáforos.

En la memoria compartido se han incluido una serie de variables y elementos de sincronización como semáforos para proporcionar a los mineros la información suficiente para no comprometer la integridad de la red.

5. La estructura compartida con información de red tendrá al menos un campo con los PIDs de todos los procesos activos (attached) que se denominará miners_pid.

La red es capaz de tener un array con el pid de todos los mineros el cual va siendo actualizado con la funcion signUp.

6. Los mineros pueden unirse en cualquier momento.

Los mineros pueden unirse en cualquier momento siempre que no haya una ronda en curso, en cuyo caso entra a una lista de espera y se une cuando termine la ronda en curso.

7. Los mineros pueden acabar y salir de la red en cualquier momento.

El minero podrá salir de la red una vez haya terminado la ronda en la que está trabajando. Se ha decidido esto para no comprometer el sistema de votación.

8. La información sobre la red debe adaptarse de forma dinámica a los cambios anteriores. Uno de los mineros, el ganador se encarga de actualizar los datos de red compartidos.

Al terminar cada ronda el minero ganador es capaz de actualizar todos los datos de red.

9. El funcionamiento de la red será robusto frente a altas o bajas de mineros, en particular se prestará atención a la validez de la información compartida y a evitar bloqueos (deadlocks).

La red está diseñada de forma que no ocurran interbloqueos.

11. Si se considera necesario para evitar bloqueos provocados por la ausencia de alguno de los mineros, se podrán establecer mecanismos de timeout de 2 o 3 segundos. Una forma sencilla de hacerlo es mediante el uso de alarm. Si un minero está esperando a empezar una ronda, pero pasan más de 2 segundos sin recibir la señal de inicio puede deberse a que el minero ganador de la ronda anterior ha sufrido un problema. Al recibir la alarma puede detectar este problema y terminar.

No se ha considerado necesario.

Monitor

1. Crea un proceso hijo.

El monitor es capaz de crear un proceso hijo

2. Recibe de los mineros mensajes con los nuevos bloques.

El monitor recibe los bloques de los mineros con distinta prioridad, uno para un minero perdedor y dos si es para el minero ganador.

3. El proceso padre dispone de un buffer de memoria para los 10 últimos bloques. Tras recibir un nuevo mensaje, comprueba si el identificador del bloque recibido ya está en el buffer. Si es así, comprueba si la solución y el objetivo coinciden. En caso afirmativo, imprime por pantalla "Verified block X with solution Y for target Z", donde X, Y y Z son los valores correspondientes. En caso de error imprime el mensaje "Error in block X with solution Y for target Z". Si el identificador no se encuentra en el buffer, se incorpora eliminando el más antiguo.

Se ha implementado un buffer circular que asegura el borrado del bloque más antiguo. Se ha añadido un mensaje que indica que el bloque no estaba en los buffers, por lo tanto, es un bloque nuevo. El monitor es capaz de imprimir los mensajes especificados.

4. El proceso padre enviará por tubería una copia de cada nuevo bloque al proceso hijo.

El proceso padre es capaz de comunicar los bloques nuevos a sus hijos.

5. El proceso hijo mantendrá una cadena con todos los bloques recibidos del padre.

El proceso hijo mantiene una cadena de bloques reservada de forma dinámica. Esto permite que el blockchain del monito se pueda extender de forma indefinida.

6. El proceso hijo imprimirá en un fichero la cadena de bloques completa cada 5 segundos.

Cada cinco segundos se envía un SIGALRM para que el proceso hijo imprima la cadena de bloques. Sin embargo, para asegurar la integridad del proceso SIGALRM es bloqueado en cada iteración del bucle y el proceso solo imprimirá una vez terminada dicha iteración.

7. El proceso monitor finaliza al recibir la señal SIGINT. Padre e hijo finalizan de manera ordenada.

Padre e hijo logran liberar recursos y terminar de forma ordenada, primero el hijo y luego el padre, asegurando que no quedan procesos huérfanos.

8. Se podrá reiniciar el proceso monitor mediante una nueva llamada de ejecución. El proceso monitor retomará el trabajo.

Es posible reiniciar el proceso monitor de forma que siga trabajando mediante una nueva llamada de ejecución.

Limitaciones del proyecto:

A priori, el proyecto cumple con todos los requisitos pero tiene una serie de limitaciones entre las cuales las más destacables son:

-El proceso monitor no imprime cada 5 segundos sino que cada 5 segundos se lanza SIGALRM y cuando termina la iteración del proceso hijo se imprime el blockchain por lo que es posible que el tiempo tenga una desviación positiva de los 5 segundos especificados como requisito.

-Los mineros deben acabar la ronda antes de terminar incluso si se le envia SIGINT, esto es necesario para asegurar que la red es robusta pero choca con el requisito de que los mineros abandonen la red según se les señalice.

-Los mineros tienen un array estático de blockchain por lo que el número de rondas que pueden hacer tiene que ser menor que el tamaño de dicho array estático. Esto también lleva a que no se haya implementado el requisito del segundo parámetro de la función en caso de que sea 0 o negativo. Es decir, el número de rondas esta comprendida entre 1 y dicho tamaño.

-Al finalizar todos los procesos, los recursos de memoria compartida quedan como existentes para asegurar una red robusta y por lo tanto, valgrind tiene quejas sobre memoria no liberada aunque creemos que esto es lo que se pretendía y por lo tanto no es una limitación, pero si es digno de mención.

-No se ha considerado necesario para la robustez de la red el implementar un control para que en caso de que el proceso pase x segundos sin seguir adelante termine

-Se ha utilizado numerosas veces la funcion sleep para suspender el proceso 1 segundo o 2. Esto no es estrictamente necesario para el funcionamiento de los programas pero, al ser un proyecto experimental, creemos conveniente dejar unos segundos entre mensaje y mensaje para entender que está pasando.

-Los elementos de memoria compartida para reiniciar la red deben ser borrados de forma manual con el comando implementado make memClean

Autoevaluación:

Calificación base:

Se ha llegado al nivel 5 por lo que el máximo de la calificación base debería de ser un 10.

Modificadores

En cuanto a los modificadores, no hemos podido encontrar ninguna situación en la que el programa falle. Es robusto y no tiene perdidas de memoria pero si es cierto que hay

que borrar de manera manual la memoria compartida con make memClean si se quiere reiniciar la red. Aun asi, consideramos que no se ha aportado la suficiente documentación y por lo tanto creemos que como modificador deberíamos de estar en algún punto intermedio entre el nivel 2 y el 3 (75% y 120%). Por lo tanto, tomaremos un punto medio y diremos que el modificador es de un 100% y no afecta a la nota.

Penalizaciones

Consideramos que no hemos modularizado lo suficiente. De ser posible, nos habría gustado implementar modulos tanto para manejar la estructura Block como para manejar la estructura netData. Por lo que en ese sentido tenemos una penalización de -1 punto.

Nota final: 9