Proyecto Final: *Miner Rush*

FECHA DE ENTREGA: ANTES DEL 10 DE MAYO
(HORA LÍMITE: 9 DE MAYO A LAS 23:59)

> «*There are three eras of currency: commodity based, politically based, and now, math based.*»
>
> Chris Dixon.

Descripción general

NOTA: En el Proyecto existen partes que se han declarado Opcionales y se identifican con la etiqueta ***[OPCIONAL]***. Los estudiantes podrán obtener la máxima calificación sin necesidad de implementarlas, si bien si se implementan podrá subir la nota final del proyecto hasta un máximo de 10, con un máximo de 2 puntos para estas partes opcionales.

Introducción

El objetivo de este proyecto es programar el software para poder crear una red de mineros de bloques inspirada en la tecnología **Blockchain**, aunque con diferencias importantes respecto a este tipo de sistemas[1].

La red se compone de un conjunto de ***mineros*** que tendrán como misión la resolución de una prueba de esfuerzo (*Proof of Work*, POW). Todos los mineros tendrán que intentar resolver de forma concurrente la misma prueba, y solo uno, el primero en encontrarla, recibirá la recompensa: una moneda.

La ***POW*** consiste en resolver un reto matemático, habitualmente hallar el operando que permite obtener un determinado resultado tras aplicar una función hash no invertible, es decir que la única forma de encontrar el operando es por fuerza bruta. En particular, dado un objetivo (*target*) t y la función hash f se debe encontrar la solución s tal que $f(s) = t$. Puesto que la

[1] Se puede consultar información sobre el funcionamiento del sistema Blockchain más implantado, el Bitcoin, en este enlace.

función f no es invertible, la única forma de encontrar s es probar todos los posibles valores hasta hallar aquel que satisface la ecuación.

En esta red distribuida cada nodo trabaja de forma independiente, y al encontrar una solución debe recibir la aprobación de al menos la mitad del resto de nodos para considerar que ha sido aceptada por la red. La solución se incorpora en un contenedor de información denominado ***bloque***. Cada minero tendrá por tanto una copia idéntica de la cadena de bloques, o al menos de segmentos de la misma.

Además, dentro de la red habrá un ***monitor***, encargado de mostrar la evolución del sistema. Será necesario que el monitor esté activo para que el sistema muestre la salida.

Funcionamiento del sistema

Mineros La red se compone de un conjunto de mineros.

- Cada minero será un proceso, Minero, encargado de resolver la POW usando múltiples hilos en paralelo. ***[OPCIONAL]***En la implementación opcional, el Minero estará formado por dos procesos: el proceso padre Minero, y el proceso hijo, Registrador, encargado de mantener actualizado un fichero de registro. Estos procesos se comunican por medio de una tubería.
- El minero que consiga ser el primero en resolver la POW solicita al resto de mineros que validen su solución mediante una votación, obteniendo una moneda si se aprueba. Además, envía el bloque resultante al monitor, si está activo, a través de una cola de mensajes.
- Los mineros se pueden lanzar desde una misma shell, o desde shells diferentes.
- Una vez se ha lanzado el primer minero, cualquier otro minero lanzado se une a la red de mineros.
- Los mineros pueden unirse en cualquier momento.
- Los mineros pueden acabar y salir de la red en cualquier momento.
- La información sobre la red debe adaptarse de forma dinámica a los cambios anteriores.
- El funcionamiento de la red debe ser 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*).

Nota. *Para simplificar la codificación, se permite suponer que el número máximo de mineros que van a participar en el proceso de minado de una red nunca excederá una determinada constante (un valor razonable inicial para esta constante es de* `100`*). A partir de ese valor, el sistema se considera lleno y se rechazan los nuevos mineros.*

Monitor El sistema debe contar con un monitor, que permite a los mineros enviar datos informativos sobre la ejecución del sistema, de manera que se cree una salida única para toda la red. Por tanto, el monitor ha de ser el primero en arrancar en el Sistema.

- El monitor está formado por dos procesos, el proceso padre, Comprobador, encargado de recibir los bloques por cola de mensajes y validarlos, y el proceso hijo, Monitor, encargado de mostrar la salida unificada. Estos procesos se comunican usando memoria compartida y un esquema de productor–consumidor.
- El monitor deberá arrancar en primer lugar.
- El monitor deberá terminar el último. Si en algún momento se para el monitor, el sistema detendrá su ejecución.

Para implementar este sistema, se escribirán **dos programas en C**, uno correspondiente al minero y otro al monitor.

Recursos de la red

Bloque Cada desafío resuelto se almacena en un bloque, que contiene, entre otros, los siguientes datos:

- Un identificador del bloque, que se incrementa en cada ronda.
- El *target*, resultado del que se desea hallar el operando.
- La solución, operando que permite hallar el *target*.
- El PID del minero «ganador», el que ha resuelto la POW.
- Las carteras de los mineros actuales, incluyendo sus PID y las monedas que han obtenido hasta ese momento.
- El número de votos totales que se han usado para evaluar ese bloque, y el número de votos positivos.

Segmento de memoria del sistema En memoria compartida se almacena una estructura con la información del sistema, entre otros:

- Un listado con los PID de los mineros activos.
- Un listado con los votos de cada minero.
- Un listado con las monedas obtenidas por cada minero.
- El último bloque resuelto, y el bloque actual que se está resolviendo.

Además, se pueden incluir los semáforos que se consideren necesarios para sincronizar el sistema.

Cola de mensajes Se utiliza una cola de mensajes para realizar la comunicación entre los mineros y el monitor.

[OPCIONAL]Tuberías ***[OPCIONAL]***La comunicación entre los dos procesos que conforman el minero (Minero y Registrador) se hace usando tuberías.

Segmento de memoria de monitoreo La comunicación entre los dos procesos que conforman el monitor (Comprobador y Monitor) se hace usando una cola circular almacenada en un segmento de memoria compartida diferente al segmento del sistema. Además, se usa un esquema productor–consumidor.

Actores y requisitos

Proceso Minero

Arranque Este proceso arranca a través de un programa que se debe ejecutar con dos parámetros:

```
./miner <N_SECONDS> <N_THREADS>
```

donde `miner` es el nombre del ejecutable, `<N_SECONDS>` es el número de segundos que va a ejecutar el proceso y `<N_THREADS>` es el número de hilos que se van a utilizar.

[OPCIONAL]Creación de Registrador El proceso principal, Minero, crea un proceso hijo, Registrador, que será el encargado de registrar los bloques en un fichero.

Conexión a memoria El proceso Minero abre el segmento de memoria compartida con la información del sistema, detectando si es el primero minero o no:

- Si es el primer minero, debe comprobar si el monitor está activo, y posteriormente, inicializar el sistema, dando tamaño al segmento, enlazándolo a su espacio de memoria, estableciendo el valor por defecto de los distintos campos, y creando los semáforos oportunos.
- Si no es el primer minero, simplemente debe abrir el segmento y enlazarlo, garantizando que se haya inicializado antes. Esto se puede hacer, por ejemplo, comprobando que el segmento ya tiene tamaño y que el último semáforo que se vaya a crear ya está disponible, y si no es así repitiendo una pequeña espera inactiva hasta que el sistema esté listo.

Registro del minero Tanto si es el primero como si no, Minero debe registrar su PID en el sistema, saliendo en el caso de que el sistema esté lleno.

Preparación Minero se prepara para empezar a minar en su primera ronda.

- Si es el primer minero, establece el objetivo inicial para el primer bloque (se puede fijar, por ejemplo, a `0`), y arranca el sistema enviando la señal `SIGUSR1` al resto de mineros para arrancar la primera ronda.
- Si no es el primer minero, espera la llegada de la señal `SIGUSR1` que indica que va a arrancar una nueva ronda (es decir, no se une en medio de la ronda, para evitar inconsistencias).

Minado Una vez arrancada la ronda, Minero empieza a minar en busca de la solución al desafío, usando la implementación de la POW disponible en los ficheros `pow.c` y `pow.h`[2], de la siguiente forma:

1. Divide el espacio de búsqueda entre tantos hilos como se haya especificado, de manera que realicen búsquedas independientes para paralelizar la tarea.
2. Crea los hilos, y espera a que terminen.
3. Cuando un hilo encuentra la solución, terminan todos los hilos (no se reutilizan para otras rondas) y se devuelve la solución al hilo principal.
4. Del mismo modo, si algún otro minero de la red encuentra la solución y notifica a través de `SIGUSR2` que ya está listo para realizar la votación, todos los hilos deben terminar y Minero pasará a la votación.

Proclamación de Ganador Tras el minado, si Minero ha obtenido la solución trata de proclamarse el proceso Ganador para esa ronda a través de algún mecanismo que evite las condiciones de carrera, por ejemplo usando un semáforo a tal efecto. En el caso de que no consiga convertirse en proceso Ganador, o bien de que haya terminado el minado por haber recibido `SIGSUR2`, se convierte en proceso Perdedor durante el resto de la ronda.

[2]En concreto, la función `pow_hash` realiza el cálculo de la función hash, mientras que la constante `POW_LIMIT` es el límite superior del rango de búsqueda (es decir, hay que probar los valores entre 0 y `POW_LIMIT` − 1).

Votación Una vez terminada la proclamación del proceso Ganador, comienza el proceso de votación.

- El proceso Ganador:
  1. Prepara la memoria compartida para la votación, introduciendo la solución obtenida en el bloque actual, e inicializando el listado de votos.
  2. Envía la señal `SIGUSR2` a todos los procesos que han participado en la ronda para arrancar la votación.
  3. Hace esperas no activas cortas hasta que todos los procesos introduzcan su voto, o hasta que transcurra un número máximo de intentos (lo que garantiza que el sistema no se bloquee si termina de forma abrupta algún minero antes de votar).
  4. Una vez terminada la votación, se comprueba el resultado, se introduce el recuento en el bloque correspondiente de memoria compartida, y si se ha recibido la aprobación se incrementa el número de monedas del proceso Ganador.
- El proceso Perdedor:
  1. Si no le ha llegado aún la señal `SIGUSR2` indicando que arranca la votación, realiza una espera no activa hasta recibirla.
  2. Una vez recibida la señal `SIGUSR2`, comprueba el bloque actual, y registra su voto como positivo o negativo, en función de si la solución es correcta o no.
  3. Entra en una espera no activa hasta recibir la señal `SIGUSR1` que indique que arranca una nueva ronda.

Envío del bloque El proceso Ganador debe enviar el bloque resuelto por cola de mensajes al monitor.

Preparación de la siguiente ronda Una vez notificado el monitor, Ganador prepara la siguiente ronda, usando como objetivo la solución actual. Terminados los preparativos, envía `SIGUSR1` a todos los mineros para notificar que empieza la siguiente ronda.

[OPCIONAL]Registro del bloque ***[OPCIONAL]***Antes de empezar a minar en la siguiente ronda, todos los mineros (el proceso Ganador y los procesos Perdedor) envían el bloque resuelto a través de una tubería a su respectivo proceso Registrador.

Finalización El proceso Minero finaliza cuando transcurre el número de segundos especificado como argumento, o cuando recibe la señal `SIGINT`. En concreto, debe desregistrarse del listado en memoria compartida, y salir liberando todos los recursos. En el caso de que sea el último minero de la red, debe eliminar además todos los recursos compartidos, y notificar al monitor el cierre del sistema enviándole un bloque de finalización por cola de mensajes. ***[OPCIONAL]***En caso de implementar el registrador, espera la finalización de su proceso Registrador, quien debe detectar el cierre de la tubería, e imprime un mensaje de aviso en el caso de que no termine con el código de salida `EXIT_SUCCESS`.

Proceso Comprobador

Arranque Este proceso arranca a través de un programa que se debe ejecutar sin parámetros:

```
./monitor
```

donde `monitor` es el nombre del ejecutable.

Conexión a memoria El proceso principal, Comprobador, crea e inicializa un segmento de memoria compartida de monitoreo, dándole tamaño, enlazándolo a su espacio de memoria, y creando los semáforos oportunos.

Creación de Monitor El proceso Comprobador crea un proceso hijo, Monitor, que será el encargado de mostrar los resultados por pantalla, recibiéndolos desde Comprobador a través del segmento de monitoreo.

Recepción y validación del bloque El proceso Comprobador se encarga de recibir y validar los bloques enviados por el proceso Ganador en cada ronda:

- Recibe un bloque a través de la cola de mensajes.
- Comprueba el bloque de nuevo (con independencia de sus votos) ajustando la bandera que indica si es correcto o no.
- Una vez comprobado, lo introduce en memoria compartida para que lo lea Monitor, y vuelve a repetir el proceso de recepción/comprobación/escritura.

Escritura del bloque La introducción 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);
}
```

En concreto:

- Se debe garantizar que no se pierde ningún bloque.
- Se usa un buffer circular de 5 bloques en el segmento de memoria compartida.
- Se alojan en el segmento de memoria compartida tres semáforos sin nombre para implementar el algoritmo de productor–consumidor.

Finalización El proceso Comprobador finaliza cuando recibe un bloque de finalización a través de la cola de mensajes, o cuando le llega la señal `SIGINT`. En cualquiera de los dos casos, notifica el cierre del sistema escribiendo un bloque de finalización en memoria compartida para Monitor. Por último, espera la finalización del proceso Monitor, imprime un mensaje de aviso en el caso de que no termine con el código de salida `EXIT_SUCCESS`, y termina liberando todos sus recursos.

Proceso Monitor

Arranque El proceso Monitor arranca como hijo del proceso Comprobador.

Extracción e impresión del bloque El proceso Monitor se encarga de mostrar los bloques por pantalla:

1. Extrae un bloque, usando el esquema de productor–consumidor definido antes.
2. Lo muestra por pantalla usando el mismo formato que los procesos Registrador, y repite el ciclo de extracción/muestra.

Finalización Cuando se extrae el bloque de finalización, libera todos los recursos y termina con código de salida `EXIT_SUCCESS` si todo fue correcto, o `EXIT_FAILURE` en caso de error.

[OPCIONAL] Proceso Registrador

Arranque El proceso Registrador arranca como hijo de un proceso Minero.

Registro de bloque Mientras la tubería que lo conecta con Minero se mantenga abierta, lee los bloques, y los escribe usando la función `dprintf` en un fichero cuyo nombre dependa del PID de su proceso padre.

Formato del bloque Para imprimir el bloque se puede usar un formato similar al siguiente para bloques correctos:

```
Id:       0019
Winner:   6837
Target:   27818400
Solution: 53980520 (validated)
Votes:    6/6
Wallets:  6836:05  6837:04  6838:06  6866:02  6885:01  6908:02
```

Y a este para bloques incorrectos:

```
Id:       0019
Winner:   7072
Target:   27818400
Solution: 00000011 (rejected)
Votes:    1/6
Wallets:  7040:06  7041:04  7042:03  7072:03  7091:01  7115:02
```

Finalización Cuando se cierra la tubería, libera todos los recursos y termina con código de salida `EXIT_SUCCESS` si todo fue correcto, o `EXIT_FAILURE` en caso de error.

Entregables y ejemplos de ejecución

Entregables

A continuación se detallan los entregables que deberán resultar de este proyecto, así como sus requisitos y la puntuación asociada a cada uno de ellos.

1,50 ptos. **Entregable 1: Memoria.**

0,50 ptos. a) Realizar un diagrama que muestre el diseño del sistema, incluyendo los distintos componentes (procesos) y sus jerarquías, así como los mecanismos de comunicación y sincronización entre ellos.

0,50 ptos. b) Describir el diseño del sistema, incluyendo los aspectos más relevantes, y los principales problemas que se han encontrado durante la implementación y cómo se han abordado.

0,50 ptos. c) Detallar las limitaciones que tiene el sistema, así como las pruebas realizadas para comprobar que el sistema es correcto. Si no se han implementado todos los requisitos, o el sistema presenta errores conocidos, es importante especificarlo en este apartado.

8,50 ptos. **Entregable 2: Sistema de Minero Multiproceso.** Entregar el código del sistema multiproceso, incluyendo un fichero `Makefile` para compilarlo. Se valorarán los siguientes aspectos:

2,00 ptos. a) Funcionamiento general del proceso Minero.

2,00 ptos. b) Funcionamiento general del proceso Comprobador.

2,00 ptos. c) Funcionamiento general del proceso Monitor.

0,50 ptos. d) Gestión correcta de las señales y mecanismos de sincronización.

0,50 ptos. e) Gestión correcta de la memoria compartida.

0,50 ptos. f) Gestión correcta de las tuberías.

0,50 ptos. g) Acceso concurrente correcto a la memoria compartida.

0,50 ptos. h) Gestión correcta de la cola de mensajes.

2,00 ptos. i) Funcionamiento general del proceso Registrador.

Ejemplos de ejecución

A continuación se muestran algunos ejemplos de ejecución del sistema final. En este ejemplo se ejecuta el sistema con dos mineros casi simultáneos y un monitor:

```
$ ./monitor & ./miner 2 1 & ./miner 3 1
[1] 62178
[2] 62179
Id:         0000
Winner:     62178
Target:     00000000
Solution:   38722988 (validated)
Votes:      1/1
Wallets:    62178:01  62179:00

Id:         0001
Winner:     62178
Target:     38722988
Solution:   82781454 (validated)
Votes:      2/2
Wallets:    62178:02  62179:00

Id:         0002
Winner:     62179
Target:     82781454
Solution:   59403743 (validated)
Votes:      2/2
Wallets:    62178:02  62179:01

Id:         0003
Winner:     62179
Target:     59403743
Solution:   44638907 (validated)
Votes:      2/2
Wallets:    62178:02  62179:02

Id:         0004
Winner:     62178
Target:     44638907
Solution:   98780967 (validated)
Votes:      2/2
Wallets:    62178:03  62179:02

Id:         0005
Winner:     62178
Target:     98780967
Solution:   49574592 (validated)
Votes:      2/2
Wallets:    62178:04  62179:02

Id:         0006
Winner:     62178
Target:     49574592
Solution:   16391022 (validated)
Votes:      2/2
Wallets:    62178:05  62179:02

Id:         0007
Winner:     62179
Target:     16391022
Solution:   41194344 (validated)
Votes:      1/1
Wallets:    62179:03

Id:         0008
Winner:     62179
Target:     41194344
Solution:   61259182 (validated)
Votes:      1/1
Wallets:    62179:04

Id:         0009
Winner:     62179
Target:     61259182
Solution:   18852291 (validated)
Votes:      1/1
Wallets:    62179:05

Id:         0010
Winner:     62179
Target:     18852291
Solution:   74660642 (validated)
Votes:      1/1
Wallets:    62179:06

[1]-  Hecho                   ./miner 2 1
[2]+  Hecho                   ./miner 3 1
$
```

En este ejemplo se arranca un minero y el monitor, y un segundo más tarde se arranca otro minero, que se une al sistema:

```
$ ./monitor & ./miner 2 1 &  (sleep 1 ; ./miner 1 3)
[1] 62321
[2] 62322
Id:         0000
Winner:     62321
Target:     00000000
Solution:   38722988 (validated)
Votes:      1/1
Wallets:    62321:01

Id:         0001
Winner:     62321
Target:     38722988
Solution:   82781454 (validated)
Votes:      1/1
Wallets:    62321:02

Id:         0002
Winner:     62321
Target:     82781454
Solution:   59403743 (validated)
Votes:      1/1
Wallets:    62321:03

Id:         0003
Winner:     62321
Target:     59403743
Solution:   44638907 (validated)
Votes:      1/1
Wallets:    62321:04  62323:00

Id:         0004
Winner:     62323
Target:     44638907
Solution:   98780967 (validated)
Votes:      2/2
Wallets:    62321:04  62323:01

Id:         0005
Winner:     62323
Target:     98780967
Solution:   49574592 (validated)
Votes:      2/2
Wallets:    62321:04  62323:02

Id:         0006
Winner:     62321
Target:     49574592
Solution:   16391022 (validated)
Votes:      2/2
Wallets:    62321:05  62323:02

Id:         0007
Winner:     62323
Target:     16391022
Solution:   41194344 (validated)
Votes:      2/2
Wallets:    62321:05  62323:03

[1]-  Hecho                   ./miner 2 1
[2]+  Hecho                   ./monitor
$
```

En este ejemplo, se ejecuta primero el monitor, luego se lanza el sistema con dos mineros casi simultáneos, se cierra uno de ellos, y se añade otro más:

```
$ ./miner 3 1 & ./miner 2 1
[1] 62520
^CFinishing by signal...
$ ./miner 2 1
[1]+  Hecho                   ./miner 3 1
$
```

```
$ ./monitor
Id:         0000
Winner:     62521
Target:     00000000
Solution:   38722988 (validated)
Votes:      2/2
Wallets:    62521:01  62520:00

Id:         0001
Winner:     62520
Target:     38722988
Solution:   82781454 (validated)
Votes:      1/1
Wallets:    62520:01

Id:         0002
Winner:     62520
Target:     82781454
Solution:   59403743 (validated)
Votes:      1/1
Wallets:    62520:02

Id:         0003
Winner:     62520
Target:     59403743
Solution:   44638907 (validated)
Votes:      1/1
Wallets:    62520:03

Id:         0004
Winner:     62520
Target:     44638907
Solution:   98780967 (validated)
Votes:      1/1
Wallets:    62520:04

Id:         0005
Winner:     62520
Target:     98780967
Solution:   49574592 (validated)
Votes:      1/1
Wallets:    62520:05

Id:         0006
Winner:     62520
Target:     49574592
Solution:   16391022 (validated)
Votes:      1/1
Wallets:    62520:06

Id:         0007
Winner:     62520
Target:     16391022
Solution:   41194344 (validated)
Votes:      1/1
Wallets:    62539:00  62520:07

Id:         0008
Winner:     62539
Target:     41194344
Solution:   61259182 (validated)
Votes:      2/2
Wallets:    62539:01  62520:07

Id:         0009
Winner:     62539
Target:     61259182
Solution:   18852291 (validated)
Votes:      2/2
Wallets:    62539:02  62520:07

$
```