Sistemas Operativos, proyecto final

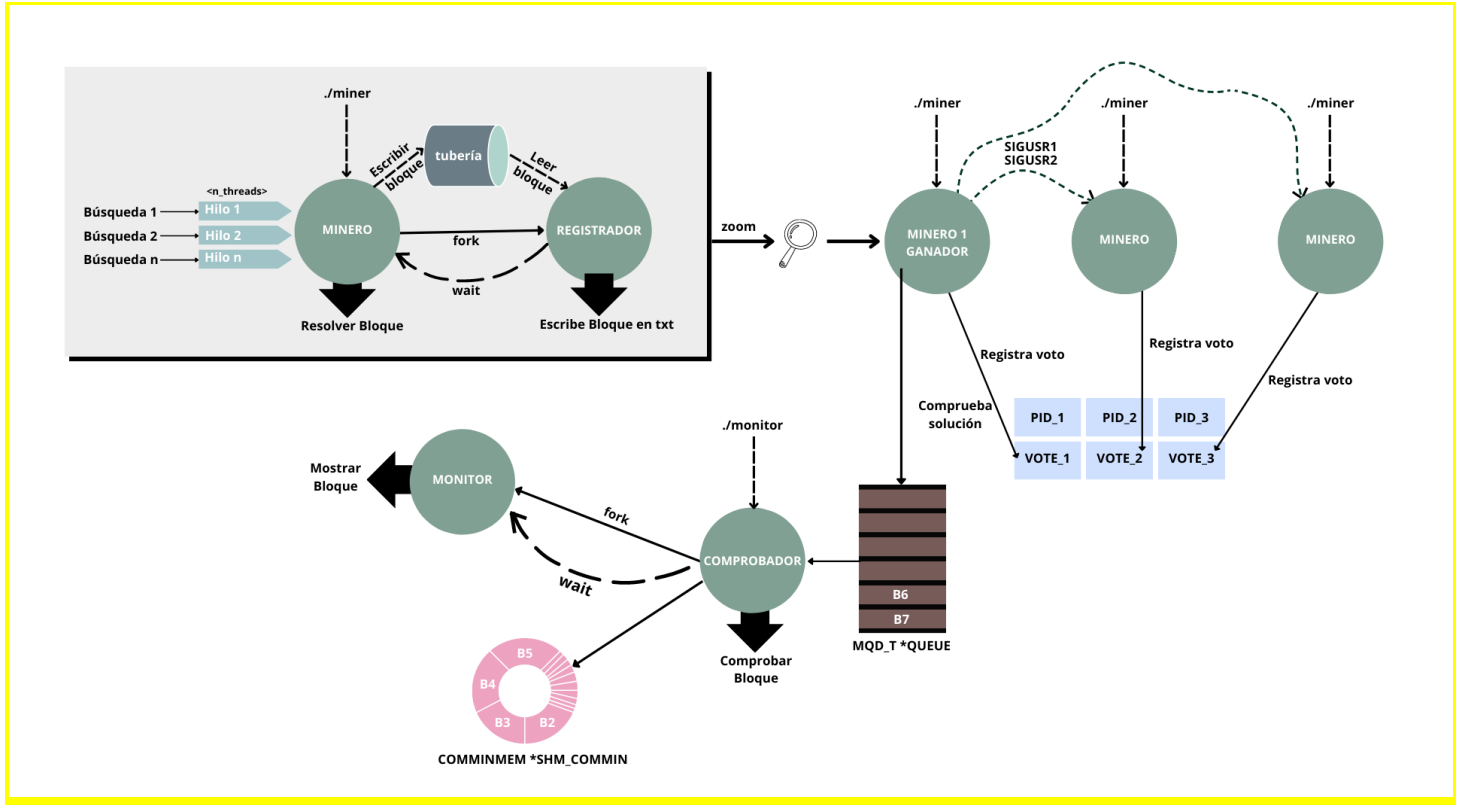
Escuela Politécnica Superior, UAM, 2022-2023

Elena Balseiro García & Paula Beltrán Marianini
Grupo 2291, Pareja 3

Autores: Elena Balseiro y Paula Beltrán

Memoria

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:



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.

minero.c:
Este módulo se encarga de todo lo referente al minero.
En la librería *minero.h* podemos encontrar cuatro estructuras: *MinerWallet*, que representa la cartera del minero, *Block*, que representa el bloque de cada ronda, *SysMemory*, la memoria compartida entre todos los procesos y *WorkerInfo*, usada por los hilos del minero.

Al ejecutar ./miner se introducen dos argumentos con el número de segundos y de hilos con el siguiente formato: ./miner <N_SECONDS> <N_THREADS>

n_seconds representan el tiempo máximo que estarán trabajando los mineros, se fija una alarma que se gestiona con un manejador de forma que cuando se alcance dicho tiempo, el programa parará.
Por otro lado, *n_threads* representa el número de hilos con los que trabajará el minero lanzado.

La estructura funcional es la siguiente:

-Se inicializa la tubería por la cual se comunicarán minero y registrador.

-Se inicializa la cola por la cual se comunicarán minero ganador y comprobador.

*Hemos creado un semáforo con nombre *semActivo* para que el minero no envíe nada si el monitor no está activo. Por eso, se inicializa a 0 y según se lanza el monitor, lo pone a 1 y al acabar lo vuelve a poner a 0.

-Se hace fork(), el padre representa al minero y el hijo el registrador.

.Padre (minero): Se divide en dos partes, en una de ellas sólo entrará el primer minero del sistema (será el que inicializa la memoria compartida) y a continuación actúa como el resto.
El resto de mineros abren y linkean la memoria compartida. Una vez entran en el sistema aumenta *currentSysMiners*, a continuación se registran en la memoria compartida (sincronización gestionada mediante semáforo *semMutex*) y proceden a minar. El minado se realiza en un bucle indefinido (que solo se corta con señales) y no empiezan a minar hasta que reciban la señal SIGUSR1 del ganador de la ronda anterior (o primer minero si es la primera ronda).

Tras recibir la señal, se incrementa *playingMiners* y se lanzan *n_threads* hilos que llaman a la función *work*, la cual busca soluciones para la target guardada en *InfoWorker.* Esta función calcula el rango asignado para cada hilo en función de su id y se hace *pow_hash* buscando la solución. En caso de encontrarla, se guarda en una variable global la posible solución y se fija a 1 la flag global *solfound* para indicar que se ha encontrado una solución.
*Hacemos diferenciación entre playingMiners y currentSysMiners ya que le enviaremos la señal para empezar a votar a *currentSysMiners* mineros pero solo esperamos que voten *playingMiners.*

Es posible que varios mineros encuentren una solución prácticamente a la vez pero solo el primero iniciará la votación. Es por ello que comprobamos que *solfound* sea 1 y *flagvotacion* esté a 0 para asegurarnos de que no hay un proceso de votación ya corriendo. Cuando esto sucede, el primer minero envía SIGUSR2 al resto para indicarles que paren de minar. Después, entran en la votación.

En el proceso de votación, el ganador vota siempre positivo y el resto lo vuelve a comprobar haciendo *pow_hash* con la posible solución. Los votos se registran en la memoria compartida. El ganador comprueba los votos cuando todos han votado y hace recuento. En caso de ser válido, actualiza la cartera y por tanto, el bloque. Después, envía el bloque al proceso hijo. También resetea los valores necesarios para la siguiente ronda.

.Hijo (registrador): El registrador, también en un bucle indefinido, lee la tubería que comunica con el padre, extrae el bloque de cada ronda en la que participa y registra en un fichero txt (cuyo nombre depende del pid del padre) toda la información del bloque.

comprobador.c:

El comprobador realiza la función del monitor, la cual se ejecuta con ./monitor. La estructura es la siguiente:

-Se hace post de *semActivo* para indicar que está preparado para leer la memoria compartida con minero.

-Se hace fork(), donde el padre es el comprobador y el hijo el monitor.

.Padre(comprobador): Recibe de la cola el bloque que le pasa el minero y vuelve a hacer pow_hash para comprobar si es válida. En ese caso, se fija correct=1 en el bloque recibido y se envía este bloque al hijo mediante la memoria circular compartida *shm_commin*

.Hijo(monitor): Lee *shm_commin*, comprueba el la flag correct e imprime por terminal toda la información del bloque.

*La estructura de esta memoria compartida *CommінMem* contiene 5 bloques, un índice que indica donde se encuentra el último guardado (la extracción se gestiona como un pop-front) y tres semáforos *semMutex*, *semFill* y *semEmpty* para aplicar el esquema productor-consumidor.

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.

Requisitos	Satisfecho
Funcionamiento general del proceso Minero	X
Funcionamiento general del proceso Registrador	X
Funcionamiento general del proceso Comprobador	X
Funcionamiento general del proceso Monitor	X
Gestión correcta de las señales y mecanismos de sincronización	*
Gestión correcta de la memoria compartida	X
Gestión correcta de las tuberías	X
Acceso concurrente correcto a la memoria compartida	X
Gestión correcta de la cola de mensajes	X

*El programa cumple con todos los requisitos a excepción del manejo de las señales de finalización. Durante el desarrollo de la práctica hemos tenido varios problemas a la hora de recibir señales para cortarlo y por tanto, ejecutar sus manejadores.
Es por esto que, a pesar de que la funcionalidad general cumple con lo esperado, el programa no puede finalizar como deseamos, y por tanto liberar los recursos correctamente. Sí que se para pero no se ejecuta correctamente el manejador que hemos establecido. Es importante acceder a dev/shm y borrar todo lo que quede abierto en caso de querer ejecutar de nuevo el programa.