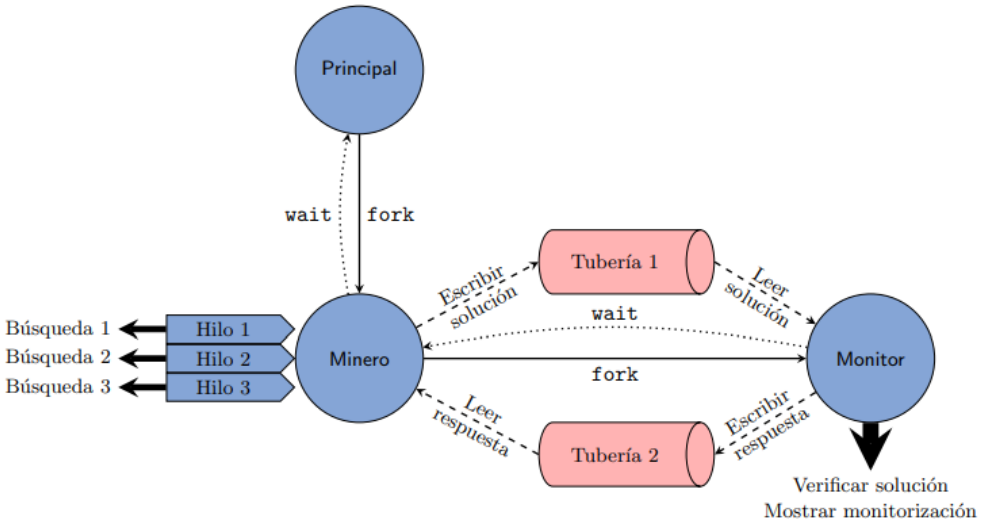
David Losada García
Luis Ignacio Pastor
david.losada@estudiante.uam.es
luisi.pastor@estudiante.uam.es

PRÁCTICA 1 SISTEMAS OPERATIVOS

FECHA: 25/02/2024

SINOPSIS DEL PROYECTO

La práctica tiene como objetivo la codificación en C de un programa que implemente un minero que resuelva un numero dado de POW (Proof Of Work), empleando para ello un número determinado de hilos. Se facilita un esquema global del sistema a diseñar para facilitar la comprensión.



PLANTEAMIENTO Y CODIFICACIÓN

A continuación, se explicará cómo funciona el código diseñado de una forma general. Se recomienda contar con el código abierto al mismo tiempo ya que se irá haciendo referencia al mismo y a funciones implementadas en este.

Para el desarrollo del proyecto hemos seguido el esquema mostrado anteriormente, el cual fue proporcionado en el enunciado de la práctica.

Comenzamos codificando principal, en nuestro caso el main. Sabemos que a través de terminal se nos pasarán los parámetros requeridos para el ejercicio, por lo que los guardamos en las variables de nombre “target”, “rounds” y “n_threads”. Estas corresponden al número a buscar, el número de rondas que se llevarán a cabo y el número de hilos que el minero podrá crear.

Tras esto, mediante un fork, creamos un proceso hijo dentro del cual llamaremos a la función “mineria”, a la cual le pasamos los parametros anteriormente mencionados, que se encargará de la búcqueda de los valores deseados. El proceso padre esperará a que termine el hijo e imprimirá por pantalla el estado con el que terminó el proceso hijo.

Ahora vamos a centrarnos en la codificación de minería. Comenzamos abriendo los dos pipes que se requieren en la implementación de esta práctica: uno que permita mandar a

monitor los datos necesarios desde minería (“send”) y otro que permita el proceso contrario (“recieved”).

Proseguimos con la creación del monitor, que como podemos ver en el esquema requiere de un proceso en nuestro minero. En el proceso hijo cerramos mediante la función “close_better” los extremos de los pipes que no se van a necesitar para la escritura/lectura (se cierran “send[1]” y “received[0]”), y procede a llamarse a la función “monitor”, al que le pasamos los file descriptors de los archivos en los que tiene que leer y escribir.

En el proceso padre, cerramos los ficheros de los pipes que no usaremos (“send[0]” y “received[1]”). Proseguimos reservando memoria para n_threads “miners”, variables de tipo “t_miner” que posteriormente inicializaremos.

Continuamos con un bucle que se ejecutará tantas veces como rondas se hayan determinado desde la terminal. En este bucle se encarga de hallar los resultados esperados con una sucesión de órdenes, que se mencionan a continuación.

Se realiza una llamada a “init_miners” para inicializar los parámetros de la estructura “miners” para la que hemos reservado memoria previamente. Los parámetros que se inicializan son los siguientes:

- **target**: se asigna el valor de target elegido
- **start**: se le asigna al minero el valor desde el que tiene que empezar a buscar;
- **end**: se le asigna al minero el valor máximo que ha de buscar;
- **result**: variable que usaremos para guardar el resultado, en caso en el que lo encuentre. Por defecto se pone a –1.
- **finish**: atributo que puede ser modificado por parte de todos los hilos para determinar si el target ya ha sido encontrado (y por tanto no es necesario que los hilos posteriores al que lo encontró se ejecuten). Comienza teniendo un valor de NOT_FOUND.

De esta forma, los hilos poseen toda la información que necesitan para poder trabajar.

Proseguimos ahora a crear cada uno de los hilos asignando un minero a cada uno y empleando la función “search”. Esta función se encargará de buscar el hash deseado. Cuando lo encuentra, modifica el valor de “finish” a FOUND para indicar al resto de hilos que dejen de buscar y guarda en el minero correspondiente el resultado en su variable “result”. También se pone “finish” a FOUND si ocurre algún error durante la creación de un hilo.

A continuación, esperamos mediante “pthread_join” a que terminen todos los hilos y usamos una función llamada “get_result” para acceder al resultado guardado por uno de los hilos.

Posteriormente, ya con el valor obtenido de los hilos, llamamos a la función “send_request”. Esta función se encarga de escribir en el archivo pasado como argumento tanto el “target” buscado como el resultado encontrado por el minero, para que el monitor pueda compararlas y verificar si es correcta o no.

Por último, obtendremos desde minero la respuesta de monitor gracias a la función “recieve_request”, que leerá la información escrita por monitor. Si la información es

correcta se imprimirá la solución por pantalla; si por el contrario resultado y “target” son distintos también se imprimirá por pantalla.

Todo este bucle se repetirá tantas veces como “rounds” se hallan pasado como argumento al programa.

Para finalizar en minero, se llama a la función “miner_exit”, encargada de liberar la memoria reservada para los mineros y cerrar los pipes. Esta función también es usada con anterior cuando ocurría algún tipo de error durante la ejecución del programa, cambiando en estos casos el parámetro “status” que se le pasa a la función por el adecuado al error. Antes de finalizar el proceso, se encarga de esperar a su hijo “monitor”, que se termina al cerrar los pipes desde el padre.

Para acabar con la explicación del código vamos a fijarnos por último en el monitor.

Su función principal es “monitor”, la cual es llamada desde el proceso hijo creado en minería. En ella se ejecutará un bucle “while” hasta que se detecte que se ha cerrado la pipe “send”. Dentro del mismo se leerá el contenido escrito en el pipe “send” desde minería, correspondientes al “target” buscado y al resultado que se cree correcto. Se escribe el resultado de la comparación en el pipe mediante la función “write_test” y se llama a la función “monitor_exit” para cerrar ambos pipes.

DETALLES

Sobre el resultado del proyecto, podemos destacar algunas cosas:

- Hemos evitado el usar variables globales aunque tuviéramos que usar mas funciones. No nos parece que usarlas sea una buen práctica, asi que hemos preferido añadir parámetros conjuntos a todos los mineros, para que todos posean la misma flag para terminar.
- Todas las funciones han sido protegidas: “*fork”, “pipe”, “malloc”, “write”, “read”, “pthread_create”*... En caso en el que alguna de estas funciones fallen, el proceso saldrá con código “*PCR_UNEXPECTED*” y se mostrará un mensaje determinado.
- Somos conscientes de que no buena práctica crear y eliminar hilos en cada iteración de rounds; no obstante, por falta de tiempo hemos decidido dejarlo como está en el producto final. Aunque no sea lo más eficiente posible, tampoco creemos que tenga mucho impacto, puesto que estamos reusando los recursos que se van liberando, así que en realidad el único problema que hay es el número de llamadas al sistema y el tiempo que tarden “*pthread_create*” y “p*thread_join*” en ejecutarse
- Hemos creado varias funciones genéricas y utilidades para evitar la repetición de código. Sobre todo, queremos destacar las funciones para liberar memoria en cada proceso, la función para imprimir el estado de un hijo en función del valor devuelto y las funciones de cerrar pipes.