Sistemas Operativos II

Práctica 4

Daniel Rivero
Dennis Quitaquís

Introducción

El objetivo de esta práctica es, basándonos en el código implementado en la práctica 3, conseguir un nivel de coordinación a la hora de crear el árbol más complejo.

Para ello utilizaremos las funciones *pthread_cond_wait*, *pthread_cond_signal* y *pthread_cond_broadcast* que nos servirán para trabajar de manera más óptima con los hilos encargados de crear el árbol, ya que en esta práctica no se crearán cuando se seleccione crear el árbol y después se eliminarán, sino que los hilos están creados desde que se inicia el programa, se despertarán cuando tengan que utilizarse y se finalizarán cuando se salga de la aplicación.

Memoria de la práctica

El flujo del programa en si es el mismo que en la anterior práctica, pero se ha mejorado la coordinación de los hilos encargados de crear el árbol tal y como explicaré a continuación.

Al iniciarse la aplicación, el hilo principal se encargará de crear los hilos que se utilizarán después para crear el árbol y, para asegurar que todos los hilos están dormidos antes de continuar con la ejecución del hilo principal y mostrar el menú, este se dormirá en la cola *cond_2* con lo cual soltará la clave del monitor que estaba usando para la creación de hilos.

En este punto, se empezarán a ejecutar los hilos secundarios y, usando la clave anterior, irán aumentando un contador y se irán durmiendo; cuando el contador iguale al número de hilos secundarios creados querrá decir que están todos los hilos secundarios dormidos y, por lo tanto, despertaré con un *signal* al hilo principal. Además, en el main haré otro lock/unlock con la misma clave que se encarga de dormir los hilos para asegurarme completamente de que no solo el contador ha igualado a la cantidad de hilos sino que efectivamente todos los hilos están dormidos y el último ha soltado la clave.

A continuación, al seleccionar la opción de crear el árbol el hilo principal leerá la primera línea del fichero, en la cual está el número de ficheros a procesar, se dormirá en la cola *cond_2* y se despertarán todos los hilos secundarios dormidos usando un *broadcast* para que se encarguen de crear el árbol de la misma forma que lo hacían en la práctica anterior.

Cuando se hayan analizado todos los ficheros, los hilos secundarios irán decrementando un contador y durmiendose nuevamente. Además, cuando el contador llegue a 0 querrá decir que todos los hilos secundarios han acabado su ejecución, con lo cual el último hilo antes de dormirse se encargará de despertar con un *signal* al hilo principal para que continúe su ejecución.

Por último, al seleccionar la opción de salir de la aplicación el hilo principal tendrá que finalizar los hilos secundarios antes de finalizar el. Para esto, se cambiará a 1 una variable global (*acaba*) y se despertarán todos los hilos secundarios con un *broadcast*, así los hilos al continuar su ejecución saldrán del bucle y acabarán su ejecución, mientras que el hilo principal esperará a que acaben con un *join*.

Una vez hayan finalizado todos los hilos secundarios, el hilo principal dejará de esperar y finalizará la aplicación.

Para comprobar el correcto funcionamiento de la creación del árbol usando múltiples hilos con las nuevas modificaciones de sincronización de hilos hemos ejecutado la aplicación con diversas cantidades de archivos y diferente número de hilos y ha funcionado correctamente, así como su ejecución con Valgrind que indica que no hay problemas de memoria sin liberar.

Como ejemplo, en la carpeta Proves hay un fichero log.txt con el log de la ejecución del programa (toda la impresión por pantalla que hace el programa según se ejecuta) y un fichero valgrind.txt con el análisis hecho por valgrind que demuestra que no hay memoria no liberada; estas pruebas se han hecho en un ordenador con 2 procesadores, utilizando 4 hilos y analizando 50 ficheros.