Hilos

Para la creación del ordenamiento por hilos hicimos uso de una estructura de árbol binario y sus propiedades, donde para determinar si un hilo es hoja o rama usamos las propiedades de árbol binario. Donde inicializamos dos estructuras globales, el arreglo que va a ser ordenado de manera que todos los hilos tengan acceso a este arreglo y por otro lado un arreglo de hilos, el cual está definido como un struct que contiene los índices iniciales y finales que deben ser ordenados por acá hilo e rama, el identificador único del hilo y el índice de los hijos.

Dentro del procedimiento principal del programa, en el main se encarga de inicializar todos los indices de ordenamiento de los hilos y cuales son sus respectivos hijos. Así mismo, en el main se determina si el nivel de profundidad es igual a 0, se aplica un *quicksort*() directamente sobre el arreglo, sino se hace el llamado a la función *rama*() el cual es un procedimiento que crea dos hilos usando la función *pthread_create*() y les informa que id del árbol contiene cada hilo. Esta llamada a función rama() después de crear los dos hilos hijos, espera que ellos terminen con *pthread_join*() para luego hacer merge de los dos arreglos. Cabe destacar que dado que el *pthread_join()* se encarga de esperar a sus hijos, no existe la necesidad de implementar mutex debido a que el comportamiento de árbol impide que existan secciones criticas en la corrida del programa.

Por consiguiente hace falta mencionar, que cuando en el procedimiento *rama*() los próximos hilos a crear son hojas, estos se crean haciendo llamada a la función *quicksort*() así mismo también se le pasa como argumento el id del árbol de cada hilo. Dentro de cada hilo hijo, no existe dependencia de datos debido a que cada uno ordena conjuntos disjuntos de indices del arreglo global.

Por ultimo cabe destacar, que el uso de variables globales y el comportamiento de los hilos como un árbol binario facilito el entendimiento del uso de memoria.

Procesos

Para la ordenación de enteros con procesos, el proceso raíz toma los argumentos de entrada y da inicio a la creación de todo el árbol de procesos, donde cada nodo hace el llamado *fork*() exactamente dos veces. Mientras el nivel de un nodo sea menor al numero máximo de niveles, este vuelve a ejecutar el mismo código de la raíz para crear los nuevos dos procesos hijos, utilizando la función *goto*() la cual facilita dar saltos a una linea especifica del código.

Una vez que se llegan a las hojas del árbol se hace la llamada de *execv*(), al cual se le da como parámetros el path del ejecutable *hoja*, el cual se encarga de la lectura del archivo binario y el ordenamiento con *quickSort*() de su parte de la secuencia de enteros, y un arreglo de parámetros que son el nombre del ejecutable, el inicio y el fin de la sección a ordenar y el nombre del archivo binario. Después de haber ordenado su parte, los procesos hojas concluyen creando un archivo de texto cuyo nombre es su pid y el cual se usara para poder comunicarse con su proceso padre.

Los procesos ramas se encargan de hacer *wait*() y leer los archivos de texto creados por sus procesos hijos, los cuales contienen dos secciones del arreglo ya ordenado, para proceder unir ambas partes con la función *merge*() y escribir un nuevo archivo de texto que sera leído por su respectivo padre. De esta manera se llega nuevamente al proceso raíz, donde finalmente se crea un archivo de texto de salida final con la secuencia de enteros ordenados.

Comparación Hilos vs Procesos

Tiempo expresado en ms	Hilos	Procesos
Prueba para 32 Enteros con 5 niveles	2.625	0.819
	2.737	0.785
	3.086	0.722
	2.842	0.691
Prueba para 32768 Enteros con 5 niveles	26.675	12.147
	17.927	16.774
Prueba para 32768 Enteros con 9 niveles	66.938	18.018
	63.415	15.911

Cabe destacar que para esta comparación se utilizo la arquitectura Intel i7 con 2 cores, ademas el número máximo de niveles que se pudieron utilizar con procesos e hilos fueron 9 y 11 niveles respectivamente. Aunque desconocemos la razón verdadera, creemos que fue causa del sistema operativo donde se realizaron las pruebas (Mac OS X).

Es interesante observar estos resultados, donde el programa basado en procesos logra tiempos mucho menores donde quizás sea causado por la eficiencia del manejo de hilos por parte del sistema de operación. Cabe destacar que desconocemos las causas de porque este resultado fue de esta forma.