Facultad de Ingeniería
Departamento de Ingeniería de Sistemas

Sistemas Operativos

Proyecto 1
Sort Concurrente

Presentado por:

Juan Camilo Chafloque
Jorge Salgado
Cristóbal Castrillón

Presentado a:
Mariela Curiel Huerfano

Bogotá D.C
Septiembre 2019

1. Descripción del problema

El problema del proyecto era ordenar un conjunto de archivos en forma concurrente utilizando procesos e hilos. Con base en este problema se tenía que implementar una variación del comando *sort* de UNIX para generar dicho ordenamiento. Una variación basada en hilos y otra variación basada en procesos. Una tercera variación era utilizar el mismo comando de ordenamiento de UNIX con llamadas al sistema a la familia EXEC para así invocar el comando *sort.*

2. Algoritmos de ordenamiento utilizados

Para la implementación de los algoritmos se utilizaron dos tipos de algoritmos de ordenamiento. El primero fue el algoritmo Mergesort, que es un algoritmo "*Divide and Conquer*". Divide el array de entrada en dos mitades y utilizando recursión se llama a ella misma para ordenar cada mitad y luego combinar las dos mitades ordenadas.

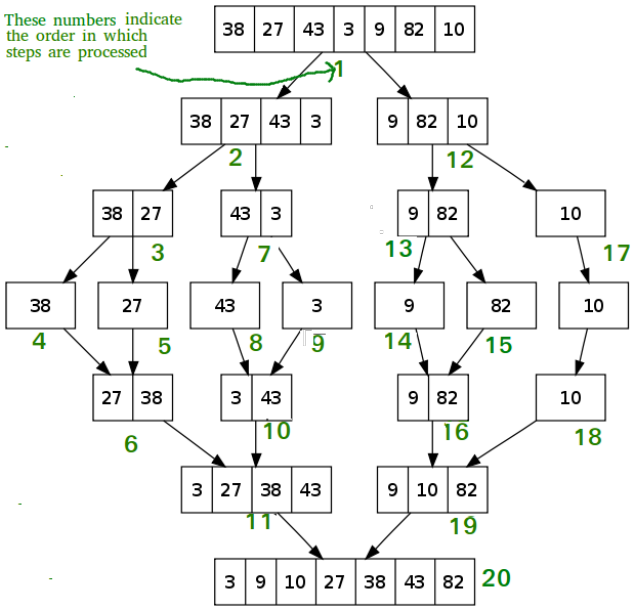

Figura 1. Diagrama del proceso del algoritmo mergesort para ordenar un conjunto de datos
Fuente: https://www.geeksforgeeks.org/merge-sort/

El otro algoritmo que se utilizó fue el algoritmo Quicksort, que al igual que el mergesort es un algoritmo "*Divide and Conquer*". Se elige un elemento como pivote, para luego dividir el arreglo alrededor del pivote seleccionado. El pivote seleccionado en la implementación fue la última posición del arreglo, como se muestra en la figura 2.

Para cada implementación (Procesos e hilos) se utilizaron ambos algoritmos. Para cada archivo que llegaba como parámetro al main se crea un proceso/hilo hijo y este ordena el archivo con el algoritmo Quicksort y guarda los datos ordenados en un archivo temporal. Luego de que todos los hijos terminen, el padre obtiene todos los archivos temporales y genera un archivo de salida final con todos los datos ordenados utilizando el algoritmo Mergesort.

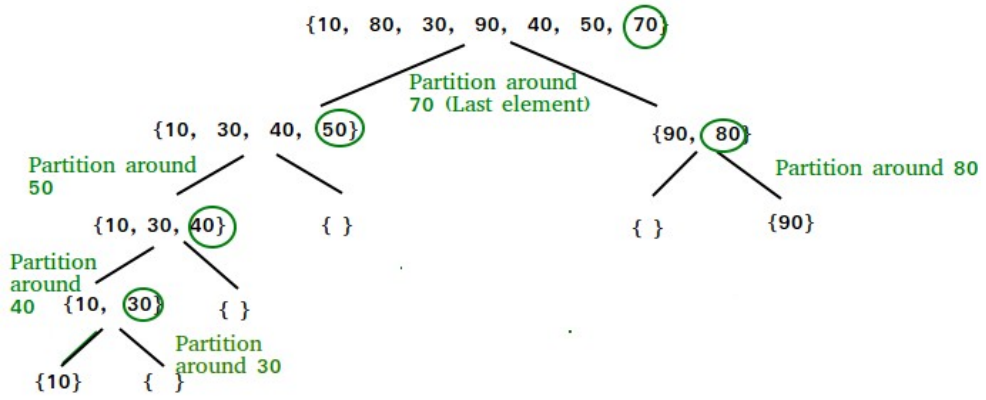

Figura 2. Diagrama del proceso del algoritmo quicksort para ordenar un conjunto de datos
Fuente: https://www.geeksforgeeks.org/quick-sort/

3. Llamada EXEC

Para la tercera implementación se necesitaba utilizar la familia de llamadas al sistema EXEC para poder utilizar el comando *sort* del sistema operativo. Con base en lo anterior, se utilizó el comando *execvp*. Dicho comando recibe como primer parámetro un *file*. Este file contiene el ejecutable del comando, que en este caso es el *sort*. Luego siguen N parámetros que indican una lista de argumentos disponibles para el comando a ejecutar. Por último, se debe terminar la lista con un puntero nulo.

Con el comando man se buscó en la terminal argumentos pertinentes del comando *sort* para poder realizar el ordenamiento de manera correcta. Los argumentos que se utilizaron fueron los siguientes:

- -n (Numeric sort): Compara de acuerdo a un valor numérico.
- -k (Key): Ordena el archivo dependiendo de un key que sería la ubicación (columna) a ordenar.
- -r (Reverse): En caso de que el usuario lo desee, el ordenamiento se hará de manera inversa.

El comando del execvp en el codigo en C fue el siguiente para ordenar normal:

execvp("sort", input, "-nk", "4,4", "-k", "5,5", "-k", "6,6" , NULL)

y el comando para ordenar de manera inversa fue:

execvp("sort", input, "-nrk", "4,4", "-rk", "5,5", "-rk", "6,6" , NULL)

En donde primero se especifica el *file* donde se encuentra el ejecutable del *sort* y luego utilizar los parámetros mencionados anteriormente para generar el ordenamiento de manera correcta. Como se puede ver el primer *key* de ordenamiento es la columna 4 y los valores son de tipo numérico, luego sigue la columna 5 y por ultimo la 6, luego se pasa el archivo que llega por parámetro al main que es el archivo a ordenar. El comando *execvp* se utilizó tanto para los ordenamientos de los hijos como el ordenamiento del padre.

4. Resultados y tablas de datos

Todas las pruebas de las medidas de rendimiento se hicieron en los computadores de la sala de Bases de Datos. El computador que fue utilizado tiene las siguientes características:

Con el comando ***uname*** se obtuvieron datos del sistema operativo del computador. El tipo de procesador del computador es x86_64. El sistema operativo es GNU/Linux.

Con el comando ***sudo lshw*** se consultaron especificaciones del computador utilizado. Para la CPU se obtuvo la siguiente información:

- Producto: Intel Core i7-6700T CPU @ 2.80GHz
- Fabricante: Intel Corp.
- Id fisico: 4b
- Versión: Intel Core i7-6700T CPU @ 2.80GHz
- Ranura: U3E1
- Tamaño: 2371 MHz
- Capacidad: 4200 MHz
- Anchura: 64 bits
- Reloj: 100 MHz

Time real (s)			
Tamaño del Log	csortp	csorth	csortpexec
1000	0,039	0,034	0,013
11000	0,532	0,556	0,088
20000	0,367	0,481	0,158

Time User (s)			
Tamaño del Log	csortp	csorth	csortpexec
1000	0,012	0,009	0,023
11000	0,51	0,508	0,143
20000	0,377	0,373	0,24

Time sys (s)			
Tamaño del Log	csortp	csorth	csortpexec
1000	0,031	0,025	0,002
11000	0,068	0,048	0,004
20000	0,128	0,108	0,021

Imagen 1. Tiempos de ejecución de cada comando para cada log.

Tiempo de respuesta			
Tamaño del Log	csortp	csorth	csortpexec
1000	0,039	0,034	0,013
11000	0,532	0,556	0,088
20000	0,367	0,481	0,158

Tiempo de CPU			
Tamaño del Log	csortp	csorth	csortpexec
1000	0,043	0,034	0,025
11000	0,578	0,556	0,147
20000	0,505	0,481	0,261

Imagen 2. Tiempos de respuesta y de CPU de cada comando para cada log

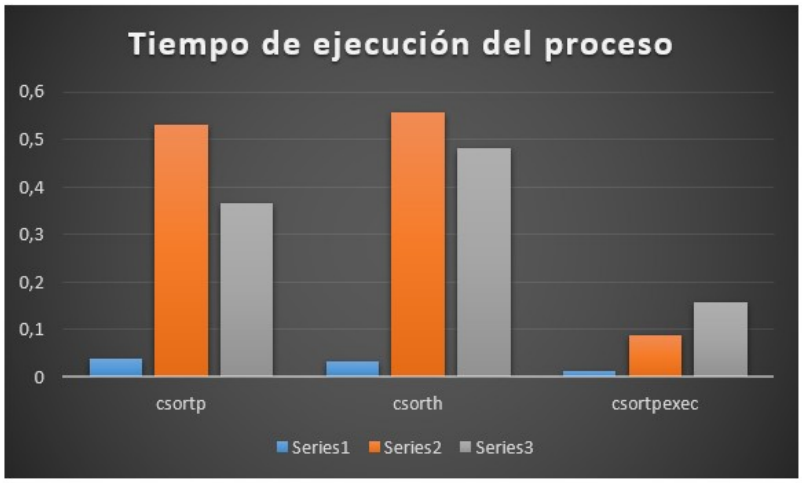

Grafico 1. Tiempos de ejecución de cada comando para cada log

Debido a que los comandos ejecutaban de manera muy rápida, la utilización del CPU no se podía calcular fácilmente. Para arreglar este problema, se utilizó la función de C sleep. Para demorar la ejecución del proceso y hacer que la medida y toma de datos sea más fácil.

Para calcular la utilización del CPU se utilizó el comando de Linux “iostat 2”. Esto permitía ver información del uso del disco. El 2 significa que se iban a mostrar estadísticas de disco recopiladas cada 2 segundos. Este comando se corrió paralelamente a los comandos de ordenamiento y luego se tomaba la utilización del CPU más alta como medida final del comando.

Utilización del CPU (%)			
Tamaño del Log	csortp	csorth	csortpexec
1000	10,93	7,61	7,31
11000	13,32	11,42	8,79
20000	11,42	27,1	9,92

Imagen 3. Utilización del CPU para cada comando y para cada log

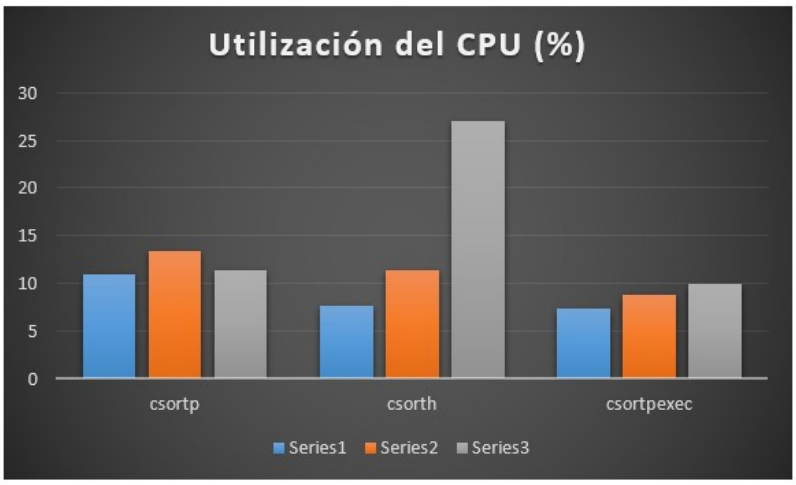

Grafico 2. Utilización del CPU de cada comando para cada log

5. Análisis de resultados

En conclusión, se puede ver que el comando csortpexec es el que mejores tiempos tiempo, haciéndolo el comando más eficiente para realizar los ordenamientos de los archivos. Además, en los resultados obtenidos en las tablas y en el grafico se puede observar que para los comandos csortp y csorth los tiempos para ordenar los logs de 11000 líneas son mayores a los tiempos para ordenar los archivos de 20000 líneas. Esto se puede deber a que los archivos de 20000 líneas tienen una mayor cantidad de datos iguales o inicialmente el archivo puede estar más ordenado que los archivos de 11000 líneas.

Con respecto a los comandos csortp y csorth, los hilos, en promedio se demoraron un poco más de tiempo que los procesos, aunque debió haber sido de la forma contraria ya que según la teoría un hilo tarda menos tiempo en ser creado que un proceso y un hilo tarda menos tiempo en terminar que un proceso, ya que los hilos, al compartir memoria y recursos, se pueden comunicar entre sí sin invocar el núcleo del sistema operativo. Sin embargo los tiempos no varían en gran cantidad y se puede concluir que los algoritmos de ordenamiento quicksort y mergesort son eficientes a la hora de ordenar archivos con grandes cantidades de lineas, al contrario de algoritmos como el bubblesort.

Finalmente se puede ver de la imagen 3 y el grafico 2, que el comando csortpexec volvió a superar a los otros dos comandos, esta vez en la utilización del CPU. El mejor funcionamiento de este comando en comparación a los otros puede deberse a que el comando sort de Linux es exclusivo para ordenar archivos por lo que el algoritmo de ordenamiento que utiliza internamente para ordenar puede ser más eficiente que el quicksort y mergesort.