Trabajo práctico 0: Infraestructura básica

Alejandro García Marra, *Padrón Nro. 91.516*
alemarra@gmail.com
Sebastián Javier Bogado, *Padrón Nro. 91.707*
sebastian.j.bogado@gmail.com
Grupo Nro. 0 - 2do. Cuatrimestre de 2012
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

El presente trabajo busca crear un programa que permita el ordenamiento de archivos a través de dos implementaciones distintas, una utilizando el algoritmo Quicksort y la otra el algoritmo Stooge sort.
Sobre este programa, luego, se realizarán una serie de mediciones con el fin de determinar los desempeños relativos de cada implementación y las posibles mejoras a realizar. Para esto haremos uso de los programas **time** y **gprof**.

1. Introducción

Muchas veces tanto para programas recién terminados, como para aquellos que llevan un tiempo en funcionamiento, se desconoce realmente qué partes del programa insumen la mayor cantidad de recursos, sean estos de tiempo, carga de cpu, etc. Poseer esta información se torna en algo crítico cuando se busca realizar una mejora de performance en dicho programa. Sería poco útil intentar optimizar a ciegas, por no decir inútil.
Haremos uso entonces de dos herramientas distintas, el profiling del código (por medio de *gprof*) y la medición de los tiempos de ejecución (por medio de *time*).

- **Profiling**: El profiling permite aprender donde el programa pasa la mayor parte de su tiempo, y cuales funciones llaman a otras mientras se ejecuta. Esta informacion puede mostrar qué piezas del programa son mas lentas de lo esperado, convirtiéndolas en candidatas para su reescritura en la etapa de optimización.
  También puede ayudarnos a descubrir cuales funciones son llamadas más o menos veces delo esperado, pudiendo así encontrar nuevos bugs (aunque el descubrimiento de bugs no es el fin principal de esta etapa)

  El profiler utiliza información recolectada en tiempo de ejecución, por lo que puede ser utilizado en programas demasiado grandes o complejos, donde un análisis por lectura de fuentes sería impracticable.
  Como consecuencia del análisis durante la ejecución, los datos con los que se corra el programa afectaran el resultado del profiler. Es decir, distintos datos de entrada pueden provocar distintas ramas de ejecución, dando po resultado que, por ejemplo, no se llamen algunas funciones.

- **Medición de Tiempos**: Permite conocer con precisión los tiempos de ejecución de un programa, discriminados entre tiempos de systema, de usario, tiempos totales, etc., así como también conocer los porcentajes para cada parte del programa, cantidad de entradas, y muchas otras opciones. La combinación con una herramienta de profiling permite exactitud a la hora de conocer la forma en que se ejecuta el programa bajo estudio, permitiendo optimizar únicamente las partes críticas del ciclo de ejecución.

2. Mediciones

2.1. Valores Obtenidos

En la tabla 1 se presentan las mediciones realizadas con **time** sobre ambos algoritmos de ordenamiento y con archivos de distintos tamaños.

Además de los archivos indicados en el enunciado, fueron agregadas mediciones sobre archivos con tamaños arbitrarios, mayores, con el fin de mostrar de mejor manera las diferencias entre algoritmos.

		Quicksort			Stooge sort		
		Ordenado	Invertido	Aleatorio	Ordenado	Invertido	Aleatorio
1kb	real*	0.00	0.00	0.00	0.00	0.00	0.00
	user*	0.00	0.00	0.00	0.00	0.00	0.00
	sys*	0.00	0.00	0.00	0.00	0.00	0.00
8kb	real	0.00	0.00	0.00	0.02	0.02	0.01
	user	0.00	0.00	0.00	0.01	0.01	0.01
	sys	0.00	0.00	0.00	0.00	0.00	0.00
16kb	real	0.00	0.00	0.00	0.00	0.02	0.02
	user	0.00	0.00	0.00	0.00	0.01	0.02
	sys	0.00	0.00	0.00	0.00	0.00	0.00
32kb	real	0.00	0.00	0.00	0.17	0.17	0.17
	user	0.00	0.00	0.00	0.17	0.17	0.17
	sys	0.00	0.00	0.00	0.00	0.00	0.00
64kb	real	0.00	0.00	0.00	1.44	1.44	1.44
	user	0.00	0.00	0.00	1.44	1.43	1.44
	sys	0.00	0.00	0.00	0.00	0.00	0.00
1024kb	real	0.04	0.03	0.03	>1500	>1500	>1500
	user	0.03	0.02	0.03	>1500	>1500	>1500
	sys	0.00	0.00	0.00	0.00	0.00	0.00

Cuadro 1: Resultados comando Time

* Referencia:

- real: %e, tiempo total real usado por el proceso.
- user: %U, total de segundos-CPU usados por el proceso directamente.
- sys : %S, total de segundos-CPU utilizados por el systema en nombre del proceso.

2.2. Análisis de los datos

La marcada diferencia entre la complejidad de los algoritmos se refleja en muestras tan chicas como la de 8kb. A partir de ahí, el Stooge sort ya hace suficiente uso del procesador como para ser notado por time, mientras que el Quicksort hace lo propio recién en la muestra más grande, de 1024kb. En este caso, el Stooge sort se tornó intolerable.

En la figura 1 se muestra el gráfico del tiempo insumido por el Quicksort para las distintas muestras.
En la figura 2 se muestra el gráfico del tiempo insumido por el Stooge sort para las distintas muestras, a excepción de la de 1024kb, porque demanda una escala que haría inapreciable la situación de las otras muestras.

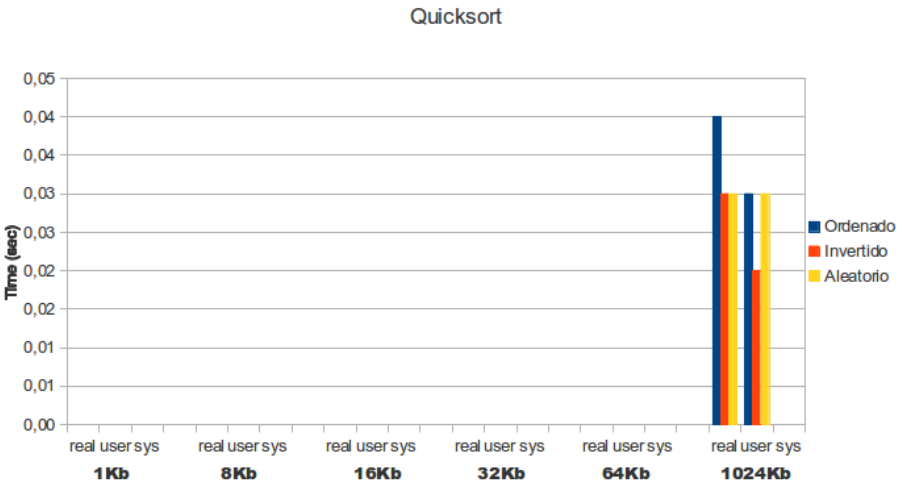

Figura 1: Tiempo tomado para distintas muestras del Quicksort.

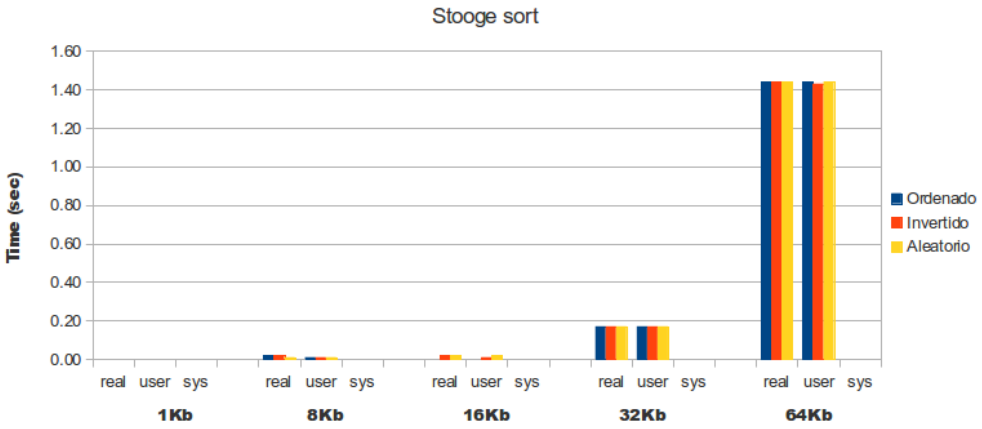

Figura 2: Tiempo tomado para distintas muestras del Stooge sort.

Gráfico de speedup, quick vs stooge

3. Profiling

Para realizar el profiling tomamos una muestra suficientemente grande como para que las funciones mismas de **gprof** tuvieran una incidencia despreciable en la prueba. En la figura 3 se muestra un ejemplo de cómo presentar las ilustraciones del informe.

Figura 3: Facultad de Ingeniería – Universidad de Buenos Aires.

4. Comandos de ejecución y corridas de prueba

Comandos de Compilación:

- make all:
- make test:
- make debug:
- make asm:
- make gprof:
- make clean:

Comandos de Ejecución:

- tp0 [OPTIONS] [file...]
- <stdout> | tp0 [OPTIONS]
- -h, –help
- -V, –version
- -q, –quicksort
- -s, –stoogesort

5. Conclusiones

Se presentó un modelo para que los alumnos puedan tomar como referencia en la redacción de sus informes de trabajos prácticos.

Referencias

[1] Intel Technology & Research, "Hyper-Threading Technology," 2006, http://www.intel.com/technology/hyperthread/.

[2] J. L. Hennessy and D. A. Patterson, "Computer Architecture. A Quantitative Approach," 3ra Edición, Morgan Kaufmann Publishers, 2000.

[3] J. Larus and T. Ball, "Rewriting Executable Files to Mesure Program Behavior," Tech. Report 1083, Univ. of Wisconsin, 1992.