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 del algoritmo Stooge sort, aproximando a un comportamiento minimalista del comando *sort*. Se comparará la versión implementada en C contra la misma función escrita en assembly

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*).

1.1. Stooge sort

El Stooge sort es un algoritmo de ordenamiento recursivo muy ineficiente, de complejidad $1\left(n^{\log 3/\log 1{,}5}\right)$.
Realiza el intercambio entre el primer y último de los elementos si es el primero es mayor (en ordenamiento ascendente). Luego, si el conjunto está compuesto por al menos tres valores, entonces aplica Stooge sort a los primeros dos tercios, luego a los dos últimos tercios, y finalmente repite con los primeros dos tercios.

2. Flujo del programa

El programa comienza verificando el primer parámetro recibido. Si no existe o no es reconocido, sucede el comportamiento por defecto, que es el mensaje de ayuda, información sobre el uso. Con esto, termina la ejecución. Otro camino rápido en la aplicación es al solicitar la versión, con el parámetro `-V` o `--version`.

En una ejecución normal, el primer parámetro comienza la lista de archivos a procesar.

Una vez en la función `sort`, se arma el conjunto de líneas a ser reordeanadas recolectándolas de a un archivo por vez. Si no hubiera archivos, se trabaja con la entrada estándar con la misma función (`parseLineas`, que se detallará en breve).

Armado el conjunto, se llama a la función stoogesort pasándole como parámetros el vector de líneas y la cantidad de elementos. Luego se imprimen por salda estándar y se libera la memoria reservada.

La lectura de cada archivo sucede en la función
`unsigned parseLineas(char** *pLinea, unsigned lineas, FILE* stream)`:

- `pLinea`: puntero a un arreglo de punteros a char
- `lineas`: cantidad de elementos del arreglo
- `stream`: archivo desde donde se llevarán a cabo las lecturas
- `return`: cantidad de elementos en el arreglo después del proceso

Esta función carga en un buffer el archivo y lo recorre en busca del caracter de fin de línea. Por cada uno encontrado, se aloca espacio para un string, y se agrega al vector. Si el archivo no termina naturalmente como una línea, se agrega

al final el `EOL`. Retorna con la cantidad de líneas cargadas (las acumuladas y las nuevas del archivo actual).

3. Mediciones

3.1. Valores Obtenidos

En la tabla 2 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.

		Stoogesort.c*			Stoogesort.S		
		Ordenado	Invertido	Aleatorio	Ordenado	Invertido	Aleatorio
1kb	real*	0.03	0.01	0.03	0.03	0.01	0.01
	user*	0.04	0.02	0.05	0.05	0.00	0.00
	sys*	0.00	0.00	0.00	0.00	0.03	0.02
8kb	real	0.38	0.38	0.40	0.38	0.36	0.38
	user	0.38	0.39	0.40	0.36	0.38	0.38
	sys	0.00	0.00	0.01	0.01	0.00	0.01
16kb	real	0.35	0.34	0.34	0.33	0.34	0.33
	user	0.33	0.36	0.35	0.32	0.34	0.34
	sys	0.01	0.00	0.00	0.02	0.00	0.00
32kb	real	10.26	10.26	10.21	9.89	9.86	9.88
	user	10.26	10.23	10.22	9.89	9.88	9.86
	sys	0.00	0.05	0.00	0.01	0.00	0.01
64kb	real	96.89	97.09	96.66	93.24	93.34	93.29
	user	96.88	97.00	96.63	93.24	93.31	93.29
	sys	0.01	0.03	0.02	0.00	0.00	0.00

Cuadro 1: Resultados comando Time

* Referencias:

Ambos fuentes utilizan la versión hecha en assembly del comparador de lineas, *strcmp.S*

- 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.

3.2. Análisis de los datos

Como era de esperarse, se aprecia una leve mejora en los tiempos de ejecución para la rutina implementada en assembly. La diferencia no es notoria ni importante, así como tampoco se logran los tiempos mínimos necesarios para que estos algoritmos puedan usarse en forma cotidiana y confiable (la naturaleza poco eficiente del algoritmo es un defecto insalvable).

3.3. Diagramas de Stack

Para las funciones:

- void stoogesort(void* array, unsigned int i, unsigned int j);

  Realiza llamados recursivos, así como también llamados a strcmp.

- int strcmp(char** array1, char** array2);

  Función hoja

Función	Stack
Stoogesort	$ra
	$fp
	$gp
	$s6
	$s5
	$s4
	$s3
	$s2
	$s1
	$s0
	padding
	Argumento 2
	Argumento 1
	Argumento 0
strcmp	$fp
	$gp

Cuadro 2: Resultados comando Time

4. Comandos de ejecución y corridas de prueba

A menos que se indique lo contrario, siempre se compilan los fuentes *.S directamente.

Comandos de Compilación:

- make all: genera el programa, modo release"
- make debug: genera el programa con flags para debugging
- make cdebug: idem, pero con el fuente stoogesort.c (Ejecutable tp1_c)
- make asm: genera los archivos Assemblies, sin generar el programa
- make gprof: genera el programa con flags para **gprof**
- make clean: remueve los archivos generados

Comandos de Ejecución:

- tp1 [OPTIONS] [file...]
- <stdout> | tp1 [OPTIONS]
- -h, –help
- -V, –version

Archivos de prueba presentes en el cd:

- 1kb ordenado, invertido, random
- 8kb ordenado, invertido, random
- 16kb ordenado, invertido, random
- 32kb ordenado, invertido, random
- 64kb ordenado, invertido, random

Archivos de fuente presentes en el cd (ver Anexo I):

- Makefile
- main.c
- parseManager.c
- parseManager.h
- sort.c
- sort.h
- stoogesort.S
- stoogesort.c
- stoogesort.h
- strcmp.S

5. Conclusiones

Con la realización de este tp, pudimos interiorizarnos en parte del trabajo realizado por el compilador, al reemplazar el código assembly (.s) generado por este a partir de un fuente (.c) por código propio.

Uno de los objetivos era obtener una mejor performance de la rutina, ya que el código generado por el compilador en muchos casos no es el más eficiente posible, agregando instrucciones innecesarias en la traducción. Este objetivo se vió completado, como muestran las mediciones, pero sin lograr un impacto importante sobre el tiempo de ejecución real de la rutina. Esto se debe a varias razones, incluidas la impericia de los programadores, pero principalmente se ve afectado por la naturaleza poco eficiente del algoritmo. No importa cuanto se optimice, el stoogesort nunca será un algoritmo veloz.

Referencias

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

[2]