66:20 Organización de computadoras
Trabajo práctico 0: Infraestructura básica.

1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa (y su correspondiente documentación) que resuelva el problema piloto que presentaremos más abajo.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 9), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo[1], y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Usaremos el programa GXemul [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo NetBSD [2].

En la clase del 15/3 hemos repasado, brevemente, los pasos necesarios para la instalación y configuración del entorno de desarrollo.

También se utilizará `gprof`[6], una herramienta de profiling, para evaluar el desempeño de programas escritos en C.

[1] `http://groups.yahoo.com/group/orga6620`

5. Descripción.

En el presente trabajo práctico, se deben implementar dos algoritmos de ordenamiento: *Mergesort*[3] y *Selection sort*[4].
Una vez implementados, procederemos a realizar mediciones para evaluar las posibildades de mejora y el desempeño relativo entre ambas implementaciones, utilizando los programas `time`[5] y `gprof`[6].

6. Implementación.

El programa debe leer los datos de entrada desde *stdin* o bien desde uno o mas archivos. La salida del programa debe imprimirse por *stdout*, mientras que los errores deben imprimirse por *stderr*. El algoritmo de ordenamiento puede seleccionarse mediante las opciones `-m` o `-s` para *mergesort* o *selection sort* respectivamente.
Mostramos el mensaje de ayuda mediante la opción `-h`.

```
$tp0 -h
tp0 [OPTIONS][file...]
-h, --help             display this help and exit.
-V, --version          display version information and exit.
-m, --merge            use the mergesort algorithm.
-s, --sel              use the selectionsort algorithm.

$echo -n "9876543210" >digits.txt
$tp0 -s  digits.txt
0123456789$

$cat letters.txt
aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ$

$tp0 < letters.txt
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz$

$tp0 letters.txt digits.txt
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz$

$ head -c 64 /dev/urandom > random.txt
$ hexdump -C random.txt
00000000  5f 8e a1 e9 2d 25 49 85  77 04 05 19 16 ca 48 fc  |_...-%I.w.....H.|
00000010  5f d0 ea 84 a2 17 5a 86  e3 93 1c 79 19 bd 32 dc  |_.....Z....y..2.|
00000020  b9 2b 76 ed 75 49 95 37  43 c4 fe 22 d5 a5 bf 56  |.+v.uI.7C.."...V|
00000030  84 6a 06 0d 56 50 f7 b5  cb 41 7c fb 2d 33 49 b7  |.j..VP...A|.-3I.|
00000040
$tp0 <random.txt  > sorted.txt
$ hexdump -C sorted.txt
00000000  84 84 85 86 8e 93 95 a1  a2 a5 b5 b7 b9 bd bf c4  |................|
00000010  ca cb d0 d5 dc e3 e9 ea  ed f7 fb fc fe 04 05 06  |................|
00000020  0d 16 17 19 19 1c 22 25  2b 2d 2d 32 33 37 41 43  |......"%+--237AC|
00000030  48 49 49 49 50 56 56 5a  5f 5f 6a 75 76 77 79 7c  |HIIIPVVZ__juvwy||
```

6.1. Portabilidad

Como es usual, es necesario que la implementación desarrollada provea un grado mínimo de portabilidad. Para satisfacer esto, el programa deberá funcionar al menos en NetBSD/pmax (usando el simulador GXemul [1]) y la versión de Linux (Knoppix, RedHat, Debian, Ubuntu) usada para correr el simulador, Linux/i386.

7. Mediciones.

Utilizar `time` para tomar el tiempo que tardan ambos algoritmos en ordenar una entrada de tamaño $100 * 2^{\{0...16\}}$, para:

- Un array ya ordenado
- Un array en orden inverso
- Diez arrays tomados al azar(tomar el promedio).

Graficar para cada algoritmo el tiempo insumido contra el tamaño de muestra, para los tres casos. Graficar el speedup de *Merge sort* contra *Selection sort* para los diversos valores de N, para los tres casos.

8. Profiling

Supongamos que queremos optimizar el *Selection sort* valiéndonos de la herramienta `gprof`.

- ¿Qué tamaño de muestra nos conviene más tomar para hacer el profiling?
- Si tuviera que elegir una y sólo una función para optimizar (asumiendo que cualquiera puede ser arbitrariamente optimizada), ¿Cuál elegiría? ¿Cuál sería el speedup máximo obtenible mediante la optimización de esa función?

9. Informe.

El informe debe incluir:

- Informe describiendo el desarrollo del trabajo práctico.
- Comando(s) para compilar el programa.
- Corridas de prueba, con los comentarios pertinentes.
- CD conteniendo todo el material digital.
- Código assembly MIPS32 generado por el compilador **(solo en formato digital)**
- Código fuente.
- Este enunciado.

10. Fechas de entrega.

- Primera entrega: Jueves 29 de Marzo.
- Revisión: Jueves 5 de Abril.
- Vencimiento: Jueves 12 de Abril.

Referencias

[1] GXemul, `http://gavare.se/gxemul/`.

[2] The NetBSD project, `http://www.netbsd.org/`.

[3] Merge-sort `http://en.wikipedia.org/wiki/Merge_sort`

[4] Selection-sort `http://en.wikipedia.org/wiki/Selection_sort`

[5] time man page `http://unixhelp.ed.ac.uk/CGI/man-cgi?time`

[6] GNU gprof `http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html`