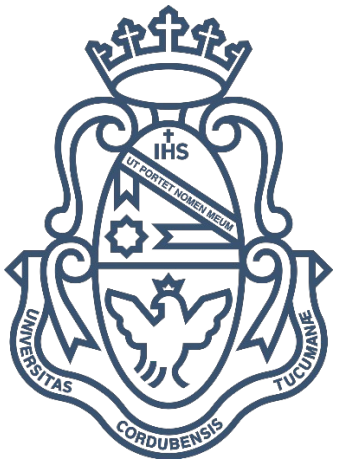

Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales
Sistemas de computación

TP1: Rendimiento de las computadoras (PARTE 3)

Grupo:
Epsilon

Profesores (por órden alfabético):
Jorge, Javier Alejandro
Lamberti, Germán Andrés
Solinas, Miguel Ángel

Alumnos (por órden alfabético):
Campos, Mariano
González, Damián Marcelo

Se procede a realizar la compilación del programa **test_gprof**, conformado por dos archivos de código fuente C. Luego se ejecuta el binario resultante. Como se puede observar, el mismo imprime texto en pantalla:

```
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ ls
test_gprof.c  test_gprof_new.c
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ gcc -Wall -pg test_gprof.c test_gprof_new.c -o test_gprof
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ ls
test_gprof  test_gprof.c  test_gprof_new.c
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ ./test_gprof

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ ls
gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
```

Obsérvese que se han pasado las flags *-pg*, que fueron menester para que al ejecutar el binario, se genere el archivo *gmon.out*. Este archivo servirá para luego pasarse como uno de los argumentos al programa **gprof**, junto con el nombre del binario ejecutable *test_gprof*. El output de **gprof** se redirige al archivo de texto *analysis.txt*:

```
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ gprof --version
GNU gprof (GNU Binutils for Ubuntu) 2.38
Based on BSD gprof, copyright 1983 Regents of the University of California.
This program is free software.  This program has absolutely no warranty.
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ gprof test_gprof gmon.out > analysis.txt
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ ls
analysis.txt  gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
```

A continuación se presenta lo mas importante del contenido de *analysis.txt*, generado por **gprof**:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 55.33      8.04     8.04        1     8.04     8.54  func1
 37.78     13.53     5.49        1     5.49     5.49  func2
  3.44     14.03     0.50        1     0.50     0.50  new_func1
  3.44     14.53     0.50                             main

        Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.07% of 14.53 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.50   14.03                 main [1]
                8.04    0.50       1/1           func1 [2]
                5.49    0.00       1/1           func2 [3]
-----------------------------------------------
                8.04    0.50       1/1           main [1]
[2]     58.8    8.04    0.50       1         func1 [2]
                0.50    0.00       1/1           new_func1 [4]
-----------------------------------------------
                5.49    0.00       1/1           main [1]
[3]     37.8    5.49    0.00       1         func2 [3]
-----------------------------------------------
                0.50    0.00       1/1           func1 [2]
[4]      3.4    0.50    0.00       1         new_func1 [4]
-----------------------------------------------
```

Luego se procede a ejecutar el otro programa de profiling: **perf**. Este programa genera el archivo *perf.data*:

```
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ perf --version
perf version 5.15.178
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ ls
analysis.txt  gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ sudo perf record ./test_gprof

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 2,305 MB perf.data (60011 samples) ]
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ sudo perf report
dmg@dmg-BANGHO:~/Documents/SdC/1_Práctica/TPs/TP_1/parte_3$ ls
analysis.txt  gmon.out  perf.data  test_gprof  test_gprof.c  test_gprof_new.c
```

Al llamar luego al comando *sudo perf report*, la terminal entra en modo *modal* con esta interfaz:

```
Samples: 60K of event 'cycles', Event count (approx.): 47947407635
Overhead  Command     Shared Object      Symbol
  54,61%  test_gprof  test_gprof         [.] func1
  37,12%  test_gprof  test_gprof         [.] func2
   3,43%  test_gprof  test_gprof         [.] new_func1
   3,42%  test_gprof  test_gprof         [.] main
   1,19%  test_gprof  [kernel.kallsyms]  [k] asm_exc_nmi
   0,02%  test_gprof  [kernel.kallsyms]  [k] pci_read32_sync
   0,01%  test_gprof  [kernel.kallsyms]  [k] __update_load_avg_cfs_rq
   0,01%  test_gprof  [kernel.kallsyms]  [k] __irqentry_text_end
   0,01%  test_gprof  [kernel.kallsyms]  [k] native_write_msr
```

Conclusión

Como se puede observar en la última imagen, ambos profilers dan resultados muy similares. De todas maneras, si uno quisiera confiar en la precisión de alguno de los dos, aquel sería **gprof** por su forma de realizar las mediciones (inyección de código). Por contraparte, puede que **perf** no sea *super fino* con las estadísticas recolectadas, pero sí es cierto que comienza a correr mas rápido y consume menos recursos que **gprof**, esto dado que **perf** no inyecta código para recoger estadísticas, sino que hace un *poll intrusivo* por medio de interrupciones de sistema al código del proceso en cuestión siendo ejecutado; ello puede hacer que alguna función *se escape* de ser detectado su run si corre lo suficientemente rápido y sucede entre poll y poll.