Universidad
Nacional
de Córdoba

Sistemas de Computación

Trabajo práctico Nº 1: "Rendimiento"

Profesor: Ing. Javier Jorge
Fecha de entrega: 03/04/23

Apellido, Nombre	Carrera	Matrícula	e-mail
Cazajous, Miguel	Ing. Computación	34980294	miguel.cazajous@unc.edu.ar
Marclé, Emiliano	Ing. Electrónica	36681020	emimarcle@mi.unc.edu.ar
Garella, Andrés E.	Ing. Computación	41378225	andres.garella@mi.unc.edu.ar

Introducción

En este informe se desarrolla el Trabajo Práctico N°1 de la asignatura Sistemas de Computación. El objetivo de este trabajo es poner en práctica los conocimientos sobre performance y rendimiento de los computadores logrando así:

1. Utilizar benchmarks de terceros para tomar decisiones de hardware.
2. Utilizar herramientas (GPROF y perf) para medir la performance de nuestro código.

Listado de Benchmarks

En principio se realizó un listado de benchmarks considerados como útiles para medir las tareas desarrolladas a diario.

Benchmark	Descripción
Hyperfine	Herramienta de benchmarking para la línea de comandos
Time	Comando de Linux para determinar tiempo de ejecución de operaciones específicas
Phoronix test suite	Plataforma automatizada de testeo y benchmarking
Google Benchmark	Herramienta de Benchmarking
Glxgears	Es un programa de prueba Open GL sencillo que reporta FPS (Frames Per Second)

Benchmarks para distintas tareas

Actividades	Benchmark
Abrir programas	Hyperfine, Time
Instalación de kernels custom	Phoronix Test Suite
Juegos o programas que usan GPU (ejemplo: terminales)	GlxGears
Programas propios (C++)	Google Benchmark

Ejemplos

Hyperfine y time

Para medir el tiempo que toma en abrir el editor de texto NeoVim. En este caso hacemos uso de las opciones del editor para salir inmediatamente cuando el programa arranca. Para otros casos tal vez no sea directo o preciso el resultado de la medición.

```
miguel 1/Entrega hyperfine "nvim --headless +qa" --warmup 5
Benchmark 1: nvim --headless +qa
  Time (mean ± σ):     135.8 ms ±   2.3 ms    [User: 104.7 ms, System: 30.7 ms]
  Range (min … max):   132.5 ms … 142.6 ms    21 runs

miguel 1/Entrega time nvim --headless +qa
nvim --headless +qa  0.10s user 0.03s system 99% cpu 0.134 total
miguel 1/Entrega
```

Phoronix Test Suite

En su momento fue útil para hacer una evaluación sobre la performance de diferentes kernels, tanto el release del kernel genérico (LTS, latest) como también pruebas en kernels custom (Xanmod, liquorix, zen, etc)

Glxgear

Una prueba rápida de la performance de la GPU. Aunque no son pruebas directas sobre el programa que se pretende use la GPU.

```
miguel 1/Entrega glxgears
Running synchronized to the vertical refresh.  The framerate should be
approximately the same as the monitor refresh rate.
722 frames in 5.0 seconds = 144.383 FPS
718 frames in 5.0 seconds = 143.416 FPS
718 frames in 5.0 seconds = 143.572 FPS
719 frames in 5.0 seconds = 143.773 FPS
X connection to :0 broken (explicit kill or server shutdown).
miguel 1/Entrega
```

Google benchmark

Para desarrollos propios un framework relativamente sencillo de usar que proporciona tiempos para n iteraciones de diferentes funciones aunque su uso está limitado solo a lenguaje C++.

```
Benchmark                        Time(ns)    CPU(ns) Iterations
----------------------------------------------------------------
BM_SetInsert/1024/1                 28928     29349     23853  133.097kB/s   33.2742k items/s
BM_SetInsert/1024/8                 32065     32913     21375  949.487kB/s   237.372k items/s
BM_SetInsert/1024/10                33157     33648     21431  1.13369MB/s   290.225k items/s
```

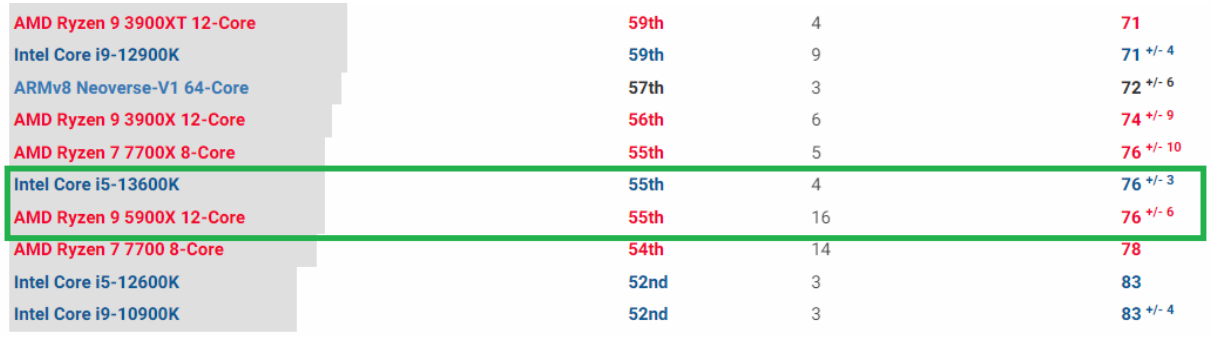
Imagen de: https://github.com/google/benchmark/blob/main/docs/user_guide.md

Rendimiento de procesadores para compilar kernel de Linux

Se busca calcular el rendimiento al compilar el kernel de Linux de los procesadores:

1. **Intel Core i5-13600K**
2. **AMD Ryzen 9 5900X 12-Core**

El rendimiento de un sistema es la capacidad que tiene este para realizar la tarea en un determinado tiempo, entonces es necesario contar con los tiempos de ejecución de la tarea para poder calcularlo.
En el sitio https://openbenchmarking.org/test/pts/build-linux-kernel-1.15.0 se proveen resultados para distintos procesadores, de un benchmark que testea el tiempo que se tarda para compilar el kernel de Linux.



AMD Ryzen 9 3900XT 12-Core	59th	4	71
Intel Core i9-12900K	59th	9	71 +/- 4
ARMv8 Neoverse-V1 64-Core	57th	3	72 +/- 6
AMD Ryzen 9 3900X 12-Core	56th	6	74 +/- 9
AMD Ryzen 7 7700X 8-Core	55th	5	76 +/- 10
Intel Core i5-13600K	55th	4	76 +/- 3
AMD Ryzen 9 5900X 12-Core	55th	16	76 +/- 6
AMD Ryzen 7 7700 8-Core	54th	14	78
Intel Core i5-12600K	52nd	3	83
Intel Core i9-10900K	52nd	3	83 +/- 4

Procesador	Tiempo promedio [s]
Intel Core i5 13600k	76 ± 3
AMD Ryzen 9 5900X 12-Core	76 ± 6

Considerando en ambos procesadores el valor promedio de 76[s] para la compilación del kernel de Linux, podemos calcular el rendimiento como:

$$\eta_{Intel\ Core\ i5} = \eta_{AMD\ Ryzen\ 9} = \frac{1}{T_{EX}} = \frac{1}{76[s]} = 0,0132\ [s^{-1}] = 1,32\%$$

Considerando que el **AMD Ryzen 9 5900X** tiene 12 núcleos y el **Intel Core i5 13600k** tiene 14, podemos decir que el primero logra un mejor aprovechamiento de sus núcleos.

Speedup del AMD Ryzen 9 7950X 16-Core:

La aceleración (speedup) cuando usamos un **AMD Ryzen 9 7950X 16-Core** se calcula tomando como rendimiento original aquel que corresponde al **AMD Ryzen 9 5900X 12-Core** y como rendimiento mejorado al del **Ryzen 9 7950X**.

AMD Ryzen Threadripper 3960X 24-Core	80th	5	44 +/- 1
AMD EPYC 7513 32-Core	79th	5	46 +/- 1
AMD Ryzen 9 7950X 16-Core	78th	51	46 +/- 5
Intel Core i9-13900K	77th	23	47 +/- 3

Procesador	Tiempo promedio [s]
AMD Ryzen 9 7950X 16-Core	46 ± 5

$$Speedup = \frac{\eta_{Mejorado}}{\eta_{Original}} = \frac{1 / T_{EX\ Mejorado}}{1 / T_{EX\ Original}} = \frac{T_{EX\ Original}}{T_{EX\ Mejorado}} = \frac{76\ [s]}{46\ [s]}$$

$$Speedup = 1,65$$

Como podemos observar, el aumento del rendimiento es aproximadamente 1,7 veces al utilizar un procesador con 16 núcleos en comparación al modelo de 12 núcleos.
Los valores de rendimiento original son casi iguales al modelo de intel, por ende podemos extrapolar este resultado para la comparación entre el Ryzen 9 7950X y el Core i5.

Comparación de costos

Procesador	Precio
Intel Core i5 13600k	300 $US
AMD Ryzen 9 5900X 12-Core	347 $US
AMD Ryzen 9 7950X 16-Core	570 $US

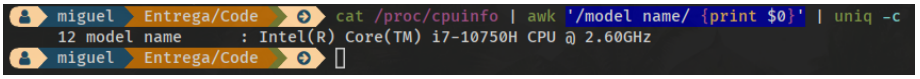
- La diferencia de precio entre el procesador de 16 núcleos y el de 12 es significativa, por lo tanto habría que determinar si es suficiente la mejora en el rendimiento para ejecutar el kernel de Linux, como para que se justifique la mejora del hardware.

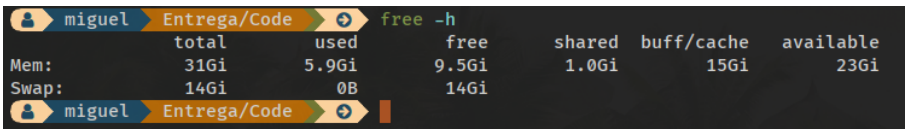
Tutorial de Profiling con GPROF y perf

Entornos de pruebas

Cada uno de los integrantes del grupo realizó el tutorial de profiling con gprof y perf, para poder comparar luego los resultados obtenidos con distintas configuraciones.
Los siguientes son los entornos de prueba utilizados por cada uno de los integrantes:

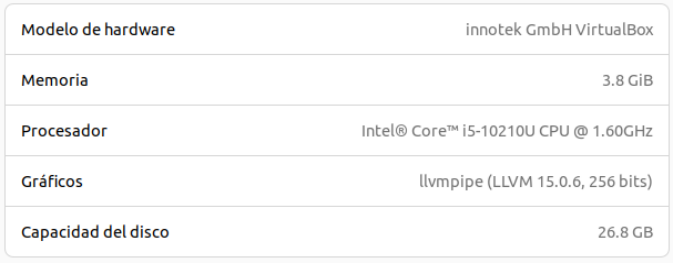
- **Intel i7 - 10750H 12x2.6Ghz / NVME 460GB / RAM: 32GB**



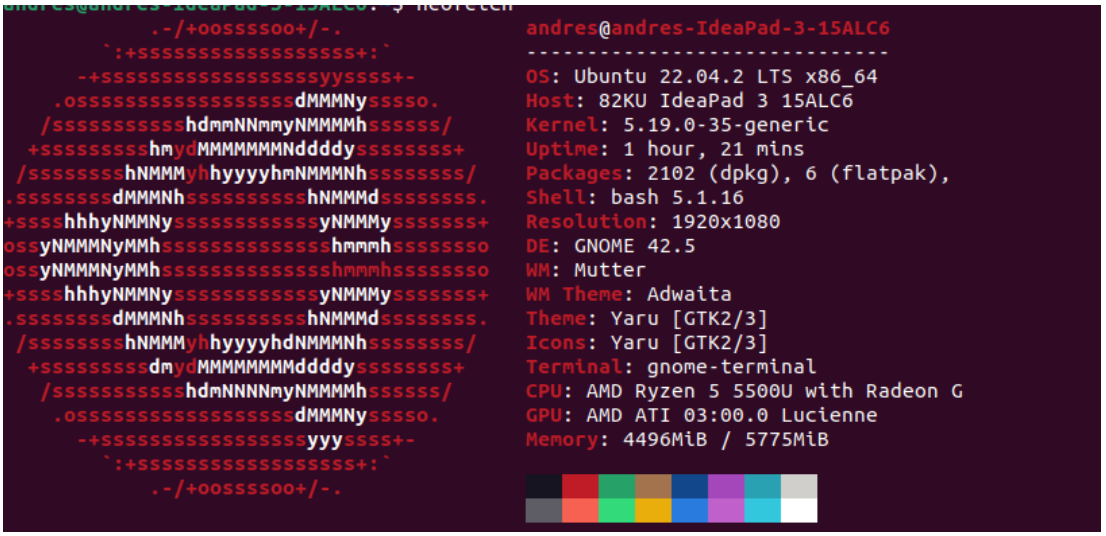


- **Intel CORE i5 NVMe 240GB 8GB RAM**

Máquina virtual con 4GB de RAM y disco duro virtual de 30GB utilizando SO Ubuntu 22.04.2 LTS.

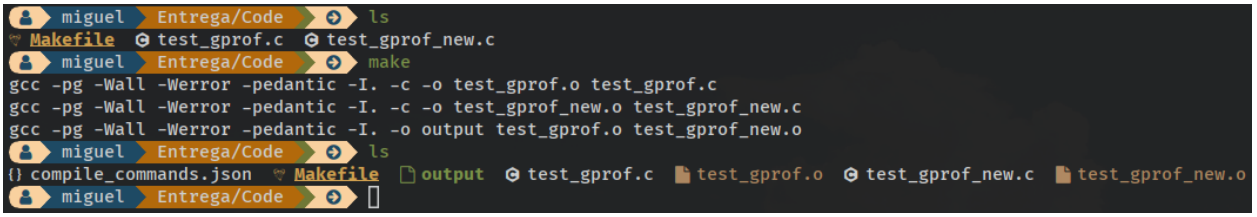


- **Ryzen 5500U NVMe 500GB 8gb RAM**



Desarrollo del tutorial de profiling

- **Paso 1: Creación de perfiles habilitada durante la compilación**



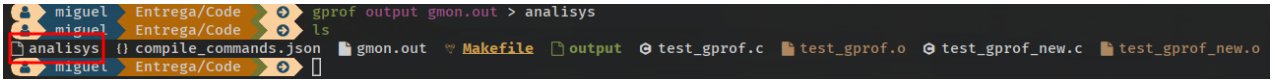
Observamos el flag -pg que es el encargado de habilitar la creación de perfiles en la compilación.

- **Paso 2: Ejecutar el código**

Luego de ejecutar el programa vemos el archivo generado **gmon.out** en el directorio de trabajo.

- **Paso 3: Ejecute la herramienta gprof**

Se ejecuta **gprof** con el nombre del ejecutable y el gmon.out, creándose el reporte **analysis.txt**.

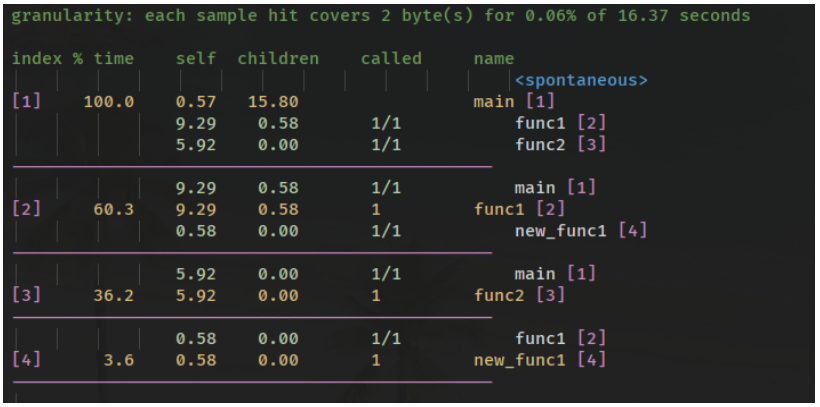


Comprensión de la información de perfil

- **% time** : porcentaje del tiempo total de funcionamiento del programa utilizado para esta función.
- **cumulative seconds:** una suma continua de los segundos contabilizados por esta función y las enumeradas anteriormente.

- **self seconds:** El número de segundos contabilizados para esta función.
- **calls:** El número de veces que esta función fue invocada.
- **self ms/call:** El número promedio de mili segundos gastados en esta función por llamada si está perfilada, de lo contrario, en blanco.
- **total ms/call:** El número promedio de mili segundos gastados en esta función y sus descendientes por llamada si está perfilada, de lo contrario, en blanco.
- **name:** El nombre de la función. El índice muestra la ubicación de la función en la lista de gprof.

Call graph



```
granularity: each sample hit covers 2 byte(s) for 0.06% of 16.37 seconds

index % time    self  children    called     name
                                             <spontaneous>
[1]    100.0    0.57   15.80                 main [1]
                9.29    0.58       1/1           func1 [2]
                5.92    0.00       1/1           func2 [3]
-----------------------------------------------
                9.29    0.58       1/1           main [1]
[2]     60.3    9.29    0.58       1         func1 [2]
                0.58    0.00       1/1           new_func1 [4]
-----------------------------------------------
                5.92    0.00       1/1           main [1]
[3]     36.2    5.92    0.00       1         func2 [3]
-----------------------------------------------
                0.58    0.00       1/1           func1 [2]
[4]      3.6    0.58    0.00       1         new_func1 [4]
-----------------------------------------------
```

Esta tabla describe el árbol de llamadas del programa y es ordenada por la cantidad total de tiempo empleado en cada función y sus hijos.

Cada entrada en esta tabla consta de varias líneas. La línea con el número de índice en el margen izquierdo enumera la función actual. Las líneas de arriba enumeran las funciones que llamaron a esta función, y las líneas debajo enumeran las funciones a las que llama.

- **index** : Un número único asignado a cada elemento de la tabla. Están ordenados numéricamente.
- **% time**: Porcentaje del tiempo 'total' que se dedicó en esta función y sus hijos.
- **self:** Cantidad total de tiempo empleado en esta función.
- **children:** Cantidad total de tiempo propagado en esta función por sus hijos.
- **called:** Número de veces que se llamó a la función. Si la función se llama a sí misma recursivamente, el número solo incluye llamadas no recursivas, y es seguido por un `+' y el número de llamadas recursivas.
- **name:** El nombre de la función actual.

Si los padres de la función no pueden ser determinados, se imprime **<spontaneous>** en el campo **name**, y todos los otros campos se dejan en blanco.

Para los hijos de la función, los campos son:

- **self:** Cantidad de tiempo que se propagó directamente del hijo a la función.
- **children:** Cantidad de tiempo que se propagó desde los hijos del hijo a la función.
- **called:** Número de veces que la función llamo este hijo ` / ' el número total de veces que el hijo fue llamado Las llamadas recursivas del hijo no se listan en el número después de `/ '.
- **name:** Este es el nombre del hijo.

Customización de la salida de gprof output utilizando flags

1. Supresión de la impresión de funciones declaradas estáticamente (privadas) usando -a

```
analisys  test_gprof.c  test_gprof_new.c  analisys2
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 93.17     15.30    15.30        2     7.65     7.94  func1
  3.60     15.89     0.59        1     0.59     0.59  new_func1
  3.48     16.46     0.57                             main
```

```
analisys  test_gprof.c  test_gprof_new.c  analisys2

granularity: each sample hit covers 2 byte(s) for 0.06% of 16.46 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.57   15.89                 main [1]
                15.30    0.59       2/2           func1 [2]
-----------------------------------------------
                15.30    0.59       2/2           main [1]
[2]     96.5   15.30    0.59       2         func1 [2]
                 0.59    0.00       1/1           new_func1 [3]
-----------------------------------------------
                 0.59    0.00       1/1           func1 [2]
[3]      3.6    0.59    0.00       1         new_func1 [3]
-----------------------------------------------
```

En este caso se elimina la información de **func2** ya que fué declarada como estática.

2. Eliminación de los textos detallados usando flag -b

```
test_gprof.c  test_gprof_new.c  analisys3
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 56.60      9.29     9.29        1     9.29     9.88  func1
 36.57     15.30     6.01        1     6.01     6.01  func2
  3.60     15.89     0.59        1     0.59     0.59  new_func1
  3.48     16.46     0.57                             main
 L
                Call graph


granularity: each sample hit covers 2 byte(s) for 0.06% of 16.46 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.57   15.89                 main [1]
                9.29    0.59       1/1           func1 [2]
                6.01    0.00       1/1           func2 [3]
-----------------------------------------------
                9.29    0.59       1/1           main [1]
[2]     60.0    9.29    0.59       1         func1 [2]
                0.59    0.00       1/1           new_func1 [4]
-----------------------------------------------
                6.01    0.00       1/1           main [1]
[3]     36.5    6.01    0.00       1         func2 [3]
-----------------------------------------------
                0.59    0.00       1/1           func1 [2]
[4]      3.6    0.59    0.00       1         new_func1 [4]
-----------------------------------------------
 L
Index by function name

   [2] func1                   [1] main
   [3] func2                   [4] new_func1
```

3. Impresión de solo perfil plano usando flag -p

test_gprof.c × test_gprof_new.c × analisys4 ×

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 56.60      9.29     9.29        1     9.29     9.88  func1
 36.57     15.30     6.01        1     6.01     6.01  func2
  3.60     15.89     0.59        1     0.59     0.59  new_func1
  3.48     16.46     0.57                             main
```

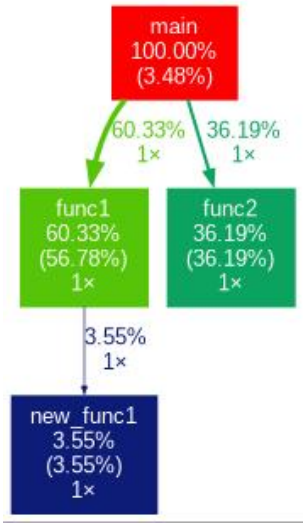
4. Impresión de información relacionada con funciones específicas en perfil plano

test_gprof.c × test_gprof_new.c × analisys5 ×

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.44      9.29     9.29        1     9.29     9.29  func1
```

Generación de un gráfico para visualizar la salida de gprof



Profiling con linux perf

perf es una herramienta sencilla para sistemas operativos basados en Linux, que permite realizar un análisis del rendimiento.
Admite contadores de rendimiento de hardware, contadores de rendimiento de software y sondas dinámicas.

- **Ejecución de perf:**

```
miguel Entrega/Code sudo perf record ./output

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
[ perf record: Woken up 10 times to write data ]
[ perf record: Captured and wrote 2.568 MB perf.data (66674 samples) ]
miguel Entrega/Code
```

- **Generación del reporte**

Mediante perf report obtenemos una visualización de los tiempos de ejecución de las diferentes funciones.

```
Samples: 66K of event 'cycles', Event count (approx.): 43006565542
Overhead  Comman  Shared Object     Symbol
  56.46%  output  output            [.] func1
  36.30%  output  output            [.] func2
   3.50%  output  output            [.] main
   3.43%  output  output            [.] new_func1
```

Resultados del tutorial de profiling utilizando otros sistemas

Profiling con GPROF

- *Entorno de prueba 2:*

```
emimarcle@emilianomarcleVB:~/GPROF$ gprof -b test_gprof gmon.out > analysis.txt
emimarcle@emilianomarcleVB:~/GPROF$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 55.17      7.26     7.26        1     7.26     7.75  func1
 37.61     12.21     4.95        1     4.95     4.95  func2
  3.72     12.70     0.49        1     0.49     0.49  new_func1
  3.50     13.16     0.46                             main


                     Call graph


granularity: each sample hit covers 4 byte(s) for 0.08% of 13.16 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.46   12.70                 main [1]
                7.26    0.49       1/1           func1 [2]
                4.95    0.00       1/1           func2 [3]
-----------------------------------------------
                7.26    0.49       1/1           main [1]
[2]     58.9    7.26    0.49       1         func1 [2]
                0.49    0.00       1/1           new_func1 [4]
-----------------------------------------------
                4.95    0.00       1/1           main [1]
[3]     37.6    4.95    0.00       1         func2 [3]
-----------------------------------------------
                0.49    0.00       1/1           func1 [2]
[4]      3.7    0.49    0.00       1         new_func1 [4]
-----------------------------------------------


Index by function name

   [2] func1                   [1] main
   [3] func2                   [4] new_func1
```

- *Entorno de prueba 3:*

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 55.44      7.54     7.54        1     7.54     8.00  func1
 37.79     12.68     5.14        1     5.14     5.14  func2
  3.38     13.14     0.46        1     0.46     0.46  new_func1
  3.38     13.60     0.46                             main
```

```
granularity: each sample hit covers 4 byte(s) for 0.07% of 13.60 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.46   13.14                 main [1]
                7.54    0.46       1/1           func1 [2]
                5.14    0.00       1/1           func2 [3]
-----------------------------------------------
                7.54    0.46       1/1           main [1]
[2]     58.8    7.54    0.46       1         func1 [2]
                0.46    0.00       1/1           new_func1 [4]
-----------------------------------------------
                5.14    0.00       1/1           main [1]
[3]     37.8    5.14    0.00       1         func2 [3]
-----------------------------------------------
                0.46    0.00       1/1           func1 [2]
[4]      3.4    0.46    0.00       1         new_func1 [4]
-----------------------------------------------
```

Profiling con perf

- *Entorno de prueba 2:*

```
emimarcle@emilianomarcleVB:~/GPROF$ sudo perf record ./test_gprof

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
[ perf record: Woken up 3 times to write data ]
[ perf record: Captured and wrote 0,596 MB perf.data (15360 samples) ]
```

```
Samples: 15K of event 'cpu-clock:pppH', Event count (approx.): 3840000000
Overhead  Command     Shared Object      Symbol
  53,80%  test_gprof  test_gprof         [.] func1
  36,42%  test_gprof  test_gprof         [.] func2
   3,56%  test_gprof  test_gprof         [.] main
   3,50%  test_gprof  test_gprof         [.] new_func1
   2,51%  test_gprof  [vboxguest]        [k] vbg_req_perform
   0,05%  test_gprof  [kernel.kallsyms]  [k] __softirqentry_text_start
   0,04%  test_gprof  [kernel.kallsyms]  [k] _raw_spin_unlock_irqrestore
   0,04%  test_gprof  [kernel.kallsyms]  [k] finish_task_switch.isra.0
   0,03%  test_gprof  [kernel.kallsyms]  [k] _raw_spin_unlock_irq
   0,01%  test_gprof  [kernel.kallsyms]  [k] exit_to_user_mode_loop
   0,01%  test_gprof  [kernel.kallsyms]  [k] __fpu_restore_sig
   0,01%  test_gprof  [kernel.kallsyms]  [k] __mod_timer
   0,01%  test_gprof  [kernel.kallsyms]  [k] __rseq_handle_notify_resume
   0,01%  test_gprof  [kernel.kallsyms]  [k] __x64_sys_rt_sigreturn
   0,01%  test_gprof  [kernel.kallsyms]  [k] restore_altstack
```

- *Entorno de prueba 3:*

```
Samples: 55K of event 'cycles', Event count (approx.): 55605154211
Overhead  Command      Shared Object       Symbol
  55,36%  test_gprof   test_gprof          [.] func1
  37,69%  test_gprof   test_gprof          [.] func2
   3,47%  test_gprof   test_gprof          [.] new_func1
   3,44%  test_gprof   test_gprof          [.] main
   0,01%  test_gprof   libc.so.6           [.] __resolv_conf_attach
   0,00%  test_gprof   [kernel.kallsyms]   [k] read_hpet.part.0
   0,00%  test_gprof   [kernel.kallsyms]   [k] do_sigaltstack.constprop.0
   0,00%  test_gprof   [kernel.kallsyms]   [k] restore_sigcontext
   0,00%  test_gprof   [kernel.kallsyms]   [k] blk_mq_flush_plug_list
   0,00%  test_gprof   [kernel.kallsyms]   [k] prepare_signal
   0,00%  test_gprof   [kernel.kallsyms]   [k] should_failslab
   0,00%  test_gprof   [kernel.kallsyms]   [k] kmem_cache_alloc
   0,00%  test_gprof   [kernel.kallsyms]   [k] __setup_rt_frame
   0,00%  test_gprof   [kernel.kallsyms]   [k] xhci_triad_to_transfer_ring
   0,00%  test_gprof   [kernel.kallsyms]   [k] ext4_ext_map_blocks
   0,00%  test_gprof   [kernel.kallsyms]   [k] blk_stat_add
   0,00%  test_gprof   [kernel.kallsyms]   [k] __cond_resched
   0,00%  test_gprof   [kernel.kallsyms]   [k] native_queued_spin_lock_slowpath.part.0
   0,00%  test_gprof   [kernel.kallsyms]   [k] __sigqueue_alloc
   0,00%  test_gprof   [kernel.kallsyms]   [k] ep_autoremove_wake_function
   0,00%  test_gprof   [kernel.kallsyms]   [k] __get_obj_cgroup_from_memcg
   0,00%  test_gprof   [kernel.kallsyms]   [k] change_pmd_range.isra.0
   0,00%  perf-exec    [kernel.kallsyms]   [k] perf_lock_task_context
   0,00%  test_gprof   [kernel.kallsyms]   [k] _raw_spin_lock_irq
```

Resumen de resultados y conclusiones sobre profiling con GPROF y perf

Configuración	tiempo total	tiempo en main	tiempo en func1	tiempo en func2	tiempo en newfunc
Intel Core i7 - NVMe 460GB - 32GB RAM	16.37s	0.57s	9.29s	5.92s	0.58s
Intel Core i5 10210U CPU @1.6GHz NVMe 240GB 8GB RAM (VM 30GB 4GB RAM)	13.16s	0.46s	7.26s	4.95s	0.49s
Ryzen 5500U - NVMe 500GB - 8GB RAM	13.6s	0.46s	7.54s	5.14s	0.46s

- La mayor parte del tiempo de ejecución transcurre en la función **func1** (entre 7.26[s] y 9.29[s]), por lo que sería importante optimizar su codificación (reduciendo la cantidad de operaciones) para maximizar así el rendimiento. En el perfilado realizado con **perf** se observa que en promedio entre el 54% y el 56% de los samples recolectados (samples overhead) cayeron en **func1**.
- La función **func2** también ocupa una porción significativa del tiempo total de ejecución (entre 4,95[s] y 5.92[s]). De acuerdo al perfilado con **perf** entre el 36,3% y 37,7% de los samples son recolectados en esta función.
- Por otro lado, desde **main** solo se llama a **func1** y **func2,** por lo que no se gasta mucho tiempo dentro de él (entre 0.46[s] y 0.57[s]).
- La función **newfunc** es llamada desde **func1** y gasta un tiempo de 0.46[s] a 0.58[s]. Con **perf,** esto se corresponde a un porcentaje entre el 3,4% y 3,5% de los samples totales recolectados