

Checklist de Tutorial de Profiling en Linux con `gprof` y `perf`

Requerimientos e instalación de herramientas

- [✓] Instalar gcc, make y las herramientas de desarrollo:

```
sudo apt update
sudo apt install build-essential
```

- [✓] Instalar linux-tools (para perf):

```
sudo apt install linux-tools-common
sudo apt install linux-tools-$(uname -r)
```

- [✓] Instalar gprof2dot y graphviz (para la visualización de gprof):

```
pip install gprof2dot
sudo apt install graphviz
```

```
valentin@valentin-Aspire-A314-31:~$ pipx install gprof2dot
installed package gprof2dot 2024.6.6, installed using Python 3.
These apps are now globally available
- gprof2dot
⚠ Note: '/home/valentin/.local/bin' is not on your PATH
environment variable. These apps will not be globally
accessible until your PATH is updated. Run `pipx
ensurepath` to automatically add it, or manually modify
your PATH in your shell's config file (i.e. ~/.bashrc).
done! ✨ 🌟 ✨
valentin@valentin-Aspire-A314-31:~$ sudo apt install graphviz
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  fonts-liberation2 libann0 libcdt5 libcggraph6 libgts-0.7-5t64
  libgts-bin libgvc6 libgvpr2 liblab-gamut1 libpathplan4
```

Parte 1: Profiling con GPROF

Preparación del código

- [✓] Crear archivo `test_gprof.c` con el siguiente contenido:

```
``c
#include <stdio.h>

void new_func1(void);

void func1(void) {
```

```

        printf("\n Inside func1 \n");
        for (int i = 0; i < 0xffffffff; i++);
        new_func1();
    }

static void func2(void) {
    printf("\n Inside func2 \n");
    for (int i = 0; i < 0xfffffaa; i++);
}

int main(void) {
    printf("\n Inside main()\n");
    for (int i = 0; i < 0xfffff; i++);
    func1();
    func2();
    return 0;
}
...

```

- [☒] Crear archivo `test_gprof_new.c` con la siguiente función:

```

```c
#include <stdio.h>

void new_func1(void) {
 printf("\n Inside new_func1()\n");
 for (int i = 0; i < 0xfffffee; i++);
}
...

```

### Compilación con soporte de profiling

- [ ☒ ] Compilar usando:

```
gcc -Wall -pg test_gprof.c test_gprof_new.c -o test_gprof
```

### Ejecución del programa

- [ ☒ ] Ejecutar:

```
./test_gprof
```

Esto generará `gmon.out`.

```

valentin@valentin-Aspire-A314-31:~$ gcc -Wall -pg test_gprof.c test_gprof_new.c -o test_gprof
valentin@valentin-Aspire-A314-31:~$./test_gprof

Inside main()

Inside func1

Inside new_func1()

Inside func2
valentin@valentin-Aspire-A314-31:~$ ls
analysis.txt mi_entorno test_gprof
Desktop Music test_gprof.c
Documents Pictures test_gprof_new.c
Downloads Public Videos
gmon.out snap

```

### Generar análisis de perfil

- [✓] Ejecutar:

```
gprof test_gprof gmon.out > analysis.txt
```

- [✓] Verificar que el archivo `analysis.txt` contiene:
  - [✓] Flat profile
  - [✓] Call graph

### Personalización del análisis con flags (ejecutar cada uno y analizar)

- [✓] `gprof -a test\_gprof gmon.out > analysis.txt`
  - Suprime funciones `static` como `func2`
- [✓] Verificar que `func2` no aparece en el output

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
68.08	23.61	23.61	2	11.80	17.32	func1
31.81	34.64	11.03	1	11.03	11.03	new_func1
0.12	34.68	0.04				main

%  
time      the percentage of the total running time of the  
          program used by this function.

cumulative  
seconds    a running sum of the number of seconds accounted  
          for by this function and those listed above it.

self  
seconds    the number of seconds accounted for by this  
          function alone. This is the major sort for this  
          listing.

calls      the number of times this function was invoked, if  
          this function is profiled, else blank.

self  
ms/call    the average number of milliseconds spent in this  
          function per call, if this function is profiled,  
          else blank.

total  
ms/call    the average number of milliseconds spent in this  
          function and its descendents per call, if this  
          function is profiled, else blank.

- [ ] `gprof -b test\_gprof gmon.out > analysis.txt`

- Elimina explicaciones detalladas

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
35.39	11.95	11.95	1	11.95	22.84	func1
32.25	22.84	10.89	1	10.89	10.89	func2
32.25	33.73	10.89	1	10.89	10.89	new_func1
0.12	33.77	0.04				main

^L

Call graph

- [✓] `gprof -p -b test\_gprof gmon.out > analysis.txt`

- Solo muestra perfil plano

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
100.00	11.95	11.95	1	11.95	11.95	func1

analysis.txt (END)

### Visualización con gprof2dot

- [✓] Instalar herramientas necesarias:

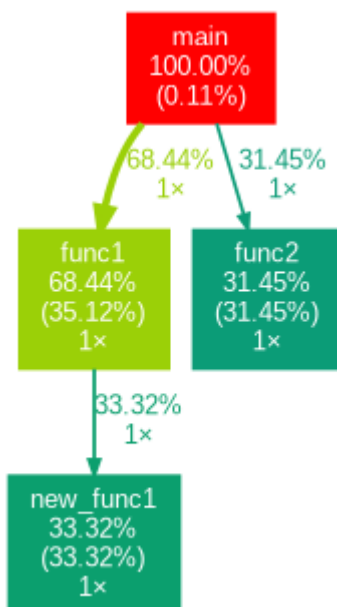
```
pip install gprof2dot
```

```
sudo apt install graphviz
```

- [✓] Generar gráfico:

```
gprof test_gprof gmon.out | gprof2dot | dot -Tpng -o output.png
```

- [✓] Verificar que se generó `output.png`



## Parte 2: Profiling con `perf`

### ### Ejecución de análisis con `perf`

- [✓] Ejecutar:

```
sudo perf record ./test_gprof
```

- [✓] Generar reporte interactivo:

```
sudo perf report
```

```
valentin@valentin-Aspire-A314-31: ~
Samples: 143K of event 'cycles:P', Event count (approx.): 83207800636
Overhead Command Shared Object Symbol
34.07% test_gprof test_gprof [.] func1
33.13% test_gprof test_gprof [.] func2
31.69% test_gprof test_gprof [.] new_func1
0.14% test_gprof test_gprof [.] main
0.04% test_gprof [kernel.kallsyms] [k] __update_load_avg_se
0.03% test_gprof [kernel.kallsyms] [k] __update_load_avg_cfs_rq
0.03% test_gprof [kernel.kallsyms] [k] update_load_avg
0.02% test_gprof [kernel.kallsyms] [k] update_cfs_group
0.02% test_gprof [kernel.kallsyms] [k] update_curr
0.02% test_gprof [kernel.kallsyms] [k] __hrtimer_run_queues
0.02% test_gprof [kernel.kallsyms] [k] __calc_delta.constprop.0
0.01% test_gprof [kernel.kallsyms] [k] update_min_vruntime
0.01% test_gprof [kernel.kallsyms] [k] native_sched_clock
0.01% test_gprof [kernel.kallsyms] [k] __raw_spin_lock_irqsave
0.01% test_gprof [kernel.kallsyms] [k] __raw_spin_lock
0.01% test_gprof [kernel.kallsyms] [k] task_tick_fair
0.01% test_gprof [kernel.kallsyms] [k] native_write_msr
0.01% test_gprof [kernel.kallsyms] [k] native_irq_return_iret
0.01% test_gprof [kernel.kallsyms] [k] timekeeping_advance
0.01% test_gprof [kernel.kallsyms] [k] psi_group_change
0.01% test_gprof [kernel.kallsyms] [k] gen8_irq_handler
0.01% test_gprof [kernel.kallsyms] [k] irqentry_exit_to_user_mode
0.01% test_gprof [kernel.kallsyms] [k] __i915_vma_retire
0.01% test_gprof [kernel.kallsyms] [k] rcu_sched_clock_irq
0.01% test_gprof [kernel.kallsyms] [k] sugov_update_single_freq
0.01% test_gprof [kernel.kallsyms] [k] reweight_entity
0.01% test_gprof [kernel.kallsyms] [k] update_curr_se
0.01% test_gprof [kernel.kallsyms] [k] ktime_get
cannot load tips.txt file, please install perf!
```

### ## Limpieza opcional

- [✓] Eliminar archivos generados si se desea:

```
rm gmon.out test_gprof analysis.txt output.png
```

### ## Fin del tutorial

- [✓] Confirmar comprensión de:

- [✓] Perfil plano
- [✓] Gráfico de llamadas
- [✓] Flags para personalización
- [✓] Visualización con `gprof2dot`
- [✓] Análisis básico con `perf`