



Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Sistemas de Computación

Entrega #1: Rendimiento

Autores:

Robles, Karen Yésica.

Rodriguez, Luciano Ariel

2024

Consignas

1- Armar una lista de benchmarks, ¿cuales les serían más útiles a cada uno ? ¿Cuáles podrían llegar a medir mejor las tareas que ustedes realizan a diario ?

Pensar en las tareas que cada uno realiza a diario y escribir en una tabla de dos entradas las tareas y que benchmark la representa mejor.

2- Cuál es el rendimiento de estos procesadores para compilar el kernel?

Intel Core i5-13600K

AMD Ryzen 9 5900X 12-Core

Cual es la aceleración cuando usamos un AMD Ryzen 9 7950X 16-Core, cual de ellos hace un uso más eficiente de la cantidad de núcleos que tiene? y cuál es más eficiente en términos de costo ?

3- Realización del tutorial descrito en time profiling adjuntando las conclusiones sobre el uso del tiempo de las funciones.

Marco teórico

Rendimiento de un sistema: es la capacidad que tiene un sistema para realizar un trabajo en un determinado tiempo. Es inversamente proporcional al tiempo, es decir, cuanto mayor sea el tiempo que necesite, menor será el rendimiento.

Los computadores ejecutan las instrucciones que componen los programas, por lo tanto el rendimiento de un computador está relacionado con el tiempo que tarda en ejecutar los programas. De esto se deduce que el tiempo es la medida del rendimiento de un computador.

Medidas de desempeño:

- Ciclos por instrucción (CPI)
- Instrucciones por ciclo (IPC)
- Rendimiento (Throughput)
- Latencia
- Speedup
- Eficiencia

El rendimiento del procesador depende de los siguientes parámetros:

1. *Frecuencia de la CPU (f_{CPU})* : es el número de ciclos por segundo al que trabaja el procesador o CPU.
2. *Periodo de la CPU (T_{CPU})* : es el tiempo que dura un ciclo y es la inversa de la frecuencia de la CPU.
3. *Ciclos por instrucción (CPI)* : las instrucciones se descomponen en microinstrucciones, que son operaciones básicas que se hacen en un ciclo de reloj. En un programa se llama CPI al promedio de microinstrucciones que tienen las instrucciones del programa, es decir, los ciclos de reloj que se tarda de media en ejecutar una instrucción.
4. *Número de instrucciones del programa* : cuantas más instrucciones haya en el programa más tiempo se tarda en ejecutar luego baja el rendimiento. El que tengamos un número reducido de instrucciones dependerá del programador y de que dispongamos de un buen compilador.
5. *Multitarea* : hace referencia a la capacidad que tiene un computador de atender simultáneamente varias tareas.

$$f_{CPU} = \frac{\text{n}^\circ \text{ ciclos}}{\text{segundo}}$$

$$T_{CPU} = \frac{1}{f_{CPU}}$$

$$CPI = \frac{\sum_{i=1}^n N^\circ \text{ Instruc}_i * CPI_i}{N^\circ \text{ InstrucTot}}$$

Rendimiento

$$T_{\text{instrucción}} = CPI * T_{CPU}$$

$$T_{\text{prog}} = N^\circ \text{ instrucciones} * CPI * T_{CPU}$$

$$\eta_{\text{prog}} = \frac{1}{T_{\text{prog}}} = \frac{1}{N^\circ \text{instruc} * CPI * T_{CPU}} = \frac{f_{CPU}}{N^\circ \text{instruc} * CPI} s^{-1}$$

Comparación de desempeño

El speedup es la razón entre el rendimiento de un sistema mejorado y el rendimiento de su implementación original

$$Speedup = \frac{Rendimiento\ Mejorado}{Rendimiento\ Original} = \frac{EX_{CPU\ Original}}{EX_{CPU\ Mejorado}}$$

Eficiencia La eficiencia mide la utilización de un recurso.

- Si Speedup n es la ganancia por mejorar el sistema con n recursos, la eficiencia mide la utilización de esos recursos

$$Eficiencia = \frac{Speedup_n}{n}$$

Programas de prueba o benchmark: La mejor y más fiable forma de calcular el rendimiento es medir el tiempo que los diversos computadores tardan en ejecutar los programas que realmente el usuario va a utilizar posteriormente. Ese será el mejor rendimiento para ese usuario, pero no para todos los usuarios, ya que el rendimiento es un valor relativo de acuerdo con la aplicación que se va a hacer.

El rendimiento de una estación de trabajo se mide analizando una serie de componentes físicos que determinan el rendimiento completo del sistema. A la hora de determinar el rendimiento global de un sistema, también hay que evaluar el sistema operativo, los equipos lógicos de red, los compiladores y las librerías gráficas, etc.

Profiling: herramientas para analizar el tiempo de ejecución del programa/uso de memoria. Los generadores de perfiles de código a menudo se usan para analizar no solo cuánto tiempo tarda en ejecutarse un programa (podemos obtenerlo de herramientas a nivel de shell como `/usr/bin/time`), sino también cuánto tiempo tarda en ejecutarse cada función o método (tiempo de CPU). Dos técnicas principales utilizadas por los perfiladores: inyección de código, muestreo(sampling).

- *inyección de código (gprof):* Estos perfiladores requieren volver a compilar el programa con banderas especiales. En cada punto donde se realiza una llamada/retorno de función, se inserta este código para verificar la hora actual del sistema (hasta microsegundos) y registrar esa información. El programa de

creación de perfiles lee y resume el archivo de registro, generando un informe sobre cuánto tiempo se dedicó a cada función/método, cuántas veces se llamó, etc.

- sampling (perf) Estos requieren privilegios de nivel de sistema operativo. El programa se ejecuta en un modo especial, con interrupciones del sistema operativo generadas a intervalos fijos. En cada una de estas interrupciones, verifica y registra qué función/método está ejecutando el programa actualmente. Esto acumula datos estadísticos sobre cuánto tiempo se dedicó a cada función. El perfilador analiza e informa sobre los datos estadísticos.

Desarrollo

1-

Tareas Diarias		Tipo de Benchmark
Procesamiento de texto	Tiempo de ejecución de un algoritmo de análisis de texto	Programas reales
Navegación por internet	Velocidad de carga de páginas web	Programas reales
Compilación de programas	Tiempo de compilación de un programa	Programas reales
Ejecución de programas	Tiempo de inicio y respuesta del sistema operativo	Benchmark sintético Benchmark reducido
Desarrollo de software	Tiempo de compilación y ejecución de programas	Benchmark sintético Benchmark reducido
Reproducción de multimedia	Rendimiento de decodificación de video	Programas reales





A partir de dichas tareas se puede elegir un procesador que se ajuste mejor a las necesidades específicas y obtener el mejor rendimiento para las tareas que se desea realizar.

2- Se analiza el rendimiento de estos procesadores para compilar el kernel:

- Intel Core i5-13600K

- AMD Ryzen 9 5900X 12-Core

Para compilar el kernel de Linux, es común evaluar el desempeño de un procesador en cuanto a la velocidad de compilación y su eficiencia energética. Se muestra una comparación general de éstos procesadores:

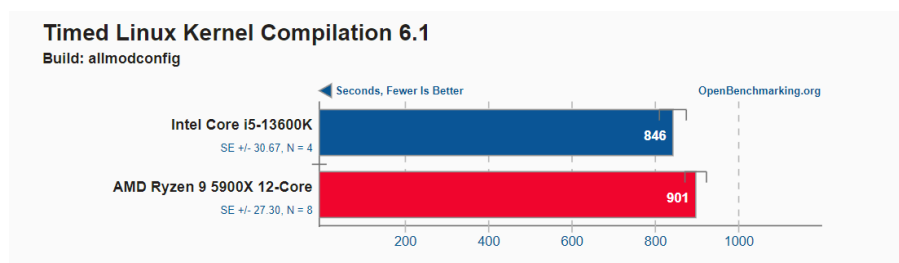
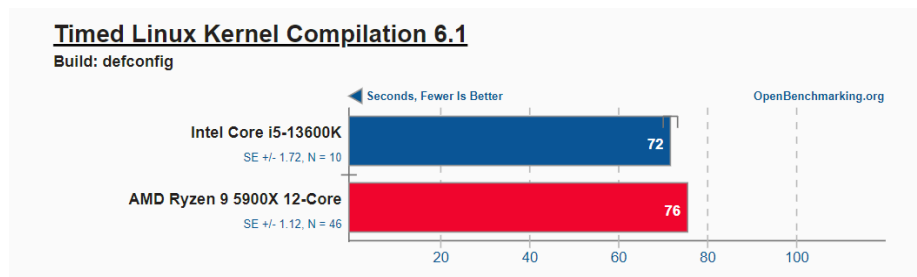
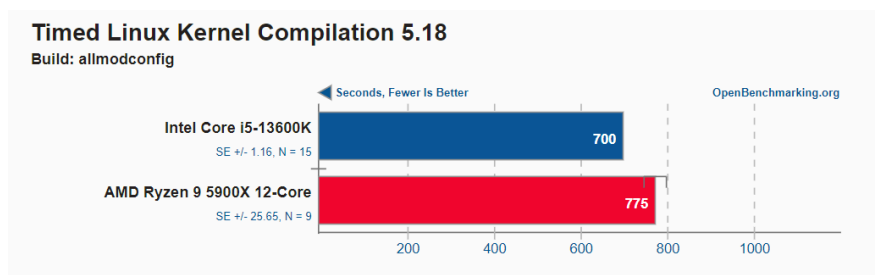
	INTEL CORE I5-13600K	AMD RYZEN 9 5900X 12-CORE
Core Count	14	12
Thread Count	20	24
CPU Clock *	5.1 GHz	3.7 GHz
Core Family	Raptor Lake	Zen 3
First Appeared	2022	2020
Overall Percentile	68th	67th
Total OpenBenchmarking.org Results	7,357	52,686
Qualified Test Combinations	354	599
Performance Per Dollar	320	0
Affiliate Shopping Links *	Amazon	
More Details On OpenBenchmarking.org		
Remove From Comparison		

* Note: Due to differences in Linux drivers/interfaces, the reported CPU clock may be displayed different depending upon the CPU - i.e. base vs. boost frequency.
 * Disclosure: OpenBenchmarking.org may earn a commission/referral on affiliate shopping links should you decide to make any purchases from the third-party shopping links.

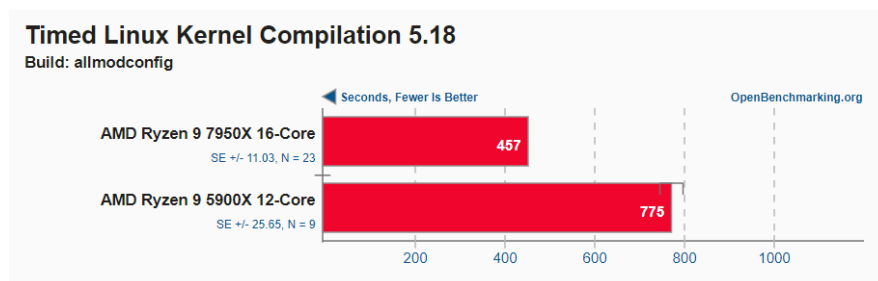
Los resultados obtenidos muestran que el procesador Intel Core i5-13600K tiene un recuento de núcleos de 14 y una frecuencia de reloj de CPU de hasta 5.1 GHz, mientras que el AMD Ryzen 9 5900X 12-Core cuenta con 12 núcleos y una frecuencia de reloj de hasta 3.7 GHz. Aunque el Ryzen tiene un recuento de hilos ligeramente mayor (24 frente a 20), el Core i5 se destaca por su velocidad de reloj más alta. Ambos procesadores pertenecen a familias distintas, con el i5-13600K perteneciendo a la familia Raptor Lake de Intel, mientras que el Ryzen 9 5900X está basado en la arquitectura Zen 3 de AMD.

El Core i5-13600K apareció por primera vez en 2022, mientras que el Ryzen 9 5900X hizo su debut en 2020. En cuanto a los resultados generales, el Core i5-13600K obtuvo un percentil general del 68th, con un total de 7,357 resultados en OpenBenchmarking.org, mientras que el Ryzen 9 5900X alcanzó el 67th percentil, con un total de 52,686 resultados registrados. Ambos procesadores han sido probados en varias combinaciones de pruebas, con el i5-13600K mostrando 354 combinaciones calificadas y el Ryzen 9 5900X con 599 combinaciones calificadas.

Comparación de tiempos de compilación del kernel de Linux:

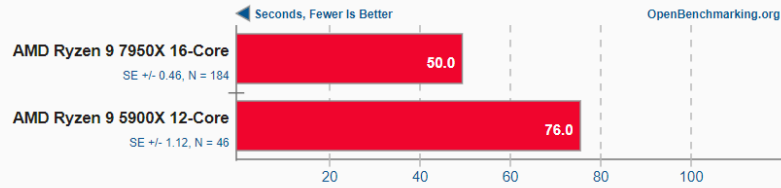


Comparación de tiempos de compilación del kernel de Linux con AMD Ryzen 9 7950X 16-Core en contraste al AMD Ryzen 9 5900X 12-Core:



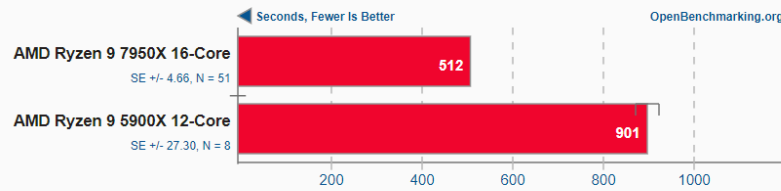
Timed Linux Kernel Compilation 6.1

Build: defconfig



Timed Linux Kernel Compilation 6.1

Build: allmodconfig



$$Rendimiento_{Ryzen\ 9\ 5900x} = \frac{1}{76} \approx 0,0132$$

$$Rendimiento_{Ryzen\ 9\ 7950x} = \frac{1}{50} \approx 0,020$$

$$Rendimiento_{i5\ 13600k} = \frac{1}{72} \approx 0,0139$$

$$Aceleración/Speedup_{Ryzen\ 9\ 5900x/7950x} = \frac{Mejor\ rendimiento}{Peor\ rendimiento} = \frac{0,020}{0,0132} \approx 1,52$$

AMD Procesador de escritorio desbloqueado Ryzen™ 9 7950X de 16 núcleos y 32 hilos

Visita la tienda de AMD

★★★★★ 963 | Buscar en esta página

200+ comprados el mes pasado

-21 % **US\$549⁹⁹**

Precio recomendado: US\$699.00

AMD Procesador de escritorio desbloqueado Ryzen 9 5900X de 12 núcleos y 24 hilos

Visita la tienda de AMD

★★★★★ 10,979 | Buscar en esta página

2K+ comprados el mes pasado

-51 % **US\$278⁵²**

Precio recomendado: US\$569.99

Intel Core i5-13600K Procesador de escritorio 14 (6 núcleos P + 8 núcleos E) con gráficos integrados - desbloqueado

Visita la tienda de Intel

★★★★★ 850 | Buscar en esta página

Opción Amazon Selección general

500+ comprados el mes pasado

-10 % **US\$294⁹⁹**

Precio recomendado: US\$329.00

Para el cálculo de eficiencia con respecto al precio y núcleos, se debe usar el speedup desde el rendimiento base, en el primer caso:

$$Speedup = \frac{72}{72} \rightarrow Eficiencia_{i5\ 13600k} = \frac{1}{\$295} \approx 0,0034$$

$$Speedup = \frac{76}{72} = 1,056 \rightarrow Eficiencia_{Ryzen\ 9\ 5900x} = \frac{1,056}{\$278} \approx 0,0038$$

$$Speedup = \frac{76}{50} = 1,52 \rightarrow Eficiencia_{Ryzen\ 9\ 7950x} = \frac{1,52}{\$550} \approx 0,0027$$

Y con respecto a los núcleos:

$$Speedup = \frac{72}{72} \rightarrow Eficiencia_{i5\ 13600k} = \frac{1}{14} \approx 0,071$$

$$Speedup = \frac{76}{72} = 1,056 \rightarrow Eficiencia_{Ryzen\ 9\ 5900x} = \frac{1,056}{12} \approx 0,0088$$

$$Speedup = \frac{76}{50} = 1,52 \rightarrow Eficiencia_{Ryzen\ 9\ 7950x} = \frac{1,52}{16} \approx 0,0095$$

Donde podemos observar que en relación a eficiencia/precio el mejor tiende a ser el Ryzen 9 5900x, y en cuanto a los núcleos, el Ryzen 9 7950x.

3 - **Time Profiling**

Time Profiling con gprof

Se utiliza la herramienta gprof. Para ello es necesario:

- Habilitar profiling durante la compilación.
- Ejecutar el código del programa para producir los datos de perfil.
- Ejecutar la herramienta gprof en el archivo de datos de generación de perfiles.

Este último paso finaliza con la generación de un archivo legible para las personas. Este archivo, contiene un par de tablas (perfil plano y gráfico de llamadas) así como otros detalles adicionales. El perfil plano proporciona una descripción general de tiempo de las funciones, la cantidad de veces que se llaman, etc. El gráfico de llamadas muestra para cada función, que funciones la llaman, que funciones llama y cuantas veces.

Se usa un código de ejemplo test_gprof.c y test_gprof_new.c que dejamos evidenciados en el repositorio de git.

Paso 1: creación de perfiles habilitada durante la compilación

En este primer paso, debemos asegurarnos de que la generación de perfiles esté habilitada cuando se complete la compilación del código. Esto es posible al agregar la opción '-pg' en el paso de compilación.

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

Paso 2: Ejecutar el código

```
$ ./test_gprof
```

```
Inside main()
Inside func1
Inside new_func1()
Inside func2
```

Esto genera un archivo llamado gmon.out en el directorio de trabajo actual.

Paso 3: Ejecutar la herramienta gprof

En este paso, la herramienta gprof se ejecuta con el nombre del ejecutable y el 'gmon.out' generado anteriormente como argumento. Esto produce un archivo de análisis que contiene toda la información de perfil deseada.

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

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
34.26	7.90	7.90	1	7.90	15.56	func1
33.22	15.56	7.66	1	7.66	7.66	new_func1
32.18	22.98	7.42	1	7.42	7.42	func2
0.35	23.06	0.08				main

```

Call graph (explanation follows)

granularity: each sample hit covers 4 byte(s) for 0.04% of 23.06 seconds

index % time    self  children  called  name
-----
[1]   100.0     0.08   22.98      1/1    <spontaneous>
      7.90     7.66      1/1    main [1]
      7.42     0.00      1/1    func1 [2]
      7.42     0.00      1/1    func2 [4]
-----
[2]    67.5     7.90     7.66      1/1    main [1]
      7.90     7.66      1/1    func1 [2]
      7.66     0.00      1/1    new_func1 [3]
-----
[3]    33.2     7.66     0.00      1/1    func1 [2]
      7.66     0.00      1/1    new_func1 [3]
-----
[4]    32.2     7.42     0.00      1/1    main [1]
      7.42     0.00      1/1    func2 [4]
-----

```

Customizar la info con flags:

Suprima la impresión de funciones declaradas estáticamente (privadas) usando -a

\$ gprof -a test_gprof gmon.out > analysis.txt

```

Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self           calls   self   total    name
time   seconds    seconds             s/call  s/call  s/call
66.44    15.32    15.32              2     7.66    11.49  func1
33.22    22.98     7.66              1     7.66     7.66  new_func1
 0.35    23.06     0.08                      0.00     0.00     0.00  main

Call graph (explanation follows)

granularity: each sample hit covers 4 byte(s) for 0.04% of 23.06 seconds

index % time    self  children  called  name
-----
[1]   100.0     0.08   22.98      2/2    <spontaneous>
      15.32     7.66      2/2    main [1]
      15.32     7.66      2/2    func1 [2]
-----
[2]    99.7    15.32     7.66      2/2    main [1]
      15.32     7.66      2/2    func1 [2]
      7.66     0.00      1/1    new_func1 [3]
-----
[3]    33.2     7.66     0.00      1/1    func1 [2]
      7.66     0.00      1/1    new_func1 [3]
-----

```

Elimine los textos detallados usando -b e imprimir solo perfil plano con -p

```
$ gprof -p -b test_gprof gmon.out > analysis.txt
```

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
34.26	7.90	7.90	1	7.90	15.56	func1
33.22	15.56	7.66	1	7.66	7.66	new_func1
32.18	22.98	7.42	1	7.42	7.42	func2
0.35	23.06	0.08				main

Imprimir información relacionada con funciones específicas en perfil plano

```
$ gprof -pfunc1 -b test_gprof gmon.out > analysis.txt
```

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
100.00	7.90	7.90	1	7.90	7.90	func1

Interpretación de la salida gprof

El perfil plano muestra la cantidad total de tiempo dedicado a la ejecución del programa de cada función. Las funciones se ordenan por orden decreciente por tiempo de ejecución empleado en ellas. En el encabezado aparece una declaración que indica cuánto tiempo equivale una muestra. Significado de los campos:

- '% time' Este es el porcentaje del tiempo total de ejecución que el programa dedicó a cada función. Estos deben sumar el 100% entre todos.
- 'cumulative seconds' Este es el número total de segundos acumulados que el ordenador dedicó a ejecutar estas funciones, más el tiempo dedicado a todas las funciones anteriores a esta en esta tabla.
- 'self seconds' Este es el número de segundos contabilizados a esta función sola. El listado del perfil plano se ordena por este número.
- 'calls' Este es el número total de veces que se invocó la función. Si la función no se llamó nunca, o el número de veces que fue llamado no se puede determinar (probablemente porque la función no se compiló con el perfilado habilitado), el campo de las llamadas ('calls') permanecerá en blanco.

- 'self ms/call' Esta cifra representa el número de milisegundos de media gastados en esta función por cada llamada, si esta función es perfilada. De lo contrario, este campo permanecerá en blanco para esta función.
- 'total ms/call' total de ms / llamada Est cifra representa el número de milisegundos de media gastados en esta función y sus descendientes por cada llamada.

El grafo de llamadas muestra qué funciones son llamadas por qué otras, y cuanto tiempo cada función usó cuando fue llamada por éstas. Las líneas de guiones dividen esta tabla en 'entradas', una para cada función. Cada entrada tiene una o más líneas. En cada entrada, la línea principal es la que comienza con un número de índice entre corchetes. Al final de esta línea aparece el nombre de la función a la que esta entrada se refiere. Las líneas anteriores de la entrada describen a los llamadores de esta función y las líneas siguientes describen sus subrutinas (también llamados 'hijos' cuando se habla del grafo de llamadas). Las entradas se ordenan por el tiempo gastado en la función y sus subrutinas.

- 'index' Las entradas están numeradas con enteros consecutivos. Por lo tanto, cada función tiene un número de índice, que aparece al principio de su línea principal.
- '% time' Este es el porcentaje del tiempo total que se gastó en esta función, incluido el tiempo dedicado en las subrutinas llamadas desde esta función.
- 'self' Esta es la cantidad total de tiempo dedicado a esta función. Esta debe ser idéntica al número impreso en el campo 'seconds' (segundos) de esta función en el perfil plano.
- 'children' Esta es la cantidad total de tiempo dedicado a las llamadas a subrutinas realizadas por esta función. Esta debe ser igual a la suma de todos los campos 'self' y 'children' de todos los hijos que aparecen justo debajo de esta función
- 'called' Este es el número de veces que se llama a la función.

Genere un gráfico

Utilizamos prof2dot que es una herramienta que puede crear una visualización de la salida de gprof.

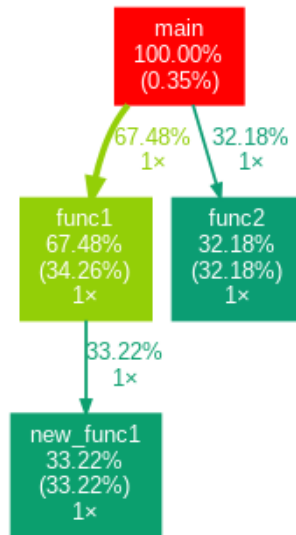
Install's:

\$ pip install gprof2dot

\$ sudo apt install graphviz

Generar el .dot y usar graphviz para el .png:

```
$ gprof -b test_gprof gmon.out | python3 -m gprof2dot -n0 -e0 -o output.dot
$ dot -Tpng output.dot -o output.png
```



Time Profiling con Linux Perf

Perf es una herramienta para crear perfiles de programas, usa perfiles estadísticos donde sondea el programa y ve qué función está funcionando. Es menos preciso pero tiene menos impacto en el rendimiento

Paso 1: Instalación de perf

El programa perf no viene preinstalado en los sistemas Linux, puede instalarlo como se muestra a continuación en Ubuntu:

```
$ sudo apt install linux-tools-common
$ sudo apt install linux-tools-5.15.0-100-generic
```

Paso 2: Registrar ciclos de CPU

El comando perf de forma predeterminada requiere privilegios sudo para permitir que los usuarios lo utilicen.

- perf record: escribir eventos en un archivo.
- perf report: para explorar el archivo grabado.

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

La grabación guarda los datos en un archivo perf.data.

Paso 3: Ver resultados de rendimiento

\$ sudo perf report

Samples: 114K of event 'cycles', Event count (approx.): 87961029263

Overhead	Command	Shared Object	Symbol
35.46%	test_gprof	test_gprof	[.] new_func1
34.06%	test_gprof	test_gprof	[.] func2
29.94%	test_gprof	test_gprof	[.] func1
0.11%	test_gprof	test_gprof	[.] main
0.04%	test_gprof	[kernel.kallsyms]	[k] delay_tsc
0.02%	test_gprof	[kernel.kallsyms]	[k] error_entry
0.01%	test_gprof	[kernel.kallsyms]	[k] pvclock_gtod_notify
0.01%	test_gprof	[kernel.kallsyms]	[k] update_vsyscall
0.01%	test_gprof	[kernel.kallsyms]	[k] timerqueue_add
0.01%	test_gprof	[kernel.kallsyms]	[k] __hrtimer_run_queues
0.01%	test_gprof	[kernel.kallsyms]	[k] __irqentry_text_end
0.01%	test_gprof	[kernel.kallsyms]	[k] __setup_rt_frame
0.01%	test_gprof	[kernel.kallsyms]	[k] hrtimer_interrupt
0.01%	test_gprof	[kernel.kallsyms]	[k] ioread32
0.01%	test_gprof	[kernel.kallsyms]	[k] get_signal
0.01%	test_gprof	[kernel.kallsyms]	[k] memcg_slab_post_alloc_hook
0.01%	test_gprof	[kernel.kallsyms]	[k] update_load_avg
0.01%	test_gprof	[kernel.kallsyms]	[k] account_user_time
0.01%	test_gprof	[kernel.kallsyms]	[k] psi_group_change
0.01%	test_gprof	[kernel.kallsyms]	[k] hrtimer_active
0.01%	test_gprof	[kernel.kallsyms]	[k] raw_notifier_call_chain
0.01%	test_gprof	libc.so.6	[.] __profil_counter
0.01%	test_gprof	[kernel.kallsyms]	[k] task_tick_fair
0.01%	test_gprof	[kernel.kallsyms]	[k] syscall_exit_to_user_mode
0.01%	test_gprof	[kernel.kallsyms]	[k] __cgroup_account_cputime_field
0.01%	test_gprof	[kernel.kallsyms]	[k] update_vdso_data.constprop.0
0.01%	test_gprof	[kernel.kallsyms]	[k] timekeeping_advance