

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias
Análisis y Diseño 2
Ing. Ricardo Morales
Aux. Kenny Eguizábal



Práctica2: Manual de GPROF

Virginia Chavarría Guzmán
200920081
Guatemala, 19 Abril de 2016

GPROF

HISTORIA

GPROF originalmente fue escrito por un grupo liderado por Sunsan L. Graham en la universidad de California, Berkeley para Berkeley unix, otra implementación fue escrita como parte del proyecto GNU por GNU Binutils en 1988 por Jay Fenlason

¿QUE ES?

GPROF es una herramienta de código abierto que puede mejorar el rendimiento del código fuente de los programas. Puede ayudar a identificar en que parte del código está el problema para que se pueda corregir. GNU GPROF este puede ayudar a identificar los cuellos de botella (“bottlenecks”) que se han generado al escribir un programa para así optimizar el rendimiento.

PARA QUE SIRVE

La herramienta GPROF nos proporciona perfiles de la ejecución de un programa, es decir, nos da información acerca de cuánto tiempo se emplea en cada función y de cuántas veces se llama. Sirve para detectar dónde el programa está invirtiendo la mayor parte de su tiempo de ejecución.

Esta herramienta nos muestra dos tipos de perfiles que son los siguientes:

- **Perfil plano:** Para cada función proporciona el tiempo y el número de veces que se llama cada función.

Flat profile:

Each sample counts as 0.01 seconds.

%	cumulative	self		self	total	
time	seconds	seconds	calls	us/call	us/call	name
48.79	5.03	5.03	110	45727.27	45727.27	h(int)
48.79	10.06	5.03	100	50300.00	96027.27	g(void)
2.42	10.31	0.25	10	25000.00	70727.27	f(void)

- **Call Graph:** Lo mismo que la anterior, pero con información acerca de que funciones la llama y que funciones llama.

index	% time	self	children	called	name
[1]	100.0	0.00	10.31		<spontaneous> main [1]
		5.04	4.58	100/100	g(void) [2]
		0.23	0.46	10/10	f(void) [4]
[2]	93.3	5.04	4.58	100/100	main [1]
		5.04	4.58	100	g(void) [2]
		4.58	0.00	100/110	h(int) [3]
		0.46	0.00	10/110	f(void) [4]
		4.58	0.00	100/110	g(void) [2]
[3]	48.9	5.04	0.00	110	h(int) [3]
[4]	6.7	0.23	0.46	10/10	main [1]
		0.23	0.46	10	f(void) [4]
		0.46	0.00	10/110	h(int) [3]

USO

Usaremos para este ejemplo el O.S Fedora 23.

Primero que nada se debe tener instalado GCC de no tenerlo se instala de la siguiente manera

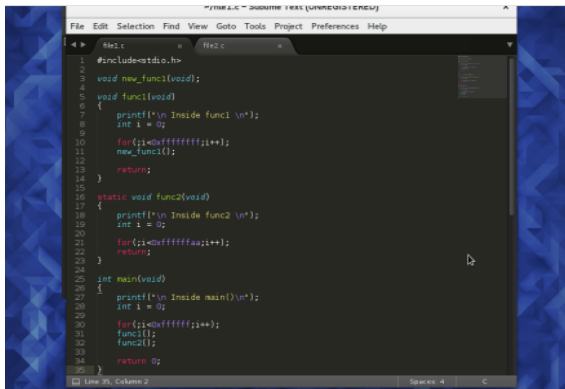
```

Actividades Terminal mar 21:56
virlinia_200920081@localhost:~$ sudo dnf install gcc
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

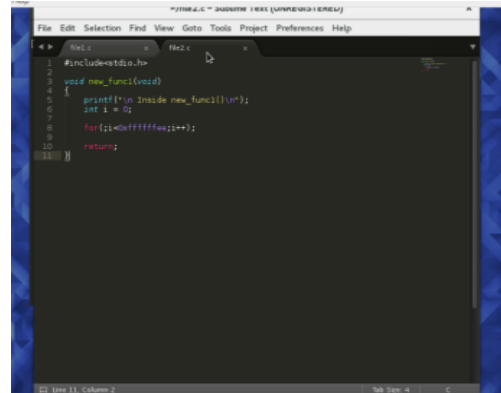
#1) Respect the privacy of others.
#2) Think before you type.
#3) With great power comes great responsibility.

[sudo] password for virlinia_200920081:
Fedora 23 - x86_64 1% [ ] 234 kB/s | 822 kB 03:04 ETA
  
```

Teniendo instalado GCC se procedera a usar GPROF, para esto se tiene 2 archivos File1.c y file2.c en las funciones los “for” se usan para consumir un poco de tiempo de ejecución.

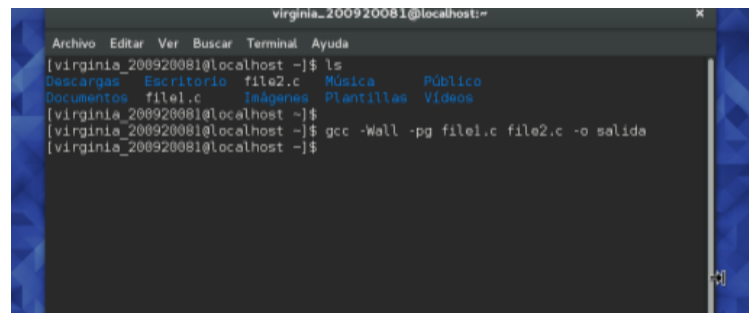


```
1 //file.c -> gmon.out (unmodified)
2
3 #include <stdio.h>
4
5 void new_func1(void);
6
7 void func1(void)
8 {
9     printf("Inside func1\n");
10    int i = 0;
11    for(i=0; i<0xffff; i++);
12    new_func1();
13    return;
14 }
15
16 static void func2(void)
17 {
18     printf("Inside func2\n");
19     int i = 0;
20     for(i=0; i<0xffff; i++);
21     return;
22 }
23
24 int main(void)
25 {
26     printf("Inside main\n");
27     int i = 0;
28     for(i=0; i<0xffff; i++);
29     func1();
30     return 0;
31 }
```



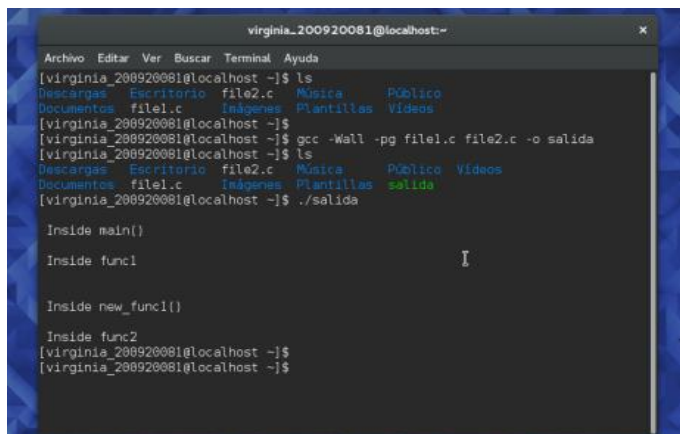
```
1 //file.c -> gmon.out (unmodified)
2
3 #include <stdio.h>
4
5 void new_func1(void);
6
7 void func1(void)
8 {
9     printf("Inside new_func1\n");
10    int i = 0;
11    for(i=0; i<0xffff; i++);
12    return;
13 }
```

Notese que para generar codigo extra para analizar el programa con GPROF se añade “-pg” al momento de compilar:

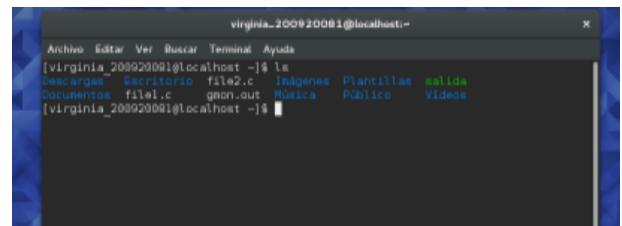


```
virginia_200920081@localhost:~$ ls
Descargas  Escritorio  file2.c  Musica  Publico
Documentos file1.c     Imágenes Plantillas Videos
[virginia_200920081@localhost ~]$ gcc -Wall -pg file1.c file2.c -o salida
[virginia_200920081@localhost ~]$
```

Luego ejecutamos la salida así “./salida”, al ejecutar estos notamos que este nos ha generado un archivo llamado gmon.out en el directorio donde se encuentra nuestro proyecto.

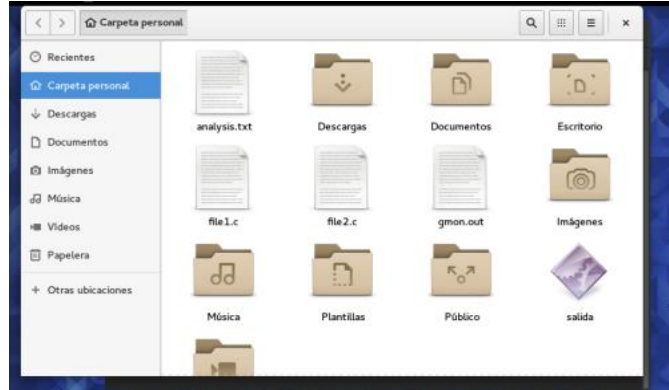
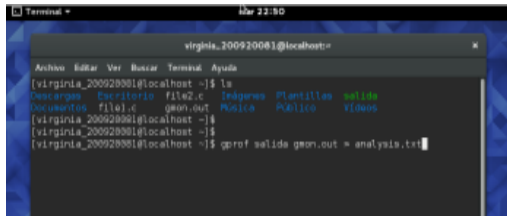


```
virginia_200920081@localhost:~$ ./salida
Inside main()
Inside func1
Inside new_func1()
Inside func2
[virginia_200920081@localhost ~]$
```



```
virginia_200920081@localhost:~$ ./salida
Descargas  Escritorio  file2.c  Imágenes Plantillas salida
Documentos file1.c     gmon.out Musica  Publico  Videos
[virginia_200920081@localhost ~]$
```

Par poder correr GPROF se usa gmon.out para generar un archivo de análisis del programa



Como se mencionó anteriormete GPROF nos generará dos tipos de perfiles el flat profile y call graph en el archivo se encuentra explicado a detalle.

```

1  Flat profile:
2
3  Each sample counts as 0.01 seconds.
4
5  % cumulative   self           total
6  time   seconds   seconds   calls   s/call   s/call   name
7  35.16      1.21      1.21         1      1.21      1.21  new_func1
8  31.65      2.31      1.09         1      1.09      1.09  func2
9  20.95      3.03      0.72         1      0.72      1.94  func1
10
11 %           the percentage of the total running time of the
12 time        program used by this function.
13
14 accumulative a running sum of the number of seconds accounted
15 seconds      for by this function and those listed above it.
16
17 self         the number of seconds accounted for by this
18 seconds      function alone. This is the major sort for this
19              listing.
20
21 calls        the number of times this function was invoked, if
22              this function is profiled, else blank.
23
24 self         the average number of milliseconds spent in this
25 ms/call      function per call, if this function is profiled,
26              else blank.
27
28 total        the average number of milliseconds spent in this
29 ms/call      function and its descendants per call, if this
30              function is profiled, else blank.
31
32 name         the name of the function. This is the minor sort
33              for this listing. The index shows the location of
34              the function in the gprof listing. If the index is
35              in parenthesis it shows where it would appear in
36              the gprof listing if it were to be printed.

```

```

1      Call graph (explanation follows)
2
3      granularity: each sample hit covers 2 byte(s) for 0.33% of 3.03 seconds
4
5      index % time    self    children    called    name
6
7      [1]    100.0    0.00    3.03
8      main [1]
9      0.72    1.21    1/1    func1 [2]
10     1.09    0.00    1/1    func2 [4]
11     -----
12     0.72    1.21    1/1    main [1]
13     0.72    1.21    1    func1 [2]
14     1.21    0.00    1/1    new_func1 [3]
15     -----
16     1.21    0.00    1/1    func1 [2]
17     1.21    0.00    1    new_func1 [3]
18     -----
19     1.09    0.00    1/1    main [1]
20     1.09    0.00    1    func2 [4]
21     -----
22
23     This table describes the call tree of the program, and was sorted by
24     the total amount of time spent in each function and its children.

```

Si queremos modificar nuestras salidas gprof nos ofrece varias flags que podremos usar como:

- a: Suprime la salida de un método static/private

```
[virginia_200920081@localhost ~]$ gprof salida -a -b gmon.out > analysis2.txt
[virginia_200920081@localhost ~]$
```

```

Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self           self      total
time  seconds    seconds   calls   s/call   s/call   name
52.60      1.81      1.81        2     0.91    1.51   func1
35.16      3.03      1.21        1     1.21    1.21  new_func1

Call graph

granularity: each sample hit covers 2 byte(s) for 0.33% of 3.03 seconds

index % time    self   children    called    name
-----
[1]   100.0      1.81    1.21        2/2        main [2]
      1.81    1.21        2        func1 [1]
      1.21    0.00        1/1        new_func1 [3]
-----
[2]   100.0      0.00    3.03         2/2        <spontaneous>
      1.81    1.21         2/2        main [2]
      1.81    1.21         2        func1 [1]
-----
[3]   40.1       1.21    0.00         1/1        func1 [1]
      1.21    0.00         1        new_func1 [3]
-----

Index by function name

[1] func1                [3] new_func1

```

Que a nuestra salida nos muestra lo siguiente:

Y que como podemos ver se ha suprimido el método fun2 el cual era estatico.

-b: Suprime toda la documentación inecesaria usando

```

[virginia_200920081@localhost ~]$ gprof -b salida gmon.out > analysis2.txt
[virginia_200920081@localhost ~]$

```

```

Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self           self      total
time  seconds    seconds   calls   s/call   s/call   name
35.16      1.21      1.21        1     1.21    1.21  new_func1
31.65      2.31      1.09        1     1.09    1.09  func2
20.95      3.03      0.72        1     0.72    1.94  func1

Call graph

granularity: each sample hit covers 2 byte(s) for 0.33% of 3.03 seconds

index % time    self   children    called    name
-----
[1]   100.0      0.00    3.03         1/1        <spontaneous>
      0.72    1.21         1/1        main [1]
      0.72    1.21         1/1        func1 [2]
      1.09    0.00         1/1        func2 [4]
-----
[2]   63.9       0.72    1.21         1/1        main [1]
      0.72    1.21         1/1        func1 [2]
      1.21    0.00         1/1        new_func1 [3]
-----
[3]   40.1       1.21    0.00         1/1        func1 [2]
      1.21    0.00         1        new_func1 [3]
-----
[4]   36.1       1.09    0.00         1/1        main [1]
      1.09    0.00         1        func2 [4]
-----

Index by function name

[2] func1                [4] func2                [3] new_func1

```

-p: Solo muestra el flat profile

```
[virginia_200920081@localhost ~]$ gprof -p salida gmon.out > analysis2.txt  
[virginia_200920081@localhost ~]$
```

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
35.16	1.21	1.21	1	1.21	1.21	new_func1
31.65	2.31	1.09	1	1.09	1.09	func2
20.95	3.03	0.72	1	0.72	1.94	func1

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

cumulative a running sum of the number of seconds accounted
seconds 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.

0 se puede mezclar para mostrar solo el flat profile sin la documentacion extra usando

```
>gprof -p -b salida gmon.out > analysis2.txt
```

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
35.16	1.21	1.21	1	1.21	1.21	new_func1
31.65	2.31	1.09	1	1.09	1.09	func2
20.95	3.03	0.72	1	0.72	1.94	func1

Si se desea mostrar la información de X función se muestra así:

```
>gprof -pfunc1 -b salida gmon.out > analysis2.txt
```

```
[virginia_200920081@localhost ~]$ gprof salida -pfunc1 -b gmon.out > analysis2.txt  
[virginia_200920081@localhost ~]$
```

Flat profile:

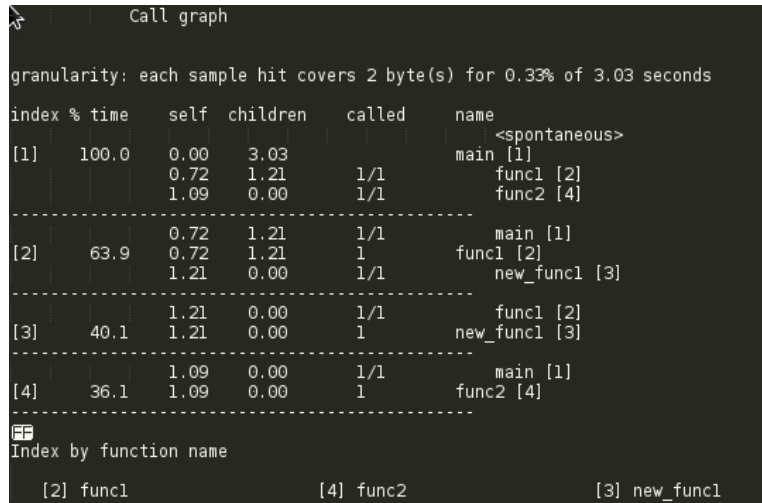
Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
63.13	0.72	0.72	1	722.86	722.86	func1

Para suprimir del archivo el flat profile se utiliza -P

```
>gprof -P -b salida gmon.out > analysis2.txt
```

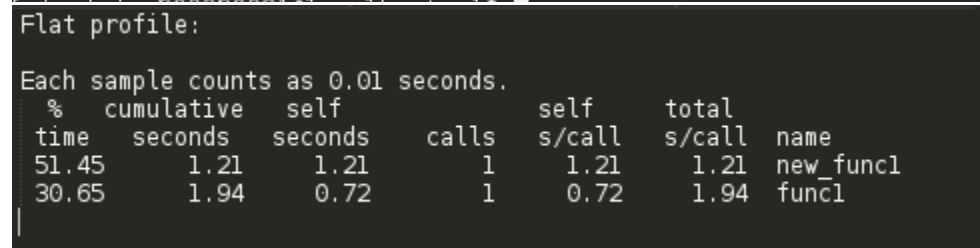
```
[virginia_200920081@localhost ~]$ gprof salida -P -b gmon.out > analysis2.txt  
[virginia_200920081@localhost ~]$
```



Para excluir una cierta función del flat profile seria :

```
>gprof -Pfunc2 -b salida gmon.out > analysis2.txt
```

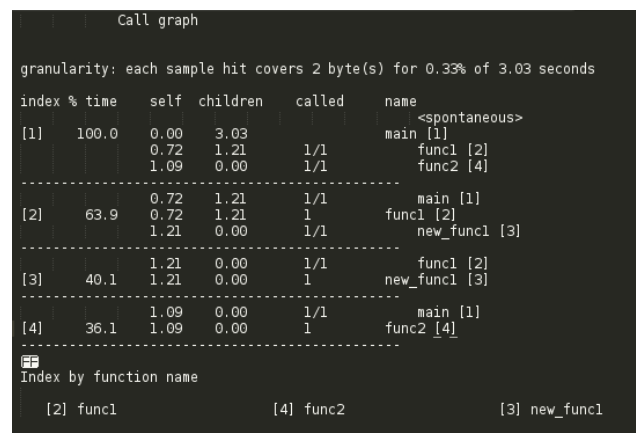
```
[virginia_200920081@localhost ~]$ gprof salida -Pfunc2 -b gmon.out > analysis2.txt
```



Para imprimir unicamente el call graph se utiliza -q

```
>gprof -q -b salida gmon.out > analysis2.txt
```

```
[virginia_200920081@localhost ~]$ gprof salida -q -b gmon.out > analysis2.txt
```



Para imprimir cierta coincidencia con el nombre de una función del call graph


```
>gprof -qfunc1 -b salida gmon.out > analysis2.txt
```

```
[virginia_200920081@localhost ~]$ gprof salida -qfunc1 -b gmon.out > analysis2.txt
```

```
Call graph

granularity: each sample hit covers 2 byte(s) for 0.33% of 3.03 seconds

index % time    self  children   called    name
-----
[2]    63.9      0.72    1.21      1/1      main (1)
      0.72    1.21      1        func1 [2]
      1.21    0.00      1/1      new_func1 [3]
-----
[3]    40.1      1.21    0.00      1/1      func1 [2]
      1.21    0.00      1        new_func1 [3]
-----
Index by function name

      [2] func1                (4) func2                [3] new_func1
```

Para suprimir el call graph de la salida se utiliza -Q

```
>gprof -Q -b salida gmon.out > analysis2.txt
```

```
[virginia_200920081@localhost ~]$ gprof salida -Q -b gmon.out > analysis2.txt
```

```
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self           self          total       name
time  seconds    seconds    calls   s/call   s/call   name
-----
35.16    1.21      1.21         1      1.21     1.21  new_func1
31.65    2.31      1.09         1      1.09     1.09  func2
20.95    3.03      0.72         1      0.72     1.94  func1
```

Para suprimir cierta función del Call graph se usa -Qnombre

```
>gprof -Qfunc1 -b salida gmon.out > analysis2.txt
```

```
[virginia_200920081@localhost ~]$ gprof salida -Qfunc1 -b gmon.out > analysis2.txt
```

```
granularity: each sample hit covers 2 byte(s) for 0.33% of 3.03 seconds

index % time    self  children   called    name
-----
[1]   100.0      0.00    3.03      1/1      <spontaneous>
      0.72    1.21      1/1      main [1]
      1.09    0.00      1/1      func1 (2)
      1.09    0.00      1/1      func2 [4]
-----
[3]    40.1      1.21    0.00      1/1      func1 (2)
      1.21    0.00      1        new_func1 [3]
-----
[4]    36.1      1.09    0.00      1/1      main [1]
      1.09    0.00      1        func2 [4]
-----
Index by function name

      (2) func1                [4] func2                [3] new_func1
```

EXPLICACIÓN

A continuación se explicará que es lo que muestra cada columna de los despliegues de GPROF tanto de flat profile como de Call graph.

Flat profile:

%time: el porcentaje del tiempo total de ejecución del programa utilizado por esta función.

Cumulative seconds: suma acumulada del número en segundos usados por cada función.

Self seconds: el número en segundos de tiempo usados por cada función individualmente.

Calls: el número de veces que esta función fue invocada, si esta función fue perfilada entonces la muestra en blanco.

Self s/call: el promedio de número en milisegundos gastados por cada función, si la función fue perfilada entonces lo muestra en blanco.

Total s/call: el promedio de número de milisegundos gastados por la función y sus hijos, al igual que anterior si esta es perfilada entonces lo muestra en blanco.

Name: nombre la función.

Call graph:

Cada entrada en esta tabla consiste en varias líneas.

Index: un número único dado a cada elemento de la tabla, estos son ordenados numéricamente, este es impreso.

Self: este es el total de tiempo gastado en esta función.

%time: tiempo en milisegundos gastados en la ejecución de cada función y sus hijos.

Children: este es número total de tiempo propagado en esta función por sus hijos.

Called: es el número de veces que la función fue llamada. Si esta función es llamada recursivamente solo incluirá las llamadas no recursivas.

Name: El nombre de la función actual, el index es impreso después del nombre, si esta función es un miembro de un ciclo, el número del ciclo es impreso entre el nombre de la función y el index de esta.

Si hay ciclos en esta tabla, entonces habrá una entrada para el ciclo como un todo, esta muestra quien la llamó (padre) y los miembros del ciclo (hijos)