DTS202TC Fundamentals of Parallel Computing

Tutorial 2: Introduction of GPROF

Xi’an Jiaotong-Liverpool University
西交利物浦大学

GPROF: LINUX GNU GCC PROFILING TOOL

Profiling is an important aspect of software programming. Through profiling one can determine the parts in program code that are time consuming and need to be re-written. This helps make your program execution faster which is always desired.

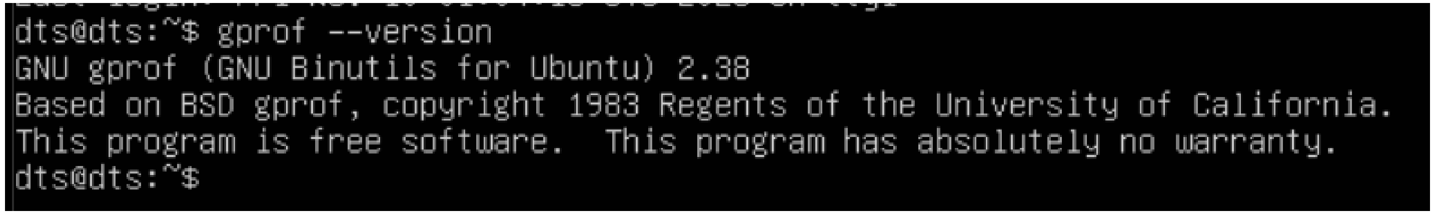
This tutorial is based on this website, for more detailed information please visit: https://www.thegeekstuff.com/2012/08/gprof-tutorial/

How to use GROF

- Have profiling enabled while compiling the code

- Execute the program code to produce the profiling data

- Run the gprof tool on the profiling data file (generated in the step above).



```
dts@dts:~$ 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.
dts@dts:~$
```

GPROF has been installed in the virtual machine environment, and you can check its version by using “gprof --version.”

Example program
test_gprof_new.c

```
#include<stdio.h>

void new_func1(void)
{
    printf("\n Inside new_func1()\n");
    int i = 0;

    // Do a loop to increase the delay effect
    for(;i<0xffffffee;i++);

    return;
}
```

Example program test_gprof.c

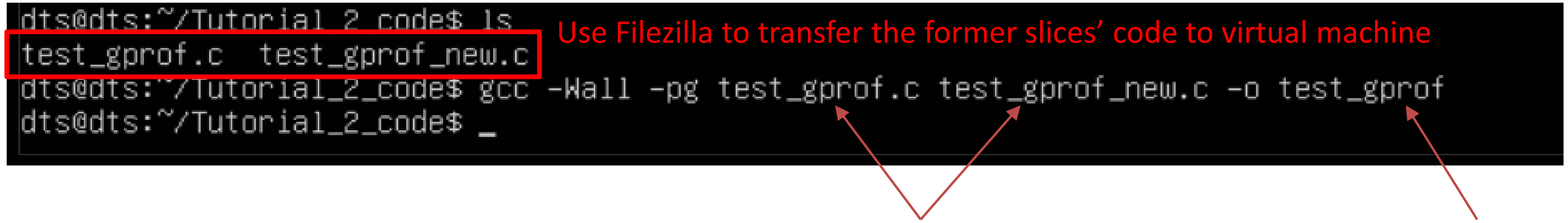
```
#include<stdio.h>

void new_func1(void);

void func1(void)
{
    printf("\n Inside func1 \n");
    int i = 0;

    // Do a loop to increase the delay
effect
    for(;i<0xffffffff;i++);
    new_func1();

    return;
}
```

```
static void func2(void)
{
    printf("\n Inside func2 \n");
    int i = 0;
    // Do a loop to increase the delay effect
    for(;i<0xffffffaa;i++);
    return;
}

int main(void)
{
    printf("\n Inside main()\n");
    int i = 0;
    // Do a loop to increase the delay effect
    for(;i<0xffffff;i++);

    func1();
    func2();

    return 0;
}
```

Step-1 : Profiling enabled while compilation

In this first step, we need to make sure that the profiling is enabled when the compilation of the code is done. This is made possible by adding the '-pg' option in the compilation step.

What is "-pg" doing:
Generate extra code to write profile information suitable for the analysis program gprof. You must use this option when compiling the source files you want data about, and you must also use it when linking.



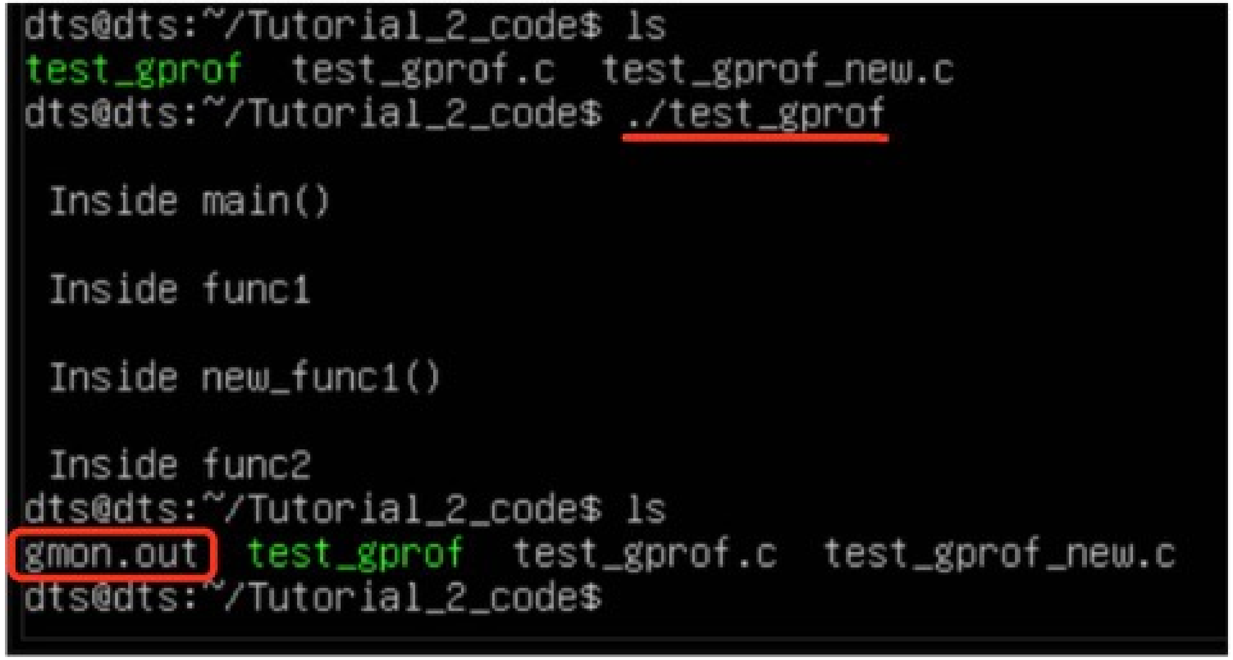
```
dts@dts:~/Tutorial_2_code$ ls
test_gprof.c  test_gprof_new.c
dts@dts:~/Tutorial_2_code$ gcc -Wall -pg test_gprof.c test_gprof_new.c -o test_gprof
dts@dts:~/Tutorial_2_code$ _
```

Use Filezilla to transfer the former slices' code to virtual machine

The file we want to compile

The name of object file

Step-2 : Execute the code

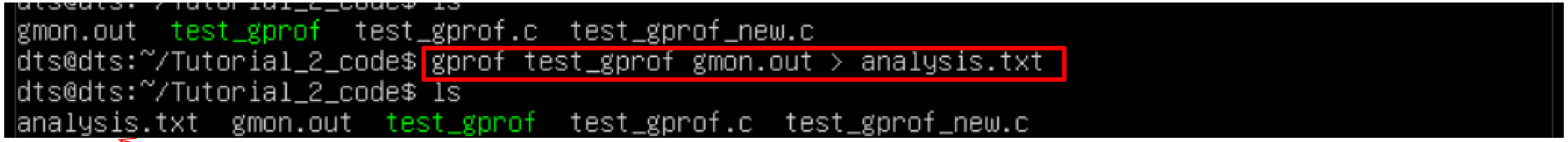


```
dts@dts:~/Tutorial_2_code$ ls
test_gprof  test_gprof.c  test_gprof_new.c
dts@dts:~/Tutorial_2_code$ ./test_gprof

 Inside main()

 Inside func1

 Inside new_func1()

 Inside func2
dts@dts:~/Tutorial_2_code$ ls
gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
dts@dts:~/Tutorial_2_code$
```

In the second step, the binary file(test_gprof) produced as a result of step-1 (above) is executed so that profiling information(gmon.out) can be generated.

Step-3 : Run the gprof tool

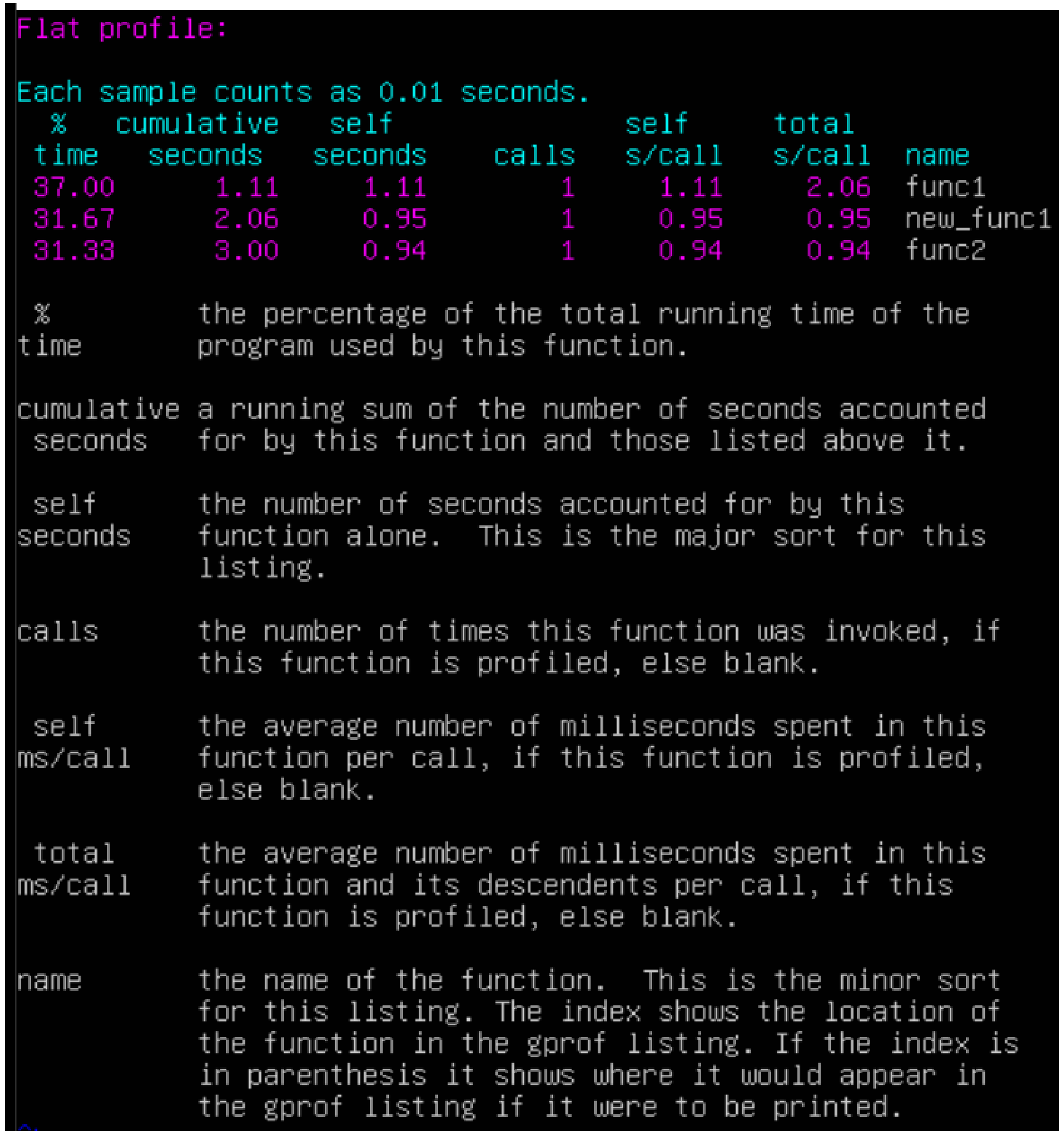
In this step, the gprof tool is run with the executable name (test_gprof) and the above generated ‘gmon.out’ as argument



```
dts@dts:~/Tutorial_2_code$ ls
gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
dts@dts:~/Tutorial_2_code$ gprof test_gprof gmon.out > analysis.txt
dts@dts:~/Tutorial_2_code$ ls
analysis.txt  gmon.out  test_gprof  test_gprof.c  test_gprof_new.c
```

Note that one can explicitly specify the output file (like in example above).

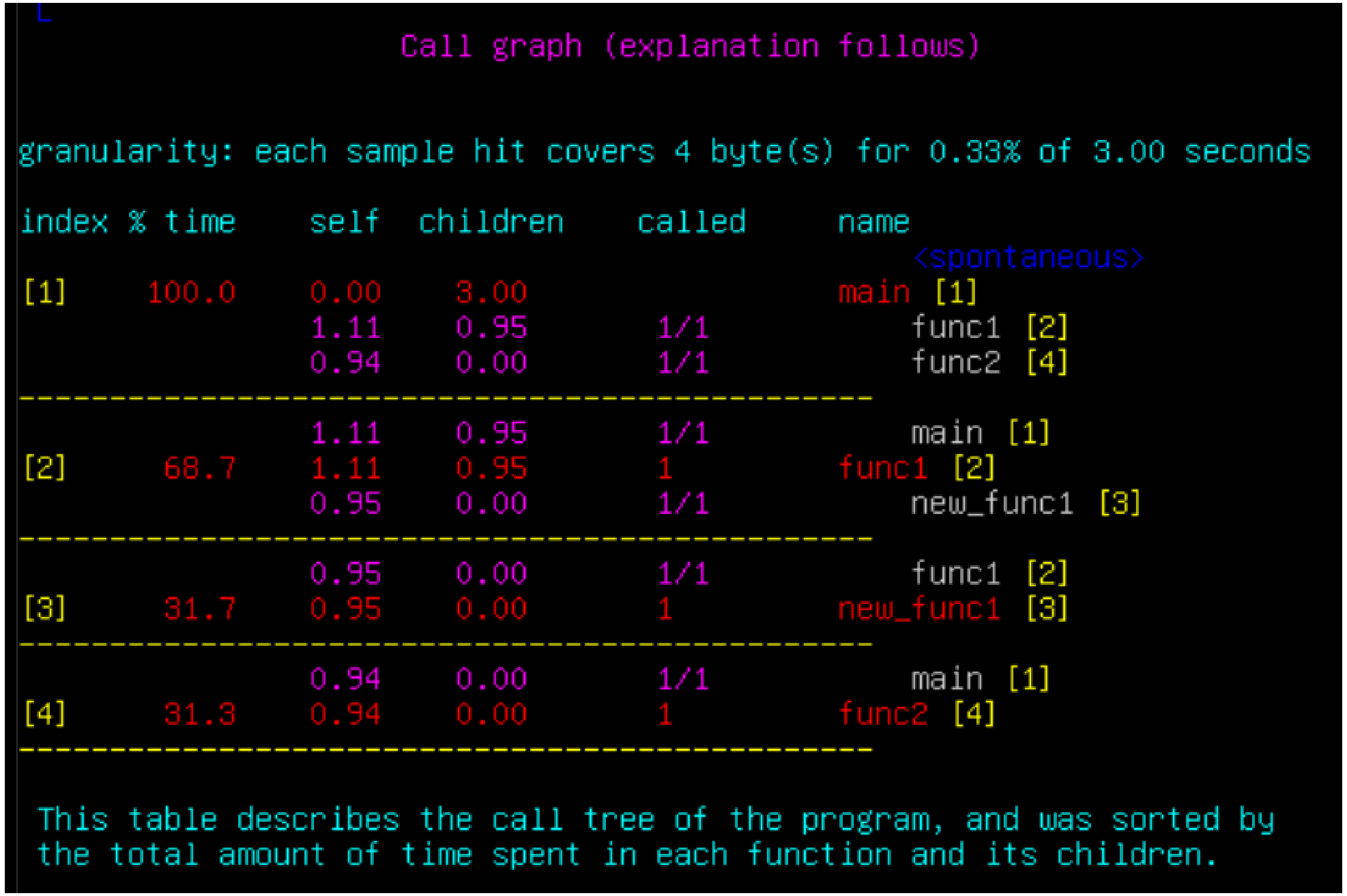
Comprehending the profiling information



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 37.00      1.11     1.11        1     1.11     2.06  func1
 31.67      2.06     0.95        1     0.95     0.95  new_func1
 31.33      3.00     0.94        1     0.94     0.94  func2

 %         the percentage of the total running time of the
time       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      the number of seconds accounted for by this
seconds    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      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

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

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.
```

As produced above, all the profiling information is now present in ‘analysis.txt’. Let's have a look at this text file.
(Use command “vi analysis.txt”)

The first part is Flat profile, it shows the running time of different part of the program.



```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.33% of 3.00 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    3.00                 main [1]
                1.11    0.95       1/1           func1 [2]
                0.94    0.00       1/1           func2 [4]
-----------------------------------------------
                1.11    0.95       1/1           main [1]
[2]     68.7    1.11    0.95       1         func1 [2]
                0.95    0.00       1/1           new_func1 [3]
-----------------------------------------------
                0.95    0.00       1/1           func1 [2]
[3]     31.7    0.95    0.00       1         new_func1 [3]
-----------------------------------------------
                0.94    0.00       1/1           main [1]
[4]     31.3    0.94    0.00       1         func2 [4]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.
```

The second pard is Call graph, it shows the call tree of the program, and was sorted by
the total amount of time spent in each function and its children.

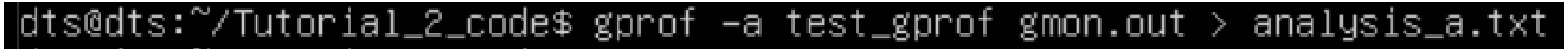
So (as already discussed) we see that this file is broadly divided into two parts :

1. Flat profile
2. Call graph

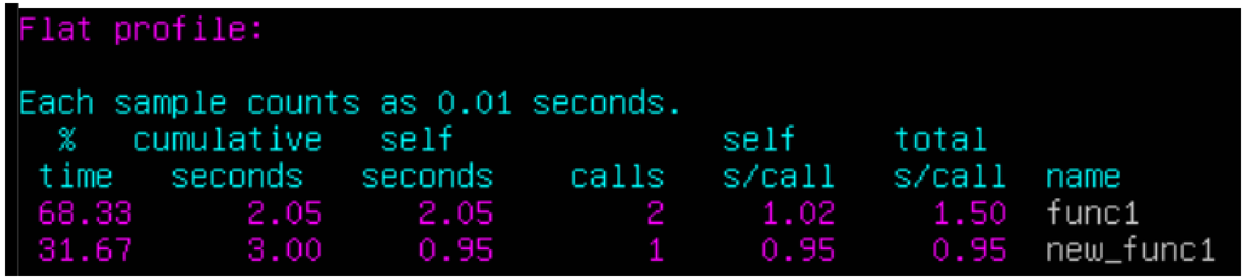
The individual columns for the (flat profile as well as call graph) are very well explained in the output itself.

Customize gprof output using flags
1. Suppress the printing of statically(private) declared functions using -a

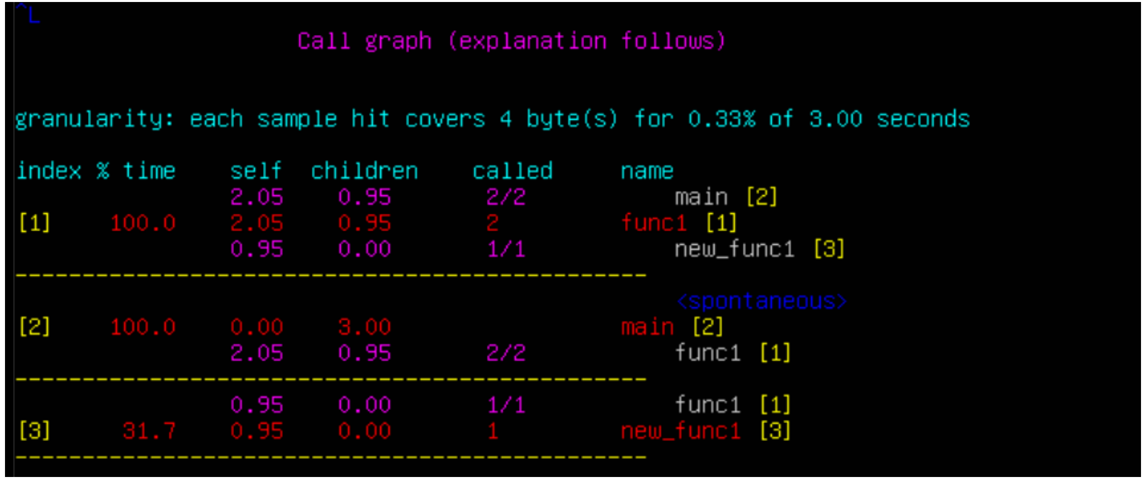
If there are some static functions whose profiling information you do not require then this can be achieved using -a option :



```
dts@dts:~/Tutorial_2_code$ gprof -a test_gprof gmon.out > analysis_a.txt
```



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  s/call   s/call  name
 68.33      2.05     2.05        2     1.02     1.50  func1
 31.67      3.00     0.95        1     0.95     0.95  new_func1
```

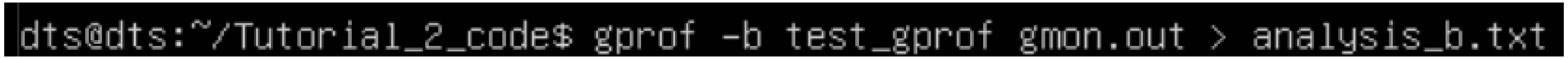


```
^L
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.33% of 3.00 seconds

index % time    self  children    called     name
                2.05    0.95       2/2           main [2]
[1]    100.0    2.05    0.95       2         func1 [1]
                0.95    0.00       1/1           new_func1 [3]
-----------------------------------------------
                                                 <spontaneous>
[2]    100.0    0.00    3.00                 main [2]
                2.05    0.95       2/2           func1 [1]
-----------------------------------------------
                0.95    0.00       1/1           func1 [1]
[3]     31.7    0.95    0.00       1         new_func1 [3]
-----------------------------------------------
```

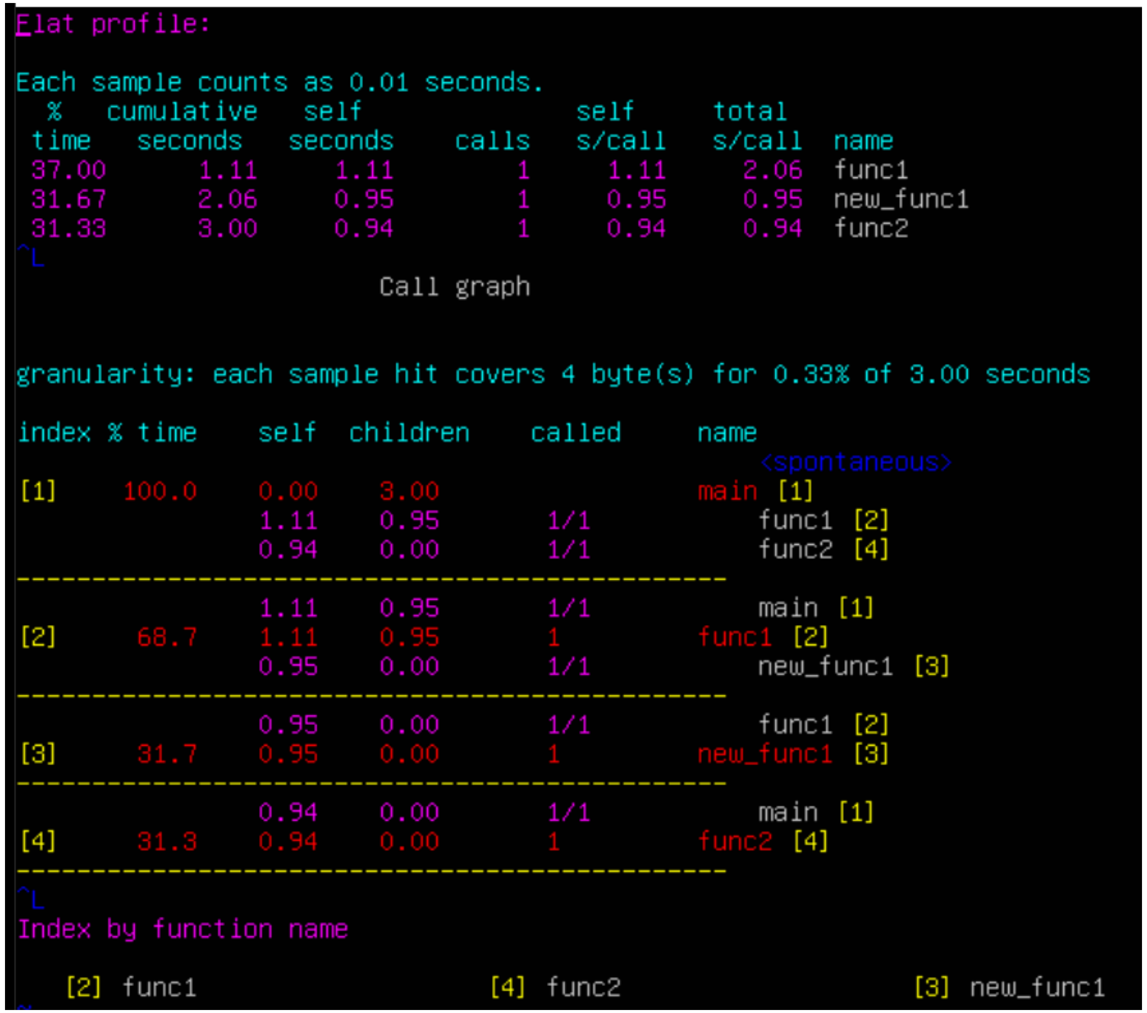
So, we see that there is no information related to func2 (which is defined static)

2. Suppress verbose blurbs using -b

Analysis file may have some describition that you do not need you can use –b to delete them



```
dts@dts:~/Tutorial_2_code$ gprof -b test_gprof gmon.out > analysis_b.txt
```

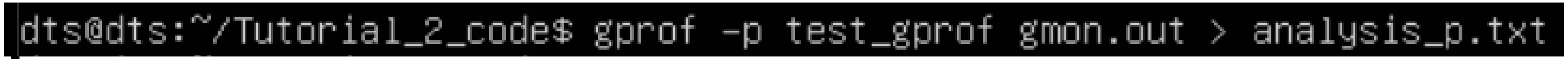


```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 37.00      1.11     1.11        1     1.11     2.06  func1
 31.67      2.06     0.95        1     0.95     0.95  new_func1
 31.33      3.00     0.94        1     0.94     0.94  func2
^L
                        Call graph


granularity: each sample hit covers 4 byte(s) for 0.33% of 3.00 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    3.00                 main [1]
                1.11    0.95       1/1           func1 [2]
                0.94    0.00       1/1           func2 [4]
-----------------------------------------------
                1.11    0.95       1/1           main [1]
[2]     68.7    1.11    0.95       1         func1 [2]
                0.95    0.00       1/1           new_func1 [3]
-----------------------------------------------
                0.95    0.00       1/1           func1 [2]
[3]     31.7    0.95    0.00       1         new_func1 [3]
-----------------------------------------------
                0.94    0.00       1/1           main [1]
[4]     31.3    0.94    0.00       1         func2 [4]
-----------------------------------------------
^L
Index by function name

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

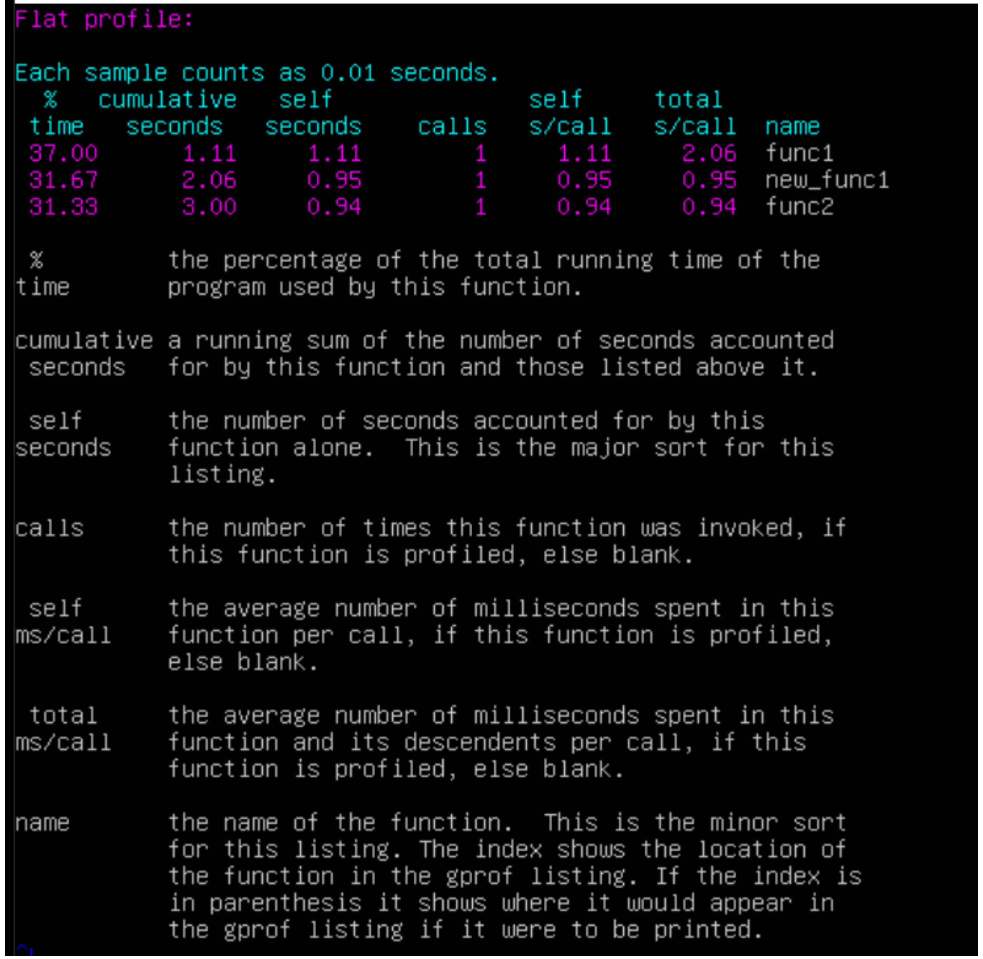
So, we see that all the verbose information is not present in the analysis file.

3. Print only flat profile using -p

If you only want to show flat profile you can use –p flag

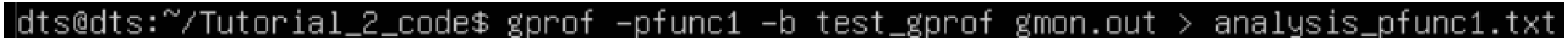


```
dts@dts:~/Tutorial_2_code$ gprof -p test_gprof gmon.out > analysis_p.txt
```

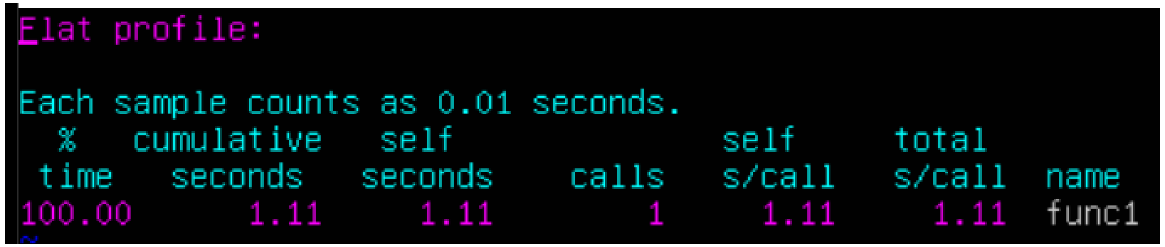


```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 37.00      1.11     1.11        1     1.11     2.06  func1
 31.67      2.06     0.95        1     0.95     0.95  new_func1
 31.33      3.00     0.94        1     0.94     0.94  func2

 %         the percentage of the total running time of the
time       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      the number of seconds accounted for by this
seconds    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      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

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

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.
```

4. Print information related to specific function in flat profile



```
dts@dts:~/Tutorial_2_code$ gprof -pfunc1 -b test_gprof gmon.out > analysis_pfunc1.txt
```



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.00      1.11     1.11        1     1.11     1.11  func1
```

5.Print only call graph information using -q

In a similar way, we can use flag –q to only output the result of call graph, the command is :

gprof –q test_gprof gmon.out > analysis_q.txt

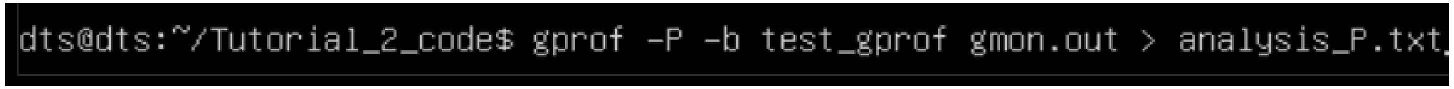
6.Print only specific function information in call graph.

Like point 4, we can choose the specific function to illustrate in call graph , the command is :

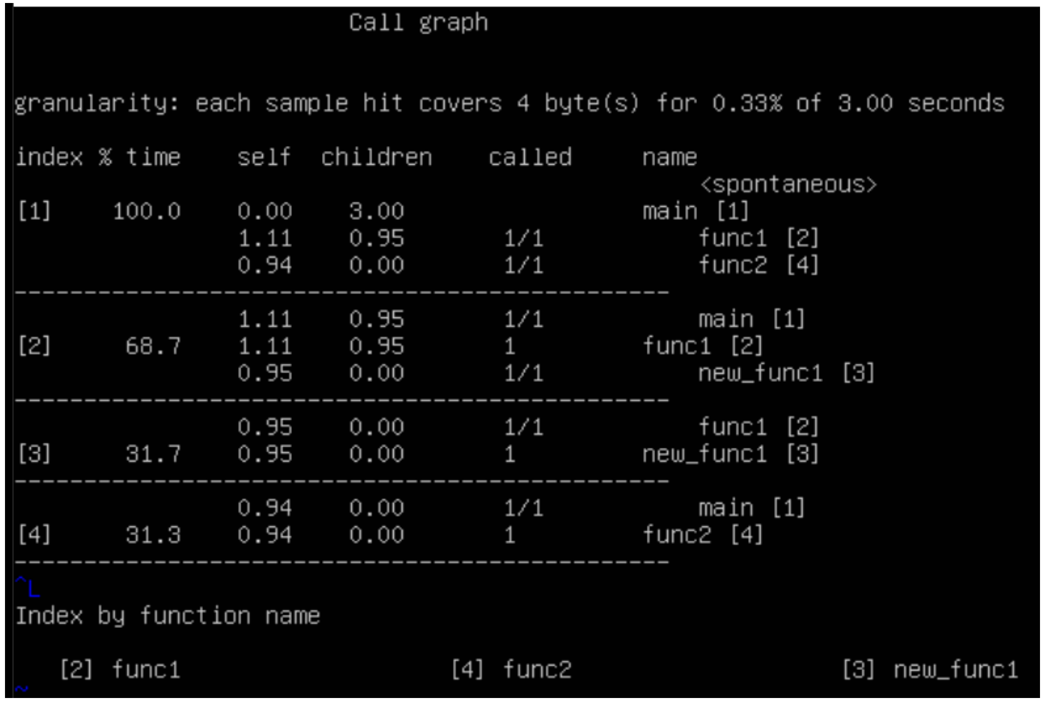
gprof –qfunc1 test_gprof gmon.out > analysis_qfunc1.txt

7.Suppress flat profile in output using -P

If flat profile is not required, then it can be suppressed using the -P option



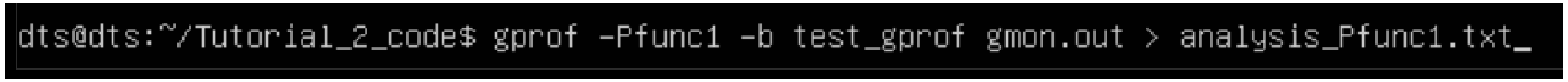
```
dts@dts:~/Tutorial_2_code$ gprof -P -b test_gprof gmon.out > analysis_P.txt
```



```
                        Call graph


granularity: each sample hit covers 4 byte(s) for 0.33% of 3.00 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    3.00                 main [1]
                1.11    0.95       1/1           func1 [2]
                0.94    0.00       1/1           func2 [4]
-----------------------------------------------
                1.11    0.95       1/1           main [1]
[2]     68.7    1.11    0.95       1         func1 [2]
                0.95    0.00       1/1           new_func1 [3]
-----------------------------------------------
                0.95    0.00       1/1           func1 [2]
[3]     31.7    0.95    0.00       1         new_func1 [3]
-----------------------------------------------
                0.94    0.00       1/1           main [1]
[4]     31.3    0.94    0.00       1         func2 [4]
-----------------------------------------------
^L
Index by function name

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

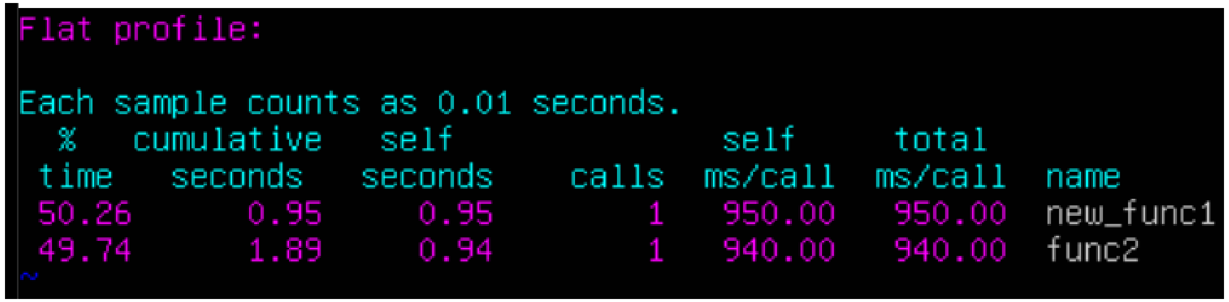
7.Suppress flat profile in output using -P

Also, if there is a requirement to print flat profile but excluding a particular function then this is also possible using -P flag by passing the function name (to exclude) along with it.



```
dts@dts:~/Tutorial_2_code$ gprof -Pfunc1 -b test_gprof gmon.out > analysis_Pfunc1.txt_
```

In the above example, we tried to exclude ‘func1’ by passing it along with the -P option to gprof.
Now let's see the analysis output:



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 50.26      0.95     0.95        1   950.00   950.00  new_func1
 49.74      1.89     0.94        1   940.00   940.00  func2
~
```

8. Suppress call graph using -Q

Like flat profile you can use –Q to suppress call graph and specifi function

gprof –Q -b test_gprof gmon.out > analysis.txt

Also, if it is desired to suppress a specific function from call graph then this can be achieved by passing the desired function name along with the -Q option to the gprof tool.

gprof –Qfunc1 -b test_gprof gmon.out > analysis.txt