Assignment No :B4

Roll No.4431

1 Title:

Gprof task distribution.

2 Problem Definition

rite a program to check task distribution using Gprof.l.

3 Learning Objectives

1. To check task distribution using Gprof.l

4 Learning Outcomes

1. Understanding of Grpof.l.

5 Related Mathematics

Let
S = s, e, x, y, fme, DD, NDD, memshared
S = Initial State
E = End State
X = Input Value i.e. Executable files
Y = Output i.e. Calculated time in milliseconds.
Fm = Main function i.e. test() function for the calculation of time.
DD = Deterministic data
NDD = Non-deterministic data
Memshared = Core that is used for execution i.e. core1/core2

6 Concepts related theory

6.1 Gprof

- Gprof is a performance analysis tool for Unix applications. It uses a hybrid of instrumentation and sampling and was created as extended version of the older "prof" tool. Unlike prof, gprof is capable of limited call graph collecting and printing.

- GPROF was originally written by a group led by Susan L. Graham at the University of California, Berkeley for Berkeley Unix Another implementation was written as part of the GNU project for GNU Binutils in 1988 by Jay Fenlason.

6.2 Profiling Data File Format

- The old BSD-derived file format used for profile data does not contain a magic cookie that allows to check whether a data file really is a gprof file. Furthermore, it does not provide a version number, thus rendering changes to the file format almost impossible. gnu gprof uses a new file format that provides these features. For backward compatibility, gnu gprof continues to support the old BSD-derived format, but not all features are supported with it. For example, basic-block execution counts cannot be accommodated by the old file format.

6.3 Insert gprof Command Summary :

After you have a profile data file gmon.out, you can run gprof to interpret the information in it. The gprof program prints a flat profile and a call graph on standard output. Typically you would redirect the output of gprof into a file with dollar ¿ dollar.

6.4 You run gprof like this:

gprof options [executable-file [profile-data files...]] [>outfile]. If you omit the executable file name, the file a.out is used. If you give no profile data file name, the file gmon.out is used. If any file is not in the proper format, or if the profile data file does not appear to belong to the executable file, an error message is printed.

6.5 Debugging gprof:

If gprof was compiled with debugging enabled, the '-d' option triggers debugging output (to stdout) which can be helpful in understanding its operation. The debugging number specified is interpreted as a sum of the following options:

2 - Topological sort : Monitor depth-first numbering of symbols during call graph analysis 4 - Cycles : Shows symbols as they are identified as cycle heads 16 - Tallying : As the call graph arcs are read, show each arc and how the total calls to each function are tallied

32 - Call graph arc sorting : Details sorting individual parents/children within each call graph entry 64 - Reading histogram and call graph records : Shows address ranges of histograms as they are read, and each call graph arc 128 - Symbol table : Reading, classifying, and sorting the symbol table from the object file. For line-by-line profiling ('-l' option), also shows line numbers being assigned to memory addresses. 256 - Static call graph : Trace operation of '-c' option

7 Implementation:

Instrumentation code is automatically inserted into the program code during compilation (for example, by using the '-pg' option of the gcc compiler), to gather caller-function data. A call to the monitor function 'mcount' is inserted before each function call.
Sampling data is saved in 'gmon.out' or in 'progname.gmon' file just before the program exits, and can be analyzed with the 'gprof' command-line tool. Several gmon files can be combined with 'gprof -s' to accumulate data from several runs of a program.
GPROF output consists of two parts: the flat profile and the call graph. The flat profile gives the total execution time spent in each function and its percentage of the total running time. Function call counts are also reported. Output is sorted by percentage, with hot spots at the top of the list.
The second part of the output is the textual call graph, which shows for each function who called it (parent) and who it called (child subroutines). There is external tool called gprof2dot capable of converting the call graph from gprof into graphical form.

8 Program Listing

```
//B4.c file:
#include<stdio.h>


static void func2(void)
{
    printf("\n Inside func2 \n");
    int i = 0;

    for(;i<0xffffffaa;i++);
    return;
}

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

    for(;i<0xffffff;i++);
    func2();

    return 0;
}


// B4.texer.l

Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
101.08     10.60    10.60        1    10.60    10.60  func2()
  0.29     10.63     0.03                             main

 %         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.
```


Call graph (explanation follows)

```
granularity: each sample hit covers 2 byte(s) for 0.09% of 10.63 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     100.0    0.03   10.60                 main [1]
                10.60    0.00       1/1           func2() [2]
-----------------------------------------------
                10.60    0.00       1/1           main [1]
[2]      99.7   10.60    0.00       1         func2() [2]
-----------------------------------------------
```

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.

Each entry in this table consists of several lines. The line with the
index number at the left hand margin lists the current function.
The lines above it list the functions that called this function,
and the lines below it list the functions this one called.
This line lists:

index A unique number given to each element of the table.
       Index numbers are sorted numerically.
       The index number is printed next to every function name so
       it is easier to look up where the function is in the table.

% time — This is the percentage of the 'total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.

self — This is the total amount of time spent in this function.

children — This is the total amount of time propagated into this function by its children.

called — This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a '+' and the number of recursive calls.

name — The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self — This is the amount of time that was propagated directly from the function into this parent.

children — This is the amount of time that was propagated from the function's children into this parent.

called — This is the number of times this parent called the function '/' the total number of times the function was called. Recursive calls to the function are not included in the number after the '/'.

name — This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word '<spontaneous>' is printed in the 'name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self — This is the amount of time that was propagated directly from the child into the function.

children — This is the amount of time that was propagated from the child's children to the function.

```
called      This is the number of times the function called
            this child '/' the total number of times the child
            was called.  Recursive calls by the child are not
            listed in the number after the '/'.

name        This is the name of the child.  The child's index
            number is printed after it.  If the child is a
            member of a cycle, the cycle number is printed
            between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The '+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.


Copyright (C) 2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


Index by function name

   [2] func2() (test.c)        [1] main
```

9 Output

```
botman@botmatrix:~$ cd Desktop/
botman@botmatrix:~/Desktop$ cd gprof/
botman@botmatrix:~/Desktop/gprof$ gcc -Wall -pg gprof.c
new_gprof.c -o gprofobj
botman@botmatrix:~/Desktop/gprof$ ls
analysis.txt  gmon.out  gprof.c   gprofobj   new_gprof~ Untitled Document~
       a.out         gprof     gprof.c~  gprof_obj  new_gprof.c
botman@botmatrix:~/Desktop/gprof$ ./gprofobj
        Inside main()

        Inside func1

        Inside new_func1()

        Inside func2

        Inside func3\
```

```
botman@botmatrix:~/Desktop/gprof$ ls
analysis.txt  gmon.out  gprof.c   gprofobj   new_gprof~ Untitled Document~
a.out         gprof     gprof.c~  gprof_obj  new_gprof.c
botman@botmatrix:~/Desktop/gprof$ gprof gprofobj gmon.out > analysis.txt
botman@botmatrix:~/Desktop/gprof$ ls
analysis.txt  gmon.out  gprof.c   gprofobj   new_gprof~ Untitled Document~
a.out         gprof     gprof.c~  gprof_obj  new_gprof.c
botman@botmatrix:~/Desktop/gprof$
```

10 Testing

10.1 Positive Testing

Sr. No.	Test Condition	Steps to be executed	Expected Result	Actual Result
1.	Give a program with lot of functions & loop	Press Enter	Display time required to evaluate	Same as Expected

10.2 Negative Testing

Sr. No.	Test Condition	Steps to be executed	Expected Result	Actual Result
1.	Give a program with no functions & loop	Press Enter	No time will display	Display time required to evaluate

11 CONCLUSION :

Hence we have successfully run the program using GPROF profiling tool.