Assignment No. B4

Aim

Gprof

Problem Definition

Write a program to check task distribution using Gprof.

Learning Objectives

- Understanding profiling
- Learn how to implement gprof tool

Learning Outcome

- Learnt about profiling and implemented gprof tool

Software And Hardware Requirements

- Latest 64-BIT Version of Linux Operating System

Mathematical Model

Let S be the system of solution set for given problem statement such that,

S = { s, e, X, Y, F, DD, NDD, Su, Fu }
where,
s = start state
such that, a program P is present.

e = end state
such that, gprof tool is used.

X = set of input
such that X = { P }
P = Program to be profiled

Y = set of output
such that Y = { A }

where,
A$\rightarrow$ the result of analysis

F = set of function
such that F = { f1, f2, f3 }
where,
f1 = function to enable profiling
f2 = function to execute code
f3 = function to print analysis to file

DD = Deterministic data
DD = { P, f1, f2 }

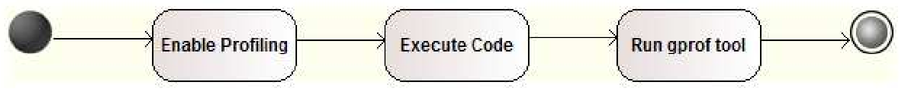
NDD = Nondeterministic data
NDD = { A, f3 }

Su = Success cases

- gprof tool is used
- analysis takes place

Fu = Failure case
gprof tool fails

State Diagram



Theory

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.

In very large projects, profiling can help in determining the parts in your program which are slower in execution than expected but also can help you find many other statistics through which many potential bugs can be spotted and sorted out.

gprof

Gprof is a performance analysis tool for Unix applications. The gprof command displays execution profiles for programs that have been compiled with the -pg flag. When such a program is executed, it runs as normal but leaves profiling information in a file called gmon.out. The command gprof reads gmon.out and prints a profile on standard output.

The gprof command works at the function level. It gives a table (flat profile) containing:

- number of times each function was called
- % of total execution time spent in the function
- average execution time per call to that function
- execution time for this function and its children

Arranged in order from most expensive function down. It also gives a call graph, a list for each function:

- which functions called this function
- which functions were called by this function

How to use gprof

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.
   -pg : 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.

2. Execute the code
   In the second step, the binary file produced as a result of step-1 (above) is executed so that profiling information can be generated. The binary is executed, a new file gmon.out is generated in the current working directory.

3. Run the gprof tool
   In this step, the gprof tool is run with the executable name and the above generated gmon.out as argument. This produces an analysis file which contains all the desired profiling information. Note that one can explicitly specify the output file or the information is produced on stdout.

Program Code

```
//test_gprof.c
#include<stdio.h>
void new_func1(void);
void func1(void)
{
    printf("\n Inside func1 \n");
    int i = 0;
    for(;i<0xffffffff;i++);
    new_func1();
    return;
}
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++);
    func1();
    func2();
    return 0;
}

//test_gprof_new.c
#include<stdio.h>
void new_func1(void)
{
    printf("\n Inside new_func1()\n");
    int i = 0;
    for(;i<0xffffffee;i++);
    return;
}
```

Output

```
root@pict:~/cl4/B_group$ g++ -pg test_gprof.c test_gprof_new.c -o test_gprof
root@pict:~/cl4/B_group$ ./test_gprof

 Inside main()
 Inside func1

 Inside new_func1()

 Inside func2
root@pict:~/cl4/B_group$ gprof test_gprof gmon.out > analysis.txt

//analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 34.07     15.29    15.29        1    15.29    15.29  new_func1()
 33.87     30.49    15.20        1    15.20    15.20  func2()
 33.05     45.32    14.83        1    14.83    30.12  func1()
  0.14     45.38     0.06                             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.
```

```
Copyright (C) 2012-2014 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.


                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.02% of 45.38 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.06   45.32                 main [1]
               14.83   15.29       1/1           func1() [2]
               15.20    0.00       1/1           func2() [4]
-----------------------------------------------
               14.83   15.29       1/1           main [1]
[2]     66.4   14.83   15.29       1         func1() [2]
               15.29    0.00       1/1           new_func1() [3]
-----------------------------------------------
               15.29    0.00       1/1           func1() [2]
[3]     33.7   15.29    0.00       1         new_func1() [3]
-----------------------------------------------
               15.20    0.00       1/1           main [1]
[4]     33.5   15.20    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.

 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) 2012-2014 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] func1()                 [4] func2()
   [3] new_func1()             [1] main
```

Conclusion

Thus, we have studied and implemented gprof profiling tool.