EEP702-Software Lab

Assignment 9 : Caculating LCM and GCD using MACROS And Use of gprof

Harshit Kumar Gupta
2013EET2369

Computer Technology
Department Of Electrical Engineering
IIT DELHI

April 18, 2014

Contents

Chapter 1

PROBLEM STATEMENT

Optimizing code for speed using inline assembling

PROBLEM 1:

Given two integers, write a time efficient c code, that spends as less time in memory access and more in calculations as possible, to get their GCD and LCM.Use directives for conditional compilation of code as follows. Find the critical parts of code that consume more percentage of time using gprof and replace those parts with inline assembly coding to optimize speed and compare the time profiling for both codes.

PROBLEM 2:

1. Find the most significant non zero bit position for largest of the two given integers using pure c code and inline assembled c code and show which one is faster.

2. Convert the given integer(assumed to be angle in degrees) to radians and find the floating point cosine value with pure c and inline assembled c and determine fastest one using gprof.

3. Use gprof on the code of assignment no-04 Library management and rewrite the code to optimize time.

Chapter 2

ABSTRACT

Basically our Purpose of Implementation is to see where does a Code hangs up or uses much of memory Space while Execution. The Use of Gprof allows us to ou to learn where your program spent its time and which functions called which other functions while it was executing. This information can show you which pieces of your program are slower than you expected, and might be candidates for rewriting to make your program execute faster. It can also tell you which functions are being called more or less often than you expected. This may help you spot bugs that had otherwise been unnoticed.

Chapter 3

INTRODUCTION

Once the program is compiled for profiling, you must run it in order to generate the information that gprof needs. Simply run the program as usual, using the normal arguments, file names, etc. The program should run normally, producing the same output as usual. It will, however, run somewhat slower than normal because of the time spent collecting and the writing the profile data. The way you run the program–the arguments and input that you give it–may have a dramatic effect on what the profile information shows. The profile data will describe the parts of the program that were activated for the particular input you use. For example, if the first command you give to your program is to quit, the profile data will show the time used in initialization and in cleanup, but not much else. Your program will write the profile data into a file called gmon.out' just before exiting. If there is already a file called gmon.out, its contents are overwritten. There is currently no way to tell the program to write the profile data under a different name, but you can rename the file afterward if you are concerned that it may be overwritten. In order to write the gmon.out' file properly, your program must exit normally: by returning from main or by calling exit. Calling the low-level function exit does not write the profile data, and neither does abnormal termination due to an unhandled signal. The gmon.out file is written in the program's current working directory at the time it exits. This means that if your program calls chdir, the 'gmon.out' file will be left in the last directory your program chdir'd to. If you don't have permission to write in this directory, the file is not written, and you will get an error message.

Chapter 4

SPECIFICATIONS AND ASSUMPTIONS

Specifications

1. Entire Code has been made in C Language.
2. Macros have been Explicitly Designed for LCM and HCF.
3. "lcm" function has been used to Compute LCM calling up the "gcd" Function.
4. gprof has been used for code analysis and Optimisation.

Assumptions

1. The Mathematical functions used are Bug-Free.
2. gmon.out is generated on the same path as of the C code.
3. User succesfully enters a Positive values for LCM and HCF Calculation.

Chapter 5

LOGIC USED/METHODOLOGY

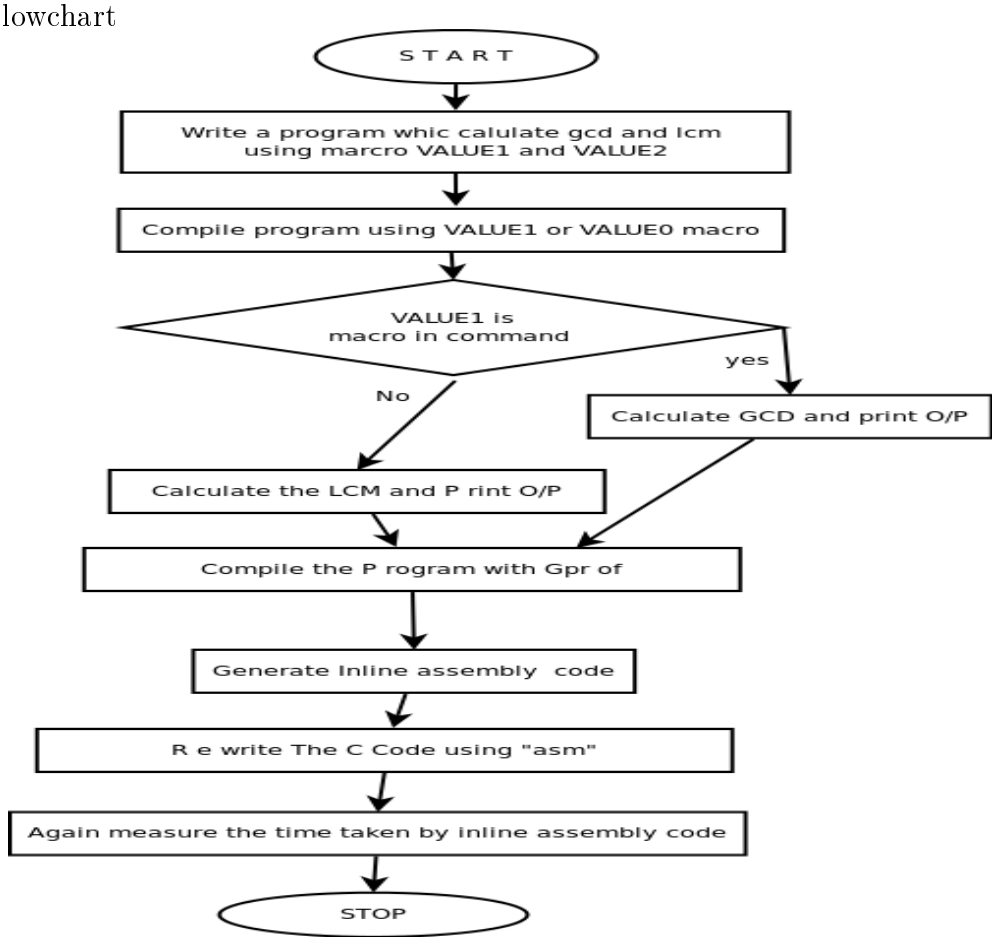
The methodology that is used for developing the program is defined below:

1. We intially Design a Proper C code for Calculating both the Values.
2. Below the Header Files Macros are Provided for LCM and HCF.
3. In the (define) Tag we put a Suitable Macro which determines what has to run.
4. The Program may Output the values Very fast as the Code is not quite Complex.
5. We run the gprof Command to our C Code which generates a File named gmon.out
6. It contains the run time of our and may be Varied increasing the Complexity of our code.
7. Results are Suitably saved in a File.

Chapter 6

FLOWCHART

Flowchart



Chapter 7

EXECUTION DIRECTIVE

For program following instructions are used.

1. gcc -MACRO pureCcode.c -o test
2. test
3. gprof options [executable-file [profile-data-files...]] [> outfile]
4. gprof test gmon.out
5. gcc -MACRO InlneAssemblyCode.c -o test
6. test
7. gprof test gmon.out

Chapter 8

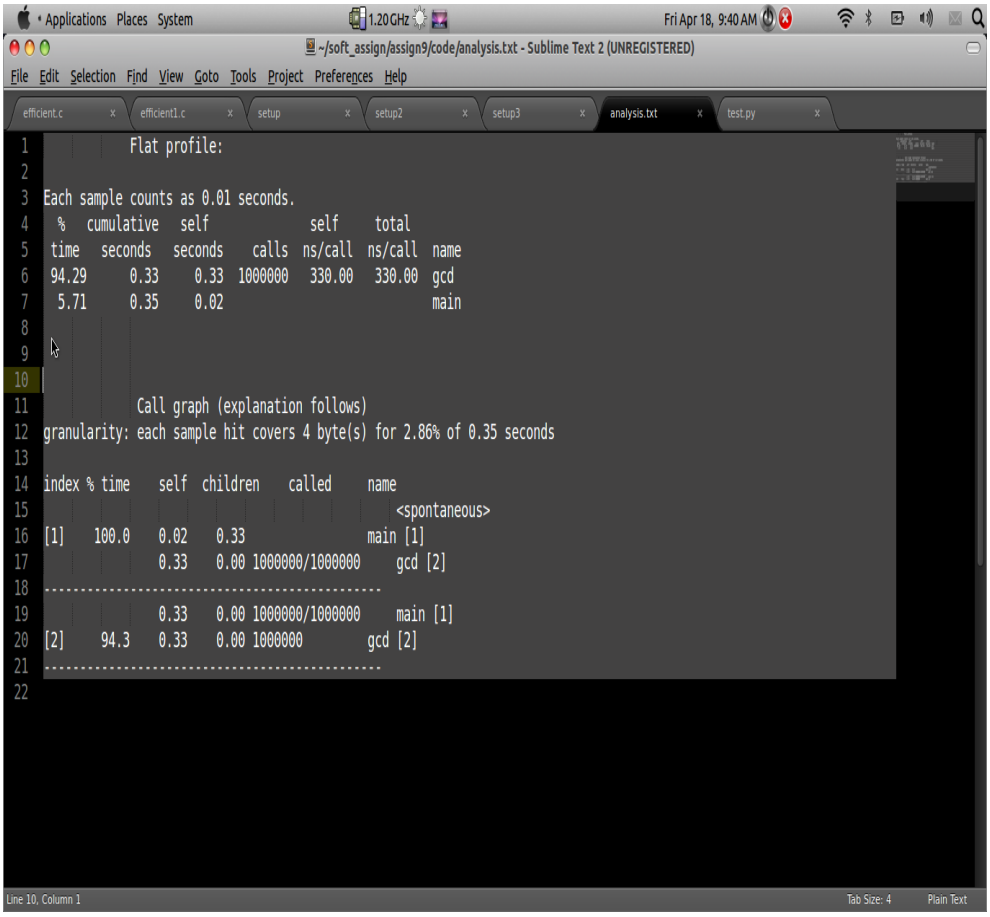
RESULTS AND CONCLUSIONS

1. For the Given Conditions Program runs Successfully and Generates a bug free Output.
2. The Profiling is Done with the Help of gprof which returns Satisfactory Outputs.
3. Writing inline assembly code does not always reduce the time taken beacuse sometimes compiler generate more optimized code which we can not write using assembly.

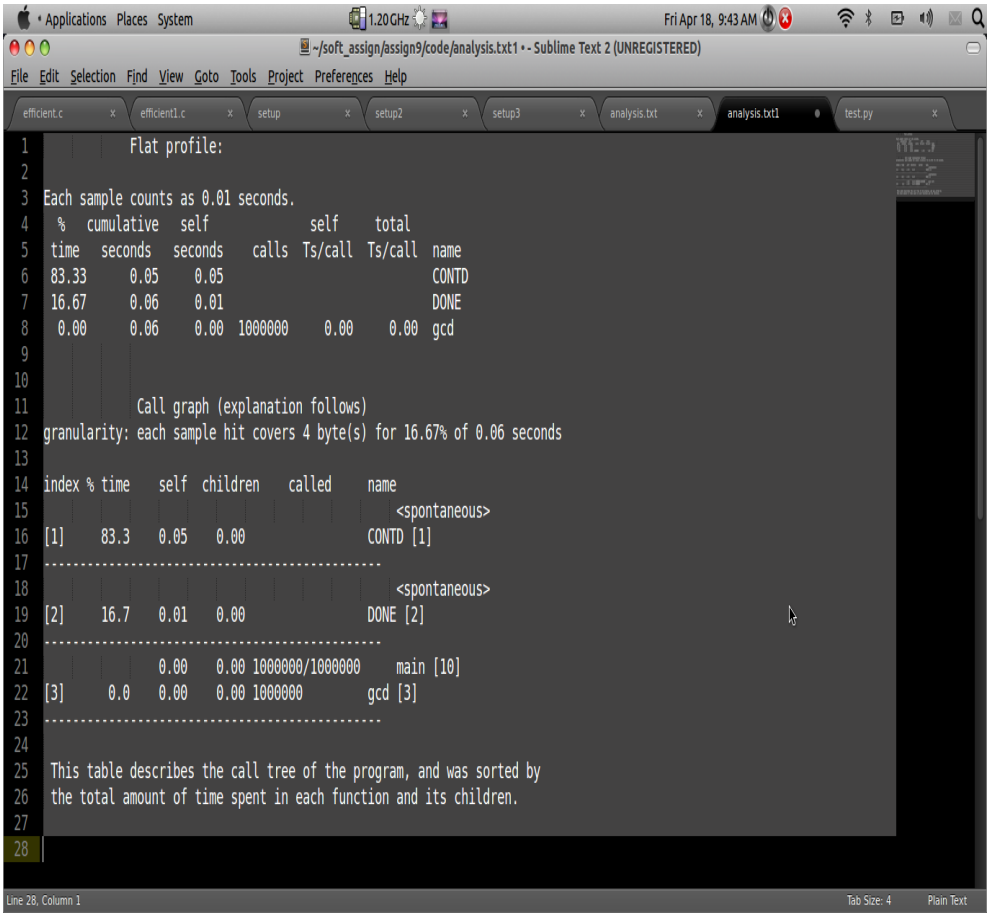
For the Given Conditions Program runs Successfully and Generates a bug free Output. The Profiling is Done with the Help of gprof which returns Satisfactory Outputs.

Chapter 9

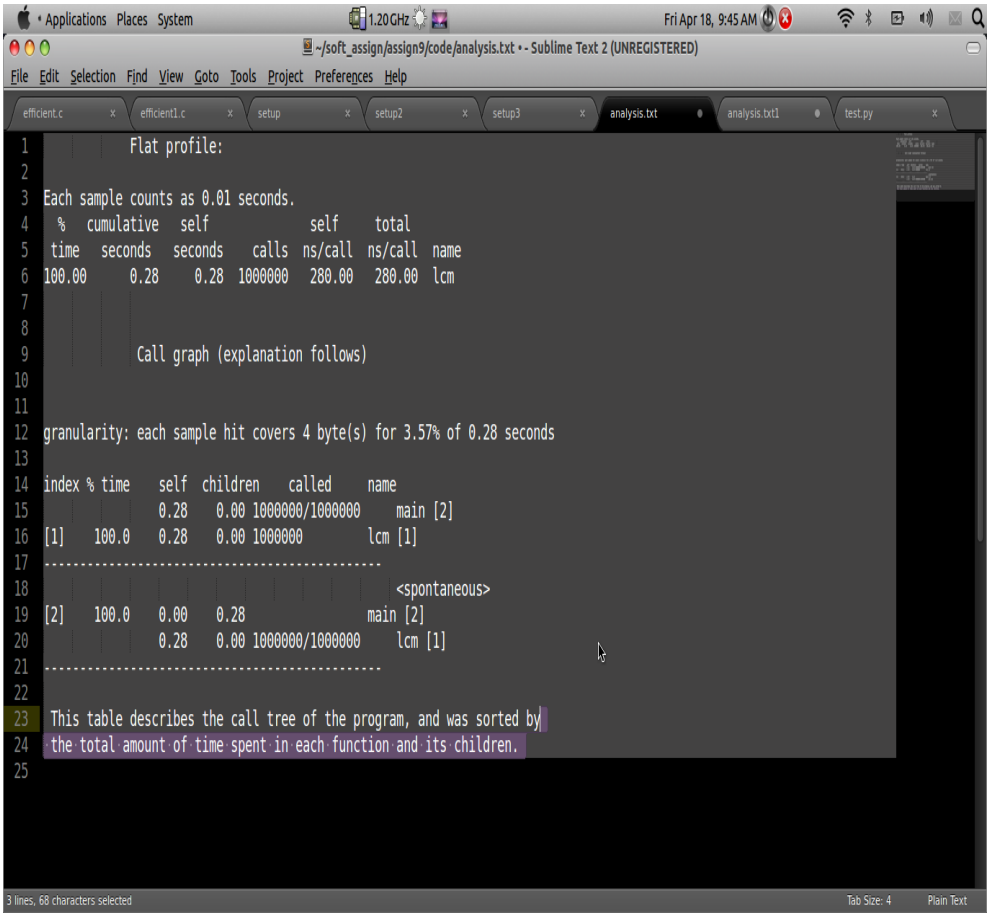
PROFILING Results



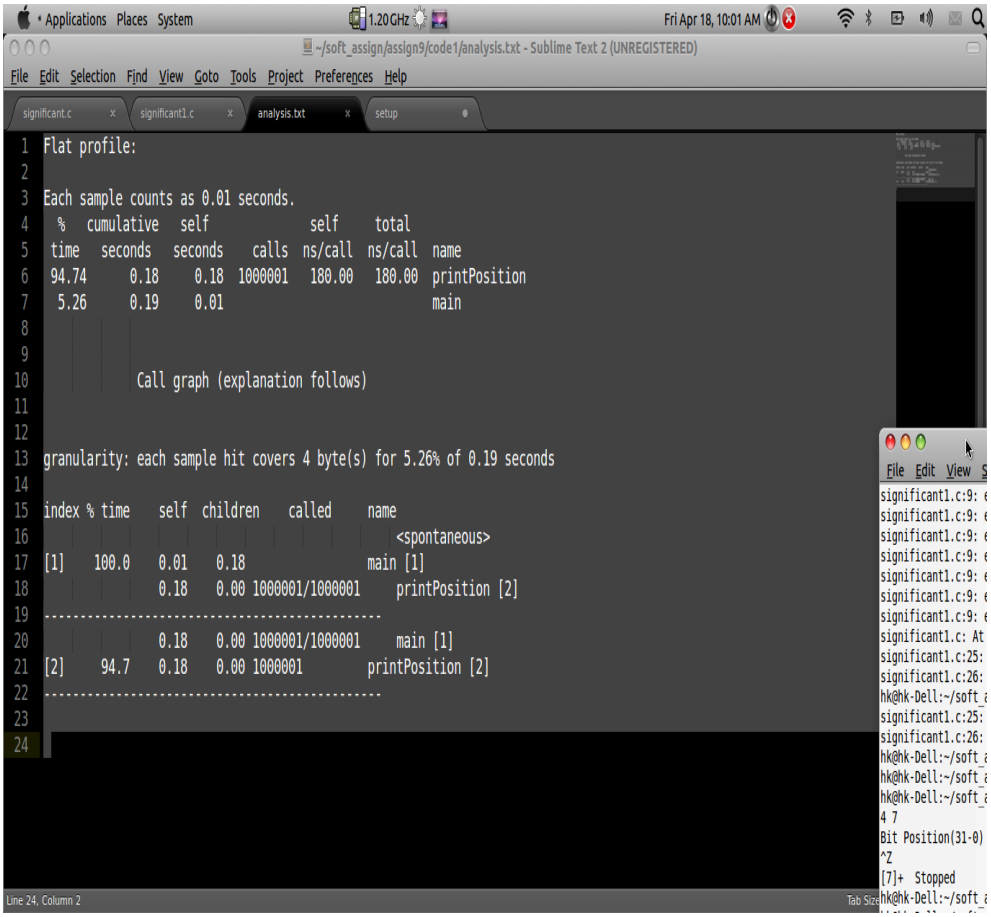
profiling for GCD in c



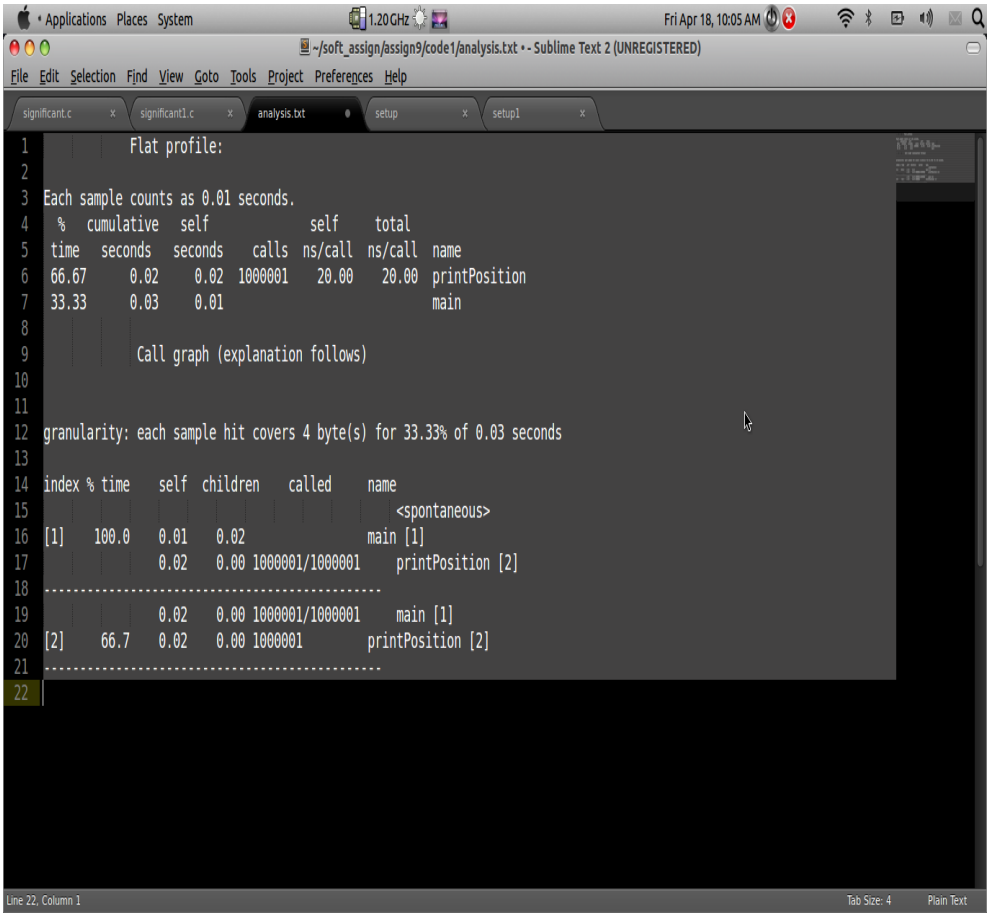
profiling for GCD in inline assembly



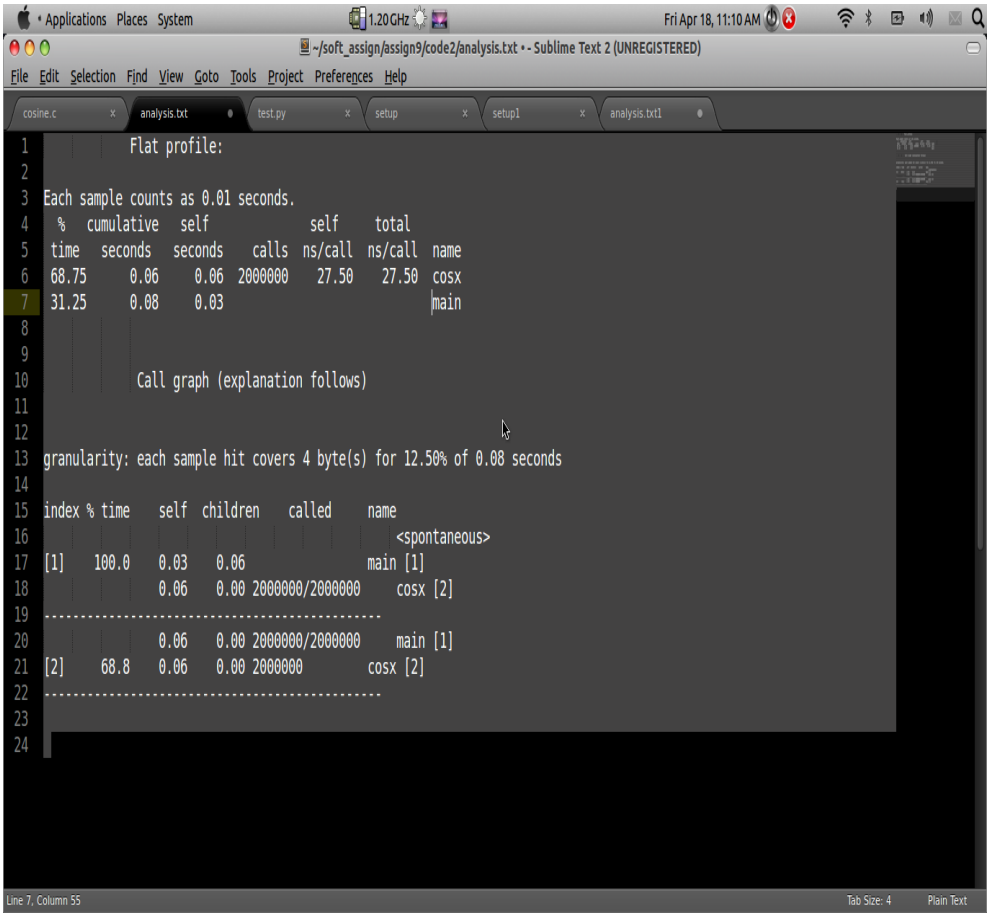
profiling for LCM in C



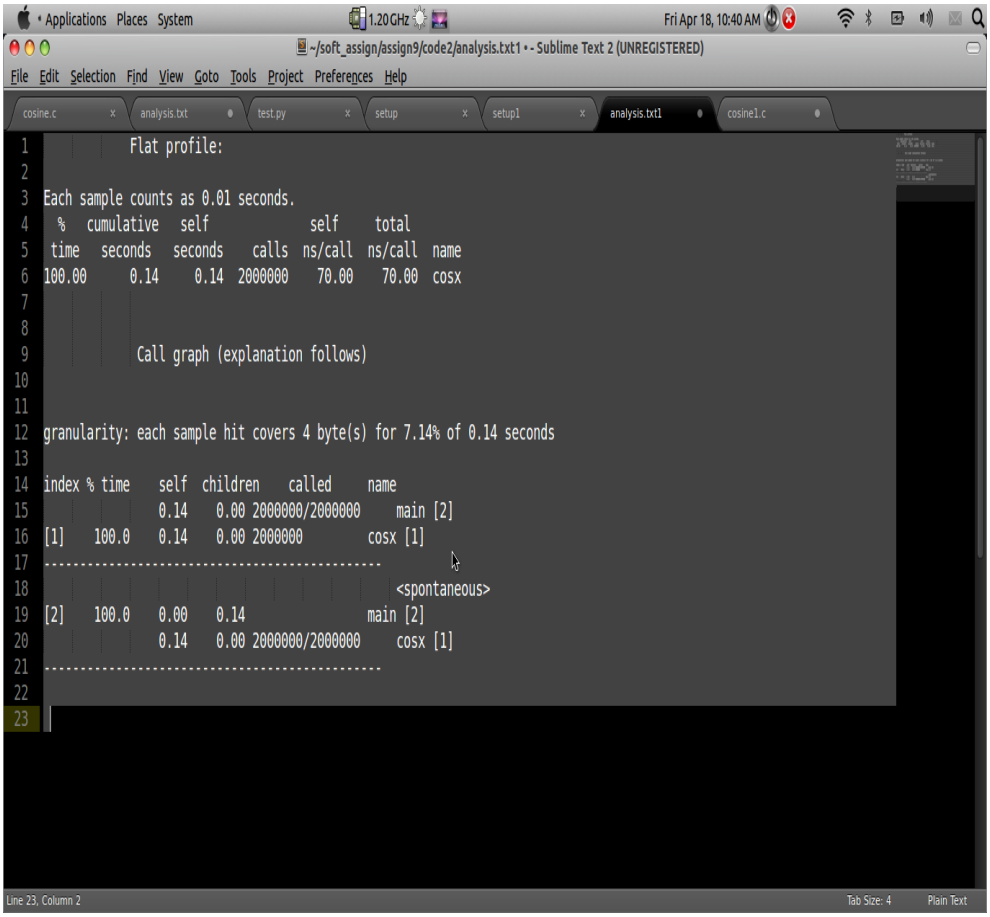
profiling for significant bit in C



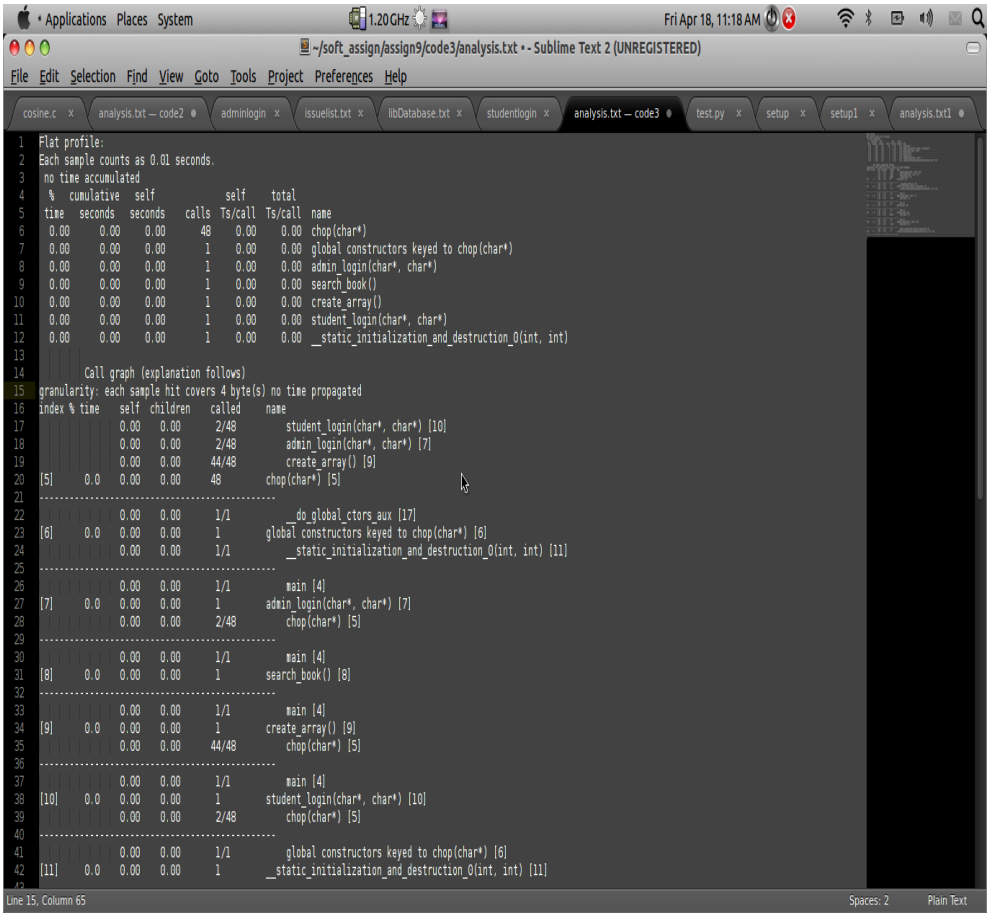
profiling for significant bit in inline assembly.



profiling for cosine in c



profiling for cosine in inline assembly



profiling for library management assignment