**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Качество и метрология программного обеспечения»**

**Тема: Измерение характеристик динамической сложности программ с помощью профилировщика Sampler**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8303 |  | Гришин К. И. |
| Преподаватель |  | Кирьянчиков В. А. |

Санкт-Петербург

2021

# Цель работы

Изучить возможности измерения динамических характеристик программ с помощью профилировщика на примере планировщика Sampler.

# Задание

Разбить программу из первой лабораторной на функциональные участки, определить узкое место и внести изменения для повешения производительности.

# Ход выполнения

## Тестовые программы

Результаты запуска программы test\_cyc.cpp и test\_sub.cpp представлены в таблицах 1 и 2 соотв. Профилирование программ проводилось с использованием параметров: 100 запусков, 10 пропусков без оптимизаций компилятора.

| исх | прием | общее время | кол-во проходов | среднее время |
| --- | --- | --- | --- | --- |
| 13 | 15 | 3793.200 | 1 | 3793.200 |
| 15 | 17 | 7459.100 | 1 | 7459.100 |
| 17 | 19 | 19624.800 | 1 | 19624.800 |
| 19 | 21 | 37528.700 | 1 | 37528.700 |
| 21 | 24 | 3963.500 | 1 | 3963.500 |
| 24 | 27 | 7777.000 | 1 | 7777.000 |
| 27 | 30 | 18443.600 | 1 | 18443.600 |
| 30 | 33 | 37498.100 | 1 | 37498.100 |
| 33 | 39 | 3679.800 | 1 | 3679.800 |
| 39 | 45 | 7540.000 | 1 | 7540.000 |
| 45 | 51 | 18709.200 | 1 | 18709.200 |
| 51 | 57 | 36343.700 | 1 | 36343.700 |

Таблица 1.Результат профилирования test\_cyc.cpp

| исх | прием | общее время | кол-во проходов | среднее время |
| --- | --- | --- | --- | --- |
| 30 | 32 | 30496332.300 | 1 | 30496332.300 |
| 32 | 34 | 61231491.300 | 1 | 61231491.300 |
| 34 | 36 | 153969222.100 | 1 | 153969222.100 |
| 36 | 38 | 308683976.700 | 1 | 308683976.700 |

Таблица 2.Результат профилирования test\_sub.cpp

Из таблиц 1 и 2 видно, что с увеличением количества итераций цикла, увеличивается время работы функциональных участков. Причем можно заметить, что что изменение количества итераций линейно влияет на время исполнения функционального участка.

Из таблицы 1 видно, что массивы одинакового размера обрабатываются за одинаковое время, что говорит о том, что кеширования не происходило.

## Программа из первой лабораторной работы

Сначала был проведен замер длительности работы программы без установленных контрольных точек. Результат работы профилировщика представлен в таблице 3.

| исх | прием | общее время | кол-во проходов | среднее время |
| --- | --- | --- | --- | --- |
| 44 | 48 | 26175.279 | 1 | 26175.279 |

Таблица 3.Результат замера длительности исполнения программы

Для профилирования программы в нее были добавлены контрольные точки для разбиения на функциональные участки (прил. А).

Результат профилирования программы представлен в таблице 4.

| исх | прием | общее время | кол-во проходов | среднее время |
| --- | --- | --- | --- | --- |
| 21 | 25 | 14.158 | 1 | 14.158 |
| 25 | 29 | 12.526 | 1 | 12.526 |
| 29 | 32 | 26.200 | 1 | 26.200 |
| 32 | 36 | 0.811 | 1 | 0.811 |
| 36 | 38 | 0.200 | 1 | 0.200 |
| 38 | 42 | 20.958 | 12 | 1.746 |
| 42 | 45 | 63.168 | 12 | 5.264 |
| 45 | 48 | 35967.942 | 4095 | 8.783 |
| 48 | 50 | 94.137 | 12 | 7.845 |
| 48 | 45 | 6228.505 | 4083 | 1.525 |
| 50 | 52 | 19.605 | 12 | 1.634 |
| 52 | 38 | 89.905 | 11 | 8.173 |
| 52 | 54 | 3.605 | 1 | 3.605 |

Таблица 4. Результат профилирования программы из первой лабораторной работы.

Добавление контрольных точек увеличило суммарное время исполнения программы до 42541.720.

Из таблицы видно, что наибольшее время занимают строки 45-48 из-за большого количества итераций цикла. Необходимо упростить тело цикла, вынеся часть вычислений наружу.

Также проведены минорные изменения остальных частей кода:

* Теперь функция не принимает указатель на переменную, в которой хранится результат, а возвращает его.
* Все операции для разыменования и взятия адреса устранены.
* Объявление и определение переменных объединены.

Исходный код модифицированной программы представлен в прил. Б.

Результат профилирования программы представлен в таблице 5.

| исх | прием | общее время | кол-во проходов | среднее время |
| --- | --- | --- | --- | --- |
| 26 | 31 | 9.826 | 1 | 9.826 |
| 31 | 35 | 12.753 | 1 | 12.753 |
| 35 | 38 | 20.053 | 1 | 20.053 |
| 38 | 42 | 2.679 | 1 | 2.679 |
| 42 | 45 | 196.984 | 1 | 196.984 |
| 45 | 51 | 36.689 | 12 | 3.057 |
| 51 | 53 | 61.816 | 12 | 5.151 |
| 53 | 55 | 21197.189 | 4095 | 5.176 |
| 55 | 58 | 91.553 | 12 | 7.629 |
| 55 | 53 | 5230.105 | 4083 | 1.281 |
| 58 | 64 | 260.384 | 12 | 21.699 |
| 64 | 45 | 73.553 | 11 | 6.687 |
| 64 | 66 | 3.395 | 1 | 3.395 |

Таблица 5. Результат профилирования модифицированной программы из первой лабораторной.

Суммарное время исполнения программы под управлением профилировщика равно 27196.979. То есть время исполнения программы под управлением профилировщика уменьшилось на 36.070%. Такой же результат ожидается и при замерах без установленных меток (табл. 6).

| исх | прием | общее время | кол-во проходов | среднее время |
| --- | --- | --- | --- | --- |
| 60 | 64 | 18555.726 | 1 | 18555.726 |

Таблица 6.Результат замера длительности исполнения модифицированной программы.

Время исполнения уменьшилось с 26175.279 до 18555.726. То есть время исполнения программы уменьшилось на 29.11 %.

# Вывод

В ходе выполнения лабораторной работы изучены возможности измерения динамических характеристик программ с помощью профилировщика на примере планировщика Sampler.

Проведено профилирование двух тестовых программ, в результате которого видно как изменяется оценка профилировщика в зависимости от количества итераций.

Проведено профилирование программы из первой лабораторной работы, в которой были обнаружены и устранены узкие места.

Программа была улучшена и при повторном измерении были измерены лучшие временные характеристики.

# Приложение А

1. #include "sampler.h"
2. #include <stdio.h>
3. #include <math.h>
5. /\* integration by the trapezoidal rule \*/
7. const double tol = 1.0e-6;
8. double sum, upper, lower;
10. /\* find f(x)=1/x \*/
11. /\* watch out for x=0 ! \*/
12. double fx(double x) {
13. return 1.0/x;
14. }
16. /\* numerical integration by the trapezoid method \*/
17. /\* function is FX, limits are LOWER and UPPER \*/
18. /\* with number of regions equal to PIECES \*/
19. /\* fixed partition is DELTA\_X, answer is SUM \*/
20. void trapez(double lower, double upper, double tol, double\* sum) {
21. SAMPLE;
22. int pieces, i;
23. double x, delta\_x, end\_sum, mid\_sum, sum1;
24. SAMPLE;
25. pieces = 1;
26. delta\_x = (upper - lower)/pieces;
27. SAMPLE;
28. end\_sum = fx(lower) + fx(upper);
29. SAMPLE;
30. \*sum = end\_sum \* delta\_x / 2.0;
31. mid\_sum = 0.0;
32. SAMPLE;
33. do {
34. SAMPLE;
35. pieces = pieces\*2;
36. sum1 = \*sum;
37. delta\_x = (upper-lower)/pieces;
38. SAMPLE;
39. for(i = 1; i <= pieces / 2; i++)
40. {
41. SAMPLE;
42. x = lower + delta\_x \* (2.0 \* i - 1.0);
43. mid\_sum = mid\_sum + fx(x);
44. SAMPLE;
45. }
46. SAMPLE;
47. \*sum = (end\_sum + 2.0 \* mid\_sum) \* delta\_x \* 0.5;
48. SAMPLE;
49. } while ( fabs(\*sum - sum1) > fabs(tol\*(\*sum)));
50. SAMPLE;
51. }
53. int main(int argc, char \*\*argv) {
54. sampler\_init(&argc, argv);
55. lower = 1.0;
56. upper = 9.0;
57. trapez(lower,upper,tol,&sum);
58. // printf("area= %.16e\n", sum);
59. return 0;
60. }

# Приложение Б

1. #include "sampler.h"
2. #include <stdio.h>
3. #include <math.h>
5. /\* integration by the trapezoidal rule \*/
7. const double tol = 1.0e-6;
8. double upper, lower;
10. /\* find f(x)=1/x \*/
11. /\* watch out for x=0 ! \*/
12. double fx(double x) {
13. return 1.0/x;
14. }
16. /\* numerical integration by the trapezoid method \*/
17. /\* function is FX, limits are LOWER and UPPER \*/
18. /\* with number of regions equal to PIECES \*/
19. /\* fixed partition is DELTA\_X, answer is SUM \*/
20. double trapez(double lower, double upper, double tol) {
21. SAMPLE;
22. double x;
23. int i;
24. double added\_sum;
25. SAMPLE;
26. int nseg = 1;
27. double dx = (upper - lower) / nseg;
28. SAMPLE;
29. double sum = 0.5 \* (fx(lower) + fx(upper));
30. SAMPLE;
31. double old\_ans = 0.0;
32. double ans = sum \* dx;
33. SAMPLE;
34. double err\_est = fmax(1.0, fabs(tol \* ans));
35. while(err\_est > fabs(tol \* ans)) {
36. SAMPLE;
37. old\_ans = ans;
38. dx = (upper - lower) / nseg;
39. x = lower + 0.5 \* dx;
40. added\_sum = 0.0;
41. SAMPLE;
42. for (i = 0; i < nseg; i++) {
43. SAMPLE;
44. added\_sum += fx(x + i \* dx);
45. SAMPLE;
46. }
47. SAMPLE;
48. sum += added\_sum;
49. nseg \*= 2;
50. ans = sum \* 0.5 \* dx;
51. err\_est = fabs(old\_ans - ans);
52. SAMPLE;
53. }
54. SAMPLE;
55. return ans;
56. }
58. int main(int argc, char \*\*argv) {
59. sampler\_init(&argc, argv);
60. lower = 1.0;
61. upper = 9.0;
62. double sum = trapez(lower,upper,tol);
63. // printf("area= %.16e\n", sum);
64. return 0;
65. }