**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

2022

# 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

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

Результат профилирования программы представлен в таблице 3.

| исх | прием | общее время | кол-во проходов | среднее время |
| --- | --- | --- | --- | --- |
| 21 | 25 | 11.066 | 1 | 11.066 |
| 25 | 28 | 12.759 | 1 | 12.759 |
| 28 | 30 | 26.035 | 1 | 26.035 |
| 30 | 33 | 1.181 | 1 | 1.181 |
| 33 | 35 | 0.949 | 1 | 0.949 |
| 35 | 39 | 26.354 | 12 | 2.196 |
| 39 | 42 | 54.489 | 12 | 4.541 |
| 42 | 45 | 38329.216 | 4095 | 9.360 |
| 45 | 47 | 140.338 | 12 | 11.695 |
| 45 | 42 | 7834.312 | 4083 | 1.919 |
| 47 | 49 | 20.639 | 12 | 1.720 |
| 49 | 35 | 71.473 | 11 | 6.498 |
| 49 | 51 | 4.466 | 1 | 4.466 |

Таблица 3. Результат профилирования программы из первой лабораторной работы.

Из таблицы видно, что наибольшее время занимают строки 28-30, однако в данных строках время обусловлено интегрируемой функцией и является частью пользовательского ввода.

В таком случае необходимо произвести оптимизацию остальных частей программы.

Было произведено улучшение программы:

* 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 4.

| исх | прием | общее время | кол-во проходов | среднее время |
| --- | --- | --- | --- | --- |
| 22 | 27 | 11.346 | 1 | 11.346 |
| 27 | 30 | 12.973 | 1 | 12.973 |
| 30 | 32 | 26.732 | 1 | 26.732 |
| 32 | 36 | 1.480 | 1 | 1.480 |
| 36 | 38 | 1.513 | 1 | 1.513 |
| 38 | 42 | 12.062 | 12 | 1.005 |
| 42 | 45 | 47.872 | 12 | 3.989 |
| 45 | 48 | 36307.406 | 4095 | 8.866 |
| 48 | 50 | 86.955 | 12 | 7.246 |
| 48 | 45 | 4754.657 | 4083 | 1.165 |
| 50 | 52 | 38.668 | 12 | 3.222 |
| 52 | 38 | 87.096 | 11 | 7.918 |
| 52 | 54 | 2.788 | 1 | 2.788 |

Таблица 4. Результат профилирования модифицированной программы из первой лабораторной.

Из таблицы видно, что большая часть участков кода занимает меньшее время, вплоть до 50%.

Время исполнения программы уменьшилось с 46533.277 до 41391.548, т. е. на 5141.729.

# 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;
25. SAMPLE;
26. pieces = 1;
27. delta\_x = (upper - lower)/pieces;
28. SAMPLE;
29. end\_sum = fx(lower) + fx(upper);
30. SAMPLE;
31. \*sum = end\_sum \* delta\_x / 2.0;
32. mid\_sum = 0.0;
33. SAMPLE;
34. do {
35. SAMPLE;
36. pieces = pieces\*2;
37. sum1 = \*sum;
38. delta\_x = (upper-lower)/pieces;
39. SAMPLE;
40. for(i = 1; i <= pieces / 2; i++)
41. {
42. SAMPLE;
43. x = lower + delta\_x \* (2.0 \* i - 1.0);
44. mid\_sum = mid\_sum + fx(x);
45. SAMPLE;
46. }
47. SAMPLE;
48. \*sum = (end\_sum + 2.0 \* mid\_sum) \* delta\_x \* 0.5;
49. SAMPLE;
50. } while ( fabs(\*sum - sum1) > fabs(tol\*(\*sum)));
51. SAMPLE;
52. }
54. int main(int argc, char \*\*argv) {
55. sampler\_init(&argc, argv);
56. lower = 1.0;
57. upper = 9.0;
58. trapez(lower,upper,tol,&sum);
59. // printf("area= %.16e\n", sum);
60. return 0;
61. }

# 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. double sum1;
24. int i;
26. SAMPLE;
27. int pieces = 1;
28. double delta\_x = (upper - lower)/pieces;
29. SAMPLE;
30. double end\_sum = fx(lower) + fx(upper);
31. SAMPLE;
33. double mid\_sum = 0.0;
34. double sum = end\_sum \* delta\_x / 2.0;
35. SAMPLE;
36. do {
37. SAMPLE;
38. pieces = pieces << 1;
39. sum1 = sum;
40. delta\_x = (upper-lower)/pieces;
41. SAMPLE;
42. for(i = 1; i <= pieces >> 1; i++)
43. {
44. SAMPLE;
45. x = lower + delta\_x \* (2.0 \* i - 1.0);
46. mid\_sum = mid\_sum + fx(x);
47. SAMPLE;
48. }
49. SAMPLE;
50. sum = (end\_sum + 2.0 \* mid\_sum) \* delta\_x \* 0.5;
51. SAMPLE;
52. } while (fabs(sum - sum1) > fabs(tol\*sum));
53. SAMPLE;
54. return sum;
55. }
57. int main(int argc, char \*\*argv) {
58. sampler\_init(&argc, argv);
59. lower = 1.0;
60. upper = 9.0;
61. double sum = trapez(lower,upper,tol);
62. // printf("area= %.16e\n", sum);
63. return 0;
64. }