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_v2

Студент гр. 8304 Алтухов А.Д.

Преподаватель Ефремов М.А.

Санкт-Петербург
2022

Цель работы.

Изучить возможности измерения динамических характеристик программ с помощью профилировщиков на примере профилировщика SAMPLER.

Ход выполнения.

Были выполнены под управлением монитора SAMPLER_v2 тестовые программы test_cyc.c и test_sub.c. Результаты представлены на рисунках 1 и 2.

исх	прием	общее время	кол-во проходов	среднее время
13	15	3093.889	1	3093.889
15	17	6371.667	1	6371.667
17	19	15623.333	1	15623.333
19	21	30878.333	1	30878.333
21	24	2972.778	1	2972.778
24	27	6683.333	1	6683.333
27	30	15303.889	1	15303.889
30	33	32942.778	1	32942.778
33	39	2960.556	1	2960.556
39	45	6047.222	1	6047.222
45	51	17267.222	1	17267.222
51	57	31368.333	1	31368.333

Рисунок 1 – Результат работы SAMPLER_v2 для программы test_cyc.c

исх	прием	общее время	кол-во проходов	среднее время
30	32	24443497.222	1	24443497.222
32	34	50713625.556	1	50713625.556
34	36	121370780.556	1	121370780.556
36	38	243180856.667	1	243180856.667

Рисунок 2 – Результат работы SAMPLER_v2 для программы test_sub.c

Под управлением монитора SAMPLER_v2 была выполнена программа из лабораторной работы №1. Результат измерений полного времени выполнения функции simps представлен на рисунке 3. Исходный код программы представлен в приложении А.

исх	прием	общее время	кол-во проходов	среднее время
49	51	7251.111	1	7251.111

Рисунок 3 – Результат работы SAMPLER_v2 для выполнения функции simps

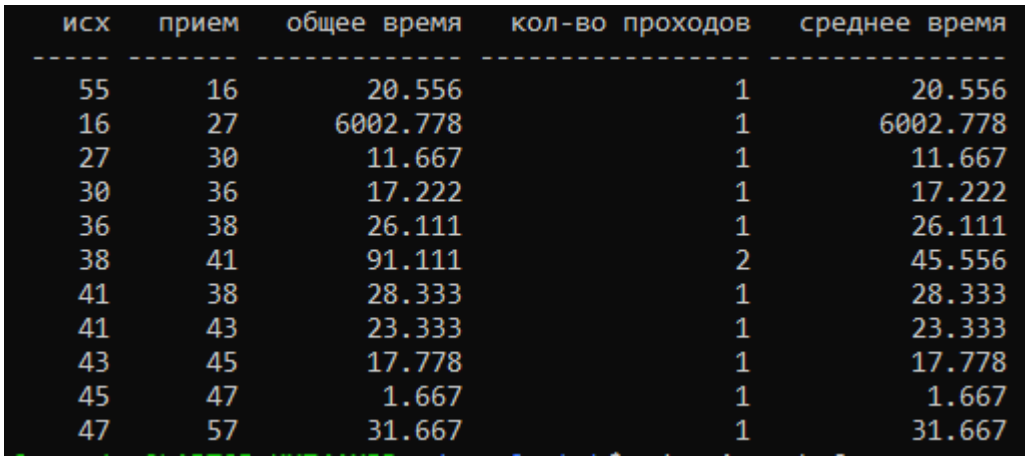
Было выполнено разбиение программы из лабораторной работы №1 на функциональные участки. Исходный код программы, разбитый на функциональные участки, представлен в приложении Б. Полученные с помощью программы SAMPLER_v2 результаты представлены на рисунке 4.

исх	прием	общее время	кол-во проходов	среднее время
55	16	17.222	1	17.222
16	27	6872.222	1	6872.222
27	30	9.444	1	9.444
30	36	16.667	1	16.667
36	38	26.667	1	26.667
38	41	76.111	2	38.056
41	38	31.111	1	31.111
41	43	21.667	1	21.667
43	45	19.444	1	19.444
45	47	13.333	1	13.333
47	57	37.222	1	37.222

Рисунок 4 – Результат работы SAMPLER_v2 для выполнения функции simps

Как видно из результатов измерения времени выполнения функциональных участков, наиболее затратным фрагментом являются строчки 16-27 с большим количеством вычислений. Для оптимизации работы программы были убраны вызовы дополнительных расчетных функций.

Была выполнена проверка измененной программы. Результат представлен на рисунке 5. Исходный код модифицированной программы представлен в приложении В.



исх	прием	общее время	кол-во проходов	среднее время
55	16	20.556	1	20.556
16	27	6002.778	1	6002.778
27	30	11.667	1	11.667
30	36	17.222	1	17.222
36	38	26.111	1	26.111
38	41	91.111	2	45.556
41	38	28.333	1	28.333
41	43	23.333	1	23.333
43	45	17.778	1	17.778
45	47	1.667	1	1.667
47	57	31.667	1	31.667

Рисунок 5 – Результат работы SAMPLER_v2 для выполнения модифицированной функции simps

Время выполнения проблемного участка составило 6002,778 мкс, по сравнению с прошлыми 6872,222 мкс. Удалось добиться сокращения времени выполнения на 869,444 мкс (12%).

Выводы.

В ходе выполнения лабораторной работы были изучены возможности измерения динамических характеристик программ с помощью профилировщиков на примере профилировщика SAMPLER_v2. Для программы, взятой из первой лабораторной работы, было выполнено измерение времени работы, с последующим выявлением узкого места и его устранения. В результате удалось добиться более быстрого выполнения программы.

Приложение А.

```
#include <stdio.h>
#include <math.h>
#include "sampler.h"

const double tol = 1.0E-6;

double fx(double x){
   return exp(-x/2);
}

double dfx(double x){
   return -(exp(-x/2))/2;
}

double simps(double lower, double upper, double tol, double* sum){
   int pieces=2;
   double delta_x=(upper-lower)/pieces;
   double odd_sum = fx(lower+delta_x);
   double even_sum =0.0;
   double end_sum =fx(lower)+fx(upper);
   double end_cor =dfx(lower)-dfx(upper);
   *sum=(end_sum+4.0*odd_sum)*delta_x/3.0;

   double sum1;
   double x;

   do
   {
      pieces=pieces*2;
```

```
            sum1=*sum;
            delta_x=(upper-lower)/pieces;
            even_sum=even_sum+odd_sum;
            odd_sum=0.0;
            for (int i=1; i<=pieces/2; i++) {

                    x=lower+delta_x*(2.0*i-1.0);
                    odd_sum=odd_sum+fx(x);

            }

*sum=(7.0*end_sum+14.0*even_sum+16.00*odd_sum+end_cor*delta_x)*delta_x
/15.0;
          } while ( (*sum!=sum1) && (fabs(*sum-sum1)<=fabs(tol*(*sum))) );
        }
        int main(int argc, char** argv)
        {
             sampler_init(&argc, argv);
           double lower=1.0;
           double upper=9.0;
           double sum = 0.0;
             SAMPLE;
           simps(lower,upper,tol,&sum);
             SAMPLE;
           //printf("\narea= %f\n", sum);
           return 0;
        }
```

Приложение Б.

```
#include <stdio.h>
#include <math.h>
#include "sampler.h"

const double tol = 1.0E-6;

double fx(double x){
   return exp(-x/2);
}

double dfx(double x){
   return -(exp(-x/2))/2;
}

double simps(double lower, double upper, double tol, double* sum){
        SAMPLE;
   int pieces=2;
   double delta_x=(upper-lower)/pieces;
   double odd_sum = fx(lower+delta_x);
   double even_sum =0.0;
   double end_sum =fx(lower)+fx(upper);
   double end_cor =dfx(lower)-dfx(upper);
   *sum=(end_sum+4.0*odd_sum)*delta_x/3.0;

   double sum1;
   double x;
   SAMPLE;
   do
   {
```

```
                SAMPLE;
            pieces=pieces*2;
            sum1=*sum;
            delta_x=(upper-lower)/pieces;
            even_sum=even_sum+odd_sum;
            odd_sum=0.0;
                SAMPLE;
            for (int i=1; i<=pieces/2; i++) {
                    SAMPLE;
                x=lower+delta_x*(2.0*i-1.0);
                odd_sum=odd_sum+fx(x);
                    SAMPLE;
            }
                SAMPLE;

*sum=(7.0*end_sum+14.0*even_sum+16.00*odd_sum+end_cor*delta_x)*delta_x
/15.0;
                    SAMPLE;
          } while ( (*sum!=sum1) && (fabs(*sum-sum1)<=fabs(tol*(*sum))) );
              SAMPLE;
        }
        int main(int argc, char** argv)
        {
              sampler_init(&argc, argv);
           double lower=1.0;
           double upper=9.0;
           double sum = 0.0;
              SAMPLE;
           simps(lower,upper,tol,&sum);
              SAMPLE;
```

```
    //printf("\narea= %f\n", sum);
    return 0;
}
```

Приложение В.

```
#include <stdio.h>
#include <math.h>
#include "sampler.h"

const double tol = 1.0E-6;

double fx(double x){
   return exp(-x/2);
}

double dfx(double x){
   return -(exp(-x/2))/2;
}

double simps(double lower, double upper, double tol, double* sum){
   SAMPLE;
   int pieces=2;
   double delta_x=(upper-lower)/pieces;
   double odd_sum = exp(-(lower+delta_x)/2);
   double even_sum =0.0;
   double end_sum =exp(-lower/2)+exp(-upper/2);
   double end_cor =-(exp(-lower/2))/2-(-(exp(-upper/2))/2);
   *sum=(end_sum+4.0*odd_sum)*delta_x/3.0;

   double sum1;
   double x;
   SAMPLE;
   do
```

```
    {
        SAMPLE;
        pieces=pieces*2;
        sum1=*sum;
        delta_x=(upper-lower)/pieces;
        even_sum=even_sum+odd_sum;
        odd_sum=0.0;
        SAMPLE;
        for (int i=1; i<=pieces/2; i++) {
            SAMPLE;
                x=lower+delta_x*(2.0*i-1.0);
                odd_sum=odd_sum+fx(x);
            SAMPLE;
        }
        SAMPLE;

*sum=(7.0*end_sum+14.0*even_sum+16.00*odd_sum+end_cor*delta_x)*delta_x
/15.0;
        SAMPLE;
    } while ( (*sum!=sum1) && (fabs(*sum-sum1)<=fabs(tol*(*sum))) );
    SAMPLE;
}
int main(int argc, char** argv)
{
    sampler_init(&argc, argv);
    double lower=1.0;
    double upper=9.0;
    double sum = 0.0;
         SAMPLE;
    simps(lower,upper,tol,&sum);
```

```
        SAMPLE;
//  printf("\narea= %f\n", sum);
    return 0;
}
```