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 _____ Сергеев А.Д.

Преподаватель _____ Кирьянчиков В. А.

Санкт-Петербург
2022

Цель работы.

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

Задание.

1) Выполнить под управлением SAMPLER тестовые программы test_cyc.c и test_sub.c и привести отчет по результатам их выполнения с анализом параметров повторения циклов, структуры описания циклов, способов профилирования процедур и проверкой их влияния на точность и чувствительность профилирования.
2) Разработанную в лаб. работе 1 программу, реализующую заданный вычислительный алгоритм, разбить на функциональные участки (ФУ) и расставить на их границах контрольные точки (КТ) для выполнения с помощью ПИМ SAMPLER измерений и получения профиля выполнения программы, представляющего времена выполнения и количество выполнений каждого ФУ.
3) Скомпилировать полученную программу. При компиляции добавить путь к sampler.h в набор путей поиска включаемых файлов (`Isampler/libsampler` при компиляции, если архив был распакован в текущий каталог), при линковке добавить путь к libsampler.a в набор путей поиска библиотек и подключить её (флаги `-LSampler/build/libsampler -lsampler` при линковке).
4) Выполнить скомпилированную программу под управлением Sampler’a с внешним зацикливанием и получить отчет по результатам профилирования. Зацикливание можно выполнять при помощи программы sampler-repeat. Использование программы приведено в разделе 4 документа «Описание работы с ПИМ SAMPLER_v2». Число повторов зависит от сложности самой программы; имеет смысл начальное число запусков взять равным 10 и увеличивать его в 5–10 раз до тех пор, пока среднее время выполнения участков не стабилизируется, или на запуски станет уходить слишком много

времени, или на результаты станет уходить слишком много дискового пространства.

5) Проанализировать полученный отчет и выявить "узкие места", приводящие к ухудшению производительности программы.
6) Ввести в программу усовершенствования для повышения производительности, получить новые профили, добавить их в отчет и объяснить полученные результаты.

Ход работы.

Под управлением монитора SAMPLER были выполнены тестовые программы test_cyc.c и test_sub.c с параметрами 20 5. Результаты представлены таблицах 1 и 2.

Таблица 1 – Результат работы SAMPLER для программы test_cyc.cpp

исх	прием	общее время	кол-во проходов	среднее время
13	15	4059.833	1	4059.833
15	17	131119.833	1	131119.833
17	19	19912.100	1	19912.100
19	21	41361.233	1	41361.233
21	24	4042.400	1	4042.400
24	27	7990.900	1	7990.900
27	30	19868.033	1	19868.033
30	33	47463.467	1	47463.467
33	39	4048.000	1	4048.000
39	45	8002.567	1	8002.567
45	51	19894.467	1	19894.467
51	57	94070.767	1	94070.767

Таблица 2 - Результат работы SAMPLER для программы test_sub.cpp

исх	прием	общее время	кол-во проходов	среднее время
30	32	21253855.367	1	21253855.367
32	34	43148903.733	1	43148903.733
34	36	109540422.367	1	109540422.367
36	38	215908309.100	1	215908309.100

Таким же образом с параметрами 15 5 была выполнена программа из лабораторной работы №1. Результат измерений для полного времени выполнения функции Bessy представлен в таблице 3. Исходный код этой программы представлен

в Приложении А.

Таблица 3 - Результат работы SAMPLER для измерения полного времени выполнения функции

исх	прием	общее время	кол-во проходов	среднее время
50	63	11942.750	1	11942.750

Программа из лабораторной работы №1 была разбита на функциональные участки. Исходный код этой программы представлен в приложении Б. Полученные с помощью монитора SAMPLER (с параметрами 15 5) результаты представлены в таблице 4.

Таблица 4 - Результат работы SAMPLER для измерения полного времени выполнения программы, разбитой на функциональные участки

исх	прием	общее время	кол-во проходов	среднее время
63	70	52.600	1	52.600
70	77	80.500	1	80.500
77	33	48.200	1	48.200
33	39	37.100	1	37.100
39	41	27.000	1	27.000
41	45	246.900	12	20.575
45	41	149.300	11	13.573
45	47	70.900	1	70.900
47	49	20832.800	1	20832.800
49	80	65.300	1	65.300
80	82	25.800	1	25.800
82	84	28.000	1	28.000

Общее время составило 21664,4 мкс. Как видно из результатов измерения времени выполнения функциональных участков – наиболее затратным фрагментом является вызов функции exp (строчки 47-49).

Была проведена оптимизация кода, реализована оптимизированная версия функции exp.

```
double my_exp( const double x ){
      double dVal, dTemp;
      int nStep = 1;
      for( dVal = 1.0, dTemp = 1.0; dTemp >= 1e-6 ; ++nStep ){
            dTemp *= x/nStep;
            dVal += dTemp;
```

```
        }
        return dVal;
    }
```

Была выполнена проверка оптимизированной программы, результат представлен в таблице 6. Исходный код модифицированной программы представлен в Приложении В.

Таблица 6 - Результат работы SAMPLER для измерения полного времени выполнения оптимизированной программы, разбитой на функциональные участки

исх	**прием**	**общее время**	**кол-во проходов**	**среднее время**
80	87	57.500	1	57.500
87	94	90.200	1	90.200
94	49	131.700	1	131.700
49	55	38.200	1	38.200
55	57	29.600	1	29.600
57	61	261.200	12	21.767
61	57	144.700	11	13.155
61	63	44.200	1	44.200
63	10	31.200	1	31.200
10	13	26.600	1	26.600
13	14	34.000	1	34.000
14	15	61.300	1	61.300
14	20	47.700	1	47.700
15	18	32.400	1	32.400
18	14	30.000	1	30.000
20	66	38.100	1	38.100
66	97	97.900	1	97.900
97	99	31.600	1	31.600
99	101	23.200	1	23.200

Общее время составило 918.4 мкс. В результате внесённых изменений удалось добиться снижения времени выполнения на 20746 мкс (96%).

Выводы.

В ходе выполнения лабораторной работы были изучены возможности измерения динамических характеристик программ с помощью профилировщиков на примере монитора SAMPLER. Для программы, разработанной в первой лабораторной работе, было выполнено измерение времени работы, с последующим выявлением неоптимальных мест и их устранения, в результате чего удалось получить более эффективную программу, сократив время работы на 96%.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

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

const double sqrtpi = 1.7724538;
const double tol = 1.0E-4;
const int terms = 12;

// infinite series expansion of the Gaussian error function
double erf (double x) {
    double x2 = x * x;
    double sum = x;
    double term = x;
    int i = 0;
    do {
        i = i + 1;
        double sum1 = sum;
        term = 2.0 * term * x2 / (1.0 + 2.0 * i);
        sum = term + sum1;
    } while (term < tol * sum);
    return 2.0 * sum * exp(-x2) / sqrtpi;
}

// complement of error function
double erfc (double x) {
    double x2,u,v,sum;
    x2 = x * x;
    v = 1.0 / (2.0 * x2);
    u = 1.0 + v * (terms + 1.0);
    int i = terms;
    do {
        sum = 1.0 + i * v / u;
        u = sum;
        i--;
    } while (i >= 1);
    return exp(-x2) / (x * sum * sqrtpi);
}


// evaluation of the gaussian error function
int main (int argc, char **argv) {
    sampler_init(&argc, argv);
    double x, er, ec;
    int done = 1;
    do {
        printf("Arg? ");
```

```
        scanf("%lf", &x);
        if (x < 0.0) done = 0;
        else {
            SAMPLE;
            if (x == 0.0) {
                er = 0.0;
                ec = 1.0;
            } else {
                if (x < 1.5) {
                    er = erf(x);
                    ec = 1.0 - er;
                } else {
                    ec = erfc(x);
                    er = 1.0 - ec;
                }
            }
            SAMPLE;
            printf("X = %.8lf; Erf = %.12lf; Erfc = %.12lf\n", x, er, ec);
        }
    } while (done);
}
```

ПРИЛОЖЕНИЕ Б.

КОД ПРОГРАММЫ С РАЗДЕЛЕНИЕМ НА ФУНКЦИОНАЛЬНЫЕ УЧАСТКИ

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

const double sqrtpi = 1.7724538;
const double tol = 1.0E-4;
const int terms = 12;

// infinite series expansion of the Gaussian error function
double erf (double x) {
    SAMPLE;
    double x2 = x * x;
    double sum = x;
    double term = x;
    int i = 0;
    SAMPLE;
    do {
        SAMPLE;
        i = i + 1;
        double sum1 = sum;
        term = 2.0 * term * x2 / (1.0 + 2.0 * i);
        sum = term + sum1;
        SAMPLE;
    } while (term < tol * sum);
    SAMPLE;
    double res = 2.0 * sum * exp(-x2) / sqrtpi;
    SAMPLE;
    return res;
}

// complement of error function
double erfc (double x) {
    SAMPLE;
    double x2,u,v,sum;
    x2 = x * x;
    v = 1.0 / (2.0 * x2);
    u = 1.0 + v * (terms + 1.0);
    int i = terms;
    SAMPLE;
    do {
        SAMPLE;
        sum = 1.0 + i * v / u;
        u = sum;
```

```
        i--;
        SAMPLE;
    } while (i >= 1);
    SAMPLE;
    double res = exp(-x2) / (x * sum * sqrtpi);
    SAMPLE;
    return res;
}

// evaluation of the gaussian error function
int main (int argc, char **argv) {
    sampler_init(&argc, argv);
    double x, er, ec;
    int done = 1;
    do {
        printf("Arg? ");
        scanf("%lf", &x);
        if (x < 0.0) done = 0;
        else {
            SAMPLE;
            if (x == 0.0) {
                SAMPLE;
                er = 0.0;
                ec = 1.0;
                SAMPLE;
            } else {
                SAMPLE;
                if (x < 1.5) {
                    SAMPLE;
                    er = erf(x);
                    ec = 1.0 - er;
                    SAMPLE;
                } else {
                    SAMPLE;
                    ec = erfc(x);
                    er = 1.0 - ec;
                    SAMPLE;
                }
                SAMPLE;
            }
            SAMPLE;
            printf("X = %.8lf; Erf = %.12lf; Erfc = %.12lf\n", x, er, ec);
        }
    } while (done);
}
```

ПРИЛОЖЕНИЕ В.

КОД ОПТИМИЗИРОВАННОЙ ПРОГРАММЫ

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

const double sqrtpi = 1.7724538;
const double tol = 1.0E-4;
const int terms = 12;

double my_exp( const double x ){
      SAMPLE;
      double dVal, dTemp;
      int nStep = 1;
      SAMPLE;
      for( dVal = 1.0, dTemp = 1.0; SAMPLE, dTemp >= 1e-6 ; ++nStep ){
            SAMPLE;
            dTemp *= x/nStep;
            dVal += dTemp;
            SAMPLE;
      }
      SAMPLE;
      return dVal;
}

// infinite series expansion of the Gaussian error function
double erf (double x) {
    SAMPLE;
    double x2 = x * x;
    double sum = x;
    double term = x;
    int i = 0;
    SAMPLE;
    do {
        SAMPLE;
        i = i + 1;
        double sum1 = sum;
        term = 2.0 * term * x2 / (1.0 + 2.0 * i);
        sum = term + sum1;
        SAMPLE;
    } while (term < tol * sum);
    SAMPLE;

    double res = 2.0 * sum * my_exp(-x2) / sqrtpi;
    SAMPLE;
    return res;
}

// complement of error function
double erfc (double x) {
    SAMPLE;
    double x2,u,v,sum;
    x2 = x * x;
    v = 1.0 / (2.0 * x2);
    u = 1.0 + v * (terms + 1.0);
    int i = terms;
    SAMPLE;
    do {
        SAMPLE;
        sum = 1.0 + i * v / u;
```

```
        u = sum;
        i--;
        SAMPLE;
    } while (i >= 1);
    SAMPLE;

    double res = my_exp(-x2) / (x * sum * sqrtpi);
    SAMPLE;
    return res;
}

// evaluation of the gaussian error function
int main (int argc, char **argv) {
    sampler_init(&argc, argv);
    double x, er, ec;
    int done = 1;
    do {
        printf("Arg? ");
        scanf("%lf", &x);
        if (x < 0.0) done = 0;
        else {
            SAMPLE;
            if (x == 0.0) {
                SAMPLE;
                er = 0.0;
                ec = 1.0;
                SAMPLE;
            } else {
                SAMPLE;
                if (x < 1.5) {
                    SAMPLE;
                    er = erf(x);
                    ec = 1.0 - er;
                    SAMPLE;
                } else {
                    SAMPLE;
                    ec = erfc(x);
                    er = 1.0 - ec;
                    SAMPLE;
                }
                SAMPLE;
            }
            SAMPLE;
            printf("X = %.8lf; Erf = %.12lf; Erfc = %.12lf\n", x, er, ec);
        }
    } while (done);
}
```