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»

Студентка гр. 8304 _____ Мельникова О.А.

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.

Были выполнены под управлением монитора SAMPLER тестовые программы test_cyc.c и test_sub.c. 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

Была выполнена под управлением монитора SAMPLER 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
82	84	62703.500	1	62703.500
84	82	2191001107.750	1	2191001107.750

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 результаты представлены в таблице 4.

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

исх	прием	общее время	кол-во проходов	среднее время
110	13	169.000	1	169.000
13	15	52.500	1	52.500
15	23	13365.000	1	13365.000
23	25	28.500	1	28.500
25	27	48.500	4	12.125
27	33	44.000	1	44.000
27	29	142.500	3	47.500
33	37	122.500	4	30.625
37	25	95.500	3	31.833
37	39	67.000	1	67.000
29	31	80.500	3	26.833
31	33	68.000	3	22.667
39	44	27.500	1	27.500
44	46	27.500	1	27.500
46	52	254.000	4	63.500
52	46	95.000	3	31.667
52	54	51.500	1	51.500
54	57	32.000	1	32.000
57	65	75.500	1	75.500
65	70	30.500	1	30.500
70	72	3771.500	1	3771.500
72	74	24.000	1	24.000
74	79	132.500	4	33.125
79	74	134.000	3	44.667
79	81	69.500	1	69.500
81	83	29.500	1	29.500
83	112	82.000	1	82.000

Общее время составило 19120 мкс. Как видно из результатов измерения времени выполнения функциональных участков – наиболее затратным фрагментом является вызов функции log (строчки 15-23) и trunc(строчки 70-72), также во время тестирования с другими данными было выявлено, что функции sqrt и sin также трудозатратны.

Была проведена оптимизация кода. Измененные участки приведены в табл. 5.

Таблица 5 – Результат оптимизации кода.

Старый код	Новый код

`t = log(xx) + euler;`	`double myLog = 0.0;`
	`double a = 0.5;`
	`double u=(double)xx;`
	`for(int i=0; i<16; i++){`
	`  u=u*u;`
	`  if(u>2.7182818) {`
	`  u=u/2.7182818;`
	`  myLog+=a;`
	`  }`
	`  a/=2;`
	`}`
	`t = myLog + euler;`

`trunc(n+0.01)`	`(int)(n+0.01)`
`sqrt(2 / (pi*x))`	`float mySqrt = 2 / (pi*x);` `__asm__ ( "fsqrt" : "+t" (mySqrt) );`

`sin(x - pi/4 - n * pi/2)`	`float toSin = x - pi/4 - n * pi/2;`
	`toSin *= 0.63661977236758134308;`
	`int sign = toSin < 0.0;`
	`toSin = sign ? -toSin : toSin;`
	`int xf = (int)toSin;`
	`toSin -= xf;`
	`if ((xf & 1) == 1)`
	`  toSin = 1 - toSin;`
	`int per = ((xf >> 1) & 1) == 1;`
	`float xxForSin = toSin * toSin;`
	`float y = toSin * (1.5707903005870776 +`
	`xxForSin * (-0.6458858977085938 +`
	`xxForSin*(0.07941798513358536 -`
	`0.0043223880120647346 * xxForSin)));`
	`float mySin = sign ^ per ? -y : y;`

Была выполнена проверка изменённой 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 функции, разбитой на функциональные участки

исх	**прием**	**общее время**	**кол-во проходов**	**среднее время**
151	13	44.500	1	44.500
13	15	67.000	1	67.000
15	21	93.000	1	93.000
21	24	54.500	1	54.500
24	26	136.500	16	8.531
26	29	103.500	16	6.469
29	32	187.500	16	11.719
32	34	163.000	16	10.188
34	36	192.000	16	12.000
36	24	199.000	15	13.267
36	38	12.500	1	12.500
38	44	29.000	1	29.000
44	46	29.500	1	29.500
46	48	56.500	3	18.833
48	54	55.500	1	55.500
48	50	107.500	2	53.750
54	58	89.500	3	29.833
58	46	76.500	2	38.250
58	60	70.500	1	70.500
50	52	60.000	2	30.000
52	54	42.000	2	21.000
60	65	28.000	1	28.000
65	67	26.500	1	26.500
67	73	248.500	4	62.125
73	67	89.500	3	29.833
73	75	49.500	1	49.500
75	78	27.500	1	27.500
78	86	80.500	1	80.500
86	91	35.000	1	35.000
91	93	33.000	1	33.000
93	95	63.500	1	63.500
95	100	76.000	4	19.000
100	95	77.000	3	25.667
100	102	49.000	1	49.000
102	104	23.500	1	23.500
104	153	96.500	1	96.500

Общее время составило 2873 мкс. В результате внесённых изменений удалось добиться снижения времени выполнения на 16247 мкс (85%).

Выводы.

В ходе выполнения лабораторной работы были изучены возможности измерения динамических характеристик программ с помощью профилировщиков на

примере профилировщика SAMPLER. Для программы, взятой из первой лабораторной работы, было выполнено измерение времени работы, с последующим выявлением узких мест и их устранения – в результате чего удалось получить более эффективную программу.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

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

float bessy(float x, float n){
   const float small = 1.0E-8;
            const float euler =             0.57721566;
            const float pi = 3.1415926;
            const float pi2 = 0.63661977;

            float x2, sum, t, ts, term, xx, y0, y1, ya, yb, yc, ans;

   if(x<12){
      xx = 0.5 * x;
      x2 = xx * xx;
      t = log(xx) + euler;
      sum = 0.0;
      term = t;
      y0 = t;
      int j = 0;
      do{
         j = j+1;
         if(j != 1) sum = sum + 1/(j-1);
         ts = t-sum;
         term = -x2 * term / (j*j) * (1-1 / (j*ts));
         y0 = y0+term;
      }while(!(abs(term) < small));
      term = xx * (t-0.5);
      sum = 0.0;
      y1 = term;
      j = 1;
      do{
         j = j+1;
         sum = sum+1/(j-1);
         ts = t-sum;
         term = (-x2 * term) / (j * (j-1)) * ((ts-0.5 / j) / (ts + 0.5 / (j-1)));
         y1 = y1+term;
      }while(!(abs(term) < small));
      y0 = pi2 * y0;
      y1 = pi2 * (y1 - 1/x);
      if(n == 0.0){
         ans = y0;
      }else if(n == 1.0){
         ans = y1;
      }else{
         ts = 2.0/x;
         ya = y0;
         yb = y1;
         int j=2;
         if(j<=trunc(n+0.01)){
            do{
               yc = ts*(j-1)*yb-ya;
               ya = yb;
               yb = yc;
               j+=1;
            }while(j<=trunc(n+0.01));
         }
         ans = yc;
      }
```

```
        return ans;
    }else{
        return sqrt(2 / (pi*x)) * sin(x - pi/4 - n * pi/2);
    }
}

int main(int argc, char** argv){
    sampler_init(&argc, argv);
    float x, ordr;
    int done = 0;
    printf("\n");
    do{
        printf("Order? \n");
        scanf("%f", &ordr);
        if(ordr < 0.0){
            done = 1;
        }else{
            do{
                printf("Arg? \n");
                scanf("%f", &x);
            }while(!(x >= 0.0));
             SAMPLE;
            printf("Y Bessel is %f \n", bessy(x,ordr));
             SAMPLE;
        }
    }while(!(done));
    return 0;
}
```

ПРИЛОЖЕНИЕ Б.

КОД ПРОГРАММЫ С РАЗДЕЛЕНИЕМ НА ФУ

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

float bessy(float x, float n){
   const float small = 1.0E-8;
            const float euler =             0.57721566;
            const float pi = 3.1415926;
            const float pi2 = 0.63661977;

            float x2, sum, t, ts, term, xx, y0, y1, ya, yb, yc, ans;
   SAMPLE;
   if(x<12){
      SAMPLE;
      xx = 0.5 * x;
      x2 = xx * xx;
      t = log(xx) + euler;
      sum = 0.0;
      term = t;
      y0 = t;
      int j = 0;
               SAMPLE;
      do{
         SAMPLE;
         j = j+1;
         SAMPLE;
         if(j != 1) {
            SAMPLE;
            sum = sum + 1/(j-1);
            SAMPLE;
         }
         SAMPLE;
         ts = t-sum;
         term = -x2 * term / (j*j) * (1-1 / (j*ts));
         y0 = y0+term;
         SAMPLE;
      }while(!(abs(term) < small));
      SAMPLE;
      term = xx * (t-0.5);
      sum = 0.0;
      y1 = term;
      j = 1;
     SAMPLE;
      do{
         SAMPLE;
         j = j+1;
         sum = sum+1/(j-1);
         ts = t-sum;
         term = (-x2 * term) / (j * (j-1)) * ((ts-0.5 / j) / (ts + 0.5 / (j-1)));
         y1 = y1+term;
         SAMPLE;
      }while(!(abs(term) < small));
      SAMPLE;
      y0 = pi2 * y0;
      y1 = pi2 * (y1 - 1/x);
      SAMPLE;
      if(n == 0.0){
```

```
        SAMPLE;
        return y0;
      }else if(n == 1.0){
        SAMPLE;
        return y1;
      }else{
        SAMPLE;
        ts = 2.0/x;
        ya = y0;
        yb = y1;
        int j=2;
        SAMPLE;
        if(j<=trunc(n+0.01)){
          SAMPLE;
          do{
                SAMPLE;
            yc = ts*(j-1)*yb-ya;
            ya = yb;
            yb = yc;
            j+=1;
            SAMPLE;
          }while(j<=trunc(n+0.01));
          SAMPLE;
        }
        SAMPLE;
        return yc;
      }
    }else{
      SAMPLE;
          float res = sqrt(2 / (pi*x)) * sin(x - pi/4 - n * pi/2);
          SAMPLE;
          return res;
    }

}

int main(int argc, char** argv){
    sampler_init(&argc, argv);
    float x, ordr;
    int done = 0;
    printf("\n");
    do{
      printf("Order? \n");
      scanf("%f", &ordr);
      if(ordr < 0.0){
        done = 1;
      }else{
        do{
          printf("Arg? \n");
          scanf("%f", &x);
        }while(!(x >= 0.0));
         SAMPLE;
         float res = bessy(x,ordr);
         SAMPLE;
        printf("Y Bessel is %f \n", res);

      }
    }while(!(done));
    return 0;
}
```

ПРИЛОЖЕНИЕ В.

КОД ПРОГРАММЫ С ОПТИМИЗАЦИЯМИ

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

float bessy(float x, float n){
   const float small = 1.0E-8;
          const float euler =          0.57721566;
          const float pi = 3.1415926;
          const float pi2 = 0.63661977;

          float x2, sum, t, ts, term, xx, y0, y1, ya, yb, yc, ans;
   SAMPLE;
   if(x<12){
      SAMPLE;
      xx = x/2;
      x2 = xx * xx;
      double myLog = 0.0;
      double a = 0.5;
      double u=(double)xx;
      SAMPLE;
      for(int i=0; i<16; i++)
      {
         SAMPLE;
         u=u*u;
         SAMPLE;
         if(u>2.7182818)
         {
            SAMPLE;
            u=u/2.7182818;
            myLog+=a;
            SAMPLE;
         }
         SAMPLE;
         a/=2;
         SAMPLE;
      }
      SAMPLE;
      t = myLog + euler;
      sum = 0.0;
      term = t;
      y0 = t;
      int j = 0;
      SAMPLE;
      do{
         SAMPLE;
         j = j+1;
         SAMPLE;
         if(j != 1) {
            SAMPLE;
            sum = sum + 1/(j-1);
            SAMPLE;
         }
         SAMPLE;
         ts = t-sum;
         term = -x2 * term / (j*j) * (1-1 / (j*ts));
         y0 = y0+term;
```

```
      SAMPLE;
    }while(!(abs(term) < small));
    SAMPLE;
    term = xx * (t-0.5);
    sum = 0.0;
    y1 = term;
    j = 1;
    SAMPLE;
    do{
      SAMPLE;
      j = j+1;
      sum = sum+1/(j-1);
      ts = t-sum;
      term = (-x2 * term) / (j * (j-1)) * ((ts-0.5 / j) / (ts + 0.5 / (j-1)));
      y1 = y1+term;
      SAMPLE;
    }while(!(abs(term) < small));
    SAMPLE;
    y0 = pi2 * y0;
    y1 = pi2 * (y1 - 1/x);
    SAMPLE;
    if(n == 0.0){
      SAMPLE;
      return y0;
    }else if(n == 1.0){
      SAMPLE;
      return y1;
    }else{
      SAMPLE;
      ts = 2.0/x;
      ya = y0;
      yb = y1;
      int j=2;
      SAMPLE;
      if(j<=(int)(n+0.01)){
        SAMPLE;
        do{
          SAMPLE;
          yc = ts*(j-1)*yb-ya;
          ya = yb;
          yb = yc;
          j+=1;
          SAMPLE;
        }while(j<=(int)(n+0.01));
        SAMPLE;
      }
      SAMPLE;
      return yc;
    }
  }else{
    SAMPLE;
    float mySqrt = 2 / (pi*x);
    __asm__ ( "fsqrt" : "+t" (mySqrt) );

    float toSin = x - pi/4 - n * pi/2;
    toSin *= 0.63661977236758134308; // 2/Pi
    int sign = toSin < 0.0;
    toSin = sign ? -toSin : toSin;
    int xf = (int)toSin;
    toSin -= xf;
    SAMPLE;
```

```
        if ((xf & 1) == 1){
            SAMPLE;
            toSin = 1 - toSin;
        }
        int per = ((xf >> 1) & 1) == 1;
        float xxForSin = toSin * toSin;
        float y = toSin * (1.5707903005870776 + xxForSin * (-0.6458858977085938 +
        xxForSin*(0.07941798513358536 - 0.0043223880120647346 * xxForSin)));
        float mySin = sign ^ per ? -y : y;
        float res = mySqrt * mySin;
        SAMPLE;
        return res;

    }

}

int main(int argc, char** argv){
    sampler_init(&argc, argv);
    float x, ordr;
    int done = 0;
    printf("\n");
    do{
        printf("Order? \n");
        scanf("%f", &ordr);
        if(ordr < 0.0){
            done = 1;
        }else{
            do{
                printf("Arg? \n");
                scanf("%f", &x);
            }while(!(x >= 0.0));
            SAMPLE;
             float res = bessy(x,ordr);
             SAMPLE;
            printf("Y Bessel is %f \n", res);
        }
    }while(!(done));
    return 0;
}
```