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
105	13	39.500	1	39.500
13	22	12329.000	1	12329.000
22	37	355.000	1	355.000
37	42	25.000	1	25.000
42	51	359.500	1	359.500
51	66	73.500	1	73.500
66	68	356.500	1	356.500
68	77	167.000	1	167.000
77	79	29.000	1	29.000
79	107	59.500	1	59.500

Как видно из результатов измерения времени выполнения функциональных участков – наиболее затратным фрагментом является вызов функции log (строчки 13-22), также во время тестирования с другими данными было выявлено, что функции trunc, 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**
148	19	52.500	1	52.500
19	43	364.500	1	364.500
43	58	230.500	1	230.500
58	63	32.000	1	32.000
63	72	389.500	1	389.500
72	87	61.500	1	61.500
87	89	66.000	1	66.000
89	98	147.000	1	147.000
98	100	28.000	1	28.000
100	150	60.500	1	60.500

В результате внесённых изменений удалось добиться снижения времени выполнения на 12362 мкс (89,6%).

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){
      xx = 0.5 * x;
      x2 = xx * xx;
      t = log(xx) + euler;
      sum = 0.0;
      term = t;
      y0 = t;
      int j = 0;
           SAMPLE;
      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));
      SAMPLE;
      term = xx * (t-0.5);
      sum = 0.0;
      y1 = term;
      j = 1;
     SAMPLE;
      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));
      SAMPLE;
      y0 = pi2 * y0;
      y1 = pi2 * (y1 - 1/x);

      if(n == 0.0){
         return y0;

      }else if(n == 1.0){
```

```
        return y1;

      }else{
        ts = 2.0/x;
        ya = y0;
        yb = y1;
        int j=2;
       SAMPLE;
        if(j<=trunc(n+0.01)){
          SAMPLE;
          do{

            yc = ts*(j-1)*yb-ya;
            ya = yb;
            yb = yc;
            j+=1;

          }while(j<=trunc(n+0.01));
          SAMPLE;
        }
        SAMPLE;
        return yc;
      }
   }else{
         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"

#define DIG        18          //разрядность чисел
#define N_BITS  16          //число бит которое считаем
const unsigned ONE=1<<(DIG-1);      //единица
const unsigned TWO=ONE<<1;          //двойка
unsigned SCALE=1<<(N_BITS+1);   //масштаб логарифмов

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){
      xx = x/2;
      x2 = xx * xx;

      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;
      sum = 0.0;
      term = t;
      y0 = t;
      int j = 0;
           SAMPLE;
      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));
```

```
  SAMPLE;
  term = xx * (t-0.5);
  sum = 0.0;
  y1 = term;
  j = 1;
 SAMPLE;
  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));
  SAMPLE;
  y0 = pi2 * y0;
  y1 = pi2 * (y1 - 1/x);

  if(n == 0.0){
     return y0;

  }else if(n == 1.0){
     return y1;

  }else{
     ts = 2.0/x;
     ya = y0;
     yb = y1;
     int j=2;
    SAMPLE;
     if(j<=(int)(n+0.01)){
     SAMPLE;
        do{

           yc = ts*(j-1)*yb-ya;
           ya = yb;
           yb = yc;
           j+=1;

        }while(j<=(int)(n+0.01));
     SAMPLE;
     }
     SAMPLE;
     return yc;

  }
  SAMPLE;
  return ans;

}else{
       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;
       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;
        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;
}
```