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. 7304 _____ Абдульманов Э.М

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы:

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

Задание:

1. Ознакомиться с документацией на монитор SAMPLER и выполнить под его управлением тестовые программы test_cyc.c и test_sub.c с анализом параметров повторения циклов, структуры описания циклов, способов профилирования процедур и проверкой их влияния на точность и чувствительность профилирования.

2. Скомпилировать и выполнить под управлением SAMPLER'a программу на C, разработанную в 1-ой лабораторной работе.

Выполнить разбиение программы на функциональные участки и снять профили для двух режимов:

1 - измерение только полного времени выполнения программы;

2 - измерение времен выполнения функциональных участков (ФУ).

Убедиться, что сумма времен выполнения ФУ соответствует полному времени выполнения программы.

3. Выявить "узкие места", связанные с ухудшением производительности программы, ввести в программу усовершенствования и получить новые профили. Объяснить смысл введенных модификаций программ.

Ход работы

Для выполнения лабораторной работы был выбран монитор Sampler_old. Программы были скомпилированы Borland C++ 3.1 на DosBox, профилирование было проведено на DosBox.

1. Профилирование тестовых файлов

Исходный код файла TEST_CYC.CPP представлен в приложении А. Результаты профилирования TEST_CYC.CPP представлены на Рисунке 1.

Список обработанных файлов.

NN	Имя обработанного файла
1.	TEST_CYC.CPP

Таблица с результатами измерений (используется 13 из 416 записей)

Исх.Поз.	Прием.Поз.	Общее время(мкс)	Кол-во прох.	Среднее время(мкс)
1 : 9	1 : 11	4335.47	1	4335.47
1 : 11	1 : 13	8670.95	1	8670.95
1 : 13	1 : 15	21677.37	1	21677.37
1 : 15	1 : 17	43343.00	1	43343.00
1 : 17	1 : 20	4343.02	1	4343.02
1 : 20	1 : 23	8670.11	1	8670.11
1 : 23	1 : 26	21673.18	1	21673.18
1 : 26	1 : 29	43348.87	1	43348.87
1 : 29	1 : 35	4335.47	1	4335.47
1 : 35	1 : 41	8670.95	1	8670.95
1 : 41	1 : 47	21678.20	1	21678.20
1 : 47	1 : 53	43343.84	1	43343.84

Рисунок 1 – Результаты профилирования TEST_CYC.CPP

Исходный код файла TEST_SUB.CPP представлен в приложении Б. Результаты профилирования TEST_SUB.CPP представлены на Рисунке 2.

Список обработанных файлов.

NN	Имя обработанного файла
1.	TEST_SUB.CPP

Таблица с результатами измерений (используется 5 из 416 записей)

Исх.Поз.	Прием.Поз.	Общее время(мкс)	Кол-во прох.	Среднее время(мкс)
1 : 24	1 : 26	433697.35	1	433697.35
1 : 26	1 : 28	867392.18	1	867392.18
1 : 28	1 : 30	2168468.29	1	2168468.29
1 : 30	1 : 32	4336936.59	1	4336936.59

Рисунок 2 - Результаты профилирования TEST_SUB.CPP

По результатам профилирования тестовых программ можно сделать выводы о том, что время выполнения функции линейно зависит от количества итераций цикла и время выполнения цикла не зависит от способа записи этого цикла.

2. Профилирование файла из лабораторной работы

Исходный код файла SIMP1.CPP для измерения общего времени представлен в приложении В. Результаты профилирования SIMP1.CPP представлены на Рисунке 3.

```
   Список обработанных файлов.

-------------------------------------------------------------------------
  NN            Имя обработанного файла
-------------------------------------------------------------------------
  1.   SIMP1.CPP
-------------------------------------------------------------------------

  Таблица с результатами измерений ( используется 2 из 416 записей )

-------------------------------------------------------------------------
Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Среднее время(мкс)
-------------------------------------------------------------------------
 1 :  48  1 :  50          245765.94            1          245765.94
-------------------------------------------------------------------------
```

Рисунок 3 - Результаты профилирования SIMP1.CPP

Исходный код файла SIMP1_FU.CPP для измерения времени выполнения ФУ представлен в приложении Г. Результаты профилирования SIMP1_FU.CPP представлены на Рисунке 4.

```
   Список обработанных файлов.

-------------------------------------------------------------------------
  NN            Имя обработанного файла
-------------------------------------------------------------------------
  1.   SIMP1_FU.CPP
-------------------------------------------------------------------------

  Таблица с результатами измерений ( используется 4 из 416 записей )

-------------------------------------------------------------------------
Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Среднее время(мкс)
-------------------------------------------------------------------------
 1 :  27  1 :  34             243.89            1             243.89
-------------------------------------------------------------------------
 1 :  34  1 :  39          243127.61           11          22102.51
-------------------------------------------------------------------------
 1 :  39  1 :  34            1012.42           10             101.24
 1 :  39  1 :  42              43.58            1              43.58
-------------------------------------------------------------------------
```

Рисунок 4 – Результаты профилирования SIMP1_FU.CPP

По результатам профилирования видно, что время выполнения ФУ составляет небольшую разницу по сравнению с измерением общего времени: время выполнения ФУ – 244427.5, общее время выполнения – 245765.94, разница

– 1.338.44. Эту разницу можно объяснить двумя причинами, то что в измерении ФУ не участвует начальная инициализация вспомогательных переменных, при которых вызываются вспомогательные функции, и использованием эмулятора.

Так же можно заменить, что самое большое количество времени тратится при вызове функции fx в цикле for. Для того, чтобы уменьшить время выполнения можно перенести код из функции fx в цикл.

Исходный код файла SIMP1_UPDATE.CPP для измерения общего времени представлен в приложении Д. Результаты профилирования SIMP1_UPDATE.CPP представлены на Рисунке 5.

```
        Список обработанных файлов.

-------------------------------------------------------------------------
   NN            Имя обработанного файла
-------------------------------------------------------------------------
   1.   SIMP1_~1.CPP
-------------------------------------------------------------------------


   Таблица с результатами измерений ( используется 2 из 416 записей )

-------------------------------------------------------------------------
Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Среднее время(мкс)
-------------------------------------------------------------------------
 1 :  48  1 :  50          226685.83            1          226685.83
-------------------------------------------------------------------------
```

Рисунок 5 – Результаты профилирования SIMP1_UPDATE.CPP

По результатам профилирования можно заметить, что общее время выполнения программы сократилось с 245765.94 до 226685.83.

Исходный код файла SIMP1_UPDATE_FU.CPP для измерения времени выполнения ФУ представлен в приложении Е. Результаты профилирования SIMP1_UPDATE_FU.CPP представлены на Рисунке 6.

```
        Список обработанных файлов.

-------------------------------------------------------------------------
   NN            Имя обработанного файла
-------------------------------------------------------------------------
   1.   SIMP1_~2.CPP
-------------------------------------------------------------------------


   Таблица с результатами измерений ( используется 4 из 416 записей )

-------------------------------------------------------------------------
Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Среднее время(мкс)
-------------------------------------------------------------------------
 1 :  27  1 :  34             243.89            1             243.89
-------------------------------------------------------------------------
 1 :  34  1 :  39          224055.05           11           20368.64
-------------------------------------------------------------------------
 1 :  39  1 :  34            1014.93           10             101.49
 1 :  39  1 :  42              43.58            1              43.58
-------------------------------------------------------------------------
```

Рисунок 6 - Результаты профилирования SIMP1_UPDATE_FU.CPP

По результатам профилирования видно, что время выполнения ФУ составляет небольшую разницу по сравнению с измерением общего времени: время выполнения ФУ – 225357.45, общее время выполнения – 226685.83, разница – 1.328.38. Эту разницу можно объяснить двумя причинами, то что в измерении ФУ не участвует начальная инициализация вспомогательных переменных, при которых вызываются вспомогательные функции, и использованием эмулятора.

Выводы:

В результате выполнения данной лабораторной работы был изучен монитор SAMPLER, с помощью которого было выполнено профилирование тестовых программ TEST_CYC.CPP и TEST_SUB.CPP. А также было проанализировано время выполнения программы, разработанной в 1-ой лабораторной работе, и время выполнения ее ФУ.

ПРИЛОЖЕНИЕ А.

TEST_CYC.CPP

```
1 #INCLUDE <STDLIB.H>
2 #INCLUDE "SAMPLER.H"
3 #DEFINE SIZE 10000
4
5 INT I, TMP, DIM[SIZE];
6
7 VOID MAIN()
8 {
9     SAMPLE;
10    FOR(I=0;I<SIZE/10;I++){ TMP=DIM[0]; DIM[0]=DIM[I]; DIM[I]=TMP; };
11    SAMPLE;
12    FOR(I=0;I<SIZE/5;I++){ TMP=DIM[0]; DIM[0]=DIM[I]; DIM[I]=TMP; };
13    SAMPLE;
14    FOR(I=0;I<SIZE/2;I++){ TMP=DIM[0]; DIM[0]=DIM[I]; DIM[I]=TMP; };
15    SAMPLE;
16    FOR(I=0;I<SIZE;I++)  { TMP=DIM[0]; DIM[0]=DIM[I]; DIM[I]=TMP; };
17    SAMPLE;
18    FOR(I=0;I<SIZE/10;I++)
19    { TMP=DIM[0]; DIM[0]=DIM[I]; DIM[I]=TMP; };
20    SAMPLE;
21    FOR(I=0;I<SIZE/5;I++)
22    { TMP=DIM[0]; DIM[0]=DIM[I]; DIM[I]=TMP; };
23    SAMPLE;
24    FOR(I=0;I<SIZE/2;I++)
25    { TMP=DIM[0]; DIM[0]=DIM[I]; DIM[I]=TMP; };
26    SAMPLE;
27    FOR(I=0;I<SIZE;I++)
28    { TMP=DIM[0]; DIM[0]=DIM[I]; DIM[I]=TMP; };
29    SAMPLE;
30    FOR(I=0;I<SIZE/10;I++)
31    { TMP=DIM[0];
32     DIM[0]=DIM[I];
33     DIM[I]=TMP;
34    };
35    SAMPLE;
36    FOR(I=0;I<SIZE/5;I++)
37    { TMP=DIM[0];
38     DIM[0]=DIM[I];
39     DIM[I]=TMP;
```

```
40    };
41    SAMPLE;
42    FOR(I=0;I<SIZE/2;I++)
43    { TMP=DIM[0];
44     DIM[0]=DIM[I];
45     DIM[I]=TMP;
46    };
47    SAMPLE;
48    FOR(I=0;I<SIZE;I++)
49    { TMP=DIM[0];
50     DIM[0]=DIM[I];
51     DIM[I]=TMP;
52    };
53    SAMPLE;
54}
```

ПРИЛОЖЕНИЕ Б.

TEST_SUB.CPP

```
1 #INCLUDE <STDLIB.H>
2 #INCLUDE "SAMPLER.H"
3 CONST UNSIGNED SIZE = 1000;
4 VOID TESTLOOP(INT NTIMES)
5 {
6  STATIC INT TESTDIM[SIZE];
7  INT TMP;
8  INT ILOOP;
9  WHILE (NTIMES > 0)
10  {
11    NTIMES --;
12    ILOOP = SIZE;
13    WHILE (ILOOP > 0)
14    {
15      ILOOP -- ;
16      TMP = TESTDIM[0];
17      TESTDIM[0] = TESTDIM[NTIMES];
18      TESTDIM[NTIMES] = TMP;
19    }
20  }
21 } /* TESTLOOP */
22 VOID MAIN()
23 {
24    SAMPLE;
25    TESTLOOP(SIZE / 10);
26    SAMPLE;
27    TESTLOOP(SIZE / 5);
28    SAMPLE;
29    TESTLOOP(SIZE / 2);
30    SAMPLE;
31    TESTLOOP(SIZE / 1);
32    SAMPLE;
33 }
```

ПРИЛОЖЕНИЕ В.

SIMP1.CPP

```
1 #INCLUDE <STDIO.H>
2 #INCLUDE <STDLIB.H>
3 #INCLUDE "MATH.H"
4 #INCLUDE "SAMPLER.H"
5
6 CONST DOUBLE TOL= 1.0E-6;
7
8  DOUBLE FX(DOUBLE X){
9          RETURN EXP(-1.0*X/2.0);
10 }
11
12 DOUBLE DFX(DOUBLE X){
13         RETURN (-1.0*EXP(-1.0*X/2.0)/2.0);
14 }
15
16 DOUBLE SIMPS(DOUBLE LOWER,DOUBLE UPPER,DOUBLE TOL,DOUBLE SUM){
17    DOUBLE X,DELTA_X,EVEN_SUM,ODD_SUM,END_SUM,END_COR,SUM1;
18    INT PIECES;
19
20    PIECES=2;
21    DELTA_X=(UPPER-LOWER)/PIECES;
22    ODD_SUM=FX(LOWER+DELTA_X);
23    EVEN_SUM=0.0;
24    END_SUM=FX(LOWER)+FX(UPPER);
25    END_COR=DFX(LOWER)-DFX(UPPER);
26    SUM=(END_SUM+4.0*ODD_SUM)*DELTA_X/3.0;
27    DO{
28          PIECES=PIECES*2;
29          SUM1=SUM;
30          DELTA_X=(UPPER-LOWER)/PIECES;
31          EVEN_SUM=EVEN_SUM+ODD_SUM;
32          ODD_SUM=0.0;
33          FOR (INT I=1;I<=PIECES/2;I++){
34                X=LOWER+DELTA_X*(2.0*I-1.0);
35                ODD_SUM=ODD_SUM+FX(X);
36          }
37
      SUM=(7.0*END_SUM+14.0*EVEN_SUM+16.00*ODD_SUM+END_COR*DELTA_X)*DELTA_X/1
5.0;
```

```
38    } WHILE ((SUM!=SUM1) & (ABS(SUM-SUM1)<=ABS(TOL*SUM)));
39
40    RETURN SUM;
41 }
42
43 INT MAIN(VOID) {
44    DOUBLE SUM=0.0;
45    DOUBLE LOWER=1.0;
46    DOUBLE UPPER=9.0;
47    DOUBLE RES =0.0;
48    SAMPLE;
49    RES=SIMPS(LOWER,UPPER,TOL,SUM);
50    SAMPLE;
51    PRINTF("AREA= %LF ",RES);
52    RETURN 0;
53 }
```

ПРИЛОЖЕНИЕ Г

SIMP1_FU.CPP

```
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>
#INCLUDE "MATH.H"
#INCLUDE "SAMPLER.H"

CONST DOUBLE TOL= 1.0E-6;

DOUBLE FX(DOUBLE X){
    RETURN EXP(-1.0*X/2.0);
}

DOUBLE DFX(DOUBLE X){
   RETURN (-1.0*EXP(-1.0*X/2.0)/2.0);
}

DOUBLE SIMPS(DOUBLE LOWER,DOUBLE UPPER,DOUBLE TOL,DOUBLE SUM){
   DOUBLE X,DELTA_X,EVEN_SUM,ODD_SUM,END_SUM,END_COR,SUM1;
   INT PIECES;

   PIECES=2;
   DELTA_X=(UPPER-LOWER)/PIECES;
   ODD_SUM=FX(LOWER+DELTA_X);
   EVEN_SUM=0.0;
   END_SUM=FX(LOWER)+FX(UPPER);
   END_COR=DFX(LOWER)-DFX(UPPER);
   SUM=(END_SUM+4.0*ODD_SUM)*DELTA_X/3.0;
   SAMPLE;
   DO{
         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++){
               X=LOWER+DELTA_X*(2.0*I-1.0);
               ODD_SUM=ODD_SUM+FX(X);
         }
         SAMPLE;
```

```
40
      SUM=(7.0*END_SUM+14.0*EVEN_SUM+16.00*ODD_SUM+END_COR*DELTA_X)*DELTA_X/1
5.0;
41    } WHILE ((SUM!=SUM1) & (ABS(SUM-SUM1)<=ABS(TOL*SUM)));
42    SAMPLE;
43    RETURN SUM;
44 }
45
46 INT MAIN(VOID) {
47    DOUBLE SUM=0.0;
48    DOUBLE LOWER=1.0;
49    DOUBLE UPPER=9.0;
50    DOUBLE RES =0.0;
51    RES=SIMPS(LOWER,UPPER,TOL,SUM);
52    PRINTF("AREA= %LF ",RES);
53    RETURN 0;
54 }
```

ПРИЛОЖЕНИЕ Д

SIMP1_UPDATE.CPP

```
1 #INCLUDE <STDIO.H>
2 #INCLUDE <STDLIB.H>
3 #INCLUDE "MATH.H"
4 #INCLUDE "SAMPLER.H"
5
6 CONST DOUBLE TOL= 1.0E-6;
7
8 DOUBLE FX(DOUBLE X){
9     RETURN EXP(-1.0*X/2.0);
10 }
11
12 DOUBLE DFX(DOUBLE X){
13    RETURN (-1.0*EXP(-1.0*X/2.0)/2.0);
14 }
15
16 DOUBLE SIMPS(DOUBLE LOWER,DOUBLE UPPER,DOUBLE TOL,DOUBLE SUM){
17    DOUBLE X,DELTA_X,EVEN_SUM,ODD_SUM,END_SUM,END_COR,SUM1;
18    INT PIECES;
19
20    PIECES=2;
21    DELTA_X=(UPPER-LOWER)/PIECES;
22    ODD_SUM=FX(LOWER+DELTA_X);
23    EVEN_SUM=0.0;
24    END_SUM=FX(LOWER)+FX(UPPER);
25    END_COR=DFX(LOWER)-DFX(UPPER);
26    SUM=(END_SUM+4.0*ODD_SUM)*DELTA_X/3.0;
27    DO{
28          PIECES=PIECES*2;
29          SUM1=SUM;
30          DELTA_X=(UPPER-LOWER)/PIECES;
31          EVEN_SUM=EVEN_SUM+ODD_SUM;
32          ODD_SUM=0.0;
33          FOR (INT I=1;I<=PIECES/2;I++){
34                X=LOWER+DELTA_X*(2.0*I-1.0);
35                ODD_SUM=ODD_SUM+EXP(-1.0*X/2.0);
36          }
37
      SUM=(7.0*END_SUM+14.0*EVEN_SUM+16.00*ODD_SUM+END_COR*DELTA_X)*DELTA_X/1
5.0;
```

```
38    } WHILE ((SUM!=SUM1) & (ABS(SUM-SUM1)<=ABS(TOL*SUM)));
39
40    RETURN SUM;
41 }
42
43 INT MAIN(VOID) {
44    DOUBLE SUM=0.0;
45    DOUBLE LOWER=1.0;
46    DOUBLE UPPER=9.0;
47    DOUBLE RES =0.0;
48    SAMPLE;
49    RES=SIMPS(LOWER,UPPER,TOL,SUM);
50    SAMPLE;
51    PRINTF("AREA= %LF ",RES);
52    RETURN 0;
53 }
```

ПРИЛОЖЕНИЕ Е

SIMP1_UPDATE_FU.CPP

```
1 #INCLUDE <STDIO.H>
2 #INCLUDE <STDLIB.H>
3 #INCLUDE "MATH.H"
4 #INCLUDE "SAMPLER.H"
5
6 CONST DOUBLE TOL= 1.0E-6;
7
8 DOUBLE FX(DOUBLE X){
9     RETURN EXP(-1.0*X/2.0);
10 }
11
12 DOUBLE DFX(DOUBLE X){
13    RETURN (-1.0*EXP(-1.0*X/2.0)/2.0);
14 }
15
16 DOUBLE SIMPS(DOUBLE LOWER,DOUBLE UPPER,DOUBLE TOL,DOUBLE SUM){
17    DOUBLE X,DELTA_X,EVEN_SUM,ODD_SUM,END_SUM,END_COR,SUM1;
18    INT PIECES;
19
20    PIECES=2;
21    DELTA_X=(UPPER-LOWER)/PIECES;
22    ODD_SUM=FX(LOWER+DELTA_X);
23    EVEN_SUM=0.0;
24    END_SUM=FX(LOWER)+FX(UPPER);
25    END_COR=DFX(LOWER)-DFX(UPPER);
26    SUM=(END_SUM+4.0*ODD_SUM)*DELTA_X/3.0;
27    SAMPLE;
28    DO{
29          PIECES=PIECES*2;
30          SUM1=SUM;
31          DELTA_X=(UPPER-LOWER)/PIECES;
32          EVEN_SUM=EVEN_SUM+ODD_SUM;
33          ODD_SUM=0.0;
34          SAMPLE;
35          FOR (INT I=1;I<=PIECES/2;I++){
36                X=LOWER+DELTA_X*(2.0*I-1.0);
37                ODD_SUM=ODD_SUM+EXP(-1.0*X/2.0);
38          }
39          SAMPLE;
```

```
40
      SUM=(7.0*END_SUM+14.0*EVEN_SUM+16.00*ODD_SUM+END_COR*DELTA_X)*DELTA_X/1
5.0;
41    } WHILE ((SUM!=SUM1) & (ABS(SUM-SUM1)<=ABS(TOL*SUM)));
42    SAMPLE;
43    RETURN SUM;
44 }
45
46 INT MAIN(VOID) {
47    DOUBLE SUM=0.0;
48    DOUBLE LOWER=1.0;
49    DOUBLE UPPER=9.0;
50    DOUBLE RES =0.0;
51    RES=SIMPS(LOWER,UPPER,TOL,SUM);
52    PRINTF("AREA= %LF ",RES);
53    RETURN 0;
54 }
```