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 _____ Щука А. 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'а 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 на рисунках 1 и 2.

исх	прием	общее время	кол-во проходов	среднее время
13	15	2724.500	1	2724.500
15	17	5509.600	1	5509.600
17	19	21113.700	1	21113.700
19	21	29048.100	1	29048.100
21	24	2421.700	1	2421.700
24	27	6710.600	1	6710.600
27	30	21085.800	1	21085.800
30	33	27679.600	1	27679.600
33	39	2645.300	1	2645.300
39	45	6723.400	1	6723.400
45	51	16509.200	1	16509.200
51	57	35612.900	1	35612.900

Рисунок 1 – Результат работы SAMPLER для программы test_cyc.cpp

исх	прием	общее время	кол-во проходов	среднее время
30	32	38266799.700	1	38266799.700
32	34	78433060.500	1	78433060.500
34	36	202983276.400	1	202983276.400
36	38	352584030.100	1	352584030.100

Рисунок 2 - Результат работы SAMPLER для программы test_sub.cpp

Была выполнена под управлением монитора SAMPLER программа из лабораторной работы №1. Результат измерений для полного времени выполнения функции Sort представлен на рисунке 3. Исходный код этой программы представлен в Приложении А.

исх	прием	общее время	кол-во проходов	среднее время
47	49	26862.534	1	26862.534

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

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 на рисунке 4.

исх	прием	общее время	кол-во проходов	среднее время
61	15	16.721	1	16.721
15	17	12.842	1	12.842
17	20	21.084	1	21.084
20	23	49.789	7	7.113
23	26	4002.116	1203	3.327
26	34	8556.842	1135	7.539
26	29	10683.584	1097	9.739
34	36	10860.384	1135	9.569
36	38	1847.668	1135	1.628
38	40	509.137	106	4.803
38	26	3290.658	1029	3.198
40	23	3608.942	1196	3.018
40	42	63.174	7	9.025
29	40	1228.116	1097	1.120
42	20	40.068	6	6.678
42	44	8.605	1	8.605
44	63	63.500	1	63.500

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

В итоге общее время выполнения – 44863,23 мкс. Разницу в 18000,7 мкс с измерением для полного времени можно объяснить многократным вызовом функции измерения, а также случайной генерацией массива из 200 элементов для сортировки и, следовательно, разным ветвлением в функции сортировки.

Как видно из результатов измерения времени выполнения функциональных участков – наиболее затратным фрагментом является тело вложенного цикла for (строчки 23-40). Внутри него самыми время затратными операциями являются проверка условия (строчки 26-29) и вызов функции swap (строчки 34-36). Для

решения этой проблемы в 27 и 35 строчках операции взятия элемента массива по индексу были заменены адресной арифметикой. Также для уменьшения времени прохождения кода была создана переменная tmp, чтобы вместо четырёх операций сложения выполнялась только одна.

Была выполнена проверка изменённой программы. Результат представлен на рисунке 5. Исходный код модифицированной программы представлен в Приложении В.

исх	прием	общее время	кол-во проходов	среднее время
63	15	22.453	1	22.453
15	18	14.700	1	14.700
18	21	17.263	1	17.263
21	24	63.795	7	9.114
24	28	5617.379	1203	4.669
28	36	8517.837	1135	7.505
28	31	7994.279	1097	7.287
36	38	7349.321	1135	6.475
38	40	1187.000	1135	1.046
40	42	139.632	106	1.317
40	28	4180.079	1029	4.062
42	24	2895.800	1196	2.421
42	44	85.984	7	12.283
31	42	1657.921	1097	1.511
44	21	38.642	6	6.440
44	46	6.932	1	6.932
46	65	95.789	1	95.789

В результате внесённых изменений общее время выполнения – 39884,8 мкс. Удалось добиться снижения времени выполнения на 4978,4 мкс (11%).

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 узкого места и его устранения – в результате чего удалось добиться более быстрого выполнения программы.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "sampler.h"
4
5 void swap(float* x, float* y)
6 {
7    float temp;
8    temp = *x;
9    *x = *y;
10    *y = temp;
11 }
12
13 void shellsort(float arr[], int num)
14 {
15    int i, j, k;
16    for (i = num / 2; i > 0; i = i / 2)
17    {
18        for (j = i; j < num; j++)
19        {
20            for (k = j - i; k >= 0; k = k - i)
21            {
22                if (arr[k + i] >= arr[k])
23                    break;
24                else
25                {
26                    swap(&arr[k], &arr[k + i]);
27                }
28            }
29        }
30    }
31 }
32
33
34 int main(int argc, char **argv)
35 {
36    sampler_init(&argc, argv);
37    const int num = 200;
38    float my_max = 100.0;
39    float arr[num];
40    int k;
41
42    for (k = 0; k < num; k++)
43    {
44        arr[k] = (float)rand() / (float)(RAND_MAX / my_max);
45    }
46
47    SAMPLE;
48    shellsort(arr, num);
49    SAMPLE;
50    return 0;
51 }
```

ПРИЛОЖЕНИЕ Б.

КОД ПРОГРАММЫ С РАЗДЕЛЕНИЕМ НА ФУ

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "sampler.h"
4
5  void swap(float* x, float* y)
6  {
7      float temp;
8      temp = *x;
9      *x = *y;
10      *y = temp;
11  }
12
13  void shellsort(float arr[], int num)
14  {
15      SAMPLE;
16      int i, j, k;
17      SAMPLE;
18      for (i = num / 2; i > 0; i = i / 2)
19      {
20       SAMPLE;
21          for (j = i; j < num; j++)
22          {
23              SAMPLE;
24              for (k = j - i; k >= 0; k = k - i)
25              {
26                  SAMPLE;
27                  if (arr[k + i] >= arr[k])
28                  {
29                      SAMPLE;
30                      break;
31                  }
32                  else
33                  {
34                      SAMPLE;
35                      swap(&arr[k], &arr[k + i]);
36                      SAMPLE;
37                  }
38                  SAMPLE;
39              }
40              SAMPLE;
41          }
42          SAMPLE;
43      }
44      SAMPLE;
45  }
46
47
48  int main(int argc, char **argv)
```

```
49  {
50      sampler_init(&argc, argv);
51      const int num = 200;
52      float my_max = 100.0;
53      float arr[num];
54      int k;
55
56      for (k = 0; k < num; k++)
57      {
58          arr[k] = (float)rand() / (float)(RAND_MAX / my_max);
59      }
60
61      SAMPLE;
62      shellsort(arr, num);
63      SAMPLE;
64      return 0;
65  }
```

ПРИЛОЖЕНИЕ В.

КОД ПРОГРАММЫ С ОПТИМИЗАЦИЯМИ

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "sampler.h"
4
5  void swap(float* x, float* y)
6  {
7      float temp;
8      temp = *x;
9      *x = *y;
10      *y = temp;
11  }
12
13  void shellsort(float arr[], int num)
14  {
15      SAMPLE;
16      int i, j, k;
17      float* tmp;
18      SAMPLE;
19      for (i = num / 2; i > 0; i = i >> 1)
20      {
21       SAMPLE;
22          for (j = i; j < num; ++j)
23          {
24              SAMPLE;
25              for (k = j - i; k >= 0; k = k - i)
26              {
27                 tmp = arr + k;
28                 SAMPLE;
29                  if (*(tmp + i) >= *(tmp))
30                  {
31                      SAMPLE;
32                      break;
33                  }
34                  else
35                  {
36                      SAMPLE;
37                      swap(tmp, tmp + i);
38                      SAMPLE;
39                  }
40                  SAMPLE;
41              }
42              SAMPLE;
43          }
44          SAMPLE;
45      }
46      SAMPLE;
47  }
48
```

```
49
50  int main(int argc, char **argv)
51  {
52      sampler_init(&argc, argv);
53      const int num = 200;
54      float my_max = 100.0;
55      float arr[num];
56      int k;
57
58      for (k = 0; k < num; k++)
59      {
60          arr[k] = (float)rand() / (float)(RAND_MAX / my_max);
61      }
62
63      SAMPLE;
64      shellsort(arr, num);
65      SAMPLE;
66      return 0;
67  }
```