Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра вычислительных систем

Отчет по лабораторной работе

по дисциплине «Архитектура вычислительных систем»

Лабораторная работа №3

«Оценка производительности подсистемы памяти»

Выполнил: студент 3 курса
группы ИП-811
Мироненко К. А

Проверил: доцент кафедры ВС
Ефимов А. В.

Новосибирск 2020-21 уч. год

Оглавление

1. Постановка задачи

Тема: оценка производительности подсистемы памяти.

Задание: *разработать программу (benchmark) для оценки производительности подсистемы памяти.*

1. Написать программу(функцию)на языке C/C++/C#для оценки производительности подсистемы памяти.

   На вход программы подать следующие аргументы.

   1) Подсистема памяти. Предусмотреть возможность указать подсистему для проверки производительности: RAM (оперативная память), HDD/SSD и flash.
   2) Размер блока данных в байтах, Кб или Мб. Если размерность не указана, то в байтах, если указана, то соответственно в Кбайтах или Мбайтах.
   3) Число испытаний, т.е. число раз повторений измерений

   Пример вызова программы: ./memory_test –m RAM –b 1024|1Kb –l 10 или

   ./memory_bandwidth --memory-type RAM|HDD|SSD|flash

   --block-size 1024|1Kb

   --launch-count 10

   В качестве блока данных использовать одномерный массив, в котором произведение числа элементов на их размерность равна требуемому размеру блока данных. Массив инициализировать случайными значениями. Для тестирования HDD/SSD и flash создать в программе файлы в соответствующих директориях.

   Измерение времени реализовать с помощью функции clock_gettime() или аналогичной с точность до наносекунд. Измерять время исключительно на запись элемента в память или считывание из неё, без операций генерации или преобразования данных.

   На выходе программы в одну строку CSV файла со следующей структурой:

   [MemoryType;BlockSize;ElementType;BufferSize;LaunchNum;Timer;WriteTime;AverageWriteTime;WriteBandwidth;AbsError(write);RelError(write);ReadTime;AverageReadTime;ReadBandwidthAbsError(read);RelError(read);], где:

MemoryType –тип памяти (RAM|HDD|SSD|flash) или модель устройства, на котором проводятся испытания;

BlockSize – размер блока данных для записи и чтения на каждом испытании;

ElementType – тип элементов используемых для заполнения массива данных;

BufferSize – размер буфера, т.е. порции данных для выполнения одно операции записи или чтения;

LaunchNum – порядковый номер испытания;

Timer – название функции обращения к таймеру (для измерения времени);

WriteTime – время выполнения отдельного испытания с номером LaunchNum [секунды];

AverageWriteTime – среднее время записи из LaunchNum испытаний [секунды];

WriteBandwidth – пропускная способность памяти (BLOCK_SIZE/AverageWriteTime) * 106

[Mb/s]

AbsError(write) – абсолютная погрешность измерения времени записи или СКО [секунды];

RelError(write) – относительная погрешность измерения времени [%];

ReadTime – время выполнения отдельного испытания LaunchNum [секунды];

AverageReadTime – среднее время записи из LaunchNum испытаний [секунды];

ReadBandwidth – пропускная способность памяти (BLOCK_SIZE/AverageReadTime) * 106

[Мб/сек.]

AbsError(read) – абсолютная погрешность измерения времени чтения или СКО [секунды];

RelError(read) – относительная погрешность измерения времени [%].

2. Написать программу(функцию) на языке C/C++/C# или скрипт (benchmark) реализующий серию испытаний программы(функции) из п.1. Оценить

пропускную способность оперативной памяти при работе с блоками данных равными объёму кэш-линии, кэш-памяти L1, L2 и L3 уровня и превышающего его. Для HDD|SSD и flash провести серию из 20 испытаний с блоками данных начиная с 4 Мб с шагом 4Мб. Результаты всех испытаний сохранить в один CSV файл со структурой, описанной в п.1.
* Для HDD|SSD и flash оценить влияние размера буфера (BufferSize) на пропускную способность памяти.

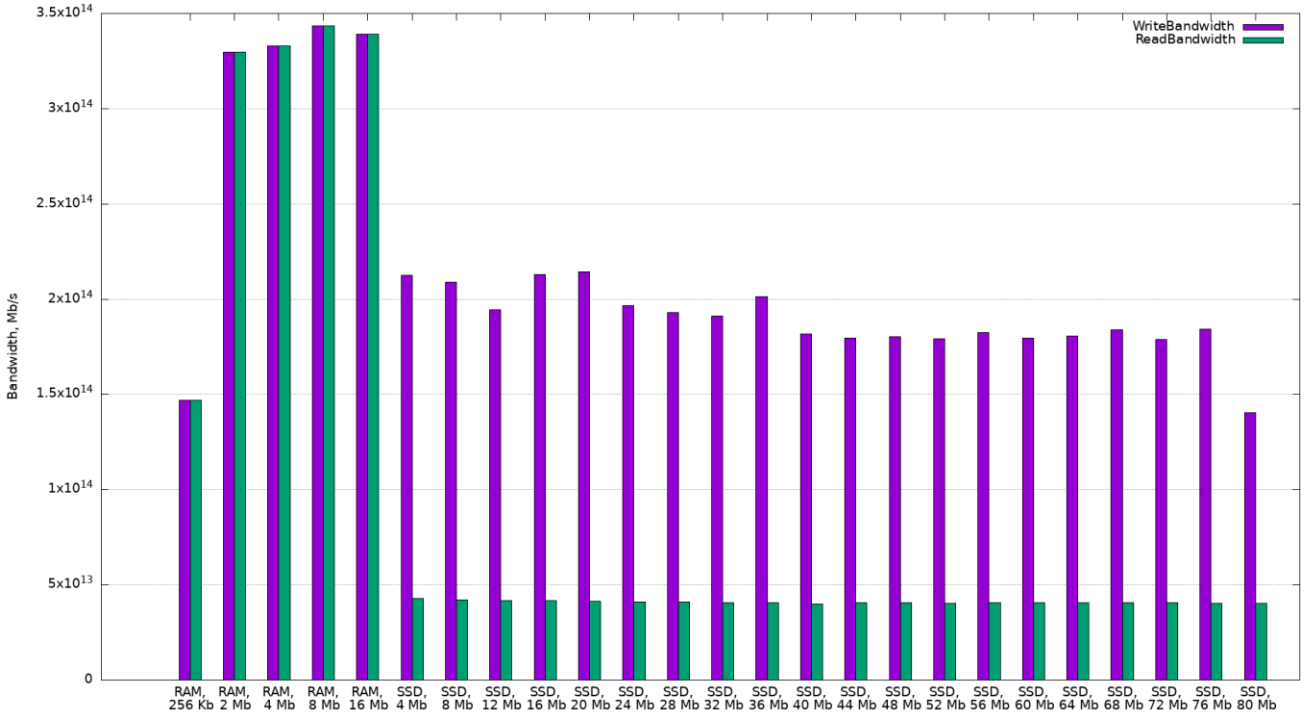
3. На основе CSV файла построить сводные таблицы и диаграммы отражающие:
    1) Зависимость пропускной способности записи и чтения от размера блока данных (BlockSize) для разного типа памяти;
    2) Зависимость погрешности измерения пропускной способности от размера блока данных для разного типа памяти;
    3) Зависимость погрешности измерений от числа испытаний LaunchNum;
    4) * Зависимость пропускной способности памяти от размера буфера для HDD|SSD и flash памяти
4. ** Оценить пропускную способность файла подкачки (windows) или раздела SWAP (linux). Сравнить с пропускной способностью RAM, HDD/SSD и flash.

2. Примеры работы программы

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	MemoryT	BlockSize	ElementT	BufferSize	LaunchNum	Timer	WriteTime	AverageWriteTime	WriteBandwidth	AbsError(write)	RelError(write)	ReadTime	AverageReadTime	ReadBandwidth	AbsError(read)	RelError(read)	
2	RAM	262144	uint8_t	1	1	clock()	2.261000e-03	1.784300e-03	1.469170e+14	4.767000e-04	2.671636e+01	2.261000e-03	1.784300e-03	1.469170e+14	4.767000e-04	2.671636e+01	
3	RAM	262144	uint8_t	1	2	clock()	2.428000e-03	1.784300e-03	1.469170e+14	6.437000e-04	3.607577e+01	2.428000e-03	1.784300e-03	1.469170e+14	6.437000e-04	3.607577e+01	
4	RAM	262144	uint8_t	1	3	clock()	2.461000e-03	1.784300e-03	1.469170e+14	6.767000e-04	3.792524e+01	2.461000e-03	1.784300e-03	1.469170e+14	6.767000e-04	3.792524e+01	
5	RAM	262144	uint8_t	1	4	clock()	9.530000e-04	1.784300e-03	1.469170e+14	8.313000e-04	4.658970e+01	9.530000e-04	1.784300e-03	1.469170e+14	8.313000e-04	4.658970e+01	
6	RAM	262144	uint8_t	1	5	clock()	3.313000e-03	1.784300e-03	1.469170e+14	1.528700e-03	8.567505e+01	3.313000e-03	1.784300e-03	1.469170e+14	1.528700e-03	8.567505e+01	
7	RAM	262144	uint8_t	1	6	clock()	2.856000e-03	1.784300e-03	1.469170e+14	1.071700e-03	6.006277e+01	2.856000e-03	1.784300e-03	1.469170e+14	1.071700e-03	6.006277e+01	
8	RAM	262144	uint8_t	1	7	clock()	1.562000e-03	1.784300e-03	1.469170e+14	2.223000e-04	1.245867e+01	1.562000e-03	1.784300e-03	1.469170e+14	2.223000e-04	1.245867e+01	
9	RAM	262144	uint8_t	1	8	clock()	6.690000e-04	1.784300e-03	1.469170e+14	1.115300e-03	6.250630e+01	6.690000e-04	1.784300e-03	1.469170e+14	1.115300e-03	6.250630e+01	
10	RAM	262144	uint8_t	1	9	clock()	6.670000e-04	1.784300e-03	1.469170e+14	1.117300e-03	6.261839e+01	6.670000e-04	1.784300e-03	1.469170e+14	1.117300e-03	6.261839e+01	
11	RAM	262144	uint8_t	1	10	clock()	6.730000e-04	1.784300e-03	1.469170e+14	1.111300e-03	6.228213e+01	6.730000e-04	1.784300e-03	1.469170e+14	1.111300e-03	6.228213e+01	
12	RAM	2097152	uint8_t	1	1	clock()	1.289200e-02	6.363200e-03	3.295751e+14	6.528800e-03	1.026025e+02	1.289200e-02	6.363200e-03	3.295751e+14	6.528800e-03	1.026025e+02	
13	RAM	2097152	uint8_t	1	2	clock()	5.587000e-03	6.363200e-03	3.295751e+14	7.762000e-04	1.219827e+01	5.587000e-03	6.363200e-03	3.295751e+14	7.762000e-04	1.219827e+01	
14	RAM	2097152	uint8_t	1	3	clock()	5.703000e-03	6.363200e-03	3.295751e+14	6.602000e-04	1.037528e+01	5.703000e-03	6.363200e-03	3.295751e+14	6.602000e-04	1.037528e+01	
15	RAM	2097152	uint8_t	1	4	clock()	5.588000e-03	6.363200e-03	3.295751e+14	7.752000e-04	1.218255e+01	5.588000e-03	6.363200e-03	3.295751e+14	7.752000e-04	1.218255e+01	
16	RAM	2097152	uint8_t	1	5	clock()	5.782000e-03	6.363200e-03	3.295751e+14	5.812000e-04	9.133769e+00	5.782000e-03	6.363200e-03	3.295751e+14	5.812000e-04	9.133769e+00	
17	RAM	2097152	uint8_t	1	6	clock()	5.602000e-03	6.363200e-03	3.295751e+14	7.612000e-04	1.196253e+01	5.602000e-03	6.363200e-03	3.295751e+14	7.612000e-04	1.196253e+01	
18	RAM	2097152	uint8_t	1	7	clock()	5.651000e-03	6.363200e-03	3.295751e+14	7.122000e-04	1.119248e+01	5.651000e-03	6.363200e-03	3.295751e+14	7.122000e-04	1.119248e+01	
19	RAM	2097152	uint8_t	1	8	clock()	5.580000e-03	6.363200e-03	3.295751e+14	7.832000e-04	1.230827e+01	5.580000e-03	6.363200e-03	3.295751e+14	7.832000e-04	1.230827e+01	
20	RAM	2097152	uint8_t	1	9	clock()	5.671000e-03	6.363200e-03	3.295751e+14	6.922000e-04	1.087817e+01	5.671000e-03	6.363200e-03	3.295751e+14	6.922000e-04	1.087817e+01	
21	RAM	2097152	uint8_t	1	10	clock()	5.576000e-03	6.363200e-03	3.295751e+14	7.872000e-04	1.237113e+01	5.576000e-03	6.363200e-03	3.295751e+14	7.872000e-04	1.237113e+01	
22	RAM	4194304	uint8_t	1	1	clock()	1.172100e-02	1.260270e-02	3.328100e+14	8.817000e-04	6.996120e+00	1.172100e-02	1.260270e-02	3.328100e+14	8.817000e-04	6.996120e+00	
23	RAM	4194304	uint8_t	1	2	clock()	1.282700e-02	1.260270e-02	3.328100e+14	2.243000e-04	1.779777e+00	1.282700e-02	1.260270e-02	3.328100e+14	2.243000e-04	1.779777e+00	
24	RAM	4194304	uint8_t	1	3	clock()	1.246600e-02	1.260270e-02	3.328100e+14	1.367000e-04	1.084688e+00	1.246600e-02	1.260270e-02	3.328100e+14	1.367000e-04	1.084688e+00	
25	RAM	4194304	uint8_t	1	4	clock()	1.332600e-02	1.260270e-02	3.328100e+14	7.233000e-04	5.739246e+00	1.332600e-02	1.260270e-02	3.328100e+14	7.233000e-04	5.739246e+00	
26	RAM	4194304	uint8_t	1	5	clock()	1.226200e-02	1.260270e-02	3.328100e+14	3.407000e-04	2.703389e+00	1.226200e-02	1.260270e-02	3.328100e+14	3.407000e-04	2.703389e+00	
27	RAM	4194304	uint8_t	1	6	clock()	1.236200e-02	1.260270e-02	3.328100e+14	2.407000e-04	1.909908e+00	1.236200e-02	1.260270e-02	3.328100e+14	2.407000e-04	1.909908e+00	
28	RAM	4194304	uint8_t	1	7	clock()	1.215600e-02	1.260270e-02	3.328100e+14	4.467000e-04	3.544479e+00	1.215600e-02	1.260270e-02	3.328100e+14	4.467000e-04	3.544479e+00	
29	RAM	4194304	uint8_t	1	8	clock()	1.265400e-02	1.260270e-02	3.328100e+14	5.130000e-05	4.070556e-01	1.265400e-02	1.260270e-02	3.328100e+14	5.130000e-05	4.070556e-01	
30	RAM	4194304	uint8_t	1	9	clock()	1.200300e-02	1.260270e-02	3.328100e+14	5.997000e-04	4.758504e+00	1.200300e-02	1.260270e-02	3.328100e+14	5.997000e-04	4.758504e+00	
31	RAM	4194304	uint8_t	1	10	clock()	1.425000e-02	1.260270e-02	3.328100e+14	1.647300e-03	1.307101e+01	1.425000e-02	1.260270e-02	3.328100e+14	1.647300e-03	1.307101e+01	
32	RAM	8388608	uint8_t	1	1	clock()	2.406700e-02	2.441500e-02	3.435842e+14	3.480000e-04	1.425353e+00	2.406700e-02	2.441500e-02	3.435842e+14	3.480000e-04	1.425353e+00	
33	RAM	8388608	uint8_t	1	2	clock()	2.350500e-02	2.441500e-02	3.435842e+14	9.100000e-04	3.727217e+00	2.350500e-02	2.441500e-02	3.435842e+14	9.100000e-04	3.727217e+00	
34	RAM	8388608	uint8_t	1	3	clock()	2.414300e-02	2.441500e-02	3.435842e+14	2.720000e-04	1.114069e+00	2.414300e-02	2.441500e-02	3.435842e+14	2.720000e-04	1.114069e+00	
35	RAM	8388608	uint8_t	1	4	clock()	2.394800e-02	2.441500e-02	3.435842e+14	4.670000e-04	1.912759e+00	2.394800e-02	2.441500e-02	3.435842e+14	4.670000e-04	1.912759e+00	
36	RAM	8388608	uint8_t	1	5	clock()	2.538500e-02	2.441500e-02	3.435842e+14	9.700000e-04	3.972967e+00	2.538500e-02	2.441500e-02	3.435842e+14	9.700000e-04	3.972967e+00	
37	RAM	8388608	uint8_t	1	6	clock()	2.422500e-02	2.441500e-02	3.435842e+14	1.900000e-04	7.782101e-01	2.422500e-02	2.441500e-02	3.435842e+14	1.900000e-04	7.782101e-01	

output

(Часть выходного csv файла)



(Зависимость пропускной способности записи и чтения от размера блока данных (BlockSize) для разного типа памяти)

Приложение Листинг

main.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>
#include <string.h>
#include <stdint.h>


int getParameters(int argc, char *argv[], char* memType, long long &blockSize, long long &launchCount)
{
   for (int i = 1 ; i < argc ; i++)
   {
      if (strcmp("-m", argv[i]) == 0 || strcmp("--memory-type", argv[i]) == 0)
      {
         i++;
         // if (strcmp("RAM", argv[i]) == 0 || strcmp("SSD", argv[i]) == 0 || strcmp("flash", argv[i]) == 0) == 0 ||
strcmp("HDD", argv[i]) == 0) == 0 )
         if (strcmp("RAM", argv[i]) == 0 || strcmp("SSD", argv[i]) == 0)
            strcpy(memType, argv[i]);
         else {
            printf("Error in arguments: invalid value for --memory-type! (possible RAM, SSD)\n");
            return 1;
         }
      }
      else if (strcmp("-b", argv[i]) == 0 || strcmp("--block-size", argv[i]) == 0)
      {
         i++;
         size_t len = strlen(argv[i]);
         int coeff = 1;

         if (argv[i][len-1] == 'b') {
            if (argv[i][len-2] == 'K')
               coeff = 1024;
            else if (argv[i][len-2] == 'M')
               coeff = 1024 * 1024;
            else {
               printf("Error in arguments: invalid value for --block-size!\nIncorrect unit of measurement (possible
Mb, Kb, or no unit of measurement if you need to set the size in bytes)\n");
               return 1;
            }
            argv[i][len - 2] = '\0';
            len -= 2;
         }
         for (size_t j = 0 ; j < len; j++)
            if (!isdigit(argv[i][j])){
               printf("Error in arguments: invalid value for --block-size!\nThe value must be a number!\n");
               return 1;
            }
         blockSize = atoll(argv[i]) * coeff;
      }
      else if (strcmp("-l", argv[i]) == 0 || strcmp("--launch-count", argv[i]) == 0)
      {
```

```
            i++;
            size_t len = strlen(argv[i]);
            for (size_t j = 0 ; j < len ; j++)
               if (!isdigit(argv[i][j])){
                  printf("Error in arguments: invalid value for --launch-count!\nThe value must be a number!\n");
                  return 1;
               }
            launchCount = atoll(argv[i]);
         }
         else {
            printf("Error in arguments: unknown key \"%s\"\n", argv[i]);
            return 1;
         }
      }
      return 0;
}

int testRAM(long long memSize, double &writeTime, double &readTime)
{
      uint8_t* origArr = (uint8_t*) malloc(memSize / sizeof(uint8_t));
      uint8_t* newArr = (uint8_t*) malloc(memSize / sizeof(uint8_t));
      clock_t start, stop;

      for (long long i = 0; i < memSize; i++)
         origArr[i] = rand() % 256;

      start = clock();
      for (long long i = 0; i < memSize; i++)
         newArr[i] = origArr[i];
      stop = clock();

      writeTime = ((double)(stop - start)) / CLOCKS_PER_SEC;
      readTime = writeTime;

      free(origArr);
      free(newArr);

      return 0;
}


int testStorageDevice(char* filepath, long long memSize, double &writeTime, double &readTime)
{
      uint8_t* arr = (uint8_t*) malloc(memSize / sizeof(uint8_t));
      FILE *fp;
      clock_t start, stop;

      for (long long i = 0; i < memSize; i++)
         arr[i] = rand() % 256;

      if ((fp = fopen(filepath, "w")) == NULL) {
         printf("Error: can't open file \"%s\"\n", filepath);
         return 1;
      }

      start = clock();
```

```
    for (long long i = 0; i < memSize; i++)
        fprintf(fp, "%c", arr[i]);
    stop = clock();
    writeTime = ((double)(stop - start)) / CLOCKS_PER_SEC;
    fclose(fp);

    if ((fp = fopen(filepath, "r")) == NULL) {
        printf("Error: can't open file \"%s\"\n", filepath);
        return 1;
    }

    start = clock();
    for (long long i = 0; i < memSize; i++)
        fscanf(fp, "%c", &arr[i]);
    stop = clock();
    readTime = ((double)(stop - start)) / CLOCKS_PER_SEC;
    fclose(fp);
    free(arr);
    if (-1 == remove(filepath))
        printf("Error: failed to delete file \"%s\"\n", filepath);
    return 0;
}


int outToCSV(char* memType, long long blockSize, long long launchCount, double* writeTime, double
averageWriteTime, double* readTime, double averageReadTime)
{
    FILE *fp;
    if (!(fp = fopen("output.csv", "a"))){
        printf("Error: can't open/find output.csv\n");
        return 1;
    }
    for (long long i = 0; i < launchCount; i++)
        fprintf(fp, "%s;%lld;%s;%lu;%lld;%s;%e;%e;%e;%e;%e;%e;%e;%e;%e;%e;\n",
        memType,
        blockSize,
        "uint8_t",
        sizeof(uint8_t),
        i + 1,
        "clock()",
        writeTime[i],
        averageWriteTime,
        blockSize / averageWriteTime * 1e6,
        abs(writeTime[i] - averageWriteTime),
        abs(writeTime[i] - averageWriteTime) / averageWriteTime * 100,
        readTime[i],
        averageReadTime,
        blockSize / averageReadTime * 1e6,
        abs(readTime[i] - averageReadTime),
        abs(readTime[i] - averageReadTime) / averageReadTime * 100);

    return 0;
}


int main(int argc, char *argv[]) {
```

```
    srand(time(0));

    char* memType = (char*) malloc(6);
    strcpy(memType, "RAM\0");
    long long blockSize = 1024;
    long long launchCount = 1;

    if (getParameters(argc, argv, memType, blockSize, launchCount))
        return 1;
    printf("arguments:\n");
    printf("memType = %s \n", memType);
    printf("blockSize = %lld bytes \n", blockSize);
    printf("launchCount = %lld \n", launchCount);


    double writeTimeSum = 0;
    double readTimeSum = 0;
    double writeTime[launchCount];
    double readTime[launchCount];

    for (long long i = 0; i < launchCount; i++) {

        if (strcmp("RAM", memType) == 0){
            if (testRAM(blockSize, writeTime[i], readTime[i]))
                return 1;
        }
        else {
            char filepath[1024];
            if (strcmp("SSD", memType) == 0)
                strcpy (filepath, "TestSSD.txt");
            if (testStorageDevice(filepath, blockSize, writeTime[i], readTime[i]))
                return 1;
        }
        writeTimeSum += writeTime[i];
        readTimeSum += readTime[i];
    }



    outToCSV(memType, blockSize, launchCount, writeTime, writeTimeSum / launchCount, readTime,
readTimeSum / launchCount);
    free(memType);
    return 0;
}
```