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.

Оглавление

[1. Постановка задачи 3](#_Toc62080435)

[2. Примеры работы программы 6](#_Toc62080436)

[*Приложение* Листинг 8](#_Toc62080437)

# Постановка задачи

***Тема:*** оценка производительности подсистемы памяти.

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

1. Написать программу(функцию)на языке С/С++/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 10В 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

[Mб/сек.]

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

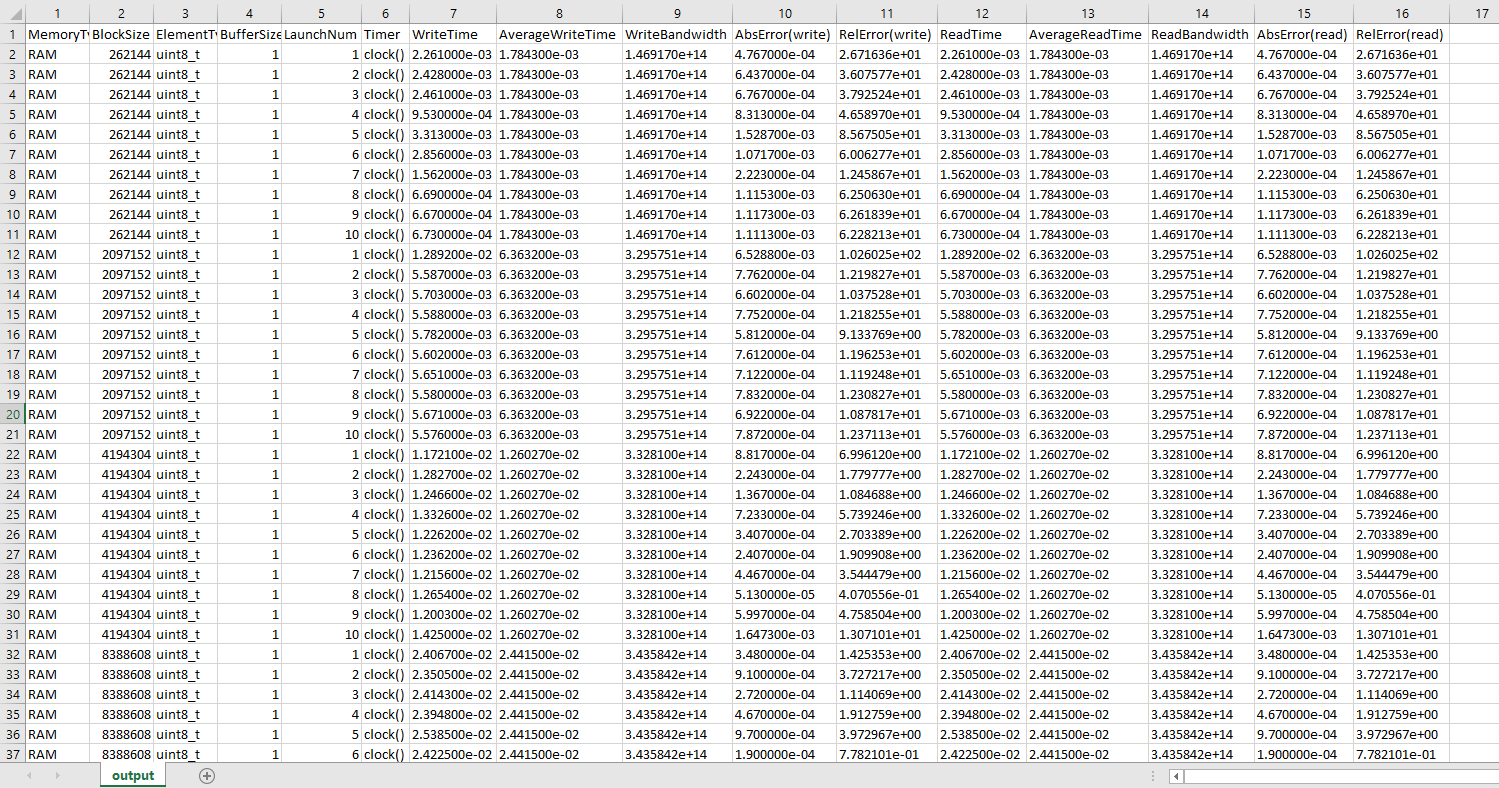
RelError(read) – относительная погрешность измерения времени [%].

1. Написать программу(функцию) на языке С/С++/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.

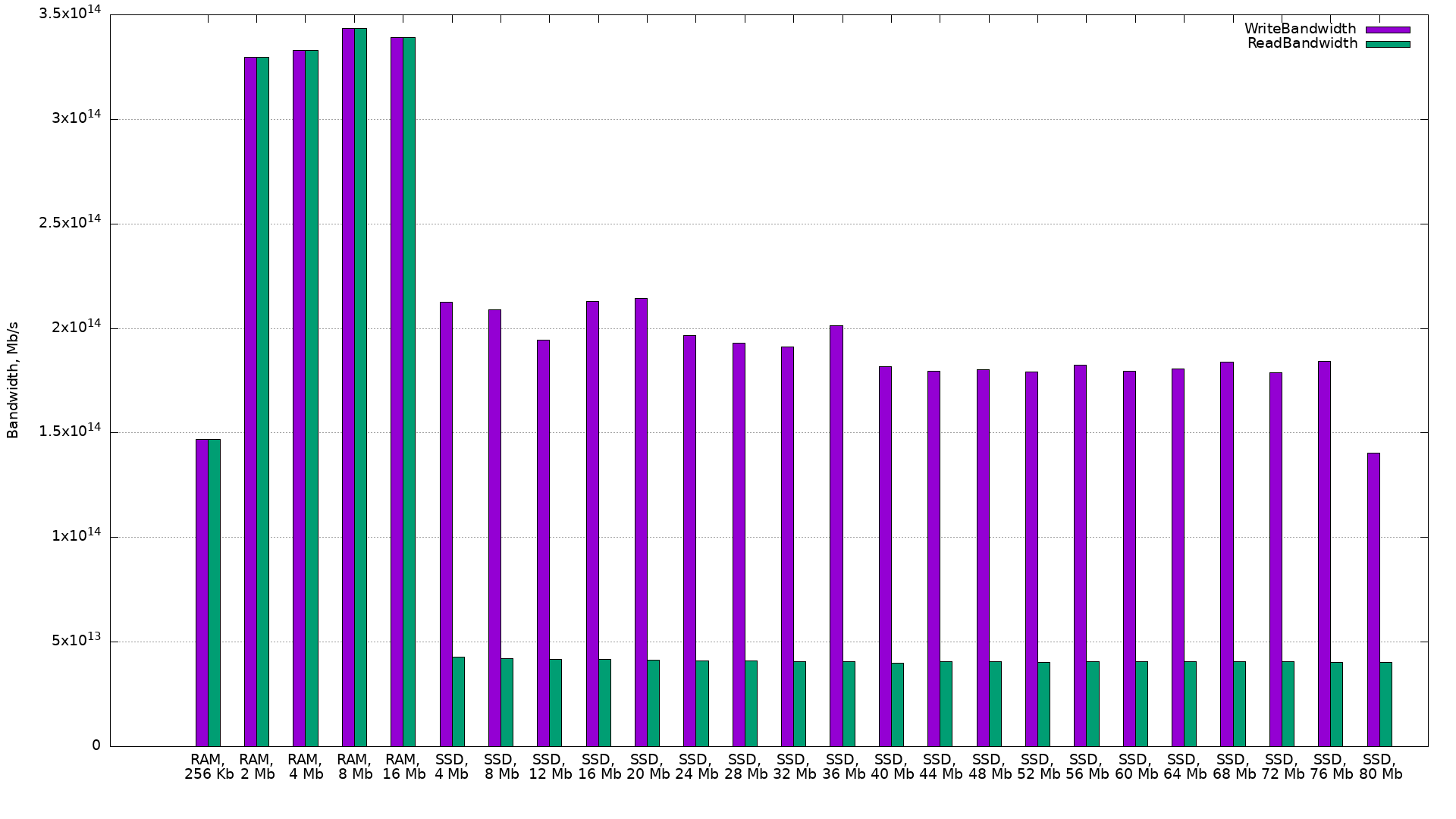
\* Для HDD|SSD и flash оценить влияние размера буфера (BufferSize) на пропускную способность памяти.

1. 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
2. \*\* Оценить пропускную способность файла подкачки (windows) или раздела SWAP (linux). Сравнить с пропускной способностью RAM, HDD/SSD и flash.

# Примеры работы программы



(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;

}