ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3  
«Оценка производительности подсистемы памяти»  
по дисциплине «Архитектура вычислительных систем»

Выполнил: студент гр. ИП-814  
Якунин А.В.

Проверил: ст. преп. Кафедры ВС   
Ткачева Т. А.

Новосибирск 2020

Содержание

[Постановка задачи 3](#_Toc50517127)

[Выполнение работы 6](#_Toc50517128)

[Результат работы 7](#_Toc50517129)

[Приложение 13](#_Toc50517130)

# Постановка задачи

*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*

*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) \* 10^6 [Mб/сек.] AbsError(read) – абсолютная погрешность измерения времени чтения или СКО [секунды] RelError(read) – относительная погрешность измерения времени [%].*

*2. 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.*

*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;*

# Выполнение работы

Инструменты, используемые в ходе работы: командная строка разработчика, Qt Creator

Размер кеша: L1 = 128 Kb, L2 = 512Kb, L3 = 3Mb.

Основные шаги выполнения работы:

1. Обработка аргументов, которые были заданы при запуске программы.
2. Перевод размера к байтам
3. Определение подсистемы памяти, которая будет оцениваться на производительность.
4. Реализация benchmark на языке С++.
5. Тестирование программы.
6. Построение сводной диаграммы зависимости пропускной способности записи и чтения от размера блока данных (BlockSize) для разного типа памяти.
7. Построение сводной диаграммы зависимости погрешности измерения пропускной способности от размера блока данных для разного типа памяти.

# Результат работы

Результат работы представлен на рисунках 1, 2, 3, 4.

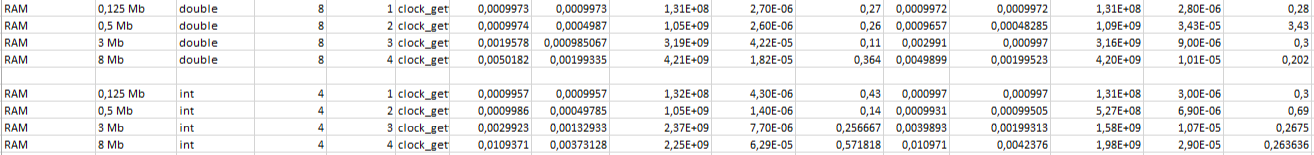


Рисунок 1. Вывод результатов испытаний RAM памяти в CSV файл (128 Kb, 512 Kb, 3 Mb, 8Mb).

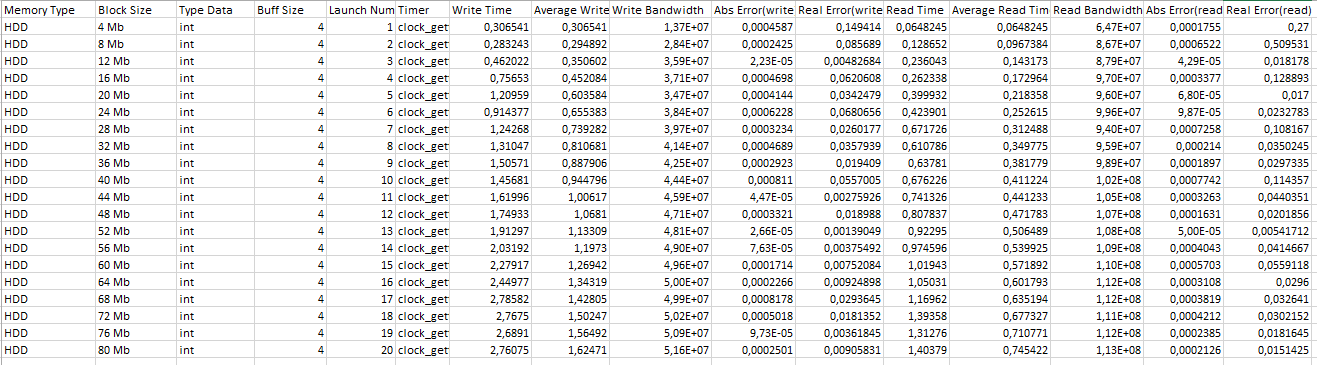
****

Рисунок 2. Вывод результатов испытаний HDD памяти в CSV файл.

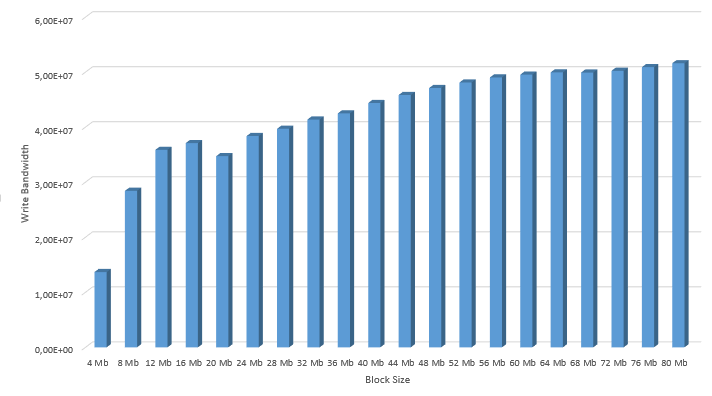


Рисунок 3. Диаграмма зависимости пропускной способности записи от размера блока данных

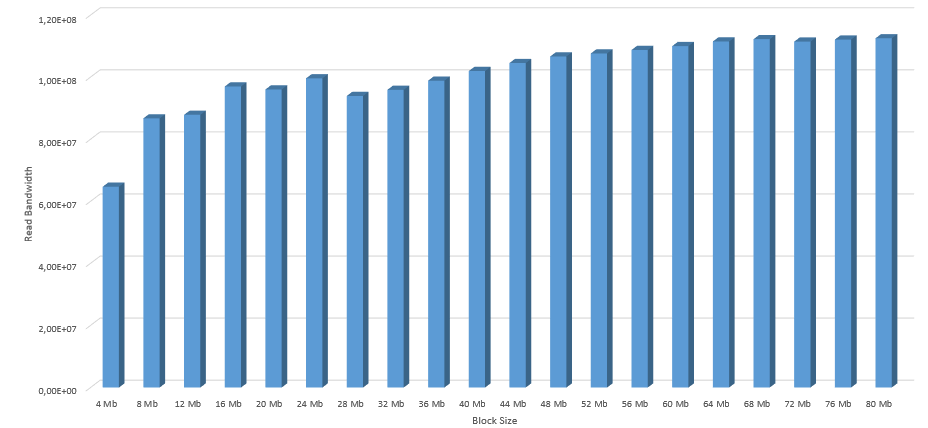


Рисунок 4. Диаграмма зависимости пропускной способности чтения от размера блока данных

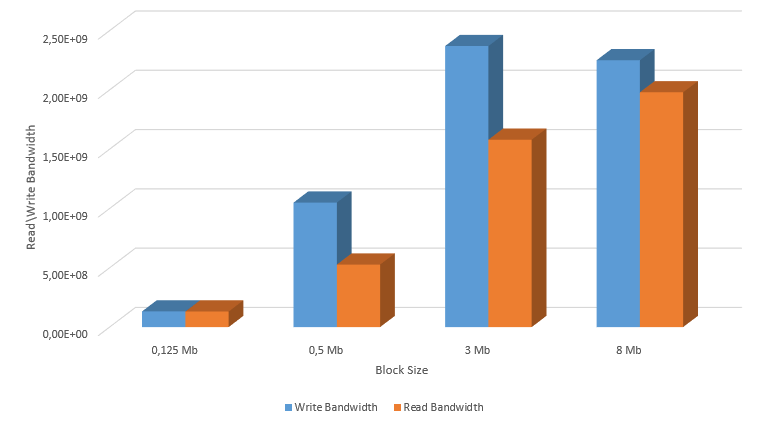


Рисунок 5. Диаграмма зависимости пропускной способности чтения и записи от размера блока данных для RAM

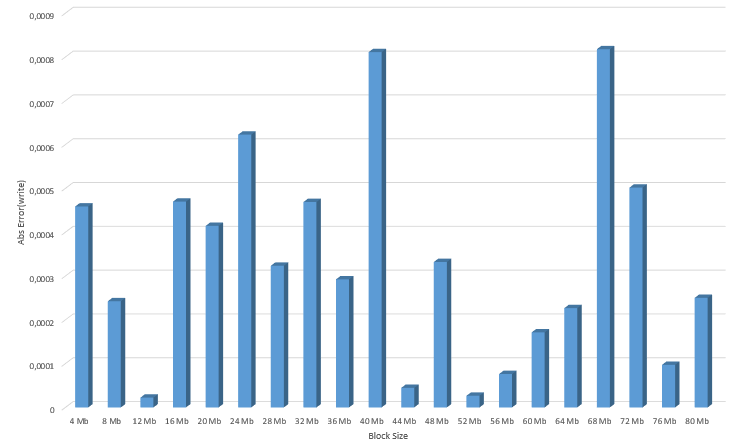


Рисунок 6. Диаграмма зависимости абсолютной погрешности записи от размера блока данных

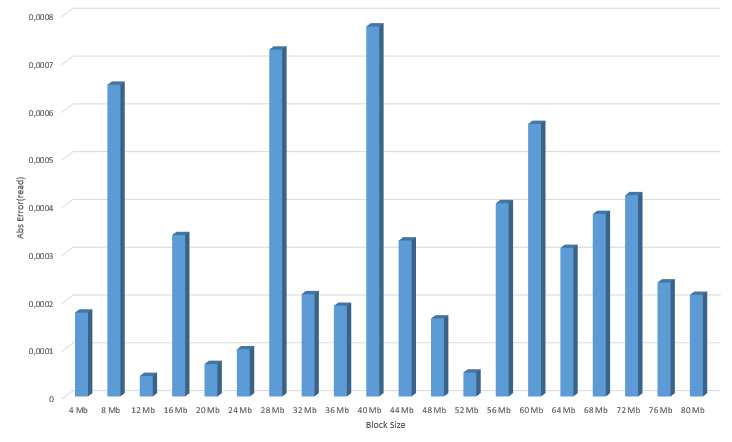


Рисунок 7. Диаграмма зависимости абсолютной погрешности чтения от размера блока данных

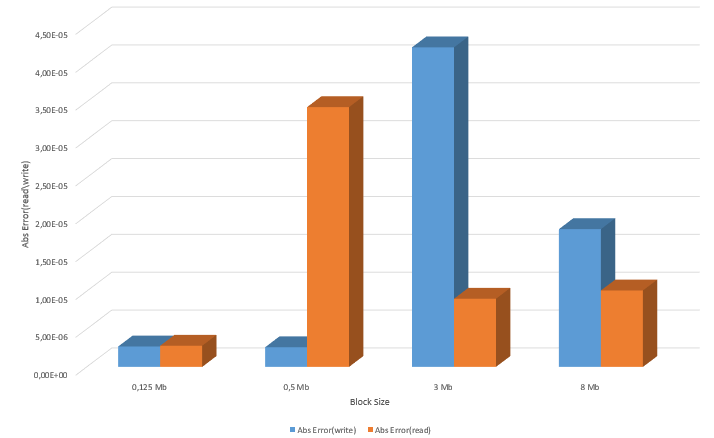


Рисунок 8 Диаграмма зависимости абсолютной погрешности чтения и записи от размера блока данных для RAM

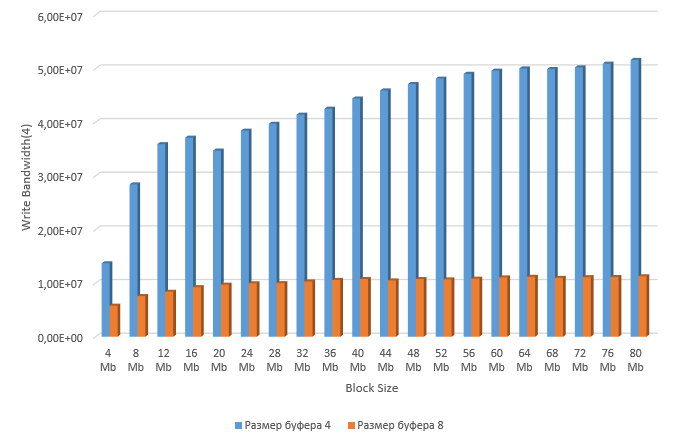


Рисунок 9. Диаграмма зависимости пропускной способности записи буфера для HDD

# Приложение

**mtest.h**

#ifndef MTEST\_H

#define MTEST\_H

#include <cmath>

#include <ctime>

#include <cstring>

#include <fstream>

#include <iomanip>

#include <iostream>

#include <windows.h>

using namespace std;

class **MTest**

{

public:

void **Gogogo**(int argc, char \*argv[]);

private:

string memType;

double blockSize;

string elementType;

int buffSize;

int launchNum;

string timer;

double writeTime;

double averageWriteTime;

double writeBandwidth;

double writeAbsError;

double writeRealError;

double readTime;

double averageReadTime;

double readBandwidth;

double readAbsError;

double readRealError;

timespec\* timeStart = new timespec;

timespec\* timeEnd = new timespec;

double\* timeWrite;

double\* timeRead;

double secondTimeStart = 0;

double\* timeCountWrite;

double\* timeCountRead;

int cash[4] = {

131072,

524288,

3145728,

0

};

string memory\_type;

int number\_of\_tests, customBlockSize;

int size\_of\_buffer = 8;

double absErrorWrite;

double realErrorWrite;

double absErrorRead;

double realErrorRead;

void **BenchTimer**(char type[], char task[], int number\_try);

void **bTimer**(char type[], char task[], int number\_try);

void **RAMTestWrite**(int size, int number\_try);

void **RAMTestRead**(int size, int number\_try);

void **ROMTestWrite**(int size, int number\_try);

void **ROMTestRead**(int size, int number\_try);

void **FlashTestWrite**(int size, int number\_try);

void **FlashTestRead**(int size, int number\_try);

double **AverageTimeCounter**(char type[], int size);

void **Accuracy**(int number\_try);

void **KeyReader**(int N, char \*key[]);

void **FileWriter**(int block, int num);

};

#endif // MTEST\_H

**mtest.cpp**

#include "mtest.h"

#define NANOSEC\_TO\_SEC 1000000000

void MTest::**Gogogo**(int argc, char \*argv[])

{

KeyReader(argc, argv);

if (memory\_type == "HDD" || memory\_type == "SSD")

{

int ROMBlockSize = 0;

int blockStep = 4 \* 1024 \* 1024;

timeWrite = new double[number\_of\_tests];

timeRead = new double[number\_of\_tests];

timeCountWrite = new double[number\_of\_tests];

timeCountRead = new double[number\_of\_tests];

for (int i = 0; i < number\_of\_tests; i++)

{

ROMBlockSize += blockStep;

ROMTestWrite(ROMBlockSize / size\_of\_buffer, i);

ROMTestRead(ROMBlockSize / size\_of\_buffer, i);

Accuracy(i);

// cout << i + 1 <<") Write = " << timeWrite[i] << " Read = " << timeRead[i] << endl;

FileWriter(ROMBlockSize, i + 1);

}

delete [] timeWrite;

delete [] timeRead;

delete [] timeCountWrite;

delete [] timeCountRead;

}

else if (memory\_type == "flash")

{

int flashBlockSize = 0;

int blockStep = 4 \* 1024 \* 1024;

timeWrite = new double[number\_of\_tests];

timeRead = new double[number\_of\_tests];

timeCountWrite = new double[number\_of\_tests];

timeCountRead = new double[number\_of\_tests];

for (int i = 0; i < number\_of\_tests; i++)

{

flashBlockSize += blockStep;

FlashTestWrite(flashBlockSize / size\_of\_buffer, i);

FlashTestRead(flashBlockSize / size\_of\_buffer, i);

Accuracy(i);

// cout << i + 1 <<") Write = " << timeWrite[i] << " Read = " << timeRead[i] << endl;

FileWriter(flashBlockSize, i + 1);

}

delete[] timeWrite;

delete[] timeRead;

delete[] timeCountWrite;

delete[] timeCountRead;

}

else if (memory\_type == "RAM")

{

number\_of\_tests = 4;

cash[3] = customBlockSize;

timeWrite = new double[number\_of\_tests];

timeRead = new double[number\_of\_tests];

timeCountWrite = new double[number\_of\_tests];

timeCountRead = new double[number\_of\_tests];

for (int i = 0; i < number\_of\_tests; i++)

{

RAMTestWrite(cash[i] / size\_of\_buffer, i);

RAMTestRead(cash[i] / size\_of\_buffer, i);

Accuracy(i);

// cout << i + 1 <<") Write = " << timeWrite[i] << " Read = " << timeRead[i] << endl;

FileWriter(cash[i], i + 1);

}

delete [] timeWrite;

delete [] timeRead;

delete [] timeCountWrite;

delete [] timeCountRead;

}

}

void MTest::**BenchTimer**(char type[], char task[], int number\_try)

{

if ((string)task == "START") clock\_gettime(CLOCK\_REALTIME, timeStart);

else if((string)task == "STOP")

{

clock\_gettime(CLOCK\_REALTIME, timeEnd);

double current = (double)((timeEnd->tv\_nsec - timeStart->tv\_nsec) + NANOSEC\_TO\_SEC \* (timeEnd->tv\_sec - timeStart->tv\_sec)) / NANOSEC\_TO\_SEC;

if ((string)type == "READ") timeRead[number\_try] = current;

else if ((string)type == "WRITE") timeWrite[number\_try] = current;

}

}

void MTest::**bTimer**(char type[], char task[], int number\_try)

{

if ((string)task == "START") secondTimeStart = clock();

else if((string)task == "STOP")

{

if ((string)type == "WRITE") timeCountWrite[number\_try] = ((double)(clock() - secondTimeStart)) / CLOCKS\_PER\_SEC;

else if ((string)type == "READ") timeCountRead[number\_try] = ((double)(clock() - secondTimeStart)) / CLOCKS\_PER\_SEC;

}

}

void MTest::**RAMTestWrite**(int size, int number\_try)

{

double\* staticMas = new double [size];

double\* dynamicMas = new double [size];

for (int i = 0; i < size; i++) staticMas[i] = rand()%100;

BenchTimer((char\*)"WRITE", (char\*)"START", number\_try);

bTimer((char\*)"WRITE", (char\*)"START", number\_try);

for (int i = 0; i < size; i++) dynamicMas[i] = staticMas[i];

BenchTimer((char\*)"WRITE", (char\*)"STOP", number\_try);

bTimer((char\*)"WRITE", (char\*)"STOP", number\_try);

delete [] staticMas;

delete [] dynamicMas;

}

void MTest::**RAMTestRead**(int size, int number\_try)

{

double\* staticMas = new double [size];

double\* dynamicMas = new double [size];

for (int i = 0; i < size; i++) staticMas[i] = rand()%100;

BenchTimer((char\*)"READ", (char\*)"START", number\_try);

bTimer((char\*)"READ", (char\*)"START", number\_try);

for (int i = 0; i < size; i++) staticMas[i] = dynamicMas[i];

BenchTimer((char\*)"READ", (char\*)"STOP", number\_try);

bTimer((char\*)"READ", (char\*)"STOP", number\_try);

delete [] staticMas;

delete [] dynamicMas;

}

void MTest::**ROMTestWrite**(int size, int number\_try)

{

double\* mas = new double[size];

for (int i = 0; i < size; i++) mas[i] = rand()%100;

BenchTimer((char\*)"WRITE", (char\*)"START", number\_try);

bTimer((char\*)"WRITE", (char\*)"START", number\_try);

ofstream fileROM("testROM.bin", ios::out);

for (int i = 0; i < size; i++) fileROM << mas[i];

fileROM.close();

BenchTimer((char\*)"WRITE", (char\*)"STOP", number\_try);

bTimer((char\*)"WRITE", (char\*)"STOP", number\_try);

delete [] mas;

}

void MTest::**ROMTestRead**(int size, int number\_try)

{

double\* mas = new double[size];

BenchTimer((char\*)"READ", (char\*)"START", number\_try);

bTimer((char\*)"READ", (char\*)"START", number\_try);

ifstream fileROM("testROM.bin", ios::in);

for (int i = 0; i < size; i++) fileROM >> mas[i];

fileROM.close();

BenchTimer((char\*)"READ", (char\*)"STOP", number\_try);

bTimer((char\*)"READ", (char\*)"STOP", number\_try);

delete [] mas;

}

void MTest::**FlashTestWrite**(int size, int number\_try)

{

int\* mas = new int[size];

for (int i = 0; i < size; i++) mas[i] = rand()%100;

BenchTimer((char\*)"WRITE", (char\*)"START", number\_try);

bTimer((char\*)"WRITE", (char\*)"START", number\_try);

ofstream fileROM("D:\testROM.bin", ios::out);

for (int i = 0; i < size; i++) fileROM << mas[i];

fileROM.close();

BenchTimer((char\*)"WRITE", (char\*)"STOP", number\_try);

bTimer((char\*)"WRITE", (char\*)"STOP", number\_try);

delete [] mas;

}

void MTest::**FlashTestRead**(int size, int number\_try)

{

int\* mas = new int[size];

BenchTimer((char\*)"READ", (char\*)"START", number\_try);

bTimer((char\*)"READ", (char\*)"START", number\_try);

ifstream fileROM("D:\testROM.bin", ios::in);

for (int i = 0; i < size; i++) fileROM >> mas[i];

fileROM.close();

BenchTimer((char\*)"READ", (char\*)"STOP", number\_try);

bTimer((char\*)"READ", (char\*)"STOP", number\_try);

delete [] mas;

}

double MTest::**AverageTimeCounter**(char type[], int size)

{

double average = 0;

if ((string)type == "WRITE")

{

for (int i = 0; i < size; i++) average += timeWrite[i];

average /= size;

}

else if ((string)type == "READ")

{

for (int i = 0; i < size; i++) average += timeRead[i];

average /= size;

}

return abs(average);

}

void MTest::**Accuracy**(int number\_try)

{

absErrorWrite = abs(timeWrite[number\_try] - timeCountWrite[number\_try]);

absErrorRead = abs(timeRead[number\_try] - timeCountRead[number\_try]);

realErrorWrite = absErrorWrite / timeCountWrite[number\_try] \* 100;

realErrorRead = absErrorRead / timeCountRead[number\_try] \* 100;

}

void MTest::**KeyReader**(int N, char \*key[])

{

string input\_block\_size, temp;

for(int i = 1; i < N; i++)

{

if((string)key[i] == "-m")

{

memory\_type = key[i + 1];

if( memory\_type != "RAM" and

memory\_type != "HDD" and

memory\_type != "SSD" and

memory\_type != "flash" )

{

cout << "\nError(input MEMORY\_TYPE)";

break;

}

}

else if((string)key[i] == "-b")

{

input\_block\_size = key[i + 1];

if( input\_block\_size[input\_block\_size.size() - 2] != 'K' and

input\_block\_size[input\_block\_size.size() - 2] != 'M')

customBlockSize = atoi(input\_block\_size.c\_str());

else

{

for (unsigned int j = 0; j < input\_block\_size.size() - 2; j++) temp[j] = input\_block\_size[j];

if(input\_block\_size[input\_block\_size.size() - 2] == 'K') customBlockSize = atoi(temp.c\_str()) \* 1024;

else customBlockSize = atoi(temp.c\_str()) \* 1024 \* 1024;

}

}

else if((string)key[i] == "-l") number\_of\_tests = atoi(key[i + 1]);

else;

}

}

void MTest::**FileWriter**(int block, int num)

{

memType = memory\_type;

elementType = "int";

blockSize = (double)block / (1024 \* 1024);

buffSize = size\_of\_buffer;

launchNum = num;

writeTime = timeWrite[num - 1];

readTime = timeRead[num - 1];

readAbsError = absErrorRead;

writeAbsError = absErrorWrite;

readRealError = realErrorRead;

writeRealError = realErrorWrite;

averageWriteTime = AverageTimeCounter((char\*)"WRITE", num);

averageReadTime = AverageTimeCounter((char\*)"READ", num);

writeBandwidth = (double)block / averageWriteTime;

readBandwidth = (double)block / averageReadTime;

timer = "clock\_gettime()";

ofstream file("text.csv", ios::app);

if(!file.is\_open()) cout << endl << "Can't open/find file template\_vmf\_end.txt" << endl;

else

{

file << memType << ";"

<< blockSize <<" Mb" << ";"

<< elementType << ";"

<< buffSize << ";"

<< launchNum << ";"

<< timer << ";"

<< writeTime << ";"

<< averageWriteTime << ";"

<< writeBandwidth << ";"

<< writeAbsError << ";"

<< writeRealError << ";"

<< readTime << ";"

<< averageReadTime << ";"

<< readBandwidth << ";"

<< readAbsError << ";"

<< readRealError << endl;

}

file.close();

}

**main.cpp**

#include "mtest.h"

int **main** (int argc, char \*argv[])

{

MTest first, second;

first.Gogogo(argc, argv);

return 0;

}