# ЭВМ и периферийные устройства

## Отчёт по лабораторной работе №6

## «Влияние КЭШ памяти на время обработки массивов»

## Выполнил: Скопинцев Н.А. ФИТ НГУ 2 курс

## Преподаватель:

## Цель работы:

1. Исследование зависимости времени доступа к данным в памяти от их объема.
2. Исследование зависимости времени доступа к данным в памяти от порядка их обхода.

## Листинг программы

#include <iostream>

#include <limits>

#include <climits>

#include <random>

#include <vector>

#include <algorithm>

#define K 10

**double** traversal(size\_t\* array, size\_t size) {

**union** ticks {

**unsigned** **long** **long** t64;

**struct** s32 {

**long** th, tl;

} t32;

} start, end;

//double cpu\_Hz = 2400000000ULL; // for 2.4 GHz CPU

size\_t k = 0;

size\_t i = 0;

**double** res = std::numeric\_limits<**double**>::max();

// прогрев

**for**(i = 0, k = 0; i < size; i++) {

k = array[k];

}

**for** (size\_t j = 0; j < K; j++) {

asm ("rdtsc\n":"=a"(start.t32.th),"=d"(start.t32.tl));

**for** (i = 0, k = 0; i < size; i++) {

k = array[k];

}

asm ("rdtsc\n":"=a"(end.t32.th),"=d"(end.t32.tl));

res = std::min(res, (**double**)(end.t64-start.t64));

}

**return** res / size;

}

**void** count(size\_t size) {

size\_t \*S, \*B, \*R;

S = **new** size\_t[size]; // Прямой обход

B = **new** size\_t[size]; // Обратный обход

R = **new** size\_t[size]; // Рандомный обход

//// �нициализация массивов///

**for** (size\_t i = 0; i < size - 1; i++) {

S[i] = i + 1;

}

S[size-1] = 0;

///////////////////////////////////////

std::cout << traversal(S, size) << "\t";

**for** (size\_t i = size - 1; i > 0; i--) {

B[i] = i - 1;

}

B[0] = size - 1;

//////////////////////////////

std::cout << traversal(B, size) << "\t";

std::vector<size\_t> v;

**for** (size\_t i = 1; i < size; i++) {

v.push\_back(i);

}

std::random\_device rd;

std::mt19937 g(rd());

std::shuffle(v.begin(), v.end(), g);

size\_t k = 0;

**for** (size\_t i = 1; i < size; i++) {

R[k] = v.back();

v.erase(v.end() - 1);

k = R[k];

}

R[k] = 0;

////////////////////////////

std::cout << traversal(R, size) << std::endl;

**delete** [] S;

**delete** [] R;

**delete** [] B;

}

**int** **main**() {

std::vector<size\_t> N {256, 512, 1024, 8388608, 24576000};

std::cout << std::fixed;

std::cout.precision(7);

size\_t tmp = 0;

std::cout << "\tПрямой обход\tОбратный обход\tСлучайный обход" << std::endl;

**for** (size\_t i = 256; i <= N.at(3); i \*= 2) {

tmp = i \* 4 / 1024;

**if** (tmp < 1024) {

std::cout << tmp << "KB\t";

}

**else** {

std::cout << tmp / 1024 << "MB\t";

}

count(i);

}

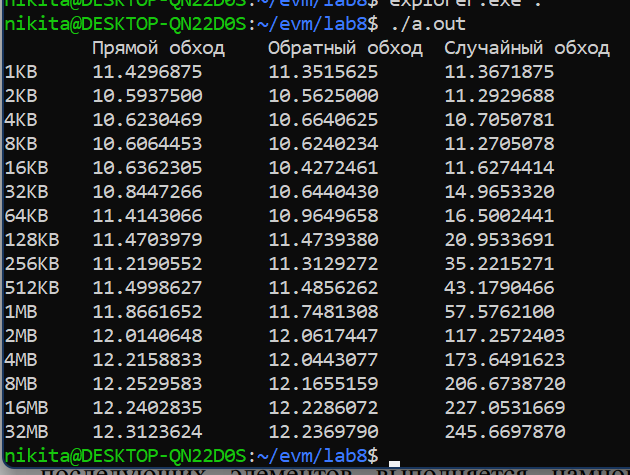
**return** 0;

}

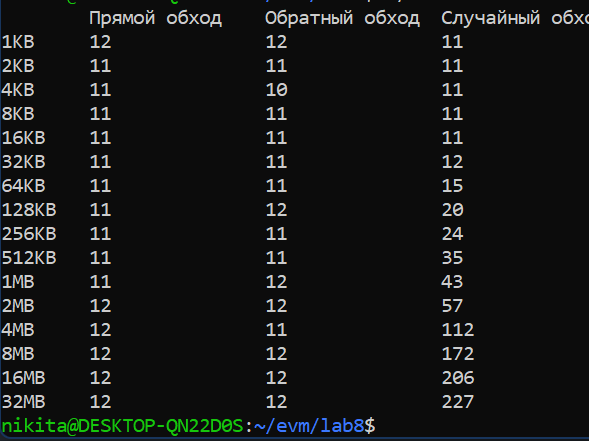
**Ключи компиляции:**

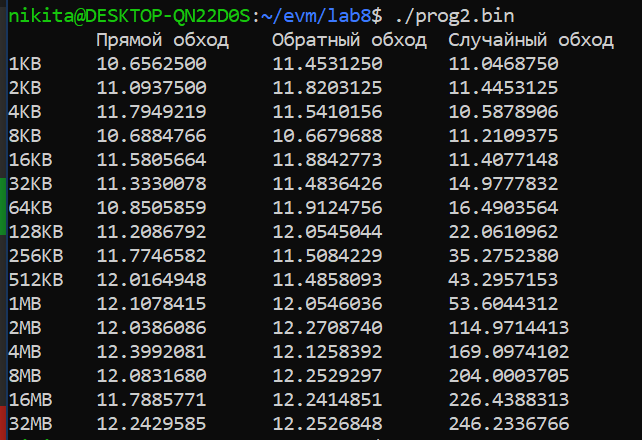
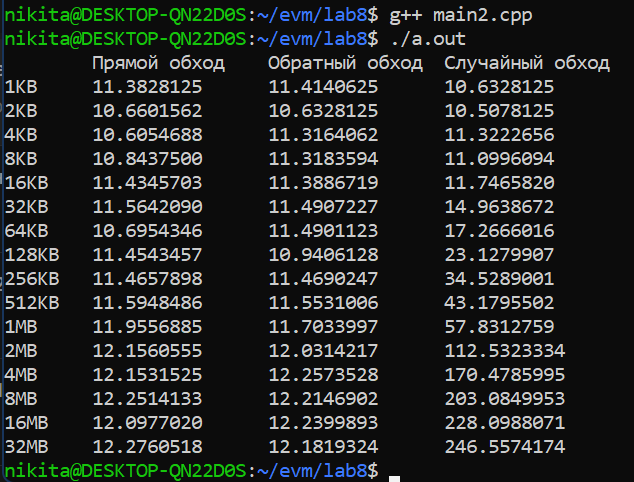
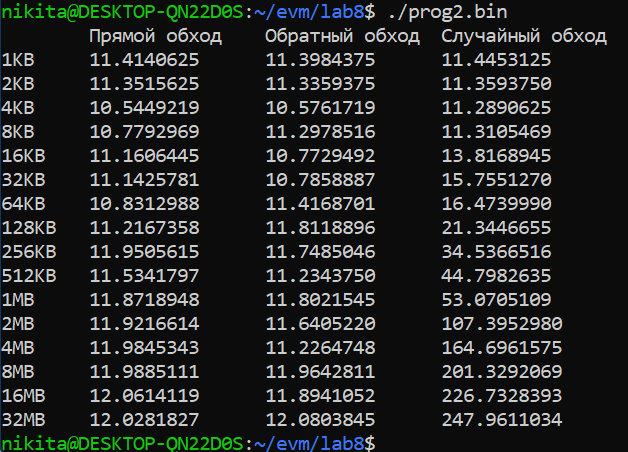
g++ main.cpp

./a.out



С volatile **(это правильная таблица)**





График

|  |  |  |  |
| --- | --- | --- | --- |
|  | Прямой обход | Обратный обход | Случайный обход |
| 1KB | 11.3671875 | 11.3906250 | 10.7265625 |
| 2KB | 11.2929688 | 10.5820312 | 10.6132812 |
| 4KB | 10.5605469 | 10.5449219 | 10.2148438 |
| 8KB | 10.5263672 | 11.2958984 | 11.3017578 |
| 16KB | 10.6044922 | 11.4506836 | 11.7163086 |
| 32KB | 10.6003418 | 10.5839844 | 12.1584473 |
| 64KB | 10.4506836 | 10.8621826 | 16.4475098 |
| 128KB | 11.4009399 | 11.0307617 | 17.6689453 |
| 256KB | 11.9041748 | 11.4349060 | 20.2542419 |
| 512KB | 11.2570343 | 11.4721985 | 35.4886627 |
| 1024KB | 11.6099396 | 11.4952850 | 43.0450897 |
| 2048KB | 12.0124550 | 11.8633652 | 57.5335884 |
| 4096KB | 12.0786400 | 11.9887466 | 112.0523586 |
| 8192KB | 12.0701180 | 12.0657406 | 172.0206184 |
| 16384KB | 12.0128775 | 12.1006446 | 206.4906855 |
| 32768KB | 12.1083001 | 12.1749483 | 227.8554127 |

## Вывод

В процессе выполнения работы была исследована зависимость времени доступа к данным в памяти от их объёма и порядка их обхода.

Был доказан очевидный факт, что последовательный обход данных наиболее аппаратно предсказуем, и потому время доступа при таком обходе наименьшее.

Было подтверждено, что при малых размерах массива время доступа при использовании различных способов обхода приблизительно равно, поскольку весь массив помещается в кэш-память.

Нельзя не отметить факт роста среднего времени доступа к элементам при их случайном обходе.