**Системное программное обеспечение**

**Лабораторная работа №2**

Съестов Дмитрий Вячеславович

P3217

#include "stdafx.h"

#include "header.h"

#define ARRAY\_LENGTH 10000

#define ARRAY\_SIZE   ARRAY\_LENGTH \* sizeof(int)

#define BINARY\_FILENAME L"..\\array.bin"

#define INPUT\_FILENAME L"..\\array.txt"

#define OUTPUT\_FILENAME L"..\\sorted.txt"

int\* loadArray(const wchar\_t\* filename)

{

    int\* arr = new int[ARRAY\_LENGTH];

    ifstream input(filename, ios\_base::in);

    for (int i = 0; i < ARRAY\_LENGTH; i++) input >> arr[i];

    input.close();

    return arr;

}

void binaryToText(const wchar\_t\* binaryFilename, const wchar\_t\* textFilename)

{

    HANDLE hFile = CreateFile(binaryFilename, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

    int\* arr = new int[ARRAY\_LENGTH];

    ReadFile(hFile, (char\*) arr, ARRAY\_SIZE, NULL, NULL);

    CloseHandle(hFile);

    ofstream output(textFilename, ios\_base::out);

    for (int i = 0; i < ARRAY\_LENGTH; i++) output << arr[i] << endl;

    output.close();

}

void serializeArray(const int\* arr, const wchar\_t\* filename)

{

    HANDLE hFile = CreateFile(filename, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

    WriteFile(hFile, (char\*) arr, ARRAY\_SIZE, NULL, NULL);

    CloseHandle(hFile);

}

//Сортировка Шелла (по возрастанию)

void sort(int\* arr, int n)

{

for (int gap = n/2; gap > 0; gap /= 2)

{

for (int i = gap; i < n; i += 1)

{

int j, temp = arr[i];

for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)

arr[j] = arr[j - gap];

arr[j] = temp;

}

}

}

time\_t elapsedMcs(time\_point<high\_resolution\_clock> startTime)

{

    auto duration = duration\_cast<microseconds>(high\_resolution\_clock::now() - startTime);

return duration.count();

}

time\_t sortHeap(const int\* arr)

{

    auto startTime = high\_resolution\_clock::now();

    //===============================

    HANDLE hHeap = GetProcessHeap();

    int\* data = (int\*) HeapAlloc(hHeap, 0, ARRAY\_SIZE);

    memcpy(data, arr, ARRAY\_SIZE);

    sort(data, ARRAY\_LENGTH);

    serializeArray(data, BINARY\_FILENAME);

    //===============================

    HeapFree(hHeap, 0, data);

    return elapsedMcs(startTime);

}

time\_t sortBasePtr(const int\* arr)

{

    serializeArray(arr, BINARY\_FILENAME);

    auto startTime = high\_resolution\_clock::now();

    //===============================

    HANDLE hFile = CreateFile(BINARY\_FILENAME, GENERIC\_READ | GENERIC\_WRITE, 0, nullptr, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, nullptr);

    DWORD dwFileSize = GetFileSize(hFile, nullptr);

    HANDLE hMapping = CreateFileMapping(hFile, nullptr, PAGE\_READWRITE, 0, ARRAY\_SIZE, nullptr);

    int\* mapData = (int\*) MapViewOfFile(hMapping, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, dwFileSize);

    int\* data = new int[ARRAY\_LENGTH];

    for (int i = 0; i <= dwFileSize / sizeof(int); i++) data[i] = \*(mapData + i);

    sort(data, dwFileSize / sizeof(int));

    for (int i = 0; i <= dwFileSize / sizeof(int); ++i) mapData[i] = \*(data + i);

    //===============================

    UnmapViewOfFile(mapData);

    CloseHandle(hMapping);

    CloseHandle(hFile);

return elapsedMcs(startTime);

}

time\_t sortMapping(const int\* arr)

{

    serializeArray(arr, BINARY\_FILENAME);

    auto startTime = high\_resolution\_clock::now();

    //===============================

    HANDLE hFile = CreateFile(BINARY\_FILENAME, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,

        NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

    DWORD dwFileSize = GetFileSize(hFile, NULL);

    HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, 0, ARRAY\_SIZE, NULL);

    int\* data = (int\*) MapViewOfFile(hMapping, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, dwFileSize);

    sort(data, dwFileSize / sizeof(int));

    //===============================

    UnmapViewOfFile(data);

    CloseHandle(hFile);

    CloseHandle(hMapping);

    return elapsedMcs(startTime);

}

int main()

{

cout << "+--------+----------------+----------------+----------------+\n"

<< "| LENGTH | HEAP | BASE POINTER | FILE MAPPING |\n"

<< "+--------+----------------+----------------+----------------+\n";

    const int\* arr = loadArray(INPUT\_FILENAME);

    time\_t time1 = sortHeap(arr);

    DeleteFile(BINARY\_FILENAME);

    time\_t time2 = sortBasePtr(arr);

    DeleteFile(BINARY\_FILENAME);

    time\_t time3 = sortMapping(arr);

    binaryToText(BINARY\_FILENAME, OUTPUT\_FILENAME);

    DeleteFile(BINARY\_FILENAME);

    delete(arr);

    printf("|%8u|%12lli mcs|%12lli mcs|%12lli mcs|\n", ARRAY\_LENGTH, time1, time2, time3);

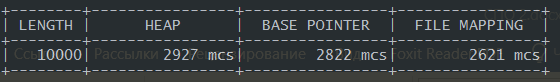
cout << "+--------+----------------+----------------+----------------+\n\n";

    system("pause");

return EXIT\_SUCCESS;

}

**Сравнение способов**



Преимущества каждого способа:

* Куча – самый простой для реализации способ.
* Базовые указатели немного более эффективны при работе с большими объёмами данных.
* Отображение файлов чаще всего даёт выигрыш в производительности по сравнению с обычным вводом-выводом. Недостаток отображения в том, что его размер зависит от архитектуры (на 32-битной нельзя создавать отображения больше 4 ГБ)