1. МИНОБРНАУКИ РОССИИ
2. САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
3. ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
4. «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
5. Кафедра Вычислительной техники

ОТЧЁТ

по лабораторной работе №2

1. по дисциплине «Операционные системы»
2. Тема: Управление памятью

|  |  |  |
| --- | --- | --- |
| Студент гр. 9308 |  | Соболев М.С. |
| Преподаватель |  | Тимофеев А.В. |

Оглавление

[1. Введение 3](#_Toc86796440)

[2. Исследование виртуального адресного пространства процесса 5](#_Toc86796441)

[2.1. Получение информации о вычислительной системе 5](#_Toc86796442)

[2.2. Определение статуса виртуальной памяти 7](#_Toc86796443)

[2.3. Определение состояния конкретного участка памяти по заданному с клавиатуры адресу 9](#_Toc86796444)

[2.4. Резервирование региона (и передача ему физической памяти) в автоматическом режиме и в режиме ввода адреса начала региона 10](#_Toc86796445)

[2.5. Вывод списка зарезервированных пользователем адресов 15](#_Toc86796446)

[2.6. Запись данных в ячейки памяти по заданным с клавиатуры адресам 16](#_Toc86796447)

[2.7. Установка защиты доступа для заданного (с клавиатуры) региона памяти и её проверку 18](#_Toc86796448)

[2.8. Возврат физической памяти и освобождение региона адресного пространства заданного (с клавиатуры) региона памяти 20](#_Toc86796449)

[2.9. Исходный код программы 23](#_Toc86796450)

[2.10. Выводы 100](#_Toc86796451)

[3. Использование проецируемых файлов для обмена данными между процессами 101](#_Toc86796452)

[3.1. Создание проецируемого файла приложением-писателем 101](#_Toc86796453)

[3.2. Открытие проецируемого файла приложением-читателем 103](#_Toc86796454)

[3.3. Исходный код программы-писателя 105](#_Toc86796455)

[3.4. Исходный код программы-читателя 110](#_Toc86796456)

[3.5. Выводы 114](#_Toc86796457)

[4. Список использованных источников 115](#_Toc86796458)

# 1. Введение

Тема работы: Управление файловой системой.

Цель работы: исследовать механизмы управления виртуальной памятью Win32.

Указания к выполнению

Задание 4.1. Исследовать виртуальное адресное пространство процесса.

1. Создайте консольное приложение с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которое выполняет:

− получение информации о вычислительной системе (функция Win32 API – GetSystemInfo);

− определение статуса виртуальной памяти (функция Win32 API – GlobalMemoryStatus);

− определение состояния конкретного участка памяти по заданному с клавиатуры адресу (функция Win32 API – VirtualQuery);

− резервирование региона в автоматическом режиме и в режиме ввода адреса начала региона (функция Win32 API – VirtualAlloc);

− резервирование региона и передача ему физической памяти в автоматическом режиме и в режиме ввода адреса начала региона (функция Win32 API – VirtualAlloc);

− запись данных в ячейки памяти по заданным с клавиатуры адресам;

− установку защиты доступа для заданного (с клавиатуры) региона памяти и ее проверку (функция Win32 API – VirtualProtect);

− возврат физической памяти и освобождение региона адресного пространства заданного (с клавиатуры) региона памяти (функция Win32 API –VirtualFree).

2. Запустите приложение и проверьте его работоспособность на нескольких наборах вводимых данных. Запротоколируйте результаты в отчет. Дайте свои комментарии в отчете относительно выполнения функций Win32 API.

3. Подготовьте итоговый отчет с развернутыми выводами по заданию.

Задание 4.2. Использование проецируемых файлов для обмена данными между процессами.

1. Создайте два консольных приложения с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которые выполняют:

− приложение-писатель создает проецируемый файл (функции Win32 API – CreateFile, CreateFileMapping), проецирует фрагмент файла в память (функции Win32 API – MapViewOfFile, UnmapViewOfFile), осуществляет ввод данных с клавиатуры и их запись в спроецированный файл;

− приложение-читатель открывает проецируемый файл (функция Win32 API – OpenFileMapping), проецирует фрагмент файла в память (функции Win32 API – MapViewOfFile, UnmapViewOfFile), считывает содержимое из спроецированного файла и отображает на экран.

2. Запустите приложения и проверьте обмен данных между процессами, удостоверьтесь в надлежащем выполнении задания. Запротоколируйте результаты в отчет. Дайте свои комментарии в отчете относительно выполнения функций Win32 API.

3. Подготовьте итоговый отчет с развернутыми выводами по заданию.

# 2. Исследование виртуального адресного пространства процесса

## 2.1. Получение информации о вычислительной системе

Реализация главного меню программы и вывод информации о вычислительной системе с использованием функции GetSystemInfo().

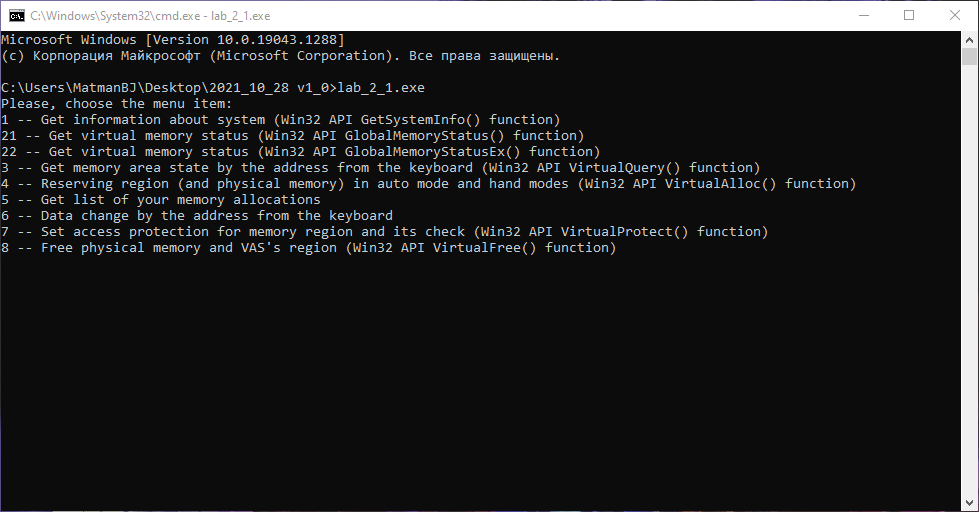


Рисунок : Главное меню программы

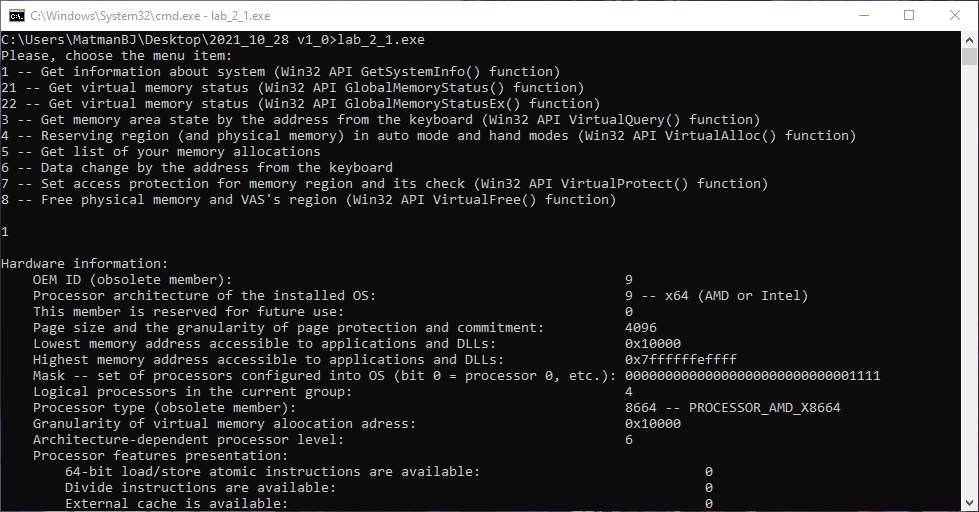


Рисунок : Вывод информации о системе

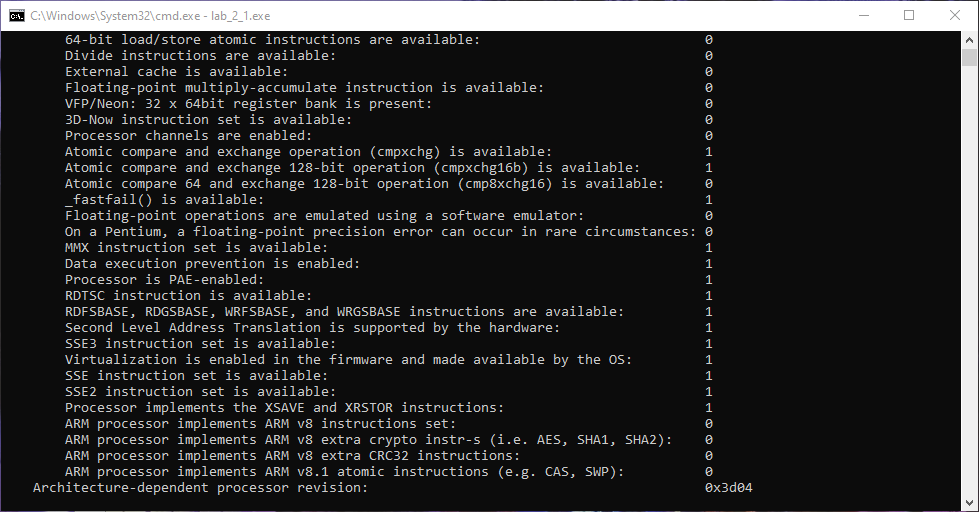


Рисунок : Вывод информации о системе

## 2.2. Определение статуса виртуальной памяти

Вывод основной информации о статусе виртуальной памяти с использованием функций GlobalMemoryStatus() и GlobalMemoryStatusEx(). Функция GlobalMemoryStatusEx() рекомендуема согласно документации Microsoft ввиду возможного получения ошибочных данных при использовании функции GlobalMemoryStatus(), однако различия в результатах обеих функций минимальны.

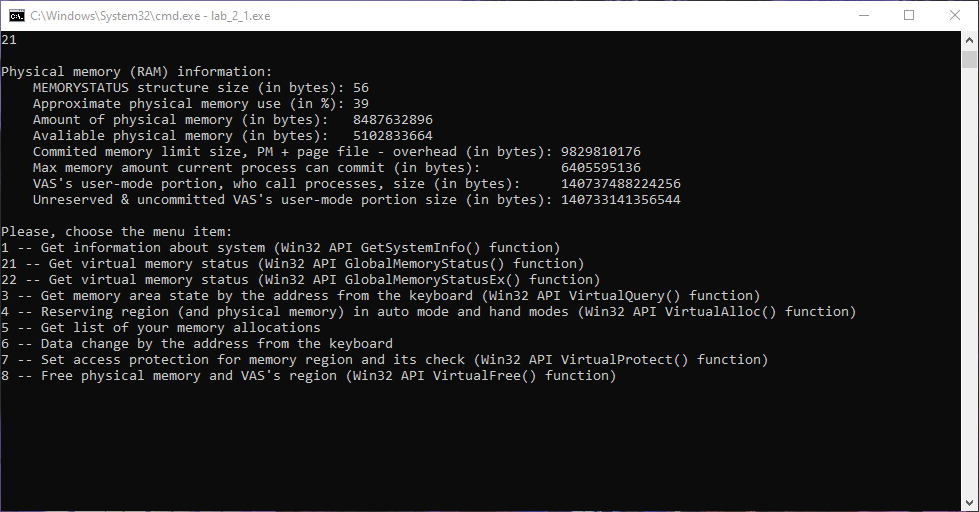


Рисунок 4: Вывод статуса виртуальной памяти

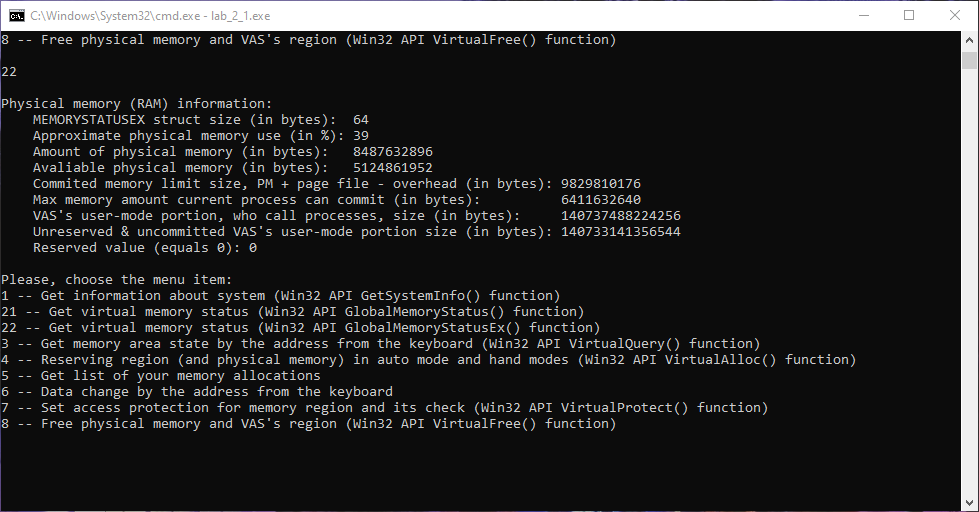


Рисунок 5: Вывод статуса виртуальной памяти

## 2.3. Определение состояния конкретного участка памяти по заданному с клавиатуры адресу

Вывод информации о состоянии конкретного участка памяти по адресу с использованием функции VirtualQuery().

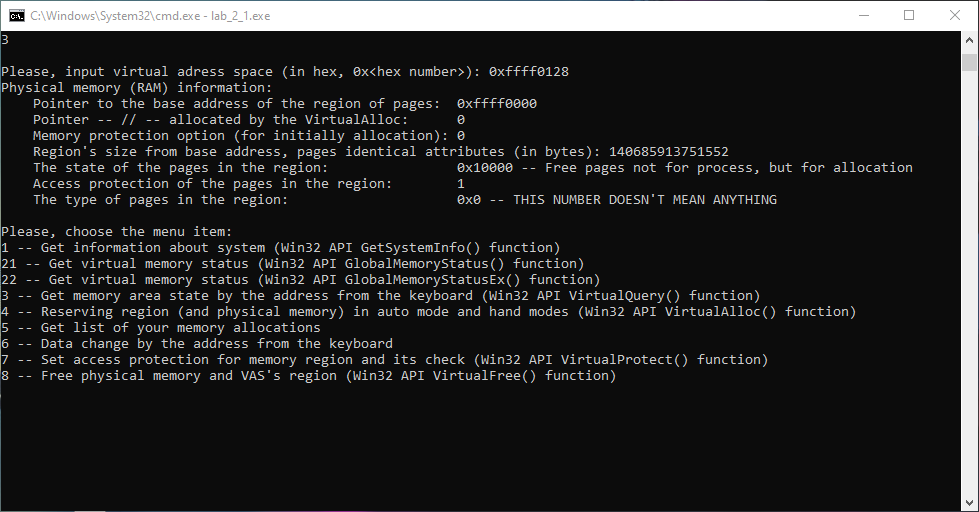


Рисунок 6: Создание каталога

## 2.4. Резервирование региона (и передача ему физической памяти) в автоматическом режиме и в режиме ввода адреса начала региона

Резервирование региона (и передача ему физической памяти) в автоматическом режиме и в режиме ввода адреса начала региона с использованием функции VirtualAlloc(). Резервирование региона, а также передача ему физической памяти реализуются с помощью соответствующих флагов, передаваемых в функцию VirtualAlloc() в качестве параметров.

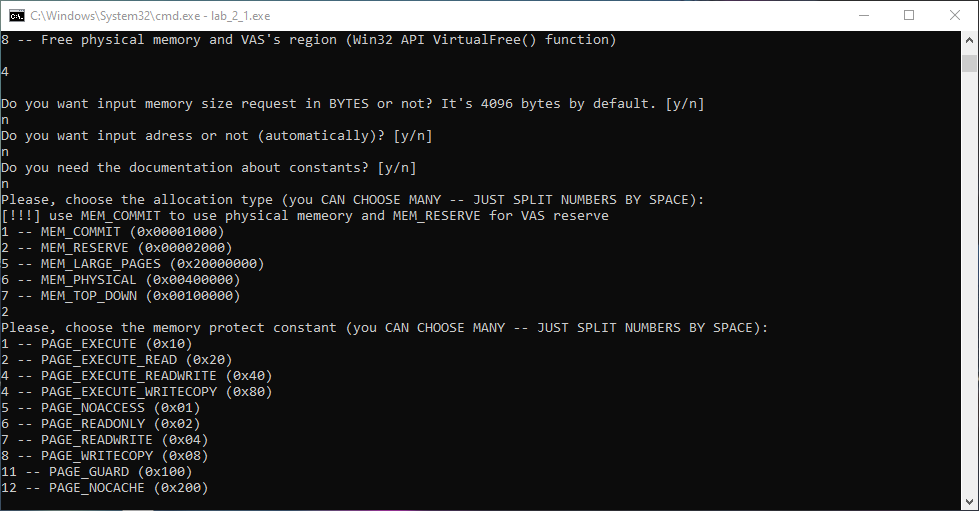


Рисунок 7: Резервирование региона в автоматическом режиме

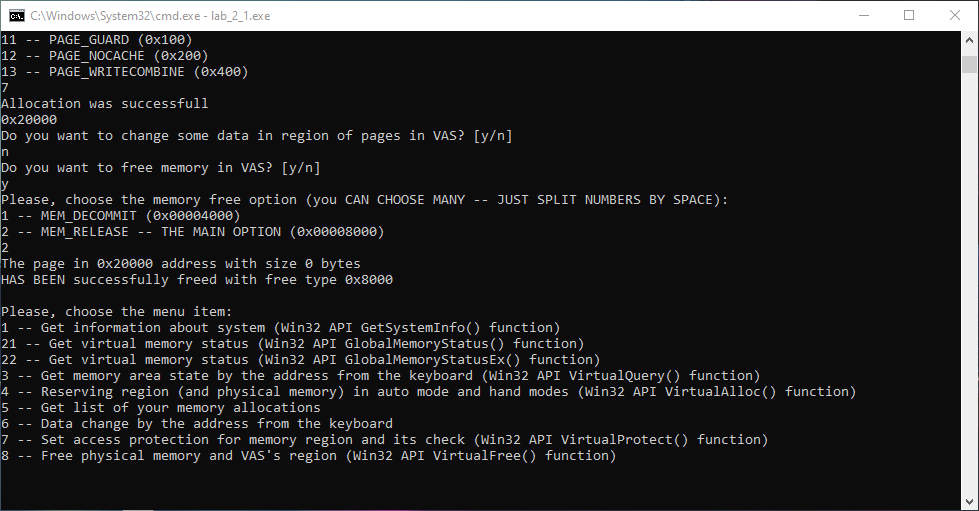


Рисунок 8: Резервирование региона в автоматическом режиме

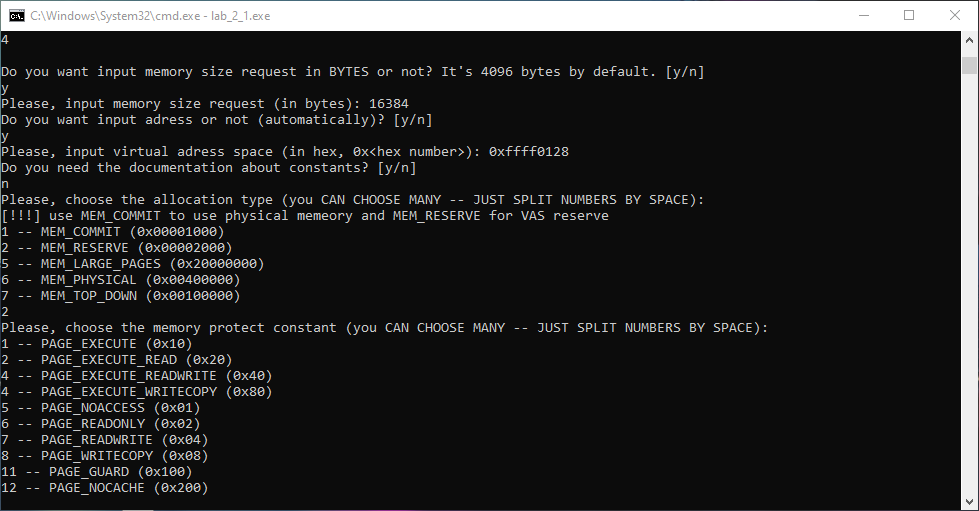


Рисунок 9: Резервирование региона в ручном режиме

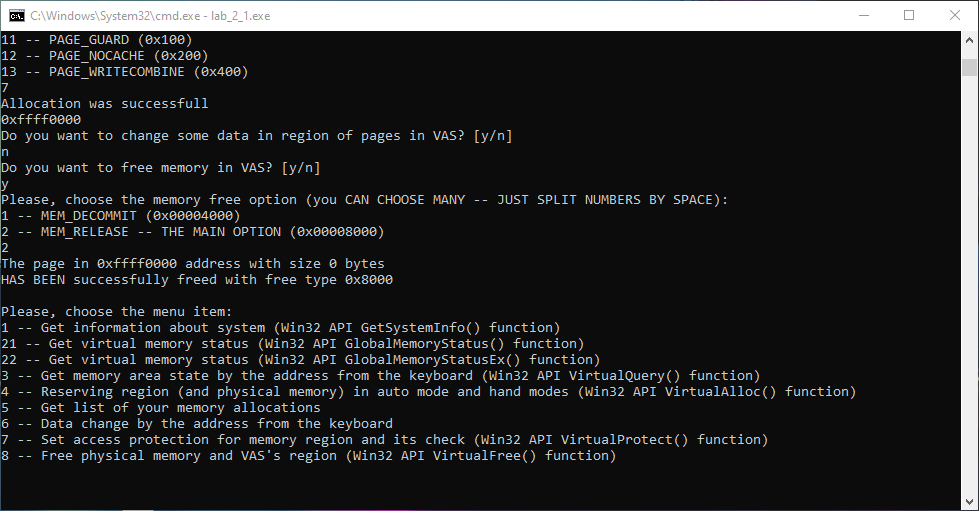


Рисунок 10: Резервирование региона в ручном режиме

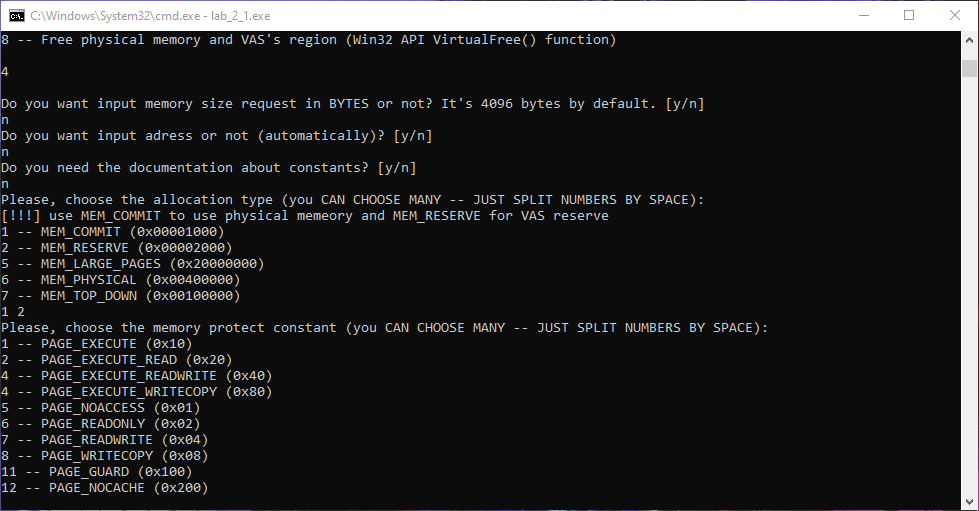


Рисунок : Резервирование региона и передача ему физической памяти в автоматическом режиме

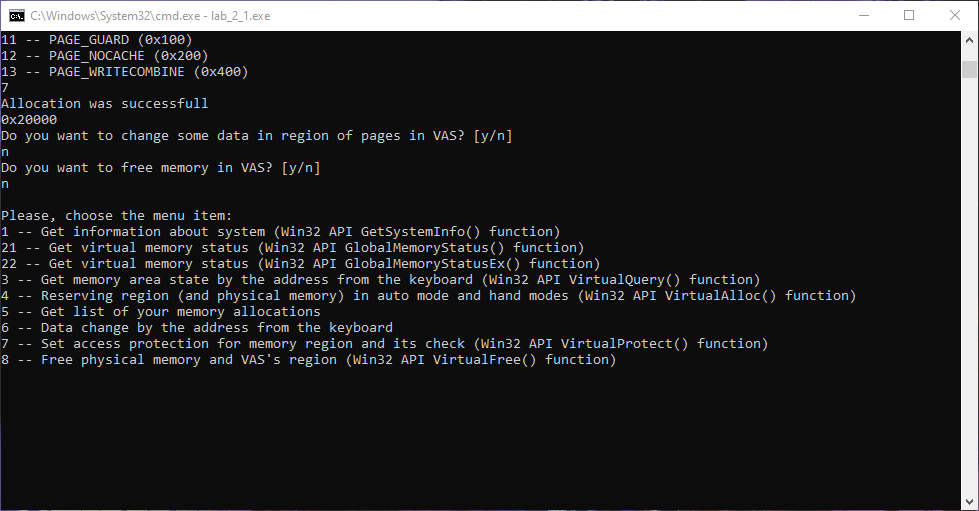


Рисунок : Резервирование региона и передача ему физической памяти в автоматическом режиме

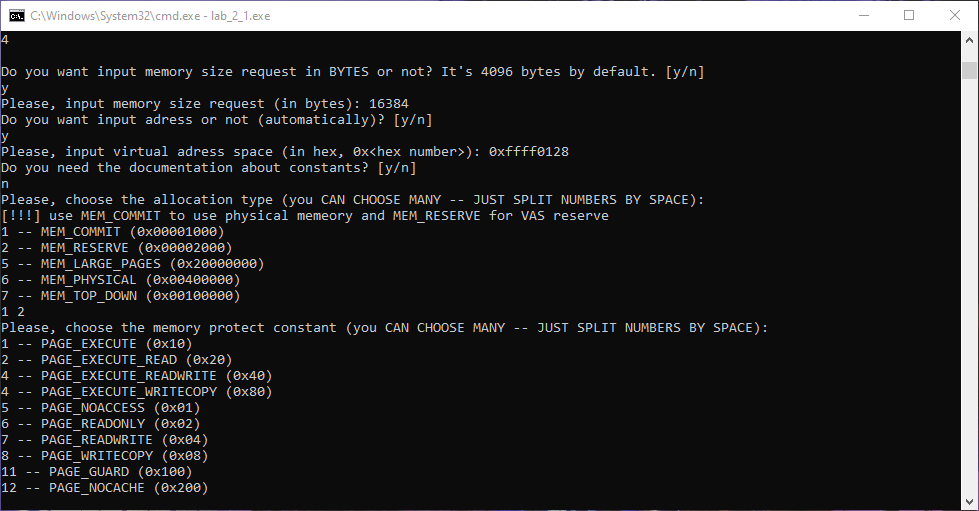


Рисунок : Резервирование региона и передача ему физической памяти в ручном режиме

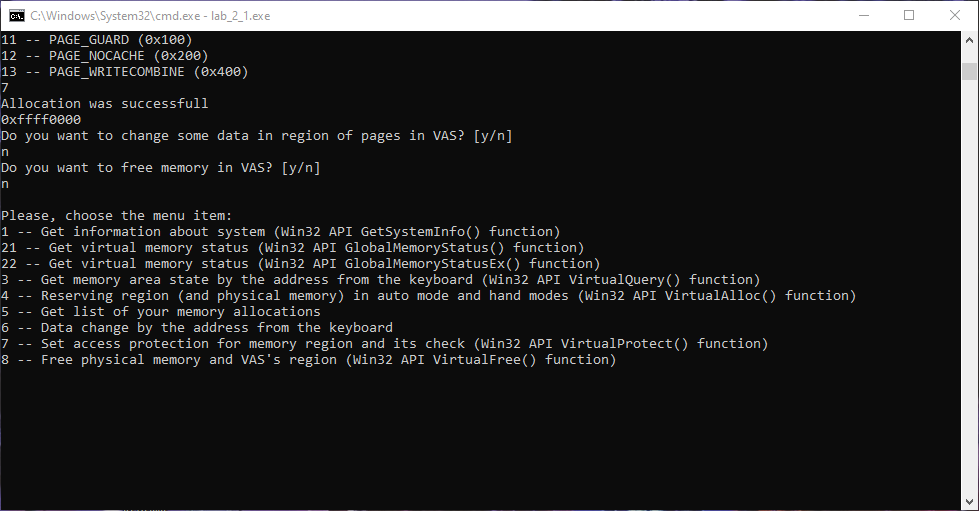


Рисунок : Резервирование региона и передача ему физической памяти в ручном режиме

## 2.5. Вывод списка зарезервированных пользователем адресов

Вывод списка зарезервированных пользователем адресов (в том числе и с физической памятью).

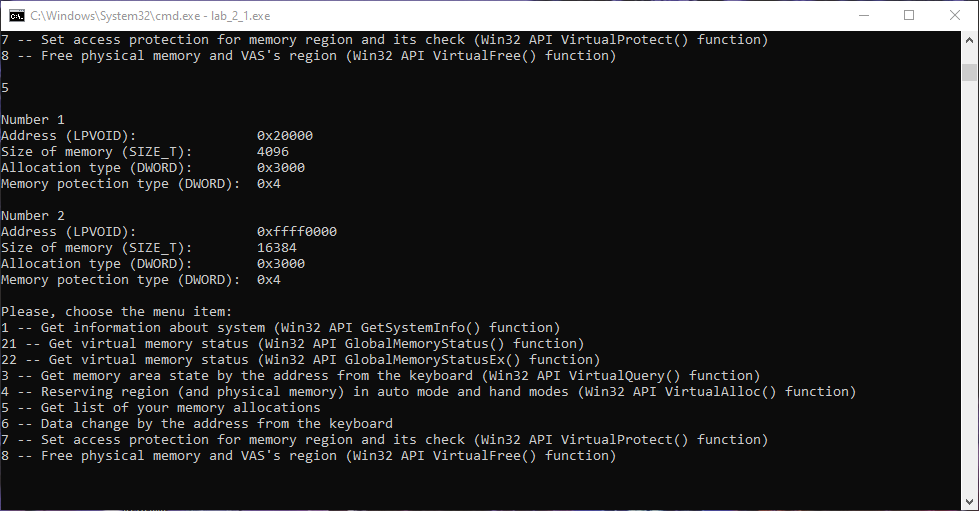


Рисунок 15: Копирование файла

## 2.6. Запись данных в ячейки памяти по заданным с клавиатуры адресам

Запись данных в ячейки памяти по заданным с клавиатуры адресам. Запись данных производится с помощью различных основных типов данных, в том числе логического, символьного, целочисленного и вещественного, которые для ввода и вывода выбирает сам пользователь. Она и просмотр данных возможны на любом участке доступного адресного пространства. В программе реализована проверка вводимого адреса. Запись данных возможна как при выборе соответствующего пункта в меню, так и при выборе соответствующего подраздела в меню выделения памяти после успешного завершения операции.

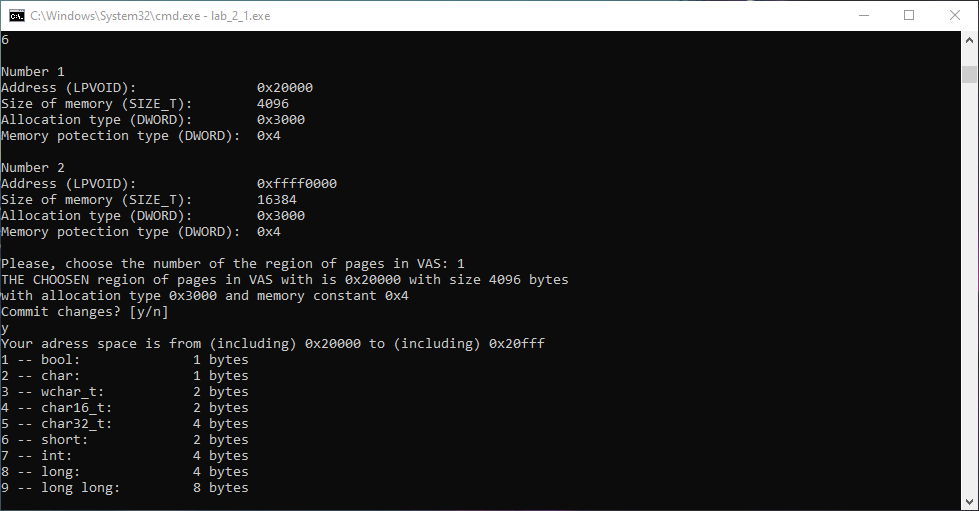


Рисунок 16: Вывод атрибутов файла

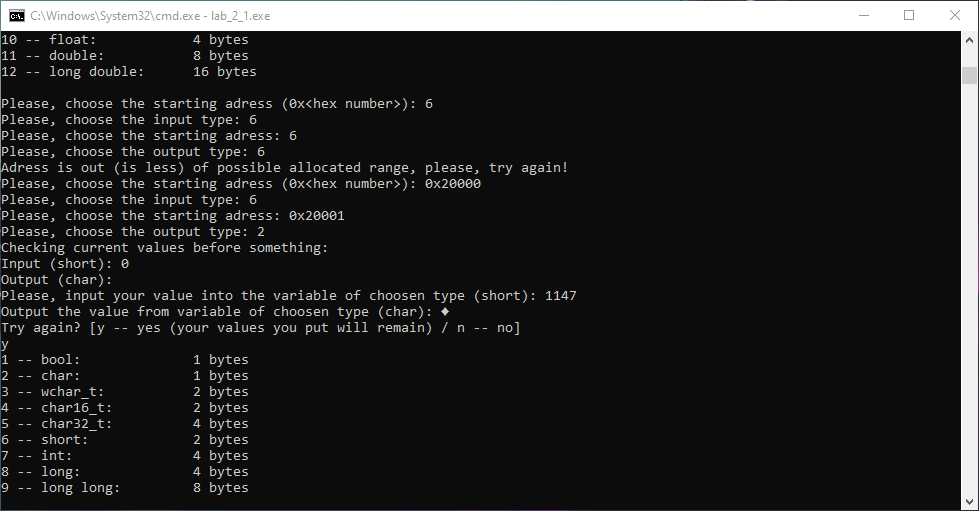


Рисунок 17: Изменение атрибутов файла

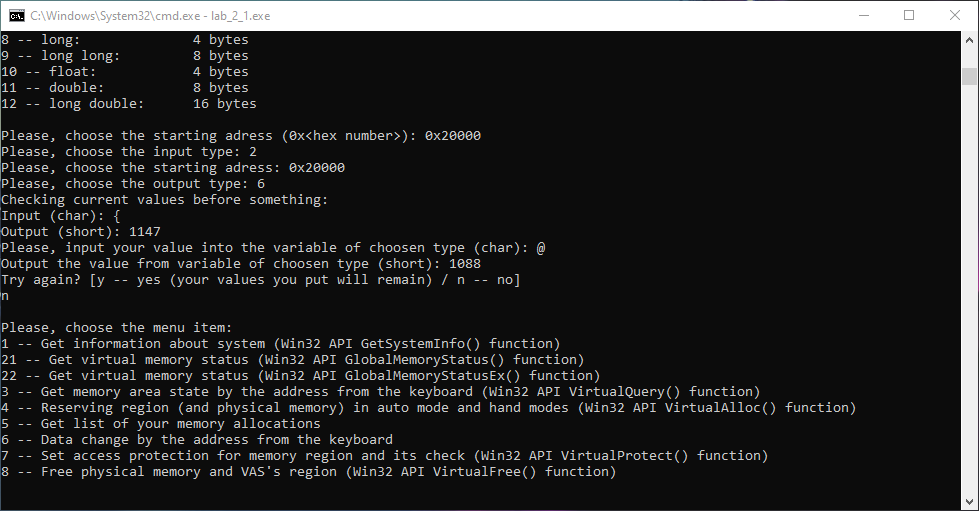


Рисунок 18: Изменённый атрибут файла в свойствах

## 2.7. Установка защиты доступа для заданного (с клавиатуры) региона памяти и её проверку

Установка константы защиты памяти для заданного (с клавиатуры) региона памяти с использованием функции VirtualProtect(). Это происходит с помощью соответствующих флагов, принимаемых функцией в качестве параметра, а в качестве проверки предыдущее и установленное значения констант выводятся. Установка константы защиты памяти производится для выделенных пользователем адресов.

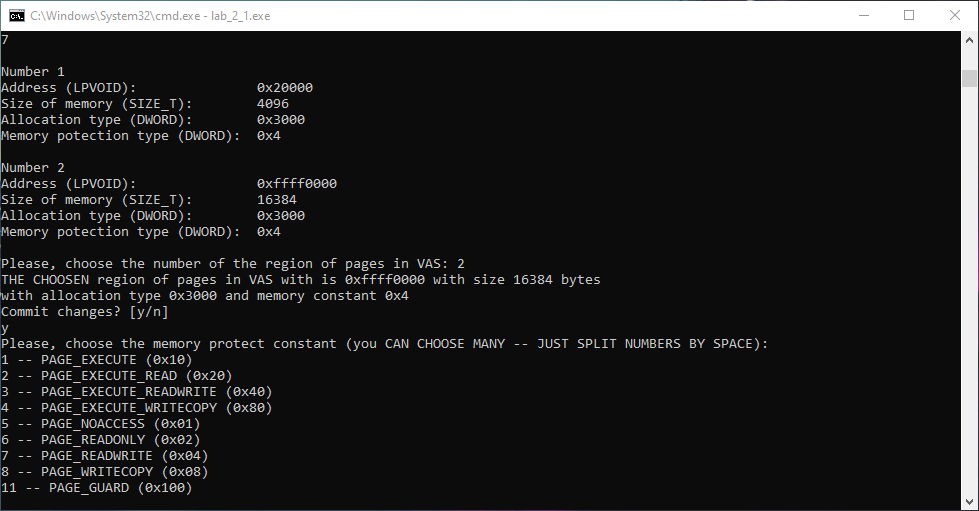


Рисунок : Установка константы защиты памяти

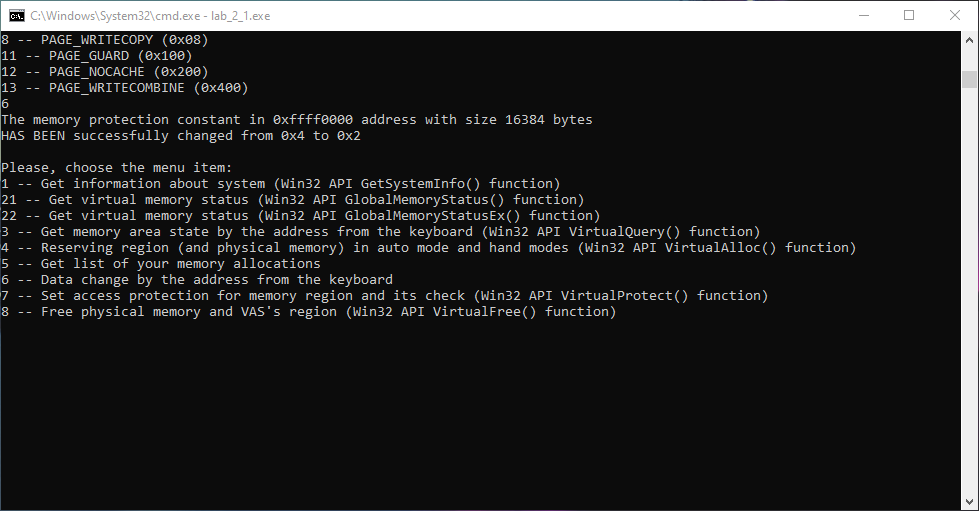


Рисунок : Установка константы защиты памяти

## 2.8. Возврат физической памяти и освобождение региона адресного пространства заданного (с клавиатуры) региона памяти

Возврат физической памяти и освобождение региона адресного пространства заданного (с клавиатуры) региона памяти. Это происходит с помощью соответствующих флагов, принимаемых функцией в качестве параметра. Возврат физической памяти и освобождение региона адресного пространства заданного региона памяти производится для выделенных пользователем адресов.

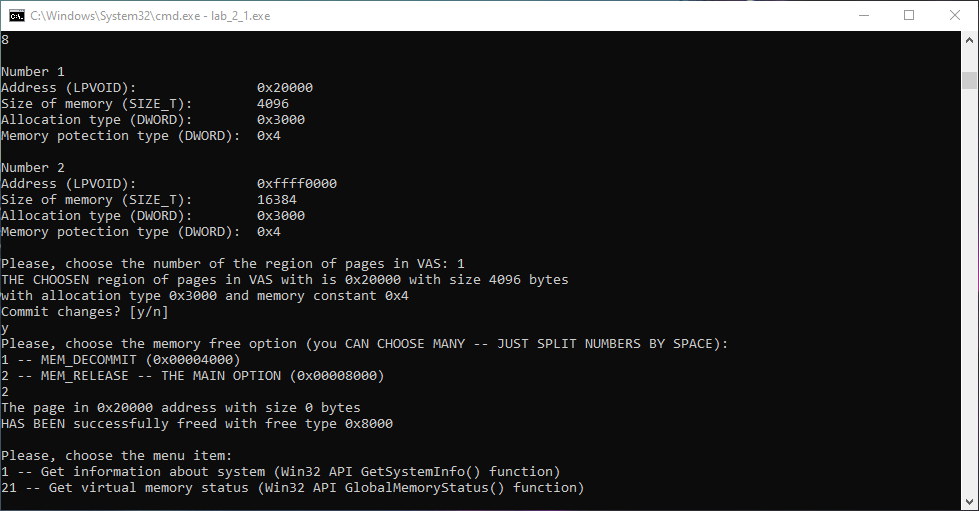


Рисунок : Возврат физической памяти и освобождение региона адресного пространства

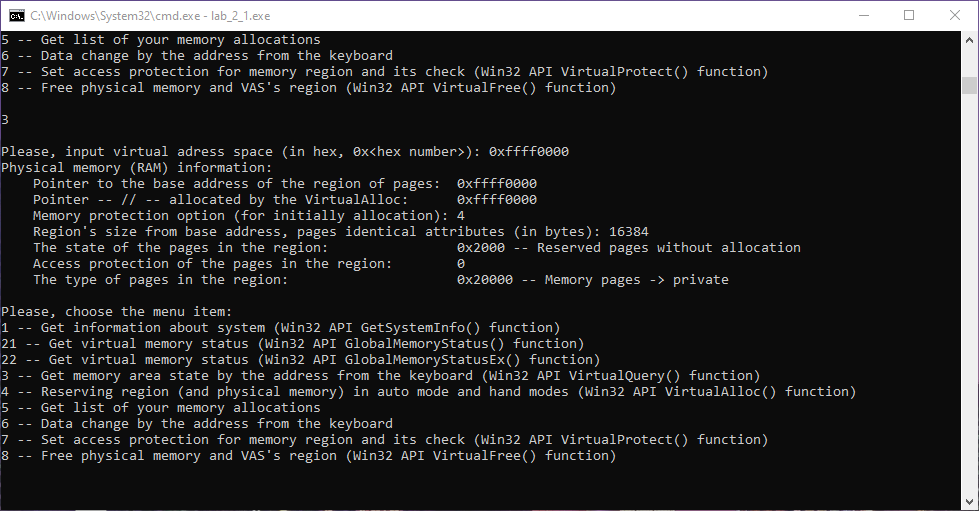


Рисунок : Проверка региона адресного пространства

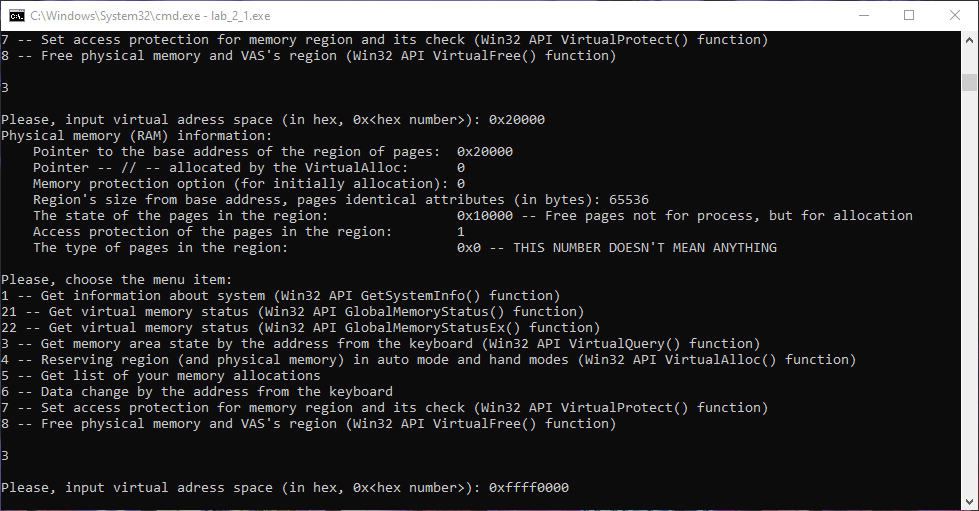


Рисунок : Проверка региона адресного пространства

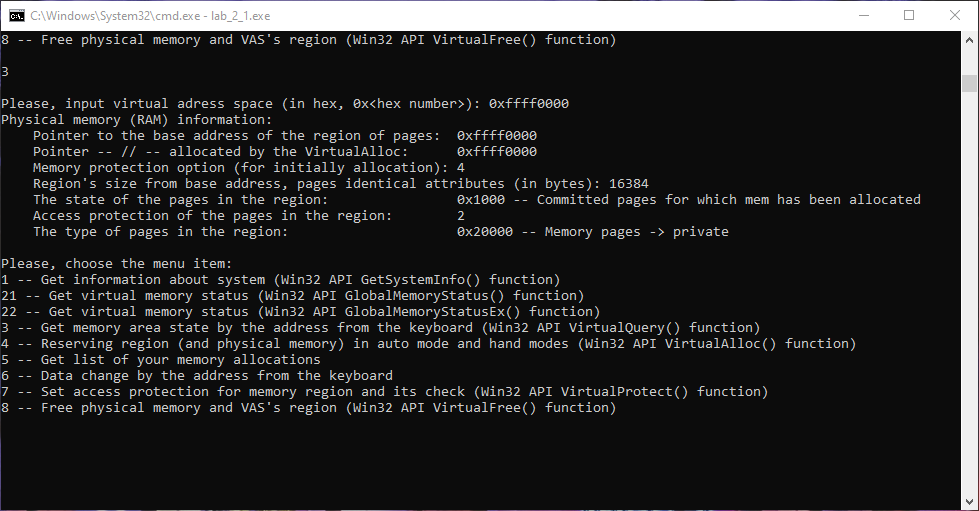


Рисунок : Возврат физической памяти

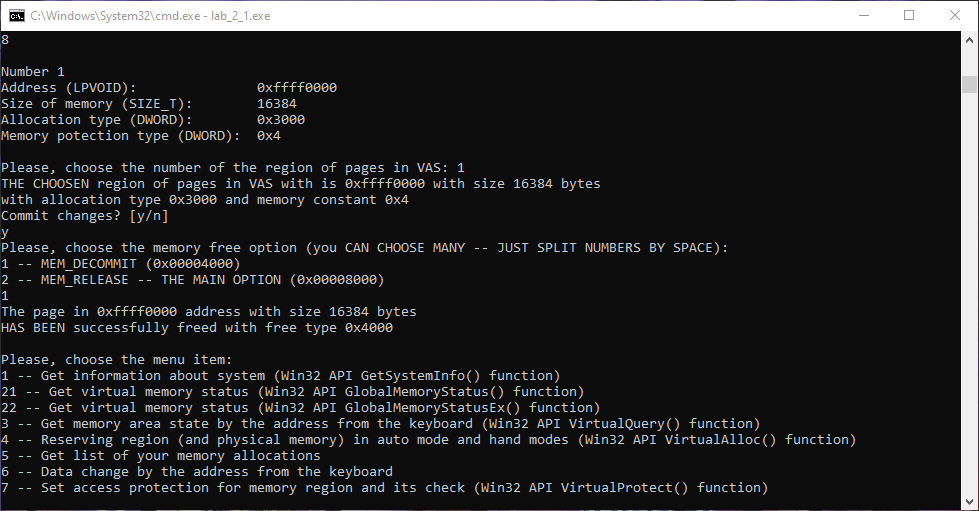


Рисунок : Проверка региона адресного пространства

## 2.9. Исходный код программы

/\*

Win32 API (WinAPI) is a set of functions in the library <windows.h>

API means "Application Programming Interface"

\*/

#include <windows.h> // for WinAPI functions

#include <bitset> // for binary output

#include <math.h> // for double making

#include <exception> // for exceptions

#include <iostream> // just for working

#include <string> // for the "string" type using

#include <vector> // for the "vector" type using

#include <tuple> // for the "tuple" type using

#include <algorithm> // for the "find" function using

using namespace std;

typedef vector<tuple<LPVOID, SIZE\_T, DWORD, DWORD>> LOCALLOC; // new thing for locating all local allocations

LOCALLOC listOfAllocations;

// ---------- FUNCTION DECLARATION ----------

bool BoolSafetyInput ();

void MainMenu ();

void Info ();

void LocalGetSystemInfo ();

void LocalGlobalMemoryStatus ();

void LocalGlobalMemoryStatusEx ();

void LocalVirtualQuery ();

void LocalListOfAllocations ();

void LocalListOfAllocationsFree ();

void LocalVirtualAlloc ();

void LocalDataChangeCore (LPVOID localVirtualAlloc, SIZE\_T localMemorySize);

void LocalDataChangeIndependent ();

void LocalVirtualProtect ();

void LocalVirtualFreeCore (LPVOID locallpAddress, SIZE\_T localdwSize);

void LocalVirtualFreeIndependent ();

// ---------- MAIN ----------

int main (int argc, char\* argv[]) // i've finally understood what it means (argc -- number of arguments, argv -- strings of arguments (including -<word> and --<word>))

{

// "GET CURRENT DIRECTORY", "SET CURRENT DIRECTORY"

int flag = -1; // "-1" for incorrect input continue the program

do

{

MainMenu();

cin >> flag;

cout << "\n";

switch (flag)

{

case 0:

cout << "Goodbye!";

break;

case 1:

LocalGetSystemInfo();

break;

case 21:

LocalGlobalMemoryStatus();

break;

case 22:

LocalGlobalMemoryStatusEx();

break;

case 3:

LocalVirtualQuery();

break;

case 4:

LocalVirtualAlloc();

break;

case 5:

LocalListOfAllocations();

break;

case 6:

LocalDataChangeIndependent();

break;

case 7:

LocalVirtualProtect ();

break;

case 8:

LocalVirtualFreeIndependent ();

break;

default:

cout << "Incorrect input! Try again.";

break;

}

}

while (flag != 0);

LocalListOfAllocationsFree ();

return 0;

}

// ---------- 0 -- Bool Safety Input ----------

bool BoolSafetyInput ()

{

string localNewVariable;

bool localNewBool = false;

bool localFlag = true;

while (localFlag == true)

{

fflush(stdin);

getline(cin, localNewVariable);

if (localNewVariable.compare("0") == 0)

{

localNewBool = false;

localFlag = false;

}

else if (localNewVariable.compare("1") == 0)

{

localNewBool = true;

localFlag = false;

}

else

{

cout << "Wrong bool number (use only 0 or 1)!\n";

}

}

return localNewBool;

}

// ---------- 0 -- MAIN MENU ----------

void MainMenu ()

{

cout << "Please, choose the menu item:\n"

<< "1 -- Get information about system (Win32 API GetSystemInfo() function)\n"

<< "21 -- Get virtual memory status (Win32 API GlobalMemoryStatus() function)\n"

<< "22 -- Get virtual memory status (Win32 API GlobalMemoryStatusEx() function)\n"

<< "3 -- Get memory area state by the address from the keyboard (Win32 API VirtualQuery() function)\n"

<< "4 -- Reserving region (and physical memory) in auto mode and hand modes (Win32 API VirtualAlloc() function)\n"

<< "5 -- Get list of your memory allocations\n"

<< "6 -- Data change by the address from the keyboard\n"

<< "7 -- Set access protection for memory region and its check (Win32 API VirtualProtect() function)\n"

<< "8 -- Free physical memory and VAS's region (Win32 API VirtualFree() function)\n"

<< "\n";

}

// ---------- 0 -- INFO ----------

void Info ()

{

cout << "Saint Petersburg Electrotechnical University \"LETI\" (ETU \"LETI\"),\n"

<< "Faculty of Computer Science and Technology \"FKTI\",\n"

<< "Department of Computer Science and Engineering,\n"

<< "Computer Systems Engineering and Informatics (09.03.01) program.\n\n"

<< "OS labortory work 2 \"Memory control\" version 1\_0 dated 2021\_10\_28\n\n"

<< "This software is under MIT License (X11 License).\n"

<< "You can see a detailed description in \"LICENSE.md\" file.\n\n"

<< "Copyight (c) 2021 Sobolev Matvey Sergeevich\n";

}

// ---------- 1 -- LOCAL GET SYSTEM INFO ----------

/\*

void GetSystemInfo(

LPSYSTEM\_INFO lpSystemInfo

);

typedef struct \_SYSTEM\_INFO {

union {

DWORD dwOemId;

struct {

WORD wProcessorArchitecture;

WORD wReserved;

} DUMMYSTRUCTNAME;

} DUMMYUNIONNAME;

DWORD dwPageSize;

LPVOID lpMinimumApplicationAddress;

LPVOID lpMaximumApplicationAddress;

DWORD\_PTR dwActiveProcessorMask;

DWORD dwNumberOfProcessors;

DWORD dwProcessorType;

DWORD dwAllocationGranularity;

WORD wProcessorLevel;

WORD wProcessorRevision;

} SYSTEM\_INFO, \*LPSYSTEM\_INFO;

\*/

void LocalGetSystemInfo()

{

SYSTEM\_INFO localSystemInfo; // creating the structure

GetSystemInfo(&localSystemInfo); // sending the pointer and getting the information

cout << "Hardware information:\n"; // information output

// DWORD dwOemId output

cout << " OEM ID (obsolete member): " << localSystemInfo.dwOemId << "\n";

// WORD wProcessorArchitecture output

if (localSystemInfo.wProcessorArchitecture == PROCESSOR\_ARCHITECTURE\_AMD64) // number 9

{

cout << " Processor architecture of the installed OS: " << localSystemInfo.wProcessorArchitecture << " -- " << "x64 (AMD or Intel)\n";

}

else if (localSystemInfo.wProcessorArchitecture == PROCESSOR\_ARCHITECTURE\_ARM) // number 5

{

cout << " Processor architecture of the installed OS: " << localSystemInfo.wProcessorArchitecture << " -- " << "ARM\n";

}

else if (localSystemInfo.wProcessorArchitecture == 0x000c /\*PROCESSOR\_ARCHITECTURE\_ARM64\*/) // number 12; with "PROCESSOR\_ARCHITECTURE\_ARM64" it doesn't work

{

cout << " Processor architecture of the installed OS: " << localSystemInfo.wProcessorArchitecture << " -- " << "ARM64\n";

}

else if (localSystemInfo.wProcessorArchitecture == PROCESSOR\_ARCHITECTURE\_IA64) // number 6

{

cout << " Processor architecture of the installed OS: " << localSystemInfo.wProcessorArchitecture << " -- " << "Intel Itanium-based\n";

}

else if (localSystemInfo.wProcessorArchitecture == PROCESSOR\_ARCHITECTURE\_INTEL) // number 0

{

cout << " Processor architecture of the installed OS: " << localSystemInfo.wProcessorArchitecture << " -- " << "x86\n";

}

else if (localSystemInfo.wProcessorArchitecture == PROCESSOR\_ARCHITECTURE\_UNKNOWN) // number 0xffff

{

cout << " Processor architecture of the installed OS: " << localSystemInfo.wProcessorArchitecture << " -- " << "Unknown architecture.\n";

}

else // other number

{

cout << " Processor architecture of the installed OS: " << localSystemInfo.wProcessorArchitecture << " -- " << "THIS NUMBER DOESN'T MEAN ANYTHING\n";

}

// WORD wReserved output

cout << " This member is reserved for future use: " << localSystemInfo.wReserved << "\n";

// DWORD dwPageSize output

cout << " Page size and the granularity of page protection and commitment: " << localSystemInfo.dwPageSize << "\n";

// LPVOID lpMinimumApplicationAddress output

cout << " Lowest memory address accessible to applications and DLLs: " << localSystemInfo.lpMinimumApplicationAddress << "\n";

// LPVOID lpMaximumApplicationAddress output

cout << " Highest memory address accessible to applications and DLLs: " << localSystemInfo.lpMaximumApplicationAddress << "\n";

// DWORD\_PTR dwActiveProcessorMask output

cout << " Mask -- set of processors configured into OS (bit 0 = processor 0, etc.): " << bitset<32>(localSystemInfo.dwActiveProcessorMask) << "\n";

// DWORD dwNumberOfProcessors output

cout << " Logical processors in the current group: " << localSystemInfo.dwNumberOfProcessors << "\n";

// DWORD dwProcessorType output

if (localSystemInfo.dwProcessorType == PROCESSOR\_INTEL\_386) // number 386

{

cout << " Processor type (obsolete member): " << localSystemInfo.dwProcessorType << " -- " << "PROCESSOR\_INTEL\_386\n";

}

else if (localSystemInfo.dwProcessorType == PROCESSOR\_INTEL\_486) // number 486

{

cout << " Processor type (obsolete member): " << localSystemInfo.dwProcessorType << " -- " << "PROCESSOR\_INTEL\_486\n";

}

else if (localSystemInfo.dwProcessorType == PROCESSOR\_INTEL\_PENTIUM) // number 586; with "PROCESSOR\_ARCHITECTURE\_ARM64" it doesn't work

{

cout << " Processor type (obsolete member): " << localSystemInfo.dwProcessorType << " -- " << "PROCESSOR\_INTEL\_PENTIUM\n";

}

else if (localSystemInfo.dwProcessorType == PROCESSOR\_INTEL\_IA64) // number 2200

{

cout << " Processor type (obsolete member): " << localSystemInfo.dwProcessorType << " -- " << "PROCESSOR\_INTEL\_IA64\n";

}

else if (localSystemInfo.dwProcessorType == PROCESSOR\_AMD\_X8664) // number 8664

{

cout << " Processor type (obsolete member): " << localSystemInfo.dwProcessorType << " -- " << "PROCESSOR\_AMD\_X8664\n";

}

/\*else if (localSystemInfo.dwProcessorType == PROCESSOR\_ARM) // Reserved

{

cout << " Processor type (obsolete member): " << localSystemInfo.dwProcessorType << " -- " << "PROCESSOR\_ARM (Reserved)\n";

}\*/

else // other number

{

cout << " Processor type (obsolete member): " << localSystemInfo.dwProcessorType << " -- " << "THIS NUMBER DOESN'T MEAN ANYTHING\n";

}

// DWORD dwAllocationGranularity output

cout << " Granularity of virtual memory aloocation adress: 0x" << hex << localSystemInfo.dwAllocationGranularity << dec << "\n";

// WORD wProcessorLevel output

/\*

If wProcessorArchitecture is PROCESSOR\_ARCHITECTURE\_INTEL, wProcessorLevel is defined by the CPU vendor.

If wProcessorArchitecture is PROCESSOR\_ARCHITECTURE\_IA64, wProcessorLevel is set to 1.

\*/

cout << " Architecture-dependent processor level: " << localSystemInfo.wProcessorLevel << "\n";

// Procaessor features output (it's not a part of the structure)

cout << " Processor features presentation:\n";

cout << " 64-bit load/store atomic instructions are available: " << IsProcessorFeaturePresent(PF\_ARM\_64BIT\_LOADSTORE\_ATOMIC) << "\n"; // number 25

cout << " Divide instructions are available: " << IsProcessorFeaturePresent(PF\_ARM\_DIVIDE\_INSTRUCTION\_AVAILABLE) << "\n"; // number 24

cout << " External cache is available: " << IsProcessorFeaturePresent(PF\_ARM\_EXTERNAL\_CACHE\_AVAILABLE) << "\n"; // number 26

cout << " Floating-point multiply-accumulate instruction is available: " << IsProcessorFeaturePresent(PF\_ARM\_FMAC\_INSTRUCTIONS\_AVAILABLE) << "\n"; // number 27

cout << " VFP/Neon: 32 x 64bit register bank is present: " << IsProcessorFeaturePresent(PF\_ARM\_VFP\_32\_REGISTERS\_AVAILABLE) << "\n"; // number 18

//cout << " VFP/Neon: 32 x 64bit register bank is present (other flag): " << IsProcessorFeaturePresent(PF\_ARM\_VFP\_EXTENDED\_REGISTERS) << "\n";

cout << " 3D-Now instruction set is available: " << IsProcessorFeaturePresent(PF\_3DNOW\_INSTRUCTIONS\_AVAILABLE) << "\n"; // number 7

cout << " Processor channels are enabled: " << IsProcessorFeaturePresent(PF\_CHANNELS\_ENABLED) << "\n"; // number 16

cout << " Atomic compare and exchange operation (cmpxchg) is available: " << IsProcessorFeaturePresent(PF\_COMPARE\_EXCHANGE\_DOUBLE) << "\n"; // number 2

cout << " Atomic compare and exchange 128-bit operation (cmpxchg16b) is available: " << IsProcessorFeaturePresent(PF\_COMPARE\_EXCHANGE128) << "\n"; // number 14

cout << " Atomic compare 64 and exchange 128-bit operation (cmp8xchg16) is available: " << IsProcessorFeaturePresent(PF\_COMPARE64\_EXCHANGE128) << "\n"; // number 15

cout << " \_fastfail() is available: " << IsProcessorFeaturePresent(PF\_FASTFAIL\_AVAILABLE) << "\n"; // number 23

cout << " Floating-point operations are emulated using a software emulator: " << IsProcessorFeaturePresent(PF\_FLOATING\_POINT\_EMULATED) << "\n"; // number 1

cout << " On a Pentium, a floating-point precision error can occur in rare circumstances: " << IsProcessorFeaturePresent(PF\_FLOATING\_POINT\_PRECISION\_ERRATA) << "\n"; // number 0

cout << " MMX instruction set is available: " << IsProcessorFeaturePresent(PF\_MMX\_INSTRUCTIONS\_AVAILABLE) << "\n"; // number 3

cout << " Data execution prevention is enabled: " << IsProcessorFeaturePresent(PF\_NX\_ENABLED) << "\n"; // number 12

cout << " Processor is PAE-enabled: " << IsProcessorFeaturePresent(PF\_PAE\_ENABLED) << "\n"; // number 9

cout << " RDTSC instruction is available: " << IsProcessorFeaturePresent(PF\_RDTSC\_INSTRUCTION\_AVAILABLE) << "\n"; // number 8

cout << " RDFSBASE, RDGSBASE, WRFSBASE, and WRGSBASE instructions are available: " << IsProcessorFeaturePresent(PF\_RDWRFSGSBASE\_AVAILABLE) << "\n"; // number 22

cout << " Second Level Address Translation is supported by the hardware: " << IsProcessorFeaturePresent(PF\_SECOND\_LEVEL\_ADDRESS\_TRANSLATION) << "\n"; // number 20

cout << " SSE3 instruction set is available: " << IsProcessorFeaturePresent(PF\_SSE3\_INSTRUCTIONS\_AVAILABLE) << "\n"; // number 13

cout << " Virtualization is enabled in the firmware and made available by the OS: " << IsProcessorFeaturePresent(PF\_VIRT\_FIRMWARE\_ENABLED) << "\n"; // number 21

cout << " SSE instruction set is available: " << IsProcessorFeaturePresent(PF\_XMMI\_INSTRUCTIONS\_AVAILABLE) << "\n"; // number 6

cout << " SSE2 instruction set is available: " << IsProcessorFeaturePresent(PF\_XMMI64\_INSTRUCTIONS\_AVAILABLE) << "\n"; // number 10

cout << " Processor implements the XSAVE and XRSTOR instructions: " << IsProcessorFeaturePresent(PF\_XSAVE\_ENABLED) << "\n"; // number 17

//cout << " ARM processor implements the the ARM v8 instructions set: " << IsProcessorFeaturePresent(PF\_ARM\_V8\_INSTRUCTIONS\_AVAILABLE) << "\n";

//cout << " ARM processor implements the ARM v8 extra cryptographic instructions (i.e. AES, SHA1 and SHA2): " << IsProcessorFeaturePresent(PF\_ARM\_V8\_CRYPTO\_INSTRUCTIONS\_AVAILABLE) << "\n";

//cout << " ARM processor implements the ARM v8 extra CRC32 instructions: " << IsProcessorFeaturePresent(PF\_ARM\_V8\_CRC32\_INSTRUCTIONS\_AVAILABLE) << "\n";

//cout << " ARM processor implements the ARM v8.1 atomic instructions (e.g. CAS, SWP): " << IsProcessorFeaturePresent(PF\_ARM\_V81\_ATOMIC\_INSTRUCTIONS\_AVAILABLE) << "\n";

cout << " ARM processor implements ARM v8 instructions set: " << IsProcessorFeaturePresent(29) << "\n"; // [crutch]

cout << " ARM processor implements ARM v8 extra crypto instr-s (i.e. AES, SHA1, SHA2): " << IsProcessorFeaturePresent(30) << "\n"; // [crutch]

cout << " ARM processor implements ARM v8 extra CRC32 instructions: " << IsProcessorFeaturePresent(31) << "\n"; // [crutch]

cout << " ARM processor implements ARM v8.1 atomic instructions (e.g. CAS, SWP): " << IsProcessorFeaturePresent(34) << "\n"; // [crutch]

// WORD wProcessorRevision output

/\*

Intel Pentium, Cyrix, or NextGen 586:

The high byte is the model and the low byte is the stepping. For example, if the value is xxyy, the model number and stepping can be displayed as follows:

Model xx, Stepping yy

Intel 80386 or 80486:

A value of the form xxyz.

If xx is equal to 0xFF, y - 0xA is the model number, and z is the stepping identifier.

If xx is not equal to 0xFF, xx + 'A' is the stepping letter and yz is the minor stepping.

ARM:

Reserved.

\*/

cout << " Architecture-dependent processor revision: 0x" << hex << localSystemInfo.wProcessorRevision << dec << "\n";

/\*if (localSystemInfo.dwProcessorType == PROCESSOR\_INTEL\_386 || localSystemInfo.dwProcessorType == PROCESSOR\_INTEL\_486)

{

if ((localSystemInfo.wProcessorRevision / 256) == 0xff)

{

cout << " Model number: " << (localSystemInfo.wProcessorRevision % 256) - (localSystemInfo.wProcessorRevision % 16) - 0xa << "\n";

cout << " Stepping identifier: " << (localSystemInfo.wProcessorRevision % 16) << "\n";

}

else

{

cout << " Stepping letter: " << (localSystemInfo.wProcessorRevision / 256) + 'A' << "\n";

cout << " Minor stepping: " << (localSystemInfo.wProcessorRevision % 256) + 'A' << "\n";

}

}

else if (localSystemInfo.dwProcessorType == PROCESSOR\_INTEL\_PENTIUM || localSystemInfo.dwProcessorType == PROCESSOR\_INTEL\_IA64 || localSystemInfo.dwProcessorType == PROCESSOR\_AMD\_X8664)

{

cout << " Model: " << (localSystemInfo.wProcessorRevision / 256) << "\n";

cout << " Stepping: " << (localSystemInfo.wProcessorRevision % 256) << "\n";

}

else

{}\*/

cout << "\n";

}

// ---------- 2 -- LOCAL GLOBAL MEMORY STATUS ----------

/\*

void GlobalMemoryStatus(

LPMEMORYSTATUS lpBuffer

);

typedef struct \_MEMORYSTATUS {

DWORD dwLength;

DWORD dwMemoryLoad;

SIZE\_T dwTotalPhys;

SIZE\_T dwAvailPhys;

SIZE\_T dwTotalPageFile;

SIZE\_T dwAvailPageFile;

SIZE\_T dwTotalVirtual;

SIZE\_T dwAvailVirtual;

} MEMORYSTATUS, \*LPMEMORYSTATUS;

\*/

void LocalGlobalMemoryStatus ()

{

MEMORYSTATUS localMemoryStatus; // creating structure

GlobalMemoryStatus(&localMemoryStatus); // sending the pointer and getting the information

cout << "Physical memory (RAM) information:\n"; // information output

// DWORD dwLength output

cout << " MEMORYSTATUS structure size (in bytes): " << localMemoryStatus.dwLength << "\n";

// DWORD dwMemoryLoad output

cout << " Approximate physical memory use (in %): " << localMemoryStatus.dwMemoryLoad << "\n";

// SIZE\_T dwTotalPhys output

cout << " Amount of physical memory (in bytes): " << localMemoryStatus.dwTotalPhys << "\n";

// SIZE\_T dwAvailPhys output

cout << " Avaliable physical memory (in bytes): " << localMemoryStatus.dwAvailPhys << "\n";

// SIZE\_T dwTotalPageFile output

cout << " Commited memory limit size, PM + page file - overhead (in bytes): " << localMemoryStatus.dwTotalPageFile << "\n";

// SIZE\_T dwAvailPageFile output

cout << " Max memory amount current process can commit (in bytes): " << localMemoryStatus.dwAvailPageFile << "\n";

// SIZE\_T dwTotalVirtual output

cout << " VAS's user-mode portion, who call processes, size (in bytes): " << localMemoryStatus.dwTotalVirtual << "\n";

// SIZE\_T dwAvailVirtual output

cout << " Unreserved & uncommitted VAS's user-mode portion size (in bytes): " << localMemoryStatus.dwAvailVirtual << "\n";

cout << "\n";

}

// ---------- 2 -- LOCAL GLOBAL MEMORY STATUS EX ----------

/\*

BOOL GlobalMemoryStatusEx(

LPMEMORYSTATUSEX lpBuffer

);

typedef struct \_MEMORYSTATUSEX {

DWORD dwLength;

DWORD dwMemoryLoad;

DWORDLONG ullTotalPhys;

DWORDLONG ullAvailPhys;

DWORDLONG ullTotalPageFile;

DWORDLONG ullAvailPageFile;

DWORDLONG ullTotalVirtual;

DWORDLONG ullAvailVirtual;

DWORDLONG ullAvailExtendedVirtual;

} MEMORYSTATUSEX, \*LPMEMORYSTATUSEX;

\*/

void LocalGlobalMemoryStatusEx ()

{

MEMORYSTATUSEX localMemoryStatusEx; // creating structure

localMemoryStatusEx.dwLength = sizeof (localMemoryStatusEx); // necessarily, without it it doesn't work!!!

bool localFlag = GlobalMemoryStatusEx(&localMemoryStatusEx); // sending the pointer and getting the information

// Physical memory refers to the actual RAM of the system

if (localFlag == true)

{

cout << "Physical memory (RAM) information:\n"; // information output

// DWORD dwLength output

cout << " MEMORYSTATUSEX struct size (in bytes): " << localMemoryStatusEx.dwLength << "\n";

// DWORD dwMemoryLoad output

cout << " Approximate physical memory use (in %): " << localMemoryStatusEx.dwMemoryLoad << "\n";

// DWORDLONG ullTotalPhys output

cout << " Amount of physical memory (in bytes): " << localMemoryStatusEx.ullTotalPhys << "\n";

// DWORDLONG ullAvailPhys output

cout << " Avaliable physical memory (in bytes): " << localMemoryStatusEx.ullAvailPhys << "\n";

// DWORDLONG ullTotalPageFile output

cout << " Commited memory limit size, PM + page file - overhead (in bytes): " << localMemoryStatusEx.ullTotalPageFile << "\n";

// DWORDLONG ullAvailPageFile output

cout << " Max memory amount current process can commit (in bytes): " << localMemoryStatusEx.ullAvailPageFile << "\n";

// DWORDLONG ullTotalVirtual output

cout << " VAS's user-mode portion, who call processes, size (in bytes): " << localMemoryStatusEx.ullTotalVirtual << "\n";

// DWORDLONG ullAvailVirtual output

cout << " Unreserved & uncommitted VAS's user-mode portion size (in bytes): " << localMemoryStatusEx.ullAvailVirtual << "\n";

// DWORDLONG ullAvailExtendedVirtual output

cout << " Reserved value (equals 0): " << localMemoryStatusEx.ullAvailExtendedVirtual << "\n";

}

else

{

cout << "Something went wrong! Last error code: " << GetLastError() << "\n";

}

cout << "\n";

}

// ---------- 3 -- LOCAL VIRTUAL QUERY ----------

/\*

SIZE\_T VirtualQuery(

LPCVOID lpAddress,

PMEMORY\_BASIC\_INFORMATION lpBuffer,

SIZE\_T dwLength

);

typedef struct \_MEMORY\_BASIC\_INFORMATION {

PVOID BaseAddress;

PVOID AllocationBase;

DWORD AllocationProtect;

WORD PartitionId;

SIZE\_T RegionSize;

DWORD State;

DWORD Protect;

DWORD Type;

} MEMORY\_BASIC\_INFORMATION, \*PMEMORY\_BASIC\_INFORMATION;

\*/

void LocalVirtualQuery ()

{

DWORD localAdress = 0x11376077;

//DWORD localAdress = -1; // creating adress variable

MEMORY\_BASIC\_INFORMATION localBuffer; // creating buffer for information write

SIZE\_T localLength; // creating size variable (for what?)

do

{

cout << "Please, input virtual adress space (in hex, 0x<hex number>): ";

cin >> hex >> localAdress >> dec;

} while (localAdress < 0x00000000 || localAdress > 0xffffffff);

// The return value is the actual number of bytes returned in the information buffer.

// If the function fails, the return value is zero. To get extended error information, call GetLastError. Possible error values include ERROR\_INVALID\_PARAMETER.

SIZE\_T localVirtualQuery = VirtualQuery ((LPVOID)localAdress, &localBuffer, sizeof(localBuffer));

// LPVOID -- pointer

// LPCVOID -- pointer to constant

// Physical memory refers to the actual RAM of the system

if (localVirtualQuery != 0)

{

cout << "Physical memory (RAM) information:\n"; // information output

// PVOID BaseAddress output

cout << " Pointer to the base address of the region of pages: " << localBuffer.BaseAddress << "\n";

// PVOID AllocationBase output

cout << " Pointer -- // -- allocated by the VirtualAlloc: " << localBuffer.AllocationBase << "\n";

// DWORD AllocationProtect output

cout << " Memory protection option (for initially allocation): " << localBuffer.AllocationProtect << "\n";

// WORD PartitionId output

//cout << " Partition ID (?): " << localBuffer.PartitionId << "\n"; // compiler can't recognize that

// SIZE\_T RegionSize output

cout << " Region's size from base address, pages identical attributes (in bytes): " << localBuffer.RegionSize << "\n";

// DWORD State output

if (localBuffer.State == MEM\_COMMIT) // number 0x1000

{

cout << " The state of the pages in the region: 0x" << hex << localBuffer.State << dec << " -- " << "Committed pages for which mem has been allocated\n";

}

else if (localBuffer.State == MEM\_FREE) // number 0x10000

{

cout << " The state of the pages in the region: 0x" << hex << localBuffer.State << dec << " -- " << "Free pages not for process, but for allocation\n";

}

else if (localBuffer.State == MEM\_RESERVE) // number 0x2000

{

cout << " The state of the pages in the region: 0x" << hex << localBuffer.State << dec << " -- " << "Reserved pages without allocation\n";

}

else // another number

{

cout << " The state of the pages in the region: 0x" << hex << localBuffer.State << dec << " -- " << "THIS NUMBER DOESN'T MEAN ANYTHING\n";

}

// DWORD Protect output

cout << " Access protection of the pages in the region: " << localBuffer.Protect << "\n";

// DWORD Type output

if (localBuffer.Type == MEM\_IMAGE) // number 0x1000000

{

cout << " The type of pages in the region: 0x" << hex << localBuffer.Type << dec << " -- " << "Memory pages -> image section\n";

}

else if (localBuffer.Type == MEM\_MAPPED) // number 0x40000

{

cout << " The type of pages in the region: 0x" << hex << localBuffer.Type << dec << " -- " << "Memory pages -> section\n";

}

else if (localBuffer.Type == MEM\_PRIVATE) // number 0x20000

{

cout << " The type of pages in the region: 0x" << hex << localBuffer.Type << dec << " -- " << "Memory pages -> private\n";

}

else // another number

{

cout << " The type of pages in the region: 0x" << hex << localBuffer.Type << dec << " -- " << "THIS NUMBER DOESN'T MEAN ANYTHING\n";

}

}

else

{

cout << "Something went wrong! Last error code: " << GetLastError() << "\n";

}

cout << "\n";

}

// ---------- 5 -- LIST OF ALLOCATIONS ----------

void LocalListOfAllocations ()

{

if (listOfAllocations.size() > 0)

{

//listOfAllocations.push\_back(tuple<LPVOID, SIZE\_T, DWORD, DWORD>((LPVOID)0x00000000, 4096, MEM\_RESET, MEM\_COMMIT)); // initialize example

//get<3>(listOfAllocations[0]) = MEM\_RESET; // change example

int j = 1;

for (LOCALLOC::const\_iterator i = listOfAllocations.begin(); i != listOfAllocations.end(); i++)

{

cout << "Number " << j << "\n";

cout << "Address (LPVOID):\t\t" << get<0>(\*i) << "\n";

cout << "Size of memory (SIZE\_T):\t" << get<1>(\*i) << "\n";

cout << "Allocation type (DWORD):\t" << hex << "0x" << get<2>(\*i) << dec << "\n";

cout << "Memory potection type (DWORD):\t" << hex << "0x" << get<3>(\*i) << dec << "\n\n";

j = j + 1;

}

}

else

{

cout << "Sorry, your HAVEN'T any region of pages in VAS! Allocate something first (choose from the main menu)!\n\n";

}

}

// ---------- 5 -- LIST OF ALLOCATIONS FREE ----------

void LocalListOfAllocationsFree ()

{

for (LOCALLOC::const\_iterator i = listOfAllocations.begin(); i != listOfAllocations.end(); i++)

{

listOfAllocations.erase(i); // erasing vector

}

}

// ---------- 5 -- LOCAL VIRTUAL ALLOC ----------

/\*

LPVOID VirtualAlloc(

[in, optional] LPVOID lpAddress,

[in] SIZE\_T dwSize,

[in] DWORD flAllocationType,

[in] DWORD flProtect

);

\*/

void LocalVirtualAlloc ()

{

DWORD localflAllocationType = 0;

DWORD localflProtect = 0;

//DWORD localAddress = -1; // creating adress variable

MEMORY\_BASIC\_INFORMATION localBuffer; // creating buffer for information write

SIZE\_T localLength; // creating size variable (for what?)

SIZE\_T localMemorySize = 4096;

char localHelp = '-';

string localChooseAllocation = "0";

string localChooseProtect = "0";

LPVOID locallpAddress = (LPVOID)0x11376077;

// The return value is the actual number of bytes returned in the information buffer.

// If the function fails, the return value is zero. To get extended error information, call GetLastError. Possible error values include ERROR\_INVALID\_PARAMETER.

//LPVOID localVirtualAlloc = VirtualAlloc (NULL, localMemorySize, MEM\_RESERVE, PAGE\_READWRITE);

// LPVOID -- pointer

// LPCVOID -- pointer to constant

localHelp = '-';

// requesting memory size request

while (localHelp != 'y' && localHelp != 'n')

{

cout << "Do you want input memory size request in BYTES or not? It's 4096 bytes by default. [y/n]\n";

cin >> localHelp;

}

// setting memory size request

if (localHelp == 'y')

{

do

{

cout << "Please, input memory size request (in bytes): ";

cin >> localMemorySize;

//cout << localMemorySize << "[memeory size request check]";

}

while (localMemorySize < 0);

}

localHelp = '-';

// requesting adress input type

while (localHelp != 'y' && localHelp != 'n')

{

cout << "Do you want input adress or not (automatically)? [y/n]\n";

cin >> localHelp;

}

// setting adress input type

if (localHelp == 'y')

{

do

{

cout << "Please, input virtual adress space (in hex, 0x<hex number>): ";

cin >> hex >> locallpAddress >> dec;

//cout << locallpAddress << "[adress check]";

}

while (locallpAddress < (LPVOID)0x00000000 || locallpAddress > (LPVOID)0xffffffff);

}

else

{

locallpAddress = NULL;

}

localHelp = '-';

// requesting help pages

while (localHelp != 'y' && localHelp != 'n')

{

cout << "Do you need the documentation about constants? [y/n]\n";

cin >> localHelp;

}

// help pages menu

if (localHelp == 'y')

{

localHelp = '-';

// requesting the documentation output

while (localHelp != 'y' && localHelp != 'n')

{

cout << "Do you need the documentation about constants? [y/n]\n";

cin >> localHelp;

}

// printing the documentation output

if (localHelp == 'y')

{

cout << "[in] flAllocationType:\n\n";

cout << "The type of memory allocation. This parameter must contain one of the following values.\n\n";

cout << "MEM\_COMMIT (0x00001000):\n\n"

<< "Allocates memory charges (from the overall size of memory and the paging files on disk) for the specified reserved memory pages.\n"

<< "The function also guarantees that when the caller later initially accesses the memory, the contents will be zero.\n"

<< "Actual physical pages are not allocated unless/until the virtual addresses are actually accessed.\n"

<< "To reserve and commit pages in one step, call VirtualAlloc with MEM\_COMMIT | MEM\_RESERVE.\n"

<< "Attempting to commit a specific address range by specifying MEM\_COMMIT without MEM\_RESERVE and a non-NULL lpAddress fails unless the entire range has already been reserved.\n"

<< "The resulting error code is ERROR\_INVALID\_ADDRESS.\n"

<< "An attempt to commit a page that is already committed does not cause the function to fail. This means that you can commit pages without first determining the current commitment state of each page.\n"

<< "If lpAddress specifies an address within an enclave, flAllocationType must be MEM\_COMMIT.\n\n";

cout << "MEM\_RESERVE (0x00002000):\n\n"

<< "Reserves a range of the process's virtual address space without allocating any actual physical storage in memory or in the paging file on disk.\n"

<< "You can commit reserved pages in subsequent calls to the VirtualAlloc function.\n"

<< "To reserve and commit pages in one step, call VirtualAlloc with MEM\_COMMIT | MEM\_RESERVE.\n"

<< "Other memory allocation functions, such as malloc and LocalAlloc, cannot use a reserved range of memory until it is released.\n\n";

cout << "MEM\_RESET (0x00080000):\n\n"

<< "Indicates that data in the memory range specified by lpAddress and dwSize is no longer of interest.\n"

<< "The pages should not be read from or written to the paging file.\n"

<< "However, the memory block will be used again later, so it should not be decommitted.\n"

<< "This value cannot be used with any other value.\n"

<< "Using this value does not guarantee that the range operated on with MEM\_RESET will contain zeros.\n"

<< "If you want the range to contain zeros, decommit the memory and then recommit it.\n"

<< "When you specify MEM\_RESET, the VirtualAlloc function ignores the value of flProtect.\n"

<< "However, you must still set flProtect to a valid protection value, such as PAGE\_NOACCESS.\n"

<< "VirtualAlloc returns an error if you use MEM\_RESET and the range of memory is mapped to a file.\n"

<< "A shared view is only acceptable if it is mapped to a paging file.\n\n";

cout << "MEM\_RESET\_UNDO (0x1000000):\n\n"

<< "MEM\_RESET\_UNDO should only be called on an address range to which MEM\_RESET was successfully applied earlier.\n"

<< "It indicates that the data in the specified memory range specified by lpAddress and dwSize is of interest to the caller and attempts to reverse the effects of MEM\_RESET.\n"

<< "If the function succeeds, that means all data in the specified address range is intact.\n"

<< "If the function fails, at least some of the data in the address range has been replaced with zeroes.\n"

<< "This value cannot be used with any other value.\n"

<< "If MEM\_RESET\_UNDO is called on an address range which was not MEM\_RESET earlier, the behavior is undefined.\n"

<< "When you specify MEM\_RESET, the VirtualAlloc function ignores the value of flProtect.\n"

<< "However, you must still set flProtect to a valid protection value, such as PAGE\_NOACCESS.\n"

<< "Windows Server 2008 R2, Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:\n"

<< "The MEM\_RESET\_UNDO flag is not supported until Windows 8 and Windows Server 2012.\n\n";

cout << "This parameter can also specify the following values as indicated.\n\n";

}

localHelp = '-';

// requesting the documentation output

while (localHelp != 'y' && localHelp != 'n')

{

cout << "Do you need the minimum size of a large page? [y/n]\n";

cin >> localHelp;

}

// printing the minimum size of a large page

if (localHelp == 'y')

{

cout << "The minimum size of a large page: " << GetLargePageMinimum() << "\n";

}

}

// choosing and setting the allocation type constant

while (localflAllocationType == 0)

{

cout << "Please, choose the allocation type (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):\n"

<< "[!!!] use MEM\_COMMIT to use physical memeory and MEM\_RESERVE for VAS reserve\n"

<< "1 -- MEM\_COMMIT (0x00001000)\n"

<< "2 -- MEM\_RESERVE (0x00002000)\n"

//<< "3 -- MEM\_RESET (0x00080000)\n" // if MEM\_RESET\_UNDO doesn't work, then MEM\_RESET usage is dangerous

//<< "4 -- MEM\_RESET\_UNDO (0x1000000)\n" // compiler declaration error

<< "5 -- MEM\_LARGE\_PAGES (0x20000000)\n"

<< "6 -- MEM\_PHYSICAL (0x00400000)\n"

<< "7 -- MEM\_TOP\_DOWN (0x00100000)\n";

//<< "8 -- MEM\_WRITE\_WATCH (0x00200000)\n"; // no GetWriteWatch and ResetWriteWatch functions in program

fflush(stdin);

std::getline(std::cin, localChooseAllocation);

// spit the string

std::string s = string(localChooseAllocation);

std::string delimiter = " ";

int i = 0;

size\_t pos = 0;

std::string token;

std::vector<string> v;

std::vector<int> vect{1, 2, 3, 4, 5, 6, 7, 8}; // all possible switch case numbers (DON'T FORGET WRITE THEM FROM MENU UP THERE)

while ((pos = s.find(delimiter)) != std::string::npos)

{

int tmpNumber = 0;

token = s.substr(0, pos);

v.push\_back(token);

tmpNumber = std::stoi(token);

if (std::find(vect.begin(), vect.end(), tmpNumber) != vect.end())

{

switch (tmpNumber) // choosing number

{

case 1:

localflAllocationType = localflAllocationType | MEM\_COMMIT;

break;

case 2:

localflAllocationType = localflAllocationType | MEM\_RESERVE;

break;

case 3:

//localflAllocationType = localflAllocationType | MEM\_RESET; // if MEM\_RESET\_UNDO doesn't work, then MEM\_RESET usage is dangerous

break;

case 4:

//localflAllocationType = localflAllocationType | MEM\_RESET\_UNDO; // compiler declaration error

break;

case 5:

localflAllocationType = localflAllocationType | MEM\_LARGE\_PAGES;

break;

case 6:

localflAllocationType = localflAllocationType | MEM\_PHYSICAL;

break;

case 7:

localflAllocationType = localflAllocationType | MEM\_TOP\_DOWN;

break;

case 8:

//localflAllocationType = localflAllocationType | MEM\_WRITE\_WATCH; // no GetWriteWatch and ResetWriteWatch functions in program

break;

default:

localflAllocationType = localflAllocationType | MEM\_RESERVE;

break;

}

vect.erase(std::remove(vect.begin(), vect.end(), tmpNumber), vect.end());

}

//std::cout << token << std::endl;

s.erase(0, pos + delimiter.length());

}

int newTMPNumber = std::stoi(s);

if (std::find(vect.begin(), vect.end(), newTMPNumber) != vect.end())

{

switch (newTMPNumber) // choosing number

{

case 1:

localflAllocationType = localflAllocationType | MEM\_COMMIT;

break;

case 2:

localflAllocationType = localflAllocationType | MEM\_RESERVE;

break;

case 3:

//localflAllocationType = localflAllocationType | MEM\_RESET; // if MEM\_RESET\_UNDO doesn't work, then MEM\_RESET usage is dangerous

break;

case 4:

//localflAllocationType = localflAllocationType | MEM\_RESET\_UNDO; // compiler declaration error

break;

case 5:

localflAllocationType = localflAllocationType | MEM\_LARGE\_PAGES;

break;

case 6:

localflAllocationType = localflAllocationType | MEM\_PHYSICAL;

break;

case 7:

localflAllocationType = localflAllocationType | MEM\_TOP\_DOWN;

break;

case 8:

//localflAllocationType = localflAllocationType | MEM\_WRITE\_WATCH; // no GetWriteWatch and ResetWriteWatch functions in program

break;

default:

localflAllocationType = localflAllocationType | MEM\_RESERVE;

break;

}

vect.erase(std::remove(vect.begin(), vect.end(), newTMPNumber), vect.end());

}

//std::cout << s << std::endl;

// end split of the string

if (localflAllocationType == 0)

{

cout << "Try again!\n";

}

}

// choosing and setting the memory protect constant

while (localflProtect == 0)

{

cout << "Please, choose the memory protect constant (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):\n"

<< "1 -- PAGE\_EXECUTE (0x10)\n"

<< "2 -- PAGE\_EXECUTE\_READ (0x20)\n"

<< "4 -- PAGE\_EXECUTE\_READWRITE (0x40)\n"

<< "4 -- PAGE\_EXECUTE\_WRITECOPY (0x80)\n"

<< "5 -- PAGE\_NOACCESS (0x01)\n"

<< "6 -- PAGE\_READONLY (0x02)\n"

<< "7 -- PAGE\_READWRITE (0x04)\n"

<< "8 -- PAGE\_WRITECOPY (0x08)\n"

//<< "9 -- PAGE\_TARGETS\_INVALID (0x40000000)\n" // compiler declaration error

//<< "10 -- PAGE\_TARGETS\_NO\_UPDATE (0x40000000)\n" // compiler declaration error

<< "11 -- PAGE\_GUARD (0x100)\n"

<< "12 -- PAGE\_NOCACHE (0x200)\n"

<< "13 -- PAGE\_WRITECOMBINE (0x400)\n";

fflush(stdin);

std::getline(std::cin, localChooseProtect);

// spit the string

std::string s = string(localChooseProtect);

std::string delimiter = " ";

int i = 0;

size\_t pos = 0;

std::string token;

std::vector<string> v;

std::vector<int> vect{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; // all possible switch case numbers (DON'T FORGET WRITE THEM FROM MENU UP THERE)

while ((pos = s.find(delimiter)) != std::string::npos)

{

int tmpNumber = 0;

token = s.substr(0, pos);

v.push\_back(token);

tmpNumber = std::stoi(token);

if (std::find(vect.begin(), vect.end(), tmpNumber) != vect.end())

{

switch (tmpNumber) // choosing number

{

case 1:

localflProtect = localflProtect | PAGE\_EXECUTE;

break;

case 2:

localflProtect = localflProtect | PAGE\_EXECUTE\_READ;

break;

case 3:

localflProtect = localflProtect | PAGE\_EXECUTE\_READWRITE;

break;

case 4:

localflProtect = localflProtect | PAGE\_EXECUTE\_WRITECOPY;

break;

case 5:

localflProtect = localflProtect | PAGE\_NOACCESS;

break;

case 6:

localflProtect = localflProtect | PAGE\_READONLY;

break;

case 7:

localflProtect = localflProtect | PAGE\_READWRITE;

break;

case 8:

localflProtect = localflProtect | PAGE\_WRITECOPY;

break;

case 9:

//localflProtect = localflProtect | PAGE\_TARGETS\_INVALID; // compiler declaration error

break;

case 10:

//localflProtect = localflProtect | PAGE\_TARGETS\_NO\_UPDATE; // compiler declaration error

break;

case 11:

localflProtect = localflProtect | PAGE\_GUARD;

break;

case 12:

localflProtect = localflProtect | PAGE\_NOCACHE;

break;

case 13:

localflProtect = localflProtect | PAGE\_WRITECOMBINE;

break;

default:

localflProtect = localflProtect | PAGE\_READWRITE;

break;

}

vect.erase(std::remove(vect.begin(), vect.end(), tmpNumber), vect.end());

}

//std::cout << token << std::endl;

s.erase(0, pos + delimiter.length());

}

int newTMPNumber = std::stoi(s);

if (std::find(vect.begin(), vect.end(), newTMPNumber) != vect.end())

{

switch (newTMPNumber) // choosing number

{

case 1:

localflProtect = localflProtect | PAGE\_EXECUTE;

break;

case 2:

localflProtect = localflProtect | PAGE\_EXECUTE\_READ;

break;

case 3:

localflProtect = localflProtect | PAGE\_EXECUTE\_READWRITE;

break;

case 4:

localflProtect = localflProtect | PAGE\_EXECUTE\_WRITECOPY;

break;

case 5:

localflProtect = localflProtect | PAGE\_NOACCESS;

break;

case 6:

localflProtect = localflProtect | PAGE\_READONLY;

break;

case 7:

localflProtect = localflProtect | PAGE\_READWRITE;

break;

case 8:

localflProtect = localflProtect | PAGE\_WRITECOPY;

break;

case 9:

//localflProtect = localflProtect | PAGE\_TARGETS\_INVALID; // compiler declaration error

break;

case 10:

//localflProtect = localflProtect | PAGE\_TARGETS\_NO\_UPDATE; // compiler declaration error

break;

case 11:

localflProtect = localflProtect | PAGE\_GUARD;

break;

case 12:

localflProtect = localflProtect | PAGE\_NOCACHE;

break;

case 13:

localflProtect = localflProtect | PAGE\_WRITECOMBINE;

break;

default:

localflProtect = localflProtect | PAGE\_READWRITE;

break;

}

vect.erase(std::remove(vect.begin(), vect.end(), newTMPNumber), vect.end());

}

//std::cout << s << std::endl;

// end split of the string

if (localflProtect == 0)

{

cout << "Try again!\n";

}

}

LPVOID localVirtualAlloc = VirtualAlloc (locallpAddress, localMemorySize, localflAllocationType, localflProtect);

if (localVirtualAlloc != NULL)

{

cout << "Allocation was successfull\n" << localVirtualAlloc << "\n";

// putting my values

localHelp = '-';

// requesting data change

while (localHelp != 'y' && localHelp != 'n')

{

cout << "Do you want to change some data in region of pages in VAS? [y/n]\n";

cin >> localHelp;

}

// data change

if (localHelp == 'y')

{

LocalDataChangeCore (localVirtualAlloc, localMemorySize);

}

// freeing memory

localHelp = '-';

// requesting freeing memory

while (localHelp != 'y' && localHelp != 'n')

{

cout << "Do you want to free memory in VAS? [y/n]\n";

cin >> localHelp;

}

// freeing memory

if (localHelp == 'y') // if free -- then freeing and checking it

{

LocalVirtualFreeCore(localVirtualAlloc, localMemorySize);

/\*if (VirtualFree (localVirtualAlloc, 0, MEM\_RELEASE))

{

cout << "Free was successfull\n";

}

else

{

cout << "Free was NOT successfull. The last error code: " << GetLastError() << "\n";

}\*/

}

else // if no -- put in in the list, i mean vector

{

listOfAllocations.push\_back(tuple<LPVOID, SIZE\_T, DWORD, DWORD>(localVirtualAlloc, localMemorySize, localflAllocationType, localflProtect));

}

}

else

{

cout << "Allocation was NOT successfull. The last error code: " << GetLastError() << "\n";

}

cout << "\n";

}

// ---------- 6 -- LOCAL DATA CHANGE CORE ----------

void LocalDataChangeCore (LPVOID localVirtualAlloc, SIZE\_T localMemorySize)

{

cout << "Your adress space is from (including) " << localVirtualAlloc << " to (including) " << localVirtualAlloc + localMemorySize - 1 << "\n";

char localRepeatMain = 'y'; // repeating all the checking

while (localRepeatMain == 'y')

{

bool localRepeat = true; // repeating input

int localStartingType = 1; // type choose for input

int localEndingType = 1; // type choose for output

SIZE\_T localStartingSize = 0; // memory size for output

SIZE\_T localEndingSize = 0; // memory size for output

LPVOID localStartingAddress = localVirtualAlloc; // starting address for input

LPVOID localEndingAddress = localVirtualAlloc; // starting address for output

// all possible types of types initializing

// input

bool\* localBool;

char\* localChar;

wchar\_t\* localWCharT;

char16\_t\* localChar16T;

char32\_t\* localChar32T;

short\* localShort;

int\* localInt;

long\* localLong;

long long\* localLongLong;

float\* localFloat;

double\* localDouble;

long double\* localLongDouble;

// output

bool\* localBoolOut;

char\* localCharOut;

wchar\_t\* localWCharTOut;

char16\_t\* localChar16TOut;

char32\_t\* localChar32TOut;

short\* localShortOut;

int\* localIntOut;

long\* localLongOut;

long long\* localLongLongOut;

float\* localFloatOut;

double\* localDoubleOut;

long double\* localLongDoubleOut;

// all possible types of types choosing

cout << "1 -- bool:\t\t" << sizeof(bool) << " bytes\n";

cout << "2 -- char:\t\t" << sizeof(char) << " bytes\n";

cout << "3 -- wchar\_t:\t\t" << sizeof(wchar\_t) << " bytes\n";

cout << "4 -- char16\_t:\t\t" << sizeof(char16\_t) << " bytes\n";

cout << "5 -- char32\_t:\t\t" << sizeof(char32\_t) << " bytes\n";

cout << "6 -- short:\t\t" << sizeof(short) << " bytes\n";

cout << "7 -- int:\t\t" << sizeof(int) << " bytes\n";

cout << "8 -- long:\t\t" << sizeof(long) << " bytes\n";

cout << "9 -- long long:\t\t" << sizeof(long long) << " bytes\n";

cout << "10 -- float:\t\t" << sizeof(float) << " bytes\n";

cout << "11 -- double:\t\t" << sizeof(double) << " bytes\n";

cout << "12 -- long double:\t" << sizeof(long double) << " bytes\n\n";

localRepeat = true; // if i will run this code again (UPT: THIS IS BUG, FIXED)

while (localRepeat == true)

{

// input and output adress and type choosing

cout << "Please, choose the starting adress (0x<hex number>): ";

cin >> hex >> localStartingAddress >> dec;

cout << "Please, choose the input type: ";

cin >> localStartingType;

cout << "Please, choose the starting adress: ";

cin >> hex >> localEndingAddress >> dec;

cout << "Please, choose the output type: ";

cin >> localEndingType;

switch (localStartingType) // starting input address size qualification

{

case 1:

localStartingSize = sizeof(bool);

break;

case 2:

localStartingSize = sizeof(char);

break;

case 3:

localStartingSize = sizeof(wchar\_t);

break;

case 4:

localStartingSize = sizeof(char16\_t);

break;

case 5:

localStartingSize = sizeof(char32\_t);

break;

case 6:

localStartingSize = sizeof(short);

break;

case 7:

localStartingSize = sizeof(int);

break;

case 8:

localStartingSize = sizeof(long);

break;

case 9:

localStartingSize = sizeof(long long);

break;

case 10:

localStartingSize = sizeof(float);

break;

case 11:

localStartingSize = sizeof(double);

break;

case 12:

localStartingSize = sizeof(long double);

break;

default:

localStartingSize = sizeof(bool);

break;

}

switch (localEndingType) // starting output size address qualification

{

case 1:

localEndingSize = sizeof(bool);

break;

case 2:

localEndingSize = sizeof(char);

break;

case 3:

localEndingSize = sizeof(wchar\_t);

break;

case 4:

localEndingSize = sizeof(char16\_t);

break;

case 5:

localEndingSize = sizeof(char32\_t);

break;

case 6:

localEndingSize = sizeof(short);

break;

case 7:

localEndingSize = sizeof(int);

break;

case 8:

localEndingSize = sizeof(long);

break;

case 9:

localEndingSize = sizeof(long long);

break;

case 10:

localEndingSize = sizeof(float);

break;

case 11:

localEndingSize = sizeof(double);

break;

case 12:

localEndingSize = sizeof(long double);

break;

default:

localEndingSize = sizeof(bool);

break;

}

if (localStartingAddress < localVirtualAlloc || localEndingAddress < localVirtualAlloc)

{

cout << "Adress is out (is less) of possible allocated range, please, try again!\n";

}

else if (localStartingSize + localStartingAddress - 1 > localVirtualAlloc + localMemorySize - 1

|| localEndingSize + localEndingAddress - 1 > localVirtualAlloc + localMemorySize - 1)

{

cout << "Address with/without memory is out (is more) of possible allocated range, please, try again!\n";

}

else

{

localRepeat = false; // if there is no errors, the program will run

}

}

// checking all values AND SETTING ADDRESSES

cout << "Checking current values before something:\n";

cout << "Input";

switch (localStartingType)

{

case 1:

localBool = (bool\*)localStartingAddress;

cout << " (bool): ";

cout << \*localBool;

break;

case 2:

localChar = (char\*)localStartingAddress;

cout << " (char): ";

cout << \*localChar;

break;

case 3:

localWCharT = (wchar\_t\*)localStartingAddress;

cout << " (wchar\_t): ";

cout << \*localWCharT;

break;

case 4:

localChar16T = (char16\_t\*)localStartingAddress;

cout << " (char16\_t): ";

cout << \*localChar16T;

break;

case 5:

localChar32T = (char32\_t\*)localStartingAddress;

cout << " (char32\_t): ";

cout << \*localChar32T;

break;

case 6:

localShort = (short\*)localStartingAddress;

cout << " (short): ";

cout << \*localShort;

break;

case 7:

localInt = (int\*)localStartingAddress;

cout << " (int): ";

cout << \*localInt;

break;

case 8:

localLong = (long\*)localStartingAddress;

cout << " (long): ";

cout << \*localLong;

break;

case 9:

localLongLong = (long long\*)localStartingAddress;

cout << " (long long): ";

cout << \*localLongLong;

break;

case 10:

localFloat = (float\*)localStartingAddress;

cout << " (float): ";

cout << \*localFloat;

break;

case 11:

localDouble = (double\*)localStartingAddress;

cout << " (double): ";

cout << \*localDouble;

break;

case 12:

localLongDouble = (long double\*)localStartingAddress;

cout << " (long double): ";

cout << \*localLongDouble;

break;

default:

localBool = (bool\*)localStartingAddress;

cout << " (bool): ";

cout << \*localBool;

break;

}

cout << "\n";

cout << "Output";

switch (localEndingType)

{

case 1:

localBoolOut = (bool\*)localEndingAddress;

cout << " (bool): ";

cout << \*localBoolOut;

break;

case 2:

localCharOut = (char\*)localEndingAddress;

cout << " (char): ";

cout << \*localCharOut;

break;

case 3:

localWCharTOut = (wchar\_t\*)localEndingAddress;

cout << " (wchar\_t): ";

cout << \*localWCharTOut;

break;

case 4:

localChar16TOut = (char16\_t\*)localEndingAddress;

cout << " (char16\_t): ";

cout << \*localChar16TOut;

break;

case 5:

localChar32TOut = (char32\_t\*)localEndingAddress;

cout << " (char32\_t): ";

cout << \*localChar32TOut;

break;

case 6:

localShortOut = (short\*)localEndingAddress;

cout << " (short): ";

cout << \*localShortOut;

break;

case 7:

localIntOut = (int\*)localEndingAddress;

cout << " (int): ";

cout << \*localIntOut;

break;

case 8:

localLongOut = (long\*)localEndingAddress;

cout << " (long): ";

cout << \*localLongOut;

break;

case 9:

localLongLongOut = (long long\*)localEndingAddress;

cout << " (long long): ";

cout << \*localLongLongOut;

break;

case 10:

localFloatOut = (float\*)localEndingAddress;

cout << " (float): ";

cout << \*localFloatOut;

break;

case 11:

localDoubleOut = (double\*)localEndingAddress;

cout << " (double): ";

cout << \*localDoubleOut;

break;

case 12:

localLongDoubleOut = (long double\*)localEndingAddress;

cout << " (long double): ";

cout << \*localLongDoubleOut;

break;

default:

localBoolOut = (bool\*)localEndingAddress;

cout << " (bool): ";

cout << \*localBoolOut;

break;

}

cout << "\n";

// setting right values for the types and size of the types

cout << "Please, input your value into the variable of choosen type";

switch (localStartingType)

{

case 1:

cout << " (bool): ";

\*localBool = BoolSafetyInput();

break;

case 2:

cout << " (char): ";

cin >> \*localChar;

break;

case 3:

cout << " (wchar\_t): ";

//cin >> \*localWCharT;

break;

case 4:

cout << " (char16\_t): ";

//cin >> \*localChar16T;

break;

case 5:

cout << " (char32\_t): ";

//cin >> \*localChar32T;

break;

case 6:

cout << " (short): ";

cin >> \*localShort;

break;

case 7:

cout << " (int): ";

cin >> \*localInt;

break;

case 8:

cout << " (long): ";

cin >> \*localLong;

break;

case 9:

cout << " (long long): ";

cin >> \*localLongLong;

break;

case 10:

cout << " (float): ";

cin >> \*localFloat;

break;

case 11:

cout << " (double): ";

cin >> \*localDouble;

break;

case 12:

cout << " (long double): ";

cin >> \*localLongDouble;

break;

default:

cout << " (bool): ";

\*localBool = BoolSafetyInput();

break;

}

// getting values from chosed types

cout << "Output the value from variable of choosen type";

switch (localEndingType)

{

case 1:

cout << " (bool): ";

cout << \*localBoolOut;

break;

case 2:

cout << " (char): ";

cout << \*localCharOut;

break;

case 3:

cout << " (wchar\_t): ";

cout << \*localWCharTOut;

break;

case 4:

cout << " (char16\_t): ";

cout << \*localChar16TOut;

break;

case 5:

cout << " (char32\_t): ";

cout << \*localChar32TOut;

break;

case 6:

cout << " (short): ";

cout << \*localShortOut;

break;

case 7:

cout << " (int): ";

cout << \*localIntOut;

break;

case 8:

cout << " (long): ";

cout << \*localLongOut;

break;

case 9:

cout << " (long long): ";

cout << \*localLongLongOut;

break;

case 10:

cout << " (float): ";

cout << \*localFloatOut;

break;

case 11:

cout << " (double): ";

cout << \*localDoubleOut;

break;

case 12:

cout << " (long double): ";

cout << \*localLongDoubleOut;

break;

default:

cout << " (bool): ";

cout << \*localBoolOut;

break;

}

cout << "\n";

localRepeat = false;

cout << "Try again? [y -- yes (your values you put will remain) / n -- no]\n";

cin >> localRepeatMain;

}

cout << "\n";

}

// ---------- 6 -- LOCAL DATA CHANGE INDEPENDENT ----------

void LocalDataChangeIndependent ()

{

bool vp = false; // at the beginning function isn't completed yet

int localChoose = 1; // default

LPVOID locallpAddress = (LPVOID)0x11376077;

SIZE\_T localdwSize = 4096;

DWORD localOldAllocationType = 0;

DWORD localOldProtect = 0; // is from list

PDWORD locallpflOldProtect = NULL; // old protection pointer

DWORD localflNewProtect = 0; // new protection

char localHelp = '-';

if (listOfAllocations.size() > 0) // if our list has something check

{

LocalListOfAllocations (); // output all possible region of pages in VAS

do

{

cout << "Please, choose the number of the region of pages in VAS: ";

cin >> localChoose;

}

while (localChoose < 1 || localChoose > listOfAllocations.size());

locallpAddress = get<0>(listOfAllocations[localChoose - 1]);

localdwSize = get<1>(listOfAllocations[localChoose - 1]);

localOldAllocationType = get<2>(listOfAllocations[localChoose - 1]);

localOldProtect = get<3>(listOfAllocations[localChoose - 1]);

cout << "THE CHOOSEN region of pages in VAS with is " << locallpAddress << " with size " << localdwSize

<< " bytes\nwith allocation type 0x" << hex << localOldAllocationType << " and memory constant 0x" << localOldProtect << dec << "\n"

<< "Commit changes? [y/n]\n";

cin >> localHelp;

LocalDataChangeCore(locallpAddress, localdwSize);

}

else

{

LocalListOfAllocations ();

}

}

// ---------- 7 -- LOCAL VIRTUAL PROTECT ----------

/\*

BOOL VirtualProtect(

[in] LPVOID lpAddress,

[in] SIZE\_T dwSize,

[in] DWORD flNewProtect,

[out] PDWORD lpflOldProtect

);

\*/

void LocalVirtualProtect ()

{

bool vp = false; // at the beginning function isn't completed yet

int localChoose = 1; // default

LPVOID locallpAddress = (LPVOID)0x11376077;

SIZE\_T localdwSize = 4096;

DWORD localOldAllocationType = 0;

DWORD localOldProtect = 0; // is from list

DWORD locallpflOldProtect; // old protection pointer (actually, it must be PDWODR)

DWORD localflNewProtect = 0; // new protection

char localHelp = '-';

string localChooseAllocation = "0";

string localChooseProtect = "0";

if (listOfAllocations.size() > 0) // if our list has something check

{

LocalListOfAllocations (); // output all possible region of pages in VAS

do

{

cout << "Please, choose the number of the region of pages in VAS: ";

cin >> localChoose;

}

while (localChoose < 1 || localChoose > listOfAllocations.size());

locallpAddress = get<0>(listOfAllocations[localChoose - 1]);

localdwSize = get<1>(listOfAllocations[localChoose - 1]);

localOldAllocationType = get<2>(listOfAllocations[localChoose - 1]);

localOldProtect = get<3>(listOfAllocations[localChoose - 1]);

cout << "THE CHOOSEN region of pages in VAS with is " << locallpAddress << " with size " << localdwSize

<< " bytes\nwith allocation type 0x" << hex << localOldAllocationType << " and memory constant 0x" << localOldProtect << dec << "\n"

<< "Commit changes? [y/n]\n";

cin >> localHelp;

// choosing and setting the NEW memory protect constant

while (localflNewProtect == 0)

{

cout << "Please, choose the memory protect constant (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):\n"

<< "1 -- PAGE\_EXECUTE (0x10)\n"

<< "2 -- PAGE\_EXECUTE\_READ (0x20)\n"

<< "3 -- PAGE\_EXECUTE\_READWRITE (0x40)\n"

<< "4 -- PAGE\_EXECUTE\_WRITECOPY (0x80)\n"

<< "5 -- PAGE\_NOACCESS (0x01)\n"

<< "6 -- PAGE\_READONLY (0x02)\n"

<< "7 -- PAGE\_READWRITE (0x04)\n"

<< "8 -- PAGE\_WRITECOPY (0x08)\n"

//<< "9 -- PAGE\_TARGETS\_INVALID (0x40000000)\n" // compiler declaration error

//<< "10 -- PAGE\_TARGETS\_NO\_UPDATE (0x40000000)\n" // compiler declaration error

<< "11 -- PAGE\_GUARD (0x100)\n"

<< "12 -- PAGE\_NOCACHE (0x200)\n"

<< "13 -- PAGE\_WRITECOMBINE (0x400)\n";

fflush(stdin);

std::getline(std::cin, localChooseProtect);

// spit the string

std::string s = string(localChooseProtect);

std::string delimiter = " ";

int i = 0;

size\_t pos = 0;

std::string token;

std::vector<string> v;

std::vector<int> vect{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; // all possible switch case numbers (DON'T FORGET WRITE THEM FROM MENU UP THERE)

while ((pos = s.find(delimiter)) != std::string::npos)

{

int tmpNumber = 0;

token = s.substr(0, pos);

v.push\_back(token);

tmpNumber = std::stoi(token);

if (std::find(vect.begin(), vect.end(), tmpNumber) != vect.end())

{

switch (tmpNumber) // choosing number

{

case 1:

localflNewProtect = localflNewProtect | PAGE\_EXECUTE;

break;

case 2:

localflNewProtect = localflNewProtect | PAGE\_EXECUTE\_READ;

break;

case 3:

localflNewProtect = localflNewProtect | PAGE\_EXECUTE\_READWRITE;

break;

case 4:

localflNewProtect = localflNewProtect | PAGE\_EXECUTE\_WRITECOPY;

break;

case 5:

localflNewProtect = localflNewProtect | PAGE\_NOACCESS;

break;

case 6:

localflNewProtect = localflNewProtect | PAGE\_READONLY;

break;

case 7:

localflNewProtect = localflNewProtect | PAGE\_READWRITE;

break;

case 8:

localflNewProtect = localflNewProtect | PAGE\_WRITECOPY;

break;

case 9:

//localflNewProtect = localflNewProtect | PAGE\_TARGETS\_INVALID; // compiler declaration error

break;

case 10:

//localflNewProtect = localflNewProtect | PAGE\_TARGETS\_NO\_UPDATE; // compiler declaration error

break;

case 11:

localflNewProtect = localflNewProtect | PAGE\_GUARD;

break;

case 12:

localflNewProtect = localflNewProtect | PAGE\_NOCACHE;

break;

case 13:

localflNewProtect = localflNewProtect | PAGE\_WRITECOMBINE;

break;

default:

localflNewProtect = localflNewProtect | PAGE\_READWRITE;

break;

}

vect.erase(std::remove(vect.begin(), vect.end(), tmpNumber), vect.end());

}

//std::cout << token << std::endl;

s.erase(0, pos + delimiter.length());

}

int newTMPNumber = std::stoi(s);

if (std::find(vect.begin(), vect.end(), newTMPNumber) != vect.end())

{

switch (newTMPNumber) // choosing number

{

case 1:

localflNewProtect = localflNewProtect | PAGE\_EXECUTE;

break;

case 2:

localflNewProtect = localflNewProtect | PAGE\_EXECUTE\_READ;

break;

case 3:

localflNewProtect = localflNewProtect | PAGE\_EXECUTE\_READWRITE;

break;

case 4:

localflNewProtect = localflNewProtect | PAGE\_EXECUTE\_WRITECOPY;

break;

case 5:

localflNewProtect = localflNewProtect | PAGE\_NOACCESS;

break;

case 6:

localflNewProtect = localflNewProtect | PAGE\_READONLY;

break;

case 7:

localflNewProtect = localflNewProtect | PAGE\_READWRITE;

break;

case 8:

localflNewProtect = localflNewProtect | PAGE\_WRITECOPY;

break;

case 9:

//localflNewProtect = localflNewProtect | PAGE\_TARGETS\_INVALID; // compiler declaration error

break;

case 10:

//localflNewProtect = localflNewProtect | PAGE\_TARGETS\_NO\_UPDATE; // compiler declaration error

break;

case 11:

localflNewProtect = localflNewProtect | PAGE\_GUARD;

break;

case 12:

localflNewProtect = localflNewProtect | PAGE\_NOCACHE;

break;

case 13:

localflNewProtect = localflNewProtect | PAGE\_WRITECOMBINE;

break;

default:

localflNewProtect = localflNewProtect | PAGE\_READWRITE;

break;

}

vect.erase(std::remove(vect.begin(), vect.end(), newTMPNumber), vect.end());

}

//std::cout << s << std::endl;

// end split of the string

if (localflNewProtect == 0)

{

cout << "Try again!\n";

}

}

// making function

vp = VirtualProtect(locallpAddress, localdwSize, localflNewProtect, &locallpflOldProtect);

// the result checking

if (vp == true)

{

cout << "The memory protection constant in " << locallpAddress << " address with size " << localdwSize

<< " bytes\nHAS BEEN successfully changed from 0x" << hex << locallpflOldProtect << " to 0x" << localflNewProtect << dec << "\n";

}

else

{

cout << "SORRY! The memory protection constant in " << locallpAddress << " address with size " << localdwSize

<< " bytes\nHASN'T BEEN successfully changed from 0x" << hex << locallpflOldProtect << " to 0x" << localflNewProtect << dec

<< "\n" << "The last error code is " << GetLastError() << "\n";

}

cout << "\n";

}

else

{

LocalListOfAllocations ();

}

}

// ---------- 8 -- LOCAL VIRTUAL FREE CORE ----------

void LocalVirtualFreeCore (LPVOID locallpAddress, SIZE\_T localdwSize)

{

bool vf = false; // at the beginning function isn't completed yet

DWORD localFree = 0;

char localRepeat = 'n'; // for start

char localHelp = '-';

string localChooseAllocation = "0";

string localChooseProtect = "0";

// choosing and setting the NEW memory protect constant

while (localFree == 0)

{

cout << "Please, choose the memory free option (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):\n"

<< "1 -- MEM\_DECOMMIT (0x00004000)\n"

<< "2 -- MEM\_RELEASE -- THE MAIN OPTION (0x00008000)\n";

//<< "3 -- MEM\_COALESCE\_PLACEHOLDERS (0x00000001)\n"

//<< "4 -- MEM\_PRESERVE\_PLACEHOLDER (0x00000002)\n";

fflush(stdin);

std::getline(std::cin, localChooseProtect);

// spit the string

std::string s = string(localChooseProtect);

std::string delimiter = " ";

int i = 0;

size\_t pos = 0;

std::string token;

std::vector<string> v;

std::vector<int> vect{1, 2, 3, 4}; // all possible switch case numbers (DON'T FORGET WRITE THEM FROM MENU UP THERE)

while ((pos = s.find(delimiter)) != std::string::npos)

{

int tmpNumber = 0;

token = s.substr(0, pos);

v.push\_back(token);

tmpNumber = std::stoi(token);

if (std::find(vect.begin(), vect.end(), tmpNumber) != vect.end())

{

switch (tmpNumber) // choosing number

{

case 1:

localFree = localFree | MEM\_DECOMMIT;

break;

case 2:

localFree = localFree | MEM\_RELEASE;

break;

case 3:

//localFree = localFree | MEM\_COALESCE\_PLACEHOLDERS; // compiler erroe

break;

case 4:

//localFree = localFree | MEM\_PRESERVE\_PLACEHOLDER; // compiler error

break;

default:

localFree = localFree | MEM\_RELEASE;

break;

}

vect.erase(std::remove(vect.begin(), vect.end(), tmpNumber), vect.end());

}

//std::cout << token << std::endl;

s.erase(0, pos + delimiter.length());

}

int newTMPNumber = std::stoi(s);

if (std::find(vect.begin(), vect.end(), newTMPNumber) != vect.end())

{

switch (newTMPNumber) // choosing number

{

case 1:

localFree = localFree | MEM\_DECOMMIT;

break;

case 2:

localFree = localFree | MEM\_RELEASE;

break;

case 3:

//localFree = localFree | MEM\_COALESCE\_PLACEHOLDERS; // compiler error

break;

case 4:

//localFree = localFree | MEM\_PRESERVE\_PLACEHOLDER; // compiler error

break;

default:

localFree = localFree | MEM\_RELEASE;

break;

}

vect.erase(std::remove(vect.begin(), vect.end(), newTMPNumber), vect.end());

}

//std::cout << s << std::endl;

// end split of the string

if (localFree == 0)

{

cout << "Try again!\n";

}

}

// making function

if ((localFree & MEM\_RELEASE) != 0) // BUG DETECTED: ((<> & <>) != <>) works, but (<> & <> != <>) DOESN'T

{

localdwSize = 0;

}

vf = VirtualFree(locallpAddress, localdwSize, localFree);

// the result checking

if (vf == true)

{

cout << "The page in " << locallpAddress << " address with size " << localdwSize

<< " bytes\nHAS BEEN successfully freed with free type 0x" << hex << localFree << dec << "\n";

}

else

{

cout << "SORRY! The page in " << locallpAddress << " address with size " << localdwSize

<< " bytes\nHASN'T BEEN successfully freed with free type 0x" << hex << localFree << dec

<< "\n" << "The last error code is " << GetLastError() << "\n";

}

}

// ---------- 8 -- LOCAL VIRTUAL FREE INDEPENDENT ----------

/\*

BOOL VirtualFree(

[in] LPVOID lpAddress,

[in] SIZE\_T dwSize,

[in] DWORD dwFreeType

);

\*/

void LocalVirtualFreeIndependent ()

{

bool vf = false; // at the beginning function isn't completed yet

int localChoose = 1; // default

LPVOID locallpAddress = (LPVOID)0x11376077;

SIZE\_T localdwSize = 4096;

DWORD localOldAllocationType = 0;

DWORD localOldProtect = 0; // is from list

PDWORD locallpflOldProtect = NULL; // old protection pointer

DWORD localflNewProtect = 0; // new protection

DWORD localFree = 0;

char localHelp = '-';

string localChooseAllocation = "0";

string localChooseProtect = "0";

if (listOfAllocations.size() > 0) // if our list has something check

{

LocalListOfAllocations (); // output all possible region of pages in VAS

do

{

cout << "Please, choose the number of the region of pages in VAS: ";

cin >> localChoose;

}

while (localChoose < 1 || localChoose > listOfAllocations.size());

locallpAddress = get<0>(listOfAllocations[localChoose - 1]);

localdwSize = get<1>(listOfAllocations[localChoose - 1]);

localOldAllocationType = get<2>(listOfAllocations[localChoose - 1]);

localOldProtect = get<3>(listOfAllocations[localChoose - 1]);

cout << "THE CHOOSEN region of pages in VAS with is " << locallpAddress << " with size " << localdwSize

<< " bytes\nwith allocation type 0x" << hex << localOldAllocationType << " and memory constant 0x" << localOldProtect << dec << "\n"

<< "Commit changes? [y/n]\n";

cin >> localHelp;

// choosing and setting the NEW memory protect constant

while (localFree == 0)

{

cout << "Please, choose the memory free option (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):\n"

<< "1 -- MEM\_DECOMMIT (0x00004000)\n"

<< "2 -- MEM\_RELEASE -- THE MAIN OPTION (0x00008000)\n";

//<< "3 -- MEM\_COALESCE\_PLACEHOLDERS (0x00000001)\n"

//<< "4 -- MEM\_PRESERVE\_PLACEHOLDER (0x00000002)\n";

fflush(stdin);

std::getline(std::cin, localChooseProtect);

// spit the string

std::string s = string(localChooseProtect);

std::string delimiter = " ";

int i = 0;

size\_t pos = 0;

std::string token;

std::vector<string> v;

std::vector<int> vect{1, 2, 3, 4}; // all possible switch case numbers (DON'T FORGET WRITE THEM FROM MENU UP THERE)

while ((pos = s.find(delimiter)) != std::string::npos)

{

int tmpNumber = 0;

token = s.substr(0, pos);

v.push\_back(token);

tmpNumber = std::stoi(token);

if (std::find(vect.begin(), vect.end(), tmpNumber) != vect.end())

{

switch (tmpNumber) // choosing number

{

case 1:

localFree = localFree | MEM\_DECOMMIT;

break;

case 2:

localFree = localFree | MEM\_RELEASE;

break;

case 3:

//localFree = localFree | MEM\_COALESCE\_PLACEHOLDERS; // compiler error

break;

case 4:

//localFree = localFree | MEM\_PRESERVE\_PLACEHOLDER; // compiler error

break;

default:

localFree = localFree | MEM\_RELEASE;

break;

}

vect.erase(std::remove(vect.begin(), vect.end(), tmpNumber), vect.end());

}

//std::cout << token << std::endl;

s.erase(0, pos + delimiter.length());

}

int newTMPNumber = std::stoi(s);

if (std::find(vect.begin(), vect.end(), newTMPNumber) != vect.end())

{

switch (newTMPNumber) // choosing number

{

case 1:

localFree = localFree | MEM\_DECOMMIT;

break;

case 2:

localFree = localFree | MEM\_RELEASE;

break;

case 3:

//localFree = localFree | MEM\_COALESCE\_PLACEHOLDERS; // compiler error

break;

case 4:

//localFree = localFree | MEM\_PRESERVE\_PLACEHOLDER; // compiler error

break;

default:

localFree = localFree | MEM\_RELEASE;

break;

}

vect.erase(std::remove(vect.begin(), vect.end(), newTMPNumber), vect.end());

}

//std::cout << s << std::endl;

// end split of the string

if (localFree == 0)

{

cout << "Try again!\n";

}

}

// making function

if ((localFree & MEM\_RELEASE) != 0) // BUG DETECTED: ((<> & <>) != <>) works, but (<> & <> != <>) DOESN'T

{

localdwSize = 0;

}

vf = VirtualFree(locallpAddress, localdwSize, localFree);

// the result checking

if (vf == true)

{

cout << "The page in " << locallpAddress << " address with size " << localdwSize

<< " bytes\nHAS BEEN successfully freed with free type 0x" << hex << localFree << dec << "\n";

listOfAllocations.erase(listOfAllocations.begin() + localChoose - 1); // erasing vector

}

else

{

cout << "SORRY! The page in " << locallpAddress << " address with size " << localdwSize

<< " bytes\nHASN'T BEEN successfully freed with free type 0x" << hex << localFree << dec

<< "\n" << "The last error code is " << GetLastError() << "\n";

}

cout << "\n";

}

else

{

LocalListOfAllocations ();

}

}

## 2.10. Выводы

В ходе выполнения первой части («Исследование виртуального адресного пространства процессов») лабораторной работы №2 «Управление памятью» были изучены основные функции управления памятью в системе Windows. Во-первых, было реализовано консольное приложение, которое давало возможность посмотреть информацию о системе, о виртуальной памяти и о её конкретном участке. Во-вторых, были созданы возможности выделения памяти (в автоматическом и ручном режимах) и её возврата, резервирования адресов и смены уровня доступа по ним. Также были добавлены возможности изменения данных по заданному адресу и просмотра всех выделенных пользователем адресов. Таким образом и было исследовано виртуальное адресное пространство.

# 3. Использование проецируемых файлов для обмена данными между процессами

## 3.1. Создание проецируемого файла приложением-писателем

В приложении-писателе создаётся (если файл с заданным именем уже существует, то будет ошибка) файл для записи и делается проецируемый файл. Далее осуществляется проецирование файла в память. Затем осуществляется запись заданного количества байт с заданным форматом вывода данных, запись осуществляется с помощью случайных значений для упрощения процесса заполнения памяти.

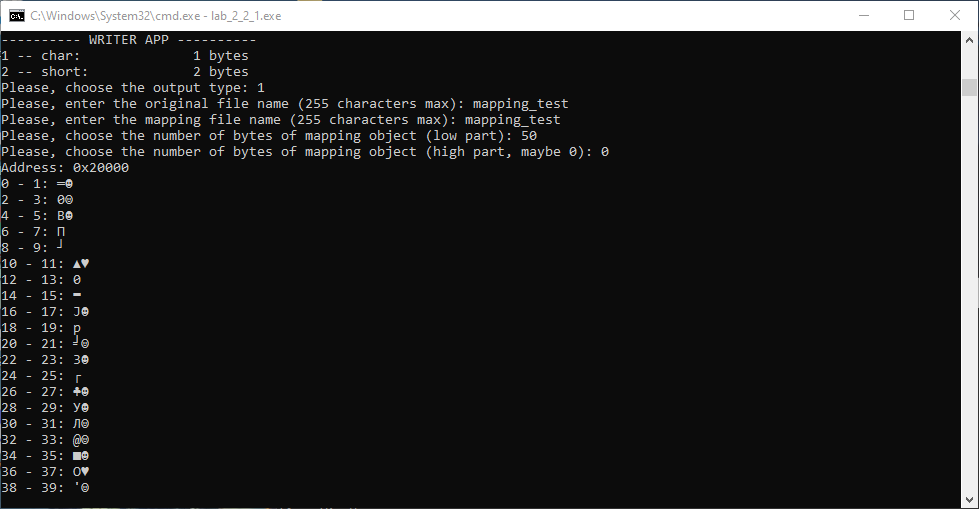


Рисунок 26: Создание проецируемого файла 1 с символьным выводом

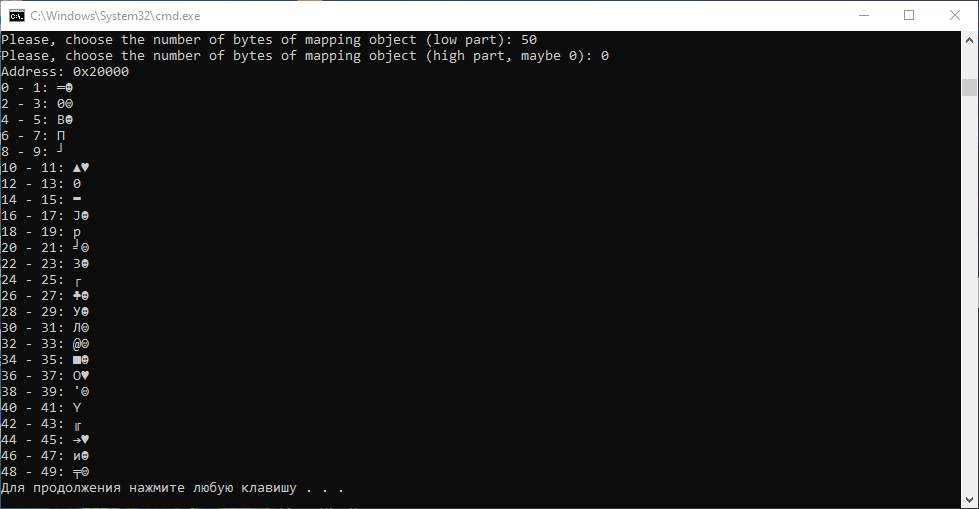


Рисунок 27: Создание проецируемого файла 1 с символьным выводом

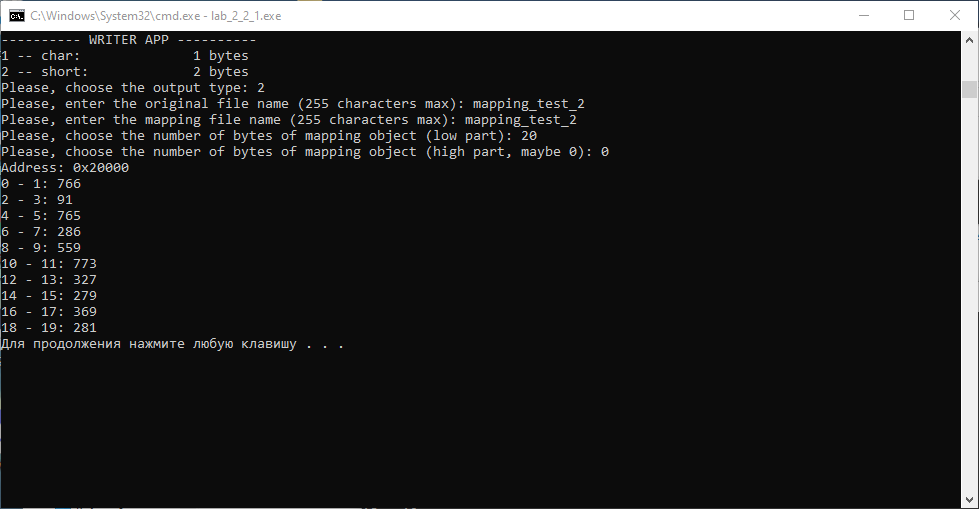


Рисунок : Создание проецируемого файла 2 с численным выводом

## 3.2. Открытие проецируемого файла приложением-читателем

В приложении-читателе открывается проецируемый файл (с тем же именем, что и был создан) для чтения. Далее осуществляется проецирование файла в память. Затем осуществляется чтение заданного количества байт с заданным форматом вывода данных, которые можно сравнить с данными, полученными при записи в приложении-писателе.

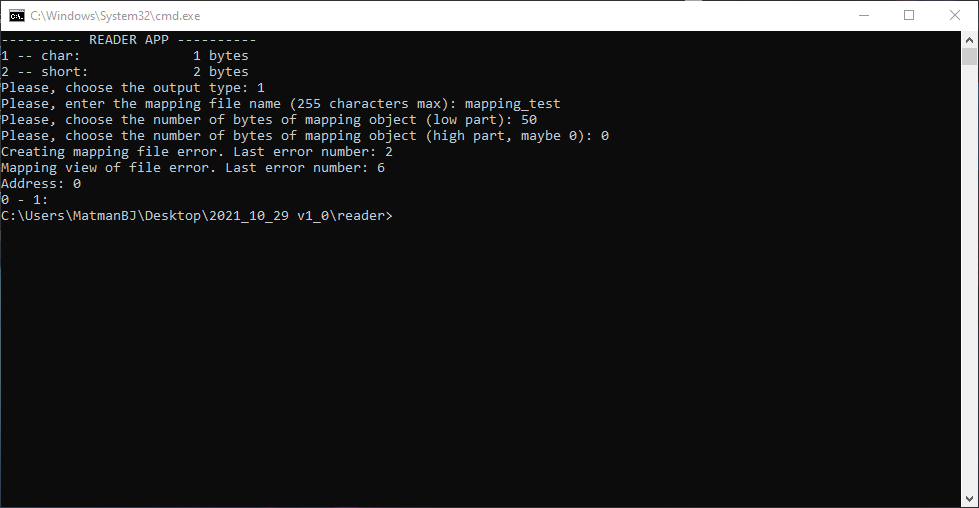


Рисунок 29: Открытие проецируемого файла 1 до его создания

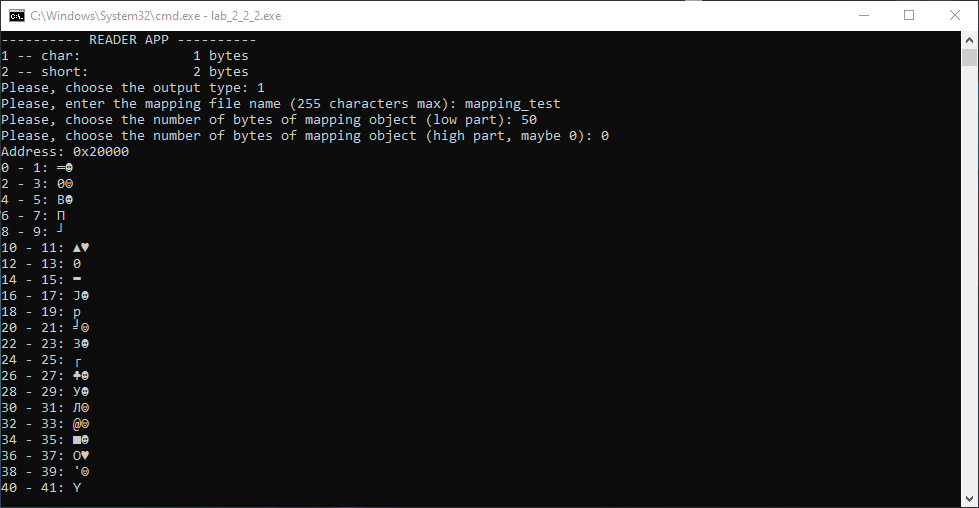


Рисунок 30: Открытие проецируемого файла 1 с символьным выводом

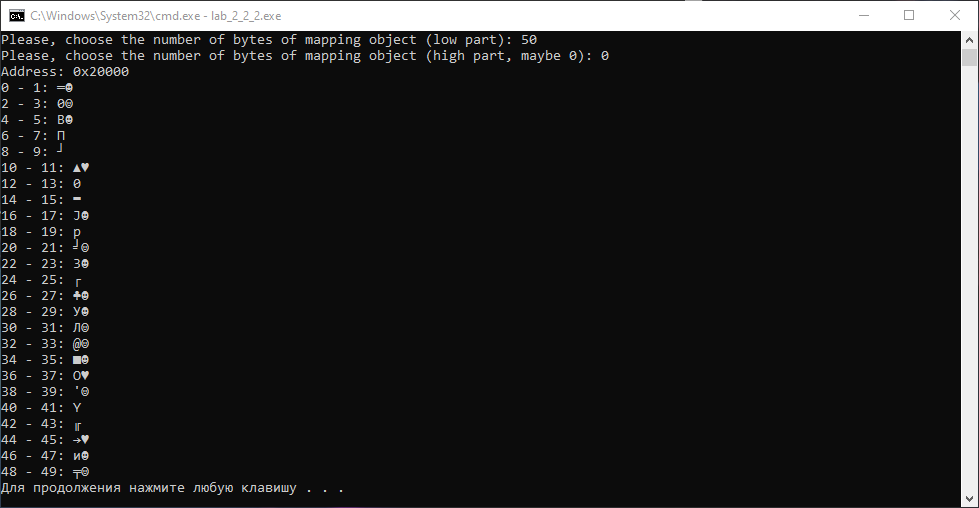


Рисунок : Открытие проецируемого файла 1 с символьным выводом

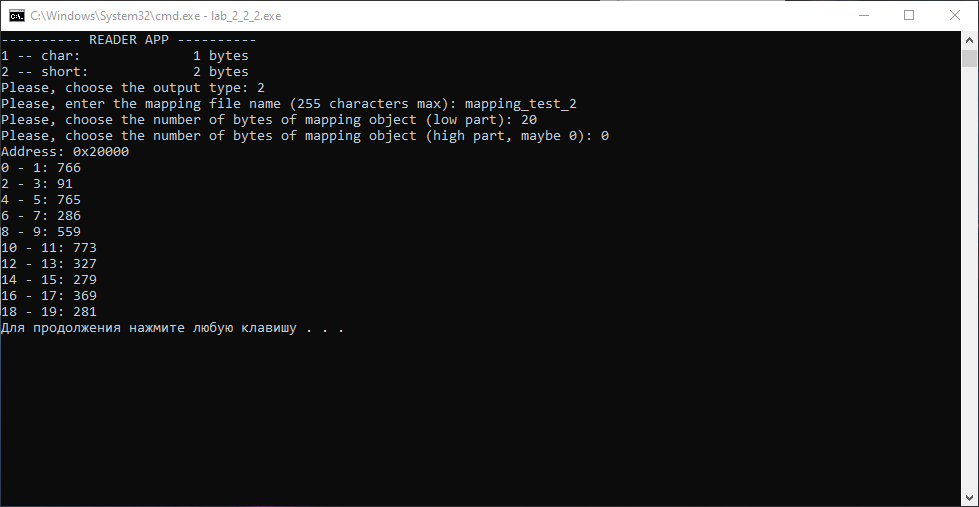


Рисунок : Открытие проецируемого файла 2 с численным выводом

## 3.3. Исходный код программы-писателя

/\*

Win32 API (WinAPI) is a set of functions in the library <windows.h>

API means "Application Programming Interface"

\*/

#include <windows.h> // for WinAPI functions

#include <iostream> // just for working

#include <ctime> // for randomization

#include <string> // for the "string" type using

using namespace std;

// ---------- MAIN ----------

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

{

cout << "---------- WRITER APP ----------\n";

int i; // loop variable

int localEndingType = 1; // type choose for output

// all possible types of types choosing

cout << "1 -- char:\t\t" << sizeof(char) << " bytes\n";

cout << "2 -- short:\t\t" << sizeof(short) << " bytes\n";

int localStartingType = 1; // type choose for input

LARGE\_INTEGER localBytes; // default

string createFileName = "localFileOne";

string mappingFileName = "localFileTwo";

// input and output adress and type choosing

do

{

cout << "Please, choose the output type: ";

cin >> localStartingType;

}

while (localStartingType < 1 || localStartingType > 2);

do

{

cout << "Please, enter the original file name (255 characters max): ";

cin >> createFileName;

}

while (createFileName.length() > 255);

do

{

cout << "Please, enter the mapping file name (255 characters max): ";

cin >> mappingFileName;

}

while (mappingFileName.length() > 255);

cout << "Please, choose the number of bytes of mapping object (low part): ";

cin >> localBytes.LowPart;

cout << "Please, choose the number of bytes of mapping object (high part, maybe 0): ";

cin >> localBytes.HighPart;

switch (localStartingType) // starting output type

{

case 1:

localStartingType = sizeof(char);

break;

case 2:

localStartingType = sizeof(short);

break;

default:

localStartingType = sizeof(char);

break;

}

srand((unsigned)time(0));

HANDLE createdFile = CreateFile(

createFileName.c\_str(), // filename

GENERIC\_READ | GENERIC\_WRITE, // desired access [all usage]

FILE\_SHARE\_DELETE | FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, // share mode [all usage]

NULL, // security attributes [NULL has been chosen because of current unusability of the other ones]

CREATE\_NEW, // creating/open files [CREATE\_NEW is safety for other files]

FILE\_ATTRIBUTE\_NORMAL, // flags and attributes [normal mode has been chosen because of current unusabilty of the other ones]

NULL // templete file handle [NULL has been chosen because of current unusability of the other ones]

);

if (createdFile == INVALID\_HANDLE\_VALUE)

{

cout << "Creating file error. Last error number: " << GetLastError() << "\n";

}

HANDLE mappingFileOld = CreateFileMapping(

createdFile, // handle of the created file [choosed the new one]

NULL, // mapping attributes [NULL has been chosen because of current unusability of the other ones]

PAGE\_READWRITE, // page protection type [PAGE\_READWRITE has been chosen because it' most convinient]

localBytes.HighPart, // high order DWORD (second part)

localBytes.LowPart, // low order DWORD (first part) [0 if the low-part means mapping file size = created file size]

mappingFileName.c\_str() // mapping filename [filename]

);

if (mappingFileOld == NULL || GetLastError() == ERROR\_ALREADY\_EXISTS)

{

cout << "Creating mapping file error. Last error number: " << GetLastError() << "\n";

}

LPVOID mappingOld = MapViewOfFile(

mappingFileOld, // handle of the mapping file

FILE\_MAP\_WRITE, // special desired access flag [FILE\_MAP\_WRITE is compatiable with PAGE\_READWRITE]

0, // OffsetHigh

0, // OffsetLow

0 // bytes to map [0 means all]

);

if (mappingOld == NULL)

{

cout << "Mapping view of file error. Last error number: " << GetLastError() << "\n";

}

cout << "Address: " << mappingOld << "\n";

// randomization and output loop

for (i = 0; i + 1 < localBytes.QuadPart; i = i + 2)

{

\*(short\*)(mappingOld + i) = (rand() % 6)\*2 + (rand() % 855);

if (localStartingType == 2)

{

cout << i << " - " << i + 1 << ": ";

cout << \*(short\*)(mappingOld + i) << "\n";

}

else

{

cout << i << " - " << i + 1 << ": ";

cout << \*(char\*)(mappingOld + i) << \*(char\*)(mappingOld + i + 0x1) << "\n";

}

}

UnmapViewOfFile(mappingOld); // unmapping current file

CloseHandle(createdFile); // closing handle of original file, NOT THE MAPPING FILE, DON'T DO THAT

//CloseHandle(mappingFileOld);

system("pause");

return 0;

}

## 3.4. Исходный код программы-читателя

/\*

Win32 API (WinAPI) is a set of functions in the library <windows.h>

API means "Application Programming Interface"

\*/

#include <windows.h> // for WinAPI functions

#include <iostream> // just for working

#include <string> // for the "string" type using

using namespace std;

// ---------- MAIN ----------

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

{

cout << "---------- READER APP ----------\n";

int i; // loop variable

int localEndingType = 1; // type choose for output

// all possible types of types initializing

LARGE\_INTEGER localBytes; // default

string mappingFileName = "localFileTwo";

// all possible types of types choosing

cout << "1 -- char:\t\t" << sizeof(char) << " bytes\n";

cout << "2 -- short:\t\t" << sizeof(short) << " bytes\n";

// input and output adress and type choosing

do

{

cout << "Please, choose the output type: ";

cin >> localEndingType;

}

while (localEndingType < 1 || localEndingType > 2);

do

{

cout << "Please, enter the mapping file name (255 characters max): ";

cin >> mappingFileName;

}

while (mappingFileName.length() > 255);

cout << "Please, choose the number of bytes of mapping object (low part): ";

cin >> localBytes.LowPart;

cout << "Please, choose the number of bytes of mapping object (high part, maybe 0): ";

cin >> localBytes.HighPart;

switch (localEndingType) // starting output type

{

case 1:

localEndingType = sizeof(char);

break;

case 2:

localEndingType = sizeof(short);

break;

default:

localEndingType = sizeof(char);

break;

}

bool localRepeat = true; // repeating input

SIZE\_T localStartingSize = 0; // memory size for output

SIZE\_T localEndingSize = 0; // memory size for output

HANDLE mappingFileNew = OpenFileMapping(

FILE\_MAP\_WRITE, // special desired access flag [FILE\_MAP\_WRITE is compatiable with PAGE\_READWRITE]

false, // inheritance mechanics -- unuseful here

mappingFileName.c\_str() // unique name of the mapping file

);

if (mappingFileNew == NULL)

{

cout << "Creating mapping file error. Last error number: " << GetLastError() << "\n";

}

// ACCESS GRANTED BY MAPPING FILENAME, SO IT MUST BE THE SAME

LPVOID mappingNew = MapViewOfFile(

mappingFileNew, // handle of the mapping file

FILE\_MAP\_WRITE, // special desired access flag [FILE\_MAP\_WRITE is compatiable with PAGE\_READWRITE]

0, // OffsetHigh

0, // OffsetLow

0 // bytes to map [0 means all]

);

if (mappingNew == NULL)

{

cout << "Mapping view of file error. Last error number: " << GetLastError() << "\n";

}

cout << "Address: " << mappingNew << "\n";

//MEMORY\_BASIC\_INFORMATION localMBI = {0};

//VirtualQueryEx(GetCurrentProcess(), mappingNew, &localMBI, sizeof(localMBI));

//out << "TRY IT: " << localMBI.RegionSize;

// output loop

// IT COVERS ALL MEMORY CLUSTERS FOR CHAR AND SHORT -- I CHECKED

for (i = 0; i + 1 < localBytes.QuadPart; i = i + 2)

{

if (localEndingType == 2)

{

cout << i << " - " << i + 1 << ": ";

cout << \*(short\*)(mappingNew + i) << "\n";

}

else

{

cout << i << " - " << i + 1 << ": ";

cout << \*(char\*)(mappingNew + i) << \*(char\*)(mappingNew + i + 0x1) << "\n";

}

}

system("pause");

return 0;

}

## 3.5. Выводы

В ходе выполнения второй части («Использование проецируемых файлов для обмена данными между процессами») лабораторной работы №2 «Управление памятью» было изучено взаимодействие с проецируемыми файлами. Было создано два приложения: приложение-писатель создавало проецируемый файл и заполняло его данными, приложение-читатель открывало проецируемый файл и считывало из него данные. При попытке открыть несуществующий файл возникала ошибка, а при удачном открытии файла информация, считываемая из него, была идентична исходным сгенерированным данным. Таким образом и было исследованы проецируемые файлы.

# 4. Список использованных источников

1. Операционные системы: электронные методические указания к лабораторным работам / Сост.: А. В. Тимофеев. СПб.: Изд-во СПбГЭТУ

«ЛЭТИ», 2016.

2. Таненбаум Э. Современные операционные системы. 2-е изд. – СПб.: Питер, 2002. – 1040 с.: ил.

3. Курс «Операционные системы» в образовательной онлайн-системе Google Класс [сайт]. URL: <https://classroom.google.com/c/Mzg3ODc4NDE5MDU4>.