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

ОТЧЁТ

по лабораторной работе №2

по дисциплине «Операционные системы»

Тема: Управление памятью

Студент гр.2307 _____ Подберёзский А.Д.

Преподаватель _____ Тимофеев А.В.

Санкт-Петербург,
2024

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().

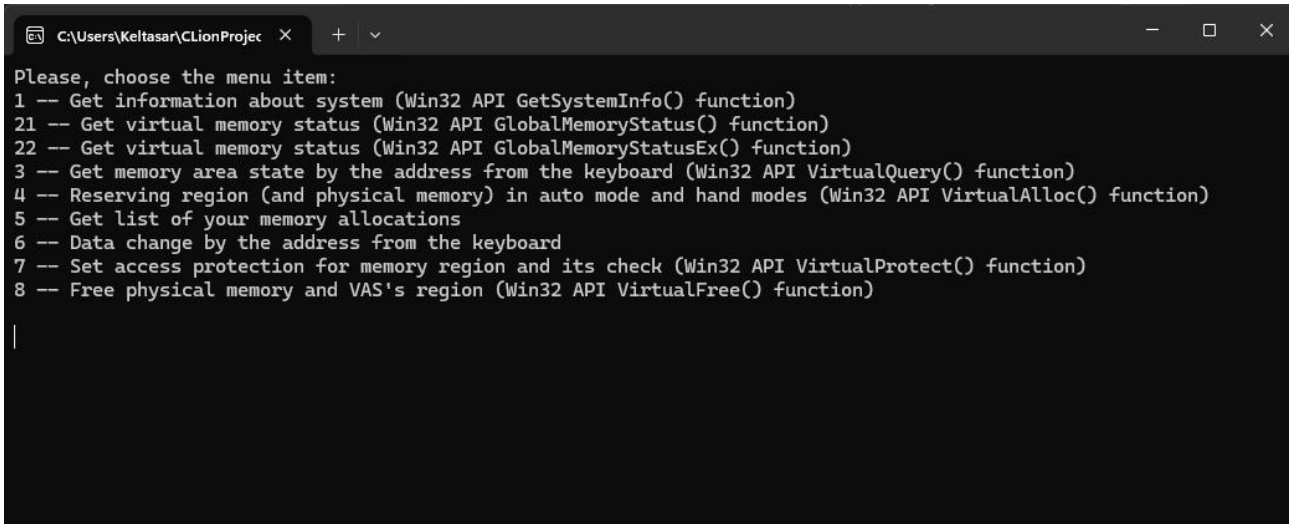

Рисунок 1: Главное меню программы

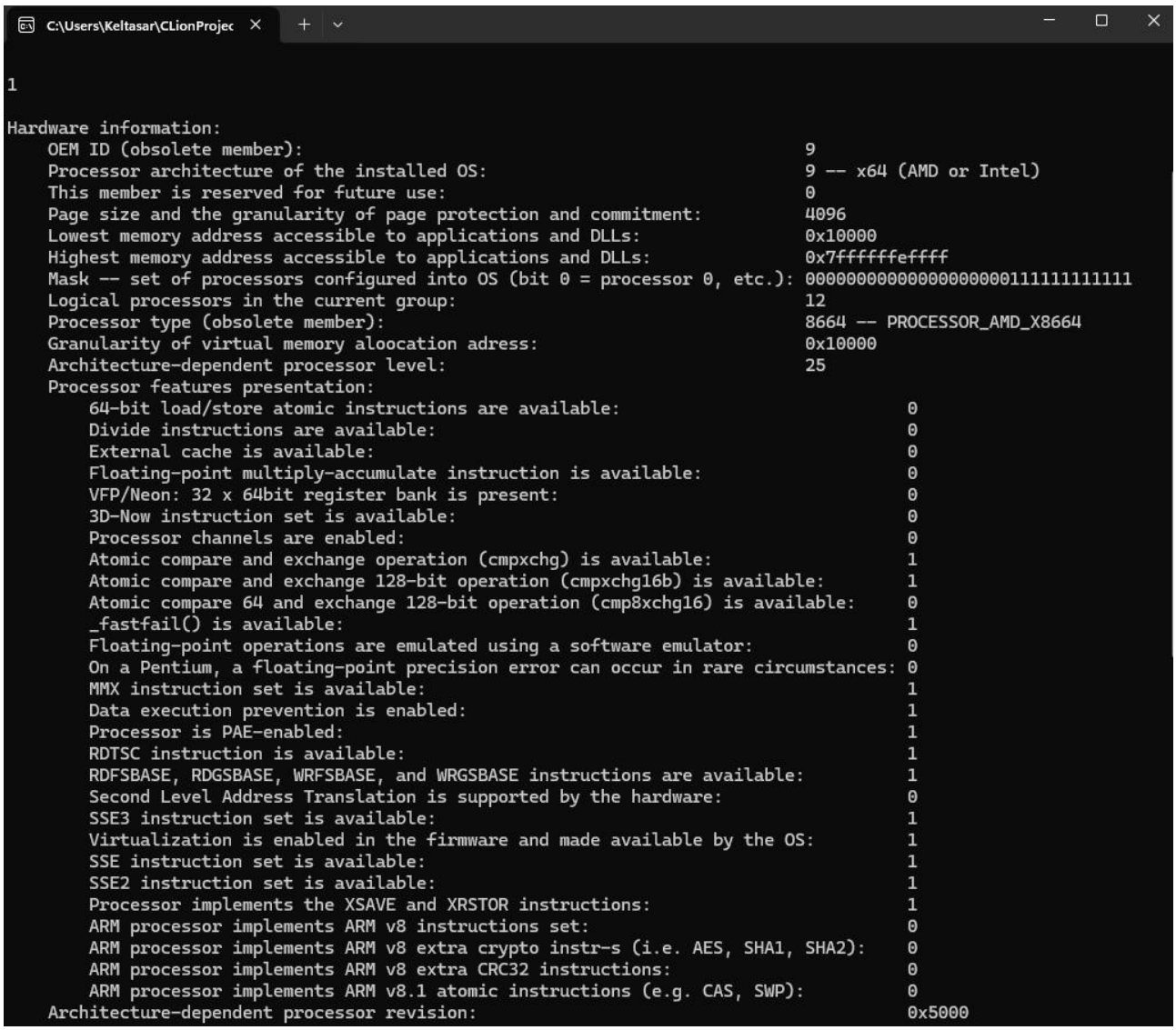

Рисунок 2: Вывод информации о системе

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.

```
21

Physical memory (RAM) information:
    MEMORYSTATUS structure size (in bytes): 56
    Approximate physical memory use (in %): 87
    Amount of physical memory (in bytes):   16477040640
    Avaliable physical memory (in bytes):   2138140672
    Commited memory limit size, PM + page file - overhead (in bytes): 31243255808
    Max memory amount current process can commit (in bytes):          10989948928
    VAS's user-mode portion, who call processes, size (in bytes):     140737488224256
    Unreserved & uncommitted VAS's user-mode portion size (in bytes): 140733134766080
```

Рисунок 3: Вывод статуса виртуальной памяти

```
22

Physical memory (RAM) information:
    MEMORYSTATUSEX struct size (in bytes):  64
    Approximate physical memory use (in %): 86
    Amount of physical memory (in bytes):   16477040640
    Avaliable physical memory (in bytes):   2205351936
    Commited memory limit size, PM + page file - overhead (in bytes): 31243255808
    Max memory amount current process can commit (in bytes):          11018346496
    VAS's user-mode portion, who call processes, size (in bytes):     140737488224256
    Unreserved & uncommitted VAS's user-mode portion size (in bytes): 140733141057536
    Reserved value (equals 0): 0
```

Рисунок 4: 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().

```
3

Please, input virtual adress space (in hex, 0x<hex number>): 0xffff0128
Physical memory (RAM) information:
    Pointer to the base address of the region of pages:  0xffff0000
    Pointer -- // -- allocated by the VirtualAlloc:      0
    Memory protection option (for initially allocation): 0
    Region's size from base address, pages identical attributes (in bytes): 847077507072
    The state of the pages in the region:                0x10000 -- Free pages not for process, but for allocation
    Access protection of the pages in the region:        1
    The type of pages in the region:                     0x0 -- THIS NUMBER DOESN'T MEAN ANYTHING
```

Рисунок 5: 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.

```
4

Do you want input memory size request in BYTES or not? It's 4096 bytes by default. [y/n]
n
Do you want input adress or not (automatically)? [y/n]
n
Do you need the documentation about constants? [y/n]
n
Please, choose the allocation type (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):
[!!!] use MEM_COMMIT to use physical memeory and MEM_RESERVE for VAS reserve
1 -- MEM_COMMIT (0x00001000)
2 -- MEM_RESERVE (0x00002000)
5 -- MEM_LARGE_PAGES (0x20000000)
6 -- MEM_PHYSICAL (0x00400000)
7 -- MEM_TOP_DOWN (0x00100000)
2
Please, choose the memory protect constant (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):
1 -- PAGE_EXECUTE (0x10)
2 -- PAGE_EXECUTE_READ (0x20)
4 -- PAGE_EXECUTE_READWRITE (0x40)
4 -- PAGE_EXECUTE_WRITECOPY (0x80)
5 -- PAGE_NOACCESS (0x01)
6 -- PAGE_READONLY (0x02)
7 -- PAGE_READWRITE (0x04)
8 -- PAGE_WRITECOPY (0x08)
11 -- PAGE_GUARD (0x100)
12 -- PAGE_NOCACHE (0x200)
13 -- PAGE_WRITECOMBINE (0x400)
7
Allocation was successfull
0x2599fbc0000
```

Рисунок 6: Резервирование региона в автоматическом режиме

```
Do you want to change some data in region of pages in VAS? [y/n]
n
Do you want to free memory in VAS? [y/n]
y
Please, choose the memory free option (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):
1 -- MEM_DECOMMIT (0x00004000)
2 -- MEM_RELEASE -- THE MAIN OPTION (0x00008000)
2
The page in 0x2599fbc0000 address with size 0 bytes
HAS BEEN successfully freed with free type 0x8000
```

Рисунок 7: Резервирование региона в автоматическом режиме

```
4

Do you want input memory size request in BYTES or not? It's 4096 bytes by default. [y/n]
y
Please, input memory size request (in bytes): 16384
Do you want input adress or not (automatically)? [y/n]
y
Please, input virtual adress space (in hex, 0x<hex number>): 0xffff0128
Do you need the documentation about constants? [y/n]
n
Please, choose the allocation type (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):
[!!!] use MEM_COMMIT to use physical memeory and MEM_RESERVE for VAS reserve
1 -- MEM_COMMIT (0x00001000)
2 -- MEM_RESERVE (0x00002000)
5 -- MEM_LARGE_PAGES (0x20000000)
6 -- MEM_PHYSICAL (0x00400000)
7 -- MEM_TOP_DOWN (0x00100000)
2
Please, choose the memory protect constant (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):
1 -- PAGE_EXECUTE (0x10)
2 -- PAGE_EXECUTE_READ (0x20)
4 -- PAGE_EXECUTE_READWRITE (0x40)
4 -- PAGE_EXECUTE_WRITECOPY (0x80)
5 -- PAGE_NOACCESS (0x01)
6 -- PAGE_READONLY (0x02)
7 -- PAGE_READWRITE (0x04)
8 -- PAGE_WRITECOPY (0x08)
11 -- PAGE_GUARD (0x100)
12 -- PAGE_NOCACHE (0x200)
13 -- PAGE_WRITECOMBINE (0x400)
7
Allocation was successfull
0xffff0000
```

Рисунок 8: Резервирование региона в ручном режиме

```
Do you want to change some data in region of pages in VAS? [y/n]
n
Do you want to free memory in VAS? [y/n]
y
Please, choose the memory free option (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):
1 -- MEM_DECOMMIT (0x00004000)
2 -- MEM_RELEASE -- THE MAIN OPTION (0x00008000)
2
The page in 0xffff0000 address with size 0 bytes
HAS BEEN successfully freed with free type 0x8000
```

Рисунок 9: Резервирование региона в ручном режиме

```
4

Do you want input memory size request in BYTES or not? It's 4096 bytes by default. [y/n]
n
Do you want input adress or not (automatically)? [y/n]
n
Do you need the documentation about constants? [y/n]
n
Please, choose the allocation type (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):
[!!!] use MEM_COMMIT to use physical memeory and MEM_RESERVE for VAS reserve
1 -- MEM_COMMIT (0x00001000)
2 -- MEM_RESERVE (0x00002000)
5 -- MEM_LARGE_PAGES (0x20000000)
6 -- MEM_PHYSICAL (0x00400000)
7 -- MEM_TOP_DOWN (0x00100000)
1 2
Please, choose the memory protect constant (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):
1 -- PAGE_EXECUTE (0x10)
2 -- PAGE_EXECUTE_READ (0x20)
4 -- PAGE_EXECUTE_READWRITE (0x40)
4 -- PAGE_EXECUTE_WRITECOPY (0x80)
5 -- PAGE_NOACCESS (0x01)
6 -- PAGE_READONLY (0x02)
7 -- PAGE_READWRITE (0x04)
8 -- PAGE_WRITECOPY (0x08)
11 -- PAGE_GUARD (0x100)
12 -- PAGE_NOCACHE (0x200)
13 -- PAGE_WRITECOMBINE (0x400)
7
```

Рисунок 10: Резервирование региона и передача ему физической памяти в автоматическом режиме

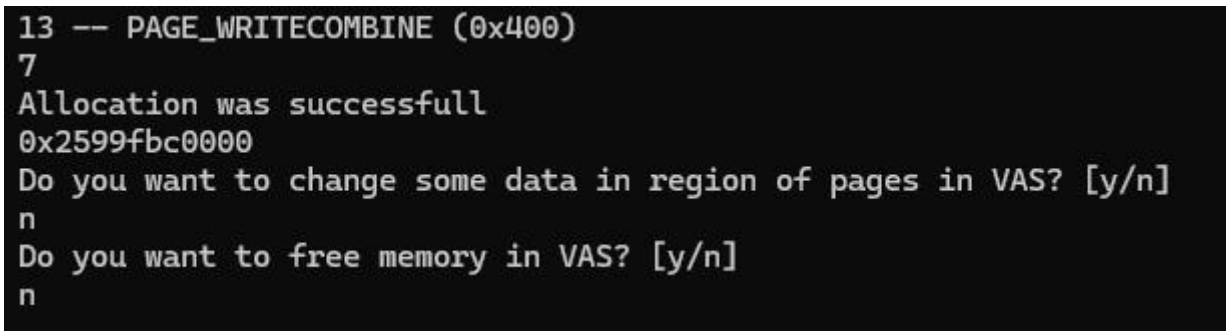

Рисунок 11: Резервирование региона и передача ему физической памяти в автоматическом режиме

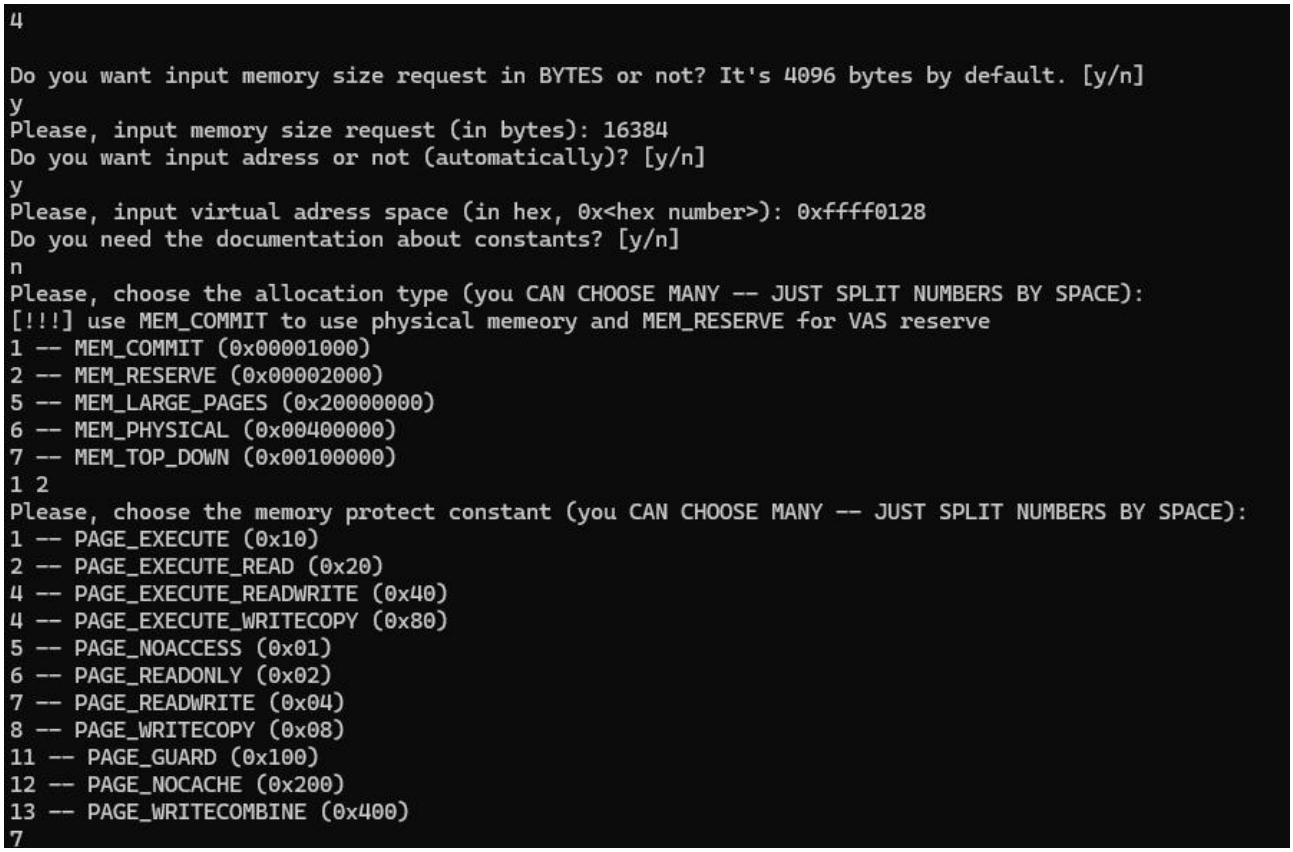

Рисунок 12: Резервирование региона и передача ему физической памяти в ручном режиме

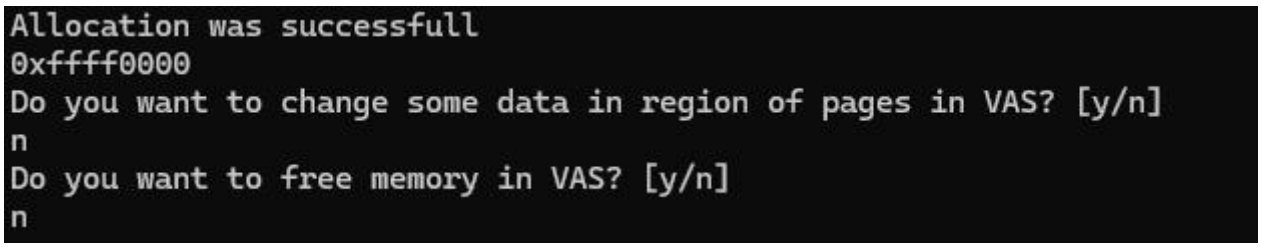

Рисунок 13: Резервирование региона и передача ему физической памяти в ручном режиме

2.5. Вывод списка зарезервированных пользователем адресов

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

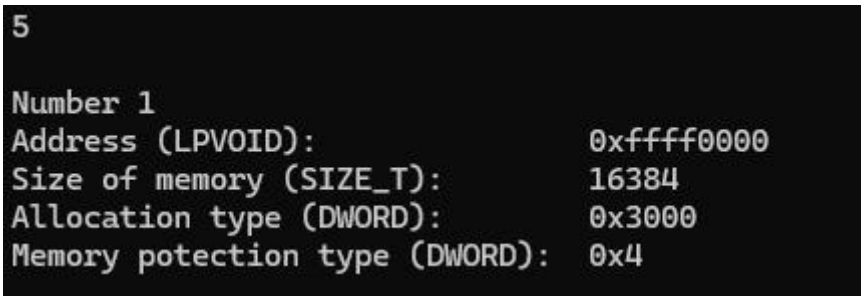

Рисунок 14: Копирование файла

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.

```
6

Number 1
Address (LPVOID):               0xffff0000
Size of memory (SIZE_T):        16384
Allocation type (DWORD):        0x3000
Memory potection type (DWORD):  0x4

Please, choose the number of the region of pages in VAS: 1
THE CHOOSEN region of pages in VAS with is 0xffff0000 with size 16384 bytes
with allocation type 0x3000 and memory constant 0x4
Commit changes? [y/n]
H
Your adress space is from (including) 0xffff0000 to (including) 0xffff3fff
1 -- bool:              1 bytes
2 -- char:              1 bytes
3 -- wchar_t:           2 bytes
4 -- char16_t:          2 bytes
5 -- char32_t:          4 bytes
6 -- short:             2 bytes
7 -- int:               4 bytes
8 -- long:              4 bytes
9 -- long long:         8 bytes
10 -- float:            4 bytes
11 -- double:           8 bytes
12 -- long double:      16 bytes
```

Рисунок 15: Вывод атрибутов файла

```
Please, choose the starting adress (0x<hex number>): 6
Please, choose the input type: 6
Please, choose the starting adress: 6
Please, choose the output type: 6
Adress is out (is less) of possible allocated range, please, try again!
Please, choose the starting adress (0x<hex number>): 0xffff0000
Please, choose the input type: 6
Please, choose the starting adress: 0xffff0001
Please, choose the output type: 2
Checking current values before something:
Input (short): 0
Output (char):
Please, input your value into the variable of choosen type (short): 1147
Output the value from variable of choosen type (char):
Try again? [y -- yes (your values you put will remain) / n -- no]
y
1 -- bool:              1 bytes
2 -- char:              1 bytes
3 -- wchar_t:           2 bytes
4 -- char16_t:          2 bytes
5 -- char32_t:          4 bytes
6 -- short:             2 bytes
7 -- int:               4 bytes
8 -- long:              4 bytes
9 -- long long:         8 bytes
10 -- float:            4 bytes
11 -- double:           8 bytes
12 -- long double:      16 bytes

Please, choose the starting adress (0x<hex number>):
```

Рисунок 16: Изменение атрибутов файла

```
Please, choose the starting adress (0x<hex number>): 0xffff0000
Please, choose the input type: 2
Please, choose the starting adress: 0xffff0000
Please, choose the output type: 6
Checking current values before something:
Input (char): {
Output (short): 1147
Please, input your value into the variable of choosen type (char): @
Output the value from variable of choosen type (short): 1088
Try again? [y -- yes (your values you put will remain) / n -- no]
```

Рисунок 17: 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.

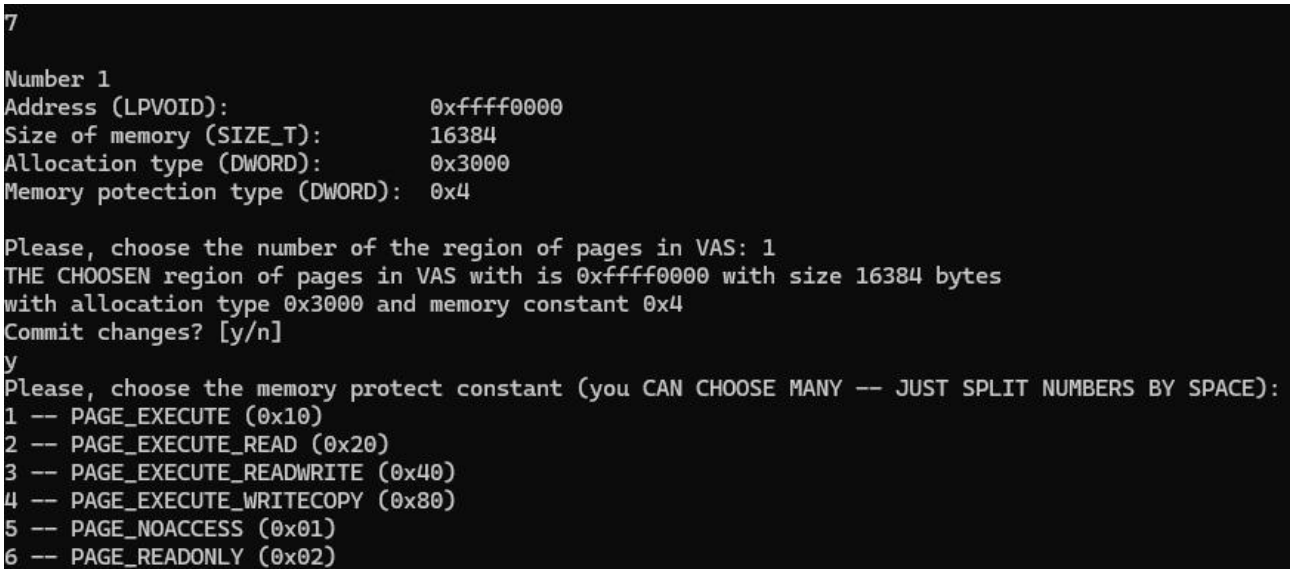

Рисунок 18: Установка константы защиты памяти

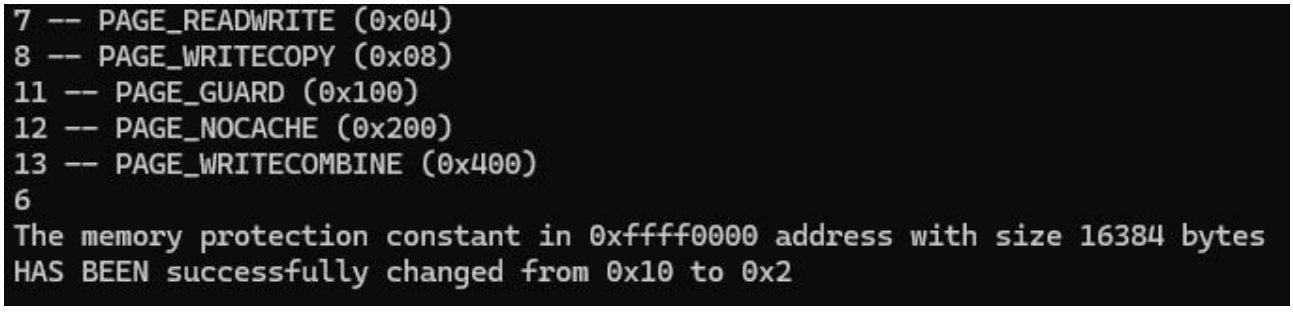

Рисунок 19: Установка константы защиты памяти

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.

```
8

Number 1
Address (LPVOID):               0xffff0000
Size of memory (SIZE_T):        16384
Allocation type (DWORD):        0x3000
Memory potection type (DWORD):  0x4

Please, choose the number of the region of pages in VAS: 1
THE CHOOSEN region of pages in VAS with is 0xffff0000 with size 16384 bytes
with allocation type 0x3000 and memory constant 0x4
Commit changes? [y/n]
y
Please, choose the memory free option (you CAN CHOOSE MANY -- JUST SPLIT NUMBERS BY SPACE):
1 -- MEM_DECOMMIT (0x00004000)
2 -- MEM_RELEASE -- THE MAIN OPTION (0x00008000)
2
The page in 0xffff0000 address with size 0 bytes
HAS BEEN successfully freed with free type 0x8000
```

Рисунок 20: Возврат физической памяти и освобождение региона адресного пространства

```
3

Please, input virtual adress space (in hex, 0x<hex number>): 0xffff0000
Physical memory (RAM) information:
    Pointer to the base address of the region of pages:  0xffff0000
    Pointer -- // -- allocated by the VirtualAlloc:      0
    Memory protection option (for initially allocation): 0
    Region's size from base address, pages identical attributes (in bytes): 70892191744
    The state of the pages in the region:                0x10000 -- Free pages not for process, but for allocation
    Access protection of the pages in the region:        1
    The type of pages in the region:                     0x0 -- THIS NUMBER DOESN'T MEAN ANYTHING
```

Рисунок 21: Проверка региона адресного пространства

2.9. Исходный код программы

```
#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;

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 ();

int main (int argc, char* argv[])
{
   // "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;
}

// ---------- 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;
}

// ---------- 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";
}
```

```
// ---------- GET SYSTEM 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_AVAILABL
E) << "\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_AVAILABL
E) << "\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";
}
```

```
// ---------- GLOBAL MEMORY STATUS ----------
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";
}

// ---------- GLOBAL MEMORY STATUS EX ----------
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";
}

// ---------- VIRTUAL QUERY ----------
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";
}

// ---------- 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";
   }
}

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

void LocalListOfAllocationsFree ()
{
   for (LOCALLOC::const_iterator i = listOfAllocations.begin(); i != listOfAllocations.end(); i++)
   {
      listOfAllocations.erase(i); // erasing vector
   }
}

// ---------- VIRTUAL ALLOC ----------
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";
}
```

```
// ---------- 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";
}

// ---------- 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 ();
```

```
    }
}

// ---------- LOCAL VIRTUAL PROTECT ----------
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 ();
    }
}

// ---------- 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";
   }
}

// ---------- VIRTUAL FREE INDEPENDENT ----------
```

```
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 по работе виртуальной памяти

Память компьютера подразделяется как минимум на два вида: основную (реальную) и вторичную. Основная память энергозависима (следовательно, нужна для кратковременного хранения данных, необходимых в данный момент) и обеспечивает быстрый доступ к данным. Вторичная память энергонезависима (следовательно, нужна для долговременного хранения) и обеспечивает более медленный доступ к данным.

Основная память при страничной организации распределена на некоторое количество кадров равного размера и каждый процесс распределён на некоторое количество страниц равного размера (и обычно равному размеру кадра памяти). Тогда блоки, или страницы (pages), процесса могут быть связаны с блоками, или кадрами (frames), памяти, то есть один кадр содержит одну страницу. Это позволяет полностью избежать внешней фрагментации памяти (когда свободная память не может быть использована вследствие, например, малых блоков для размещения процесса) и минимизировать внутреннюю фрагментацию (когда используется только часть выделенной под процесс памяти), сводя её к частичному использованию только последней страницы процесса.

При страничной организации памяти преобразование логических адресов в физические – задача аппаратного уровня. Логический адрес (номер страницы и смещение) переходит в физический адрес (номер кадра и смещение) с использованием специальной таблицы страниц. В ней для каждого процесса для каждой из его страниц содержится номер кадра (если он имеется, например, когда процесс не выгружен из памяти), при этом смещение одинаково.

Виртуальная память основана на использовании сегментов или страниц. Если процесс состоит из большого числа данных, из которых только часть необходима в данный момент, то нет нужны выгружать все данные в основную память, поэтому появляется возможность использовать её более эффективно (в

том числе, если нет возможности загрузить все данные). Если все обращения к памяти происходят через логические адреса (которые затем переходят в физические), и процесс может быть разбит на страницы, то можно решить эту проблему с помощью виртуальной памяти – возможностью организации большей памяти, которая включает в себя как основную, так и вторичную.

При организации виртуальной памяти с помощью страниц в записях таблицы страниц помимо номера кадра появляются и управляющие биты. В частности, может быть бит присутствия – наличие страницы в основной памяти, бит модификации – индикатор изменения содержимого страницы в основной памяти – и другие. Таким образом, главный механизм виртуальной памяти – это преобразование логического (виртуального) адреса (номер страницы + смещение) в физический адрес (номер страницы + смещение) с помощью таблицы страниц некоторого процесса. Виртуальные адреса формируют виртуальное адресное пространство.

2.11. 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 сообщения: «Hello, shared memory!».

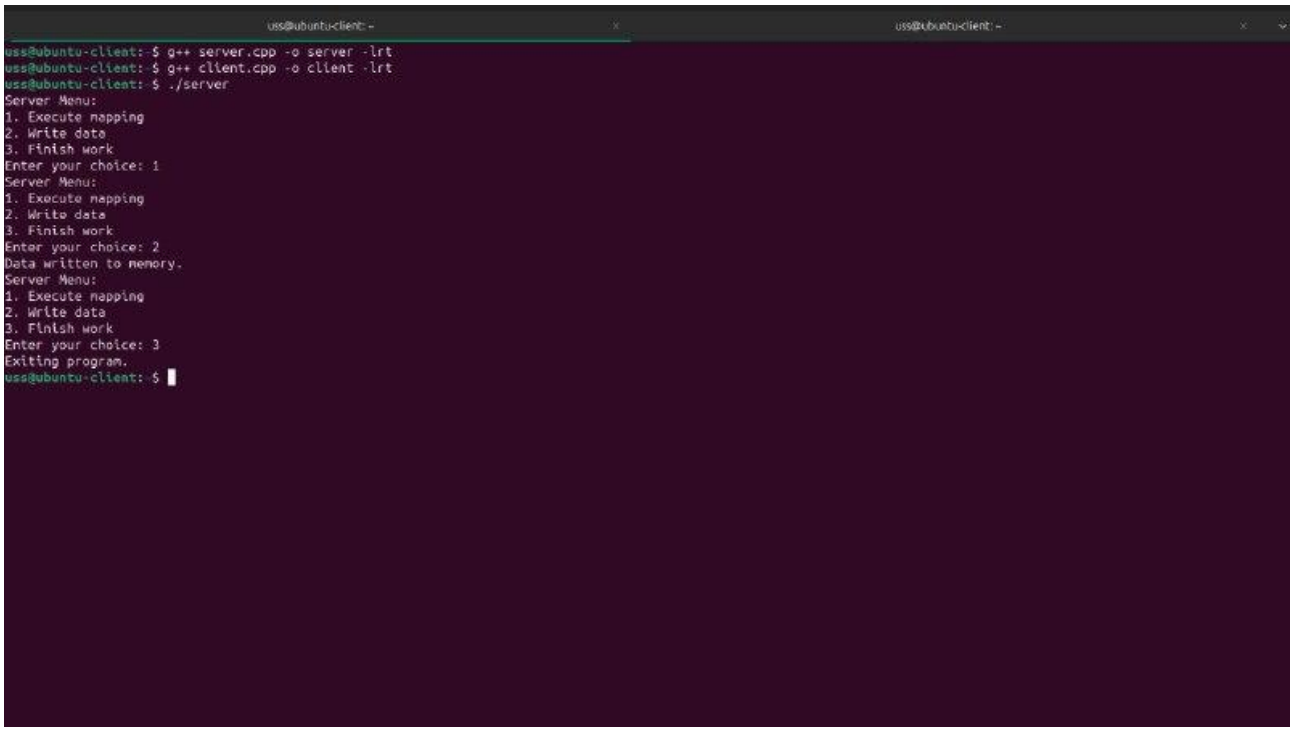

Рисунок 26: Создание проецируемого файла

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 и вывод сообщения: «Hello, shared memory!».

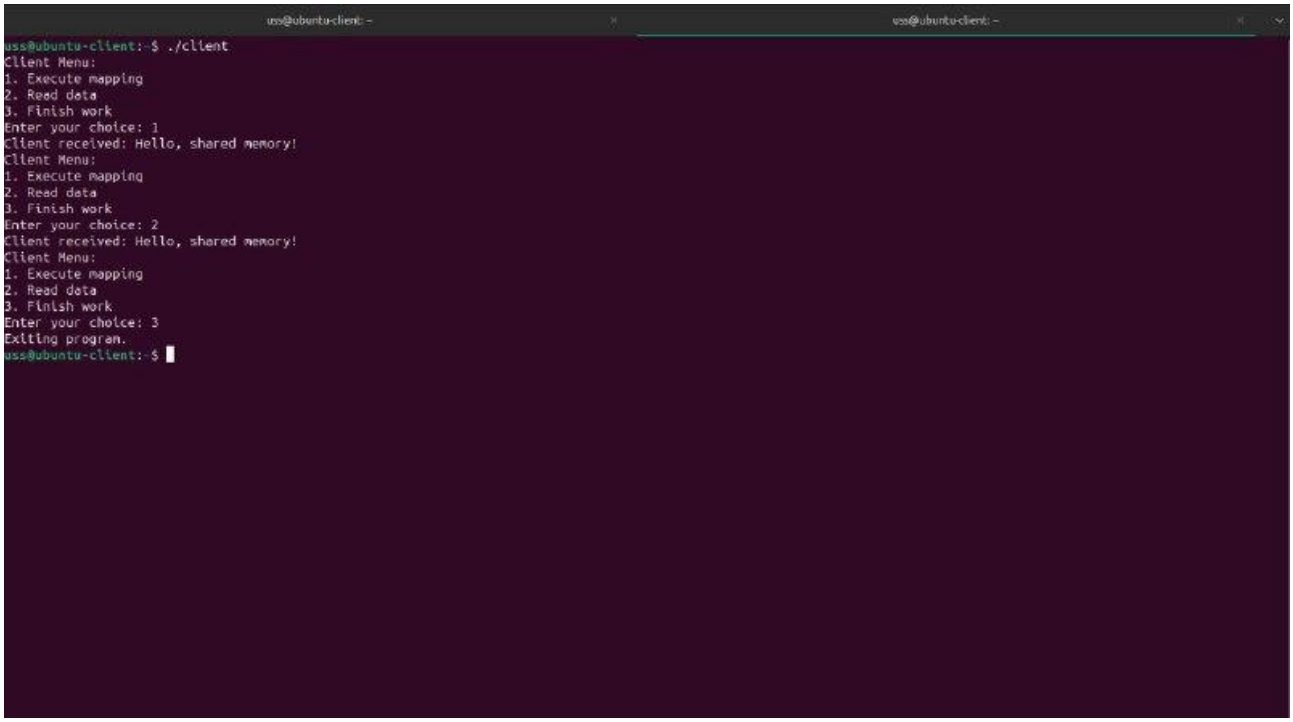

Рисунок 27: Открытие проецируемого файла и вывод сообщения

Если запустить приложение-читатель без запуска приложения-писателя, или сделать так, что приложение-писатель закончит свою работу раньше, чем приложение-читатель успеет прочитать, то возникнет следующая ошибка:

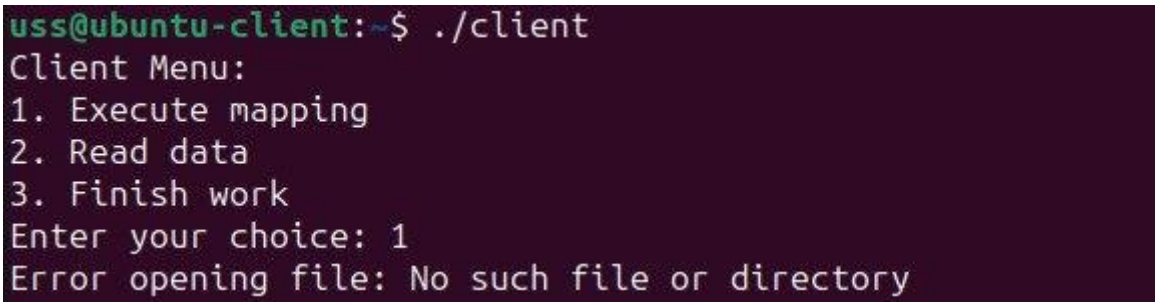

Рисунок 28: Ошибка чтения

3.3. Исходный код программы-писателя

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <cstdio>
#include <cstring>
#include <cstdlib>

#define FILENAME "shared_memory_file"
#define FILESIZE sizeof(char) * 100

void createAndMapFile(int& fd, char** ptr);
void writeToMemory(char* ptr);
void unmapAndDeleteFile(int fd, char* ptr);

int main() {
    int choice;
    int fd; // Дескриптор файла
    char *ptr = nullptr; // Указатель на начало проецируемой области в памяти

    do {
        printf("Server Menu:\n");
        printf("1. Execute mapping\n");
        printf("2. Write data\n");
        printf("3. Finish work\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                createAndMapFile(fd, &ptr);
                break;
            case 2:
                if (ptr != nullptr) {
                    writeToMemory(ptr);
                } else {
                    printf("Memory is not mapped yet.\n");
                }
                break;
            case 3:
                if (ptr != nullptr) {
                    unmapAndDeleteFile(fd, ptr);
                    ptr = nullptr;
                }
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice, please try again.\n");
        }
    } while (choice != 3);

    return 0;
}

void createAndMapFile(int& fd, char** ptr) {
```

```
    fd = open(FILENAME, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    if (fd == -1) {
        perror("Error opening file");
        exit(1);
    }

    if (ftruncate(fd, FILESIZE) == -1) {
        perror("Error setting file size");
        close(fd);
        exit(1);
    }

    *ptr = (char *)mmap(NULL, FILESIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (*ptr == MAP_FAILED) {
        perror("Error mapping file");
        close(fd);
        exit(1);
    }
    // Очистка памяти
    memset(*ptr, 0, FILESIZE);
}

void writeToMemory(char* ptr) {
    sprintf(ptr, "Hello, shared memory!");
    printf("Data written to memory.\n");
}
```

```
void unmapAndDeleteFile(int fd, char* ptr) {
    munmap(ptr, FILESIZE);
    close(fd);
    unlink(FILENAME);
}
```

3.4. Исходный код программы-читателя

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <cstdio>
#include <cstdlib>

#define FILENAME "shared_memory_file"
#define FILESIZE sizeof(char) * 100

void mapAndReadFile();

int main() {
    int choice;

    do {
        printf("Client Menu:\n");
        printf("1. Execute mapping\n");
        printf("2. Read data\n");
        printf("3. Finish work\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
            case 2: // В данном случае обработка для чтения и отображения объединены
```

```
            mapAndReadFile();
            break;
        case 3:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice, please try again.\n");
    }
  } while (choice != 3);

  return 0;
}

void mapAndReadFile() {
  int fd = open(FILENAME, O_RDONLY);
  if (fd == -1) {
    perror("Error opening file");
    exit(1);
  }

  struct stat st;
  if (fstat(fd, &st) == -1) {
    perror("Error getting file size");
    close(fd);
    exit(1);
  }

  char* ptr = (char *)mmap(NULL, st.st_size, PROT_READ, MAP_SHARED, fd,
```

```
0);
    if (ptr == MAP_FAILED) {
        perror("Error mapping file");
        close(fd);
        exit(1);
    }

    printf("Client received: %s\n", ptr);

    munmap(ptr, st.st_size);
    close(fd);
}
```

3.5. Выводы по работе проецируемых файлов

Механизм проецируемых файлов - это важная концепция в операционных системах, позволяющая процессам работать с общими данными, хранящимися в файлах, как если бы они находились в памяти. Когда файл проецируется в адресное пространство процесса, это означает, что его содержимое становится доступным для чтения и записи через указатель в памяти процесса.

На уровне операционной системы, процессы, находящиеся в разных виртуальных адресных пространствах (ВАП), могут работать с общей памятью благодаря механизму разделения памяти и использованию проецируемых файлов. Когда несколько процессов отображают один и тот же файл в свои ВАП, они могут взаимодействовать через этот файл, обмениваясь данными и синхронизируя свою работу.

Процессы могут использовать системные вызовы, такие как mmap() в UNIX-подобных системах, чтобы проецировать файлы в свои адресные пространства.

Таким образом, механизм проецируемых файлов позволяет процессам в разных ВАП работать с общей памятью, обеспечивая эффективное взаимодействие и совместное использование данных между процессами.

3.6. 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.