**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по дисциплине «Операционные Системы»**

Тема: **«исследовать механизмы управления виртуальной памятью Win32»**

|  |  |  |
| --- | --- | --- |
| Студент | Аршин А.Д |  |

С

Санкт-Петербург

2025

**Введение**

**Цель работы**

Изучение механизмов работы с виртуальным адресным пространством процесса в операционной системе Windows. Ознакомление с функциями Win32 API, предназначенными для управления виртуальной памятью, их особенностями и практическим применением.

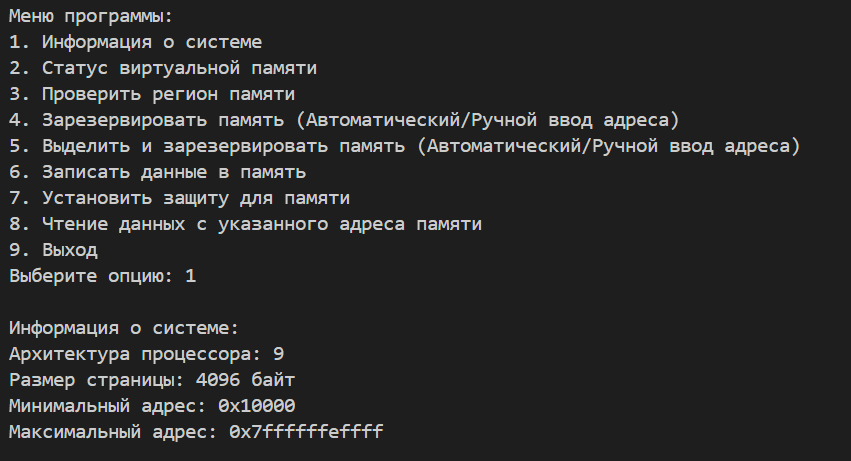
* **Постановка задачи**

Разработать консольное приложение, реализующее управление виртуальной памятью с использованием Win32 API. Программа должна:

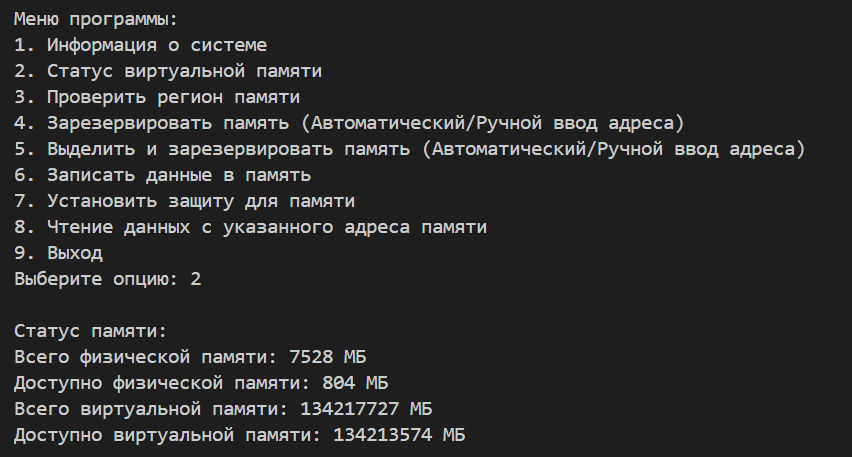
* Получать информацию о вычислительной системе (функция GetSystemInfo).
* Определять статус виртуальной памяти (функция GlobalMemoryStatus).
* Определять состояние заданного участка памяти (функция VirtualQuery).
* Выполнять раздельное резервирование региона памяти и выделение физической памяти, как в автоматическом режиме, так и с вводом начального адреса (функции VirtualAlloc, VirtualFree).
* Выполнять одновременное резервирование и выделение физической памяти в аналогичных режимах.
* Записывать данные в память по указанным адресам.
* Устанавливать защиту доступа для региона памяти и проверять ее (функция VirtualProtect).

**Пример выполнения программы:**

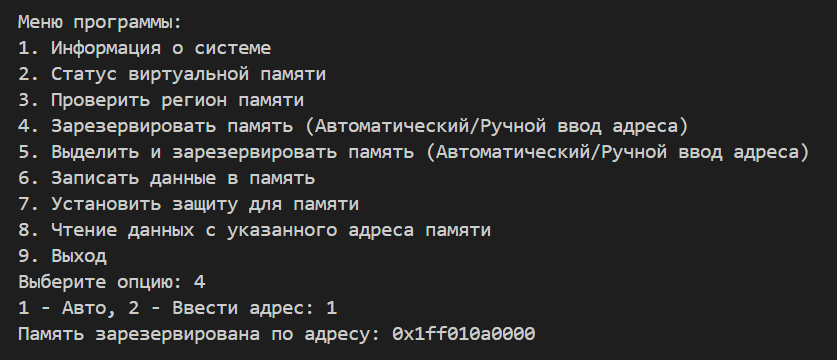
**1.**



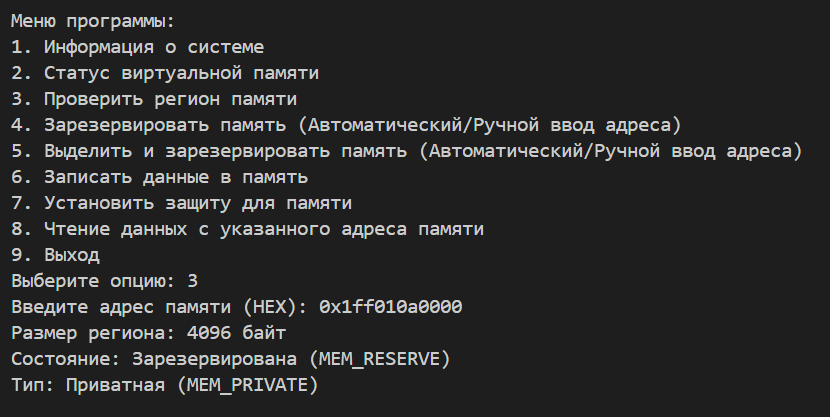
**2.**



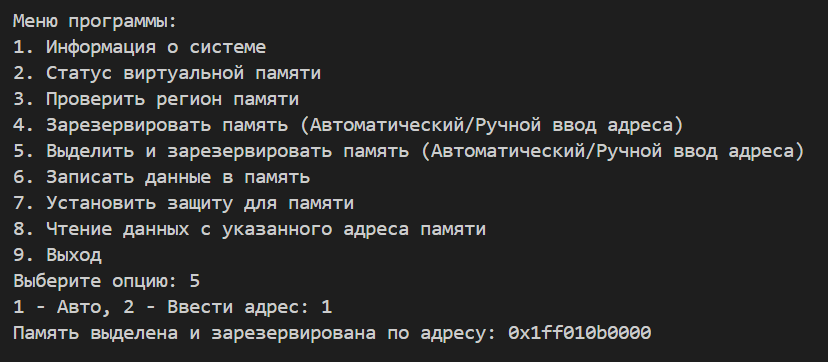
3.



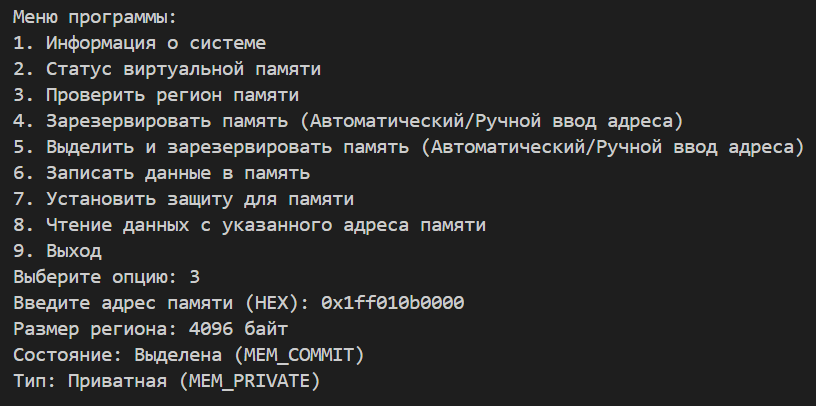
4.



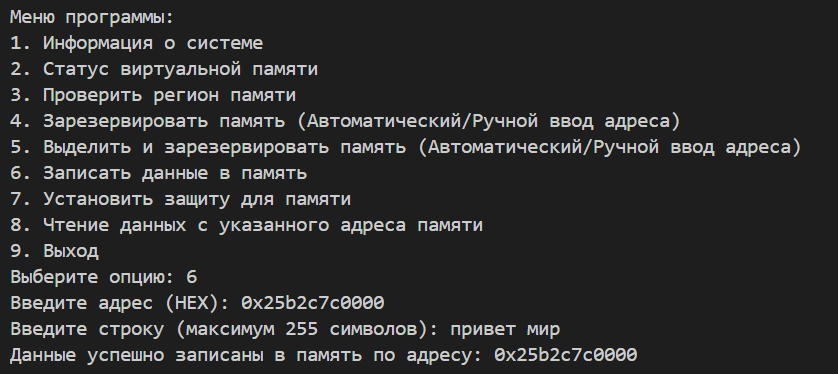
5.



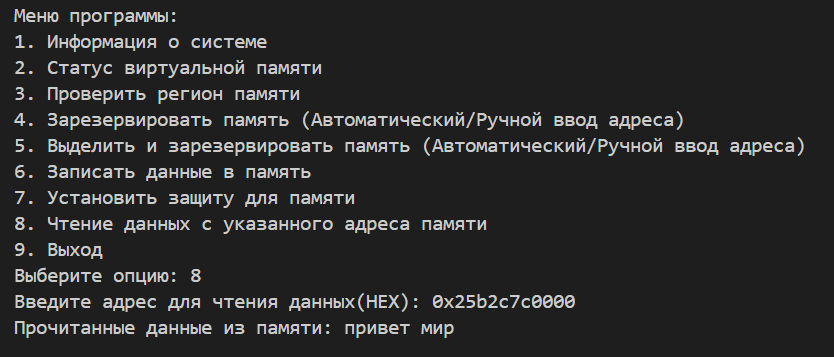
6.

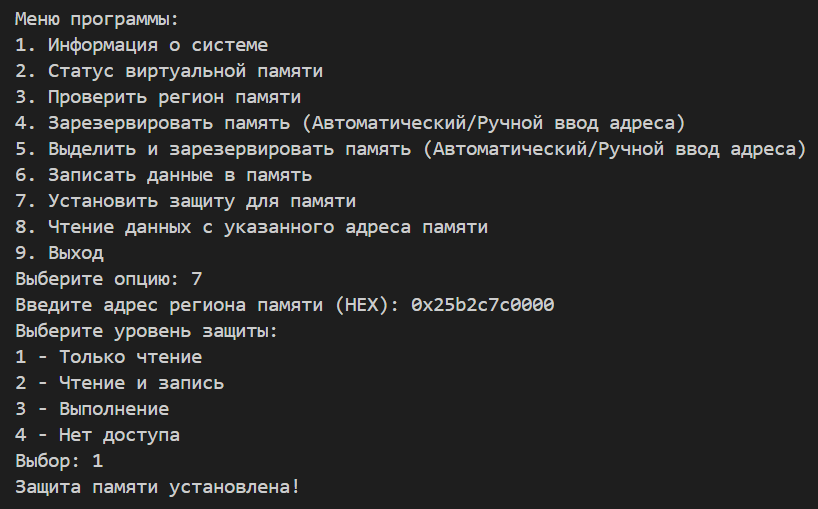


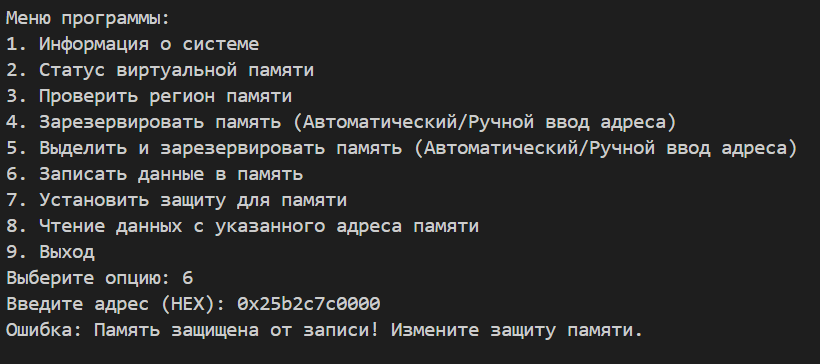
7.



8.



9. 



**Код программы**

#include <windows.h>  // Библиотека для работы с системными API Windows

#include <iostream>   // Ввод и вывод

#include <iomanip>    // Манипуляторы ввода-вывода (например, std::hex)

#include <locale>     // Локализация (установка русского языка)

#include <codecvt>    // Кодировки UTF-16

#include <fcntl.h>    // Установка режимов консоли

#include <io.h>       // Управление вводом/выводом

//---------------------------------------------------------------------------//

void GSI();//GetSystemInformation

void GMS();//GetMemoryStatus

void QMR();//QueryMemoryRegion

void RM(bool userDefined);//ReserveMemory

void AARM(bool userDefined);//AllocateAndReserveMemory

void WTM();//WriteToMemory

void PM();//ProtectMemory

void RM();//ReadMemory

//---------------------------------------------------------------------------//

int main() {

    int choice;

    \_setmode(\_fileno(stdout), \_O\_U16TEXT);

    \_setmode(\_fileno(stdin), \_O\_U16TEXT);

    do {

        std::wcout << L"\nМеню программы:\n"

                   << L"1. Информация о системе\n"

                   << L"2. Статус виртуальной памяти\n"

                   << L"3. Проверить регион памяти\n"

                   << L"4. Зарезервировать память (Автоматический/Ручной ввод адреса)\n"

                   << L"5. Выделить и зарезервировать память (Автоматический/Ручной ввод адреса)\n"

                   << L"6. Записать данные в память\n"

                   << L"7. Установить защиту для памяти\n"

                   << L"8. Чтение данных с указанного адреса памяти\n"

                   << L"9. Выход\n"

                   << L"Выберите опцию: ";

        std::wcin >> choice;

        std::wcin.ignore();

        switch (choice) {

            case 1: GSI(); break;

            case 2: GMS(); break;

            case 3: QMR(); break;

            case 4: {

                std::wcout << L"1 - Авто, 2 - Ввести адрес: ";

                int opt;

                std::wcin >> opt;

                RM(opt == 2);

                break;

            }

            case 5: {

                std::wcout << L"1 - Авто, 2 - Ввести адрес: ";

                int opt;

                std::wcin >> opt;

                AARM(opt == 2);

                break;

            }

            case 6: WTM(); break;

            case 7: PM(); break;

            case 8: RM();break;

            case 9: std::wcout << L"Выход...\n"; break;

            default: std::wcout << L"Неверный выбор!\n";

        }

    } while (choice != 9);

    return 0;

}

//---------------------------------------------------------------------------//

void GSI() {

    SYSTEM\_INFO si;

    GetSystemInfo(&si);

    std::wcout << L"\nИнформация о системе:\n"

               << L"Архитектура процессора: " << si.wProcessorArchitecture << L"\n"

               << L"Размер страницы: " << si.dwPageSize << L" байт\n"

               << L"Минимальный адрес: " << si.lpMinimumApplicationAddress << L"\n"

               << L"Максимальный адрес: " << si.lpMaximumApplicationAddress << L"\n";

}

//---------------------------------------------------------------------------//

void GMS() {

    MEMORYSTATUS memStatus;

    GlobalMemoryStatus(&memStatus);

    std::wcout << L"\nСтатус памяти:\n"

               << L"Всего физической памяти: " << memStatus.dwTotalPhys / (1024 \* 1024) << L" МБ\n"

               << L"Доступно физической памяти: " << memStatus.dwAvailPhys / (1024 \* 1024) << L" МБ\n"

               << L"Всего виртуальной памяти: " << memStatus.dwTotalVirtual / (1024 \* 1024) << L" МБ\n"

               << L"Доступно виртуальной памяти: " << memStatus.dwAvailVirtual / (1024 \* 1024) << L" МБ\n";

}

//---------------------------------------------------------------------------//

void QMR() {

    MEMORY\_BASIC\_INFORMATION mbi;

    void\* address;

    std::wcout << L"Введите адрес памяти (HEX): ";

    std::wcin >> std::hex >> address;

    if (VirtualQuery(address, &mbi, sizeof(mbi))) {

        std::wcout << L"Размер региона: " << mbi.RegionSize << L" байт\n";

        // Определение состояния памяти

        std::wcout << L"Состояние: ";

        switch (mbi.State) {

            case MEM\_COMMIT: std::wcout << L"Выделена (MEM\_COMMIT)\n"; break;

            case MEM\_RESERVE: std::wcout << L"Зарезервирована (MEM\_RESERVE)\n"; break;

            case MEM\_FREE: std::wcout << L"Свободна (MEM\_FREE)\n"; break;

            default: std::wcout << L"Неизвестное\n";

        }

        // Определение типа памяти

        std::wcout << L"Тип: ";

        switch (mbi.Type) {

            case MEM\_PRIVATE: std::wcout << L"Приватная (MEM\_PRIVATE)\n"; break;

            case MEM\_MAPPED: std::wcout << L"Отображена из файла (MEM\_MAPPED)\n"; break;

            case MEM\_IMAGE: std::wcout << L"Загружена из исполняемого файла (MEM\_IMAGE)\n"; break;

            default: std::wcout << L"Неизвестный\n";

        }

    } else {

        std::wcout << L"Ошибка VirtualQuery!\n";

    }

}

//---------------------------------------------------------------------------//

void RM(bool userDefined) {

    void\* address = nullptr;

    if (userDefined) {

        std::wcout << L"Введите начальный адрес (HEX): ";

        std::wcin >> std::hex >> address;

    }

    void\* allocated = VirtualAlloc(address, 4096, MEM\_RESERVE, PAGE\_NOACCESS);

    if (allocated) {

        std::wcout << L"Память зарезервирована по адресу: " << allocated << L"\n";

    } else {

        std::wcout << L"Ошибка резервирования памяти!\n";

    }

}

//---------------------------------------------------------------------------//

void AARM(bool userDefined) {

    void\* address = nullptr;

    if (userDefined) {

        std::wcout << L"Введите адрес (HEX): ";

        std::wcin >> std::hex >> address;

    }

    void\* allocated = VirtualAlloc(address, 4096, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE);

    if (allocated) {

        std::wcout << L"Память выделена и зарезервирована по адресу: " << allocated << L"\n";

    } else {

        std::wcout << L"Ошибка выделения памяти!\n";

    }

}

//---------------------------------------------------------------------------//

void WTM() {

    void\* address;

    wchar\_t data[256];

    std::wcout << L"Введите адрес (HEX): ";

    std::wcin >> std::hex >> address;

    std::wcin.ignore();

    MEMORY\_BASIC\_INFORMATION mbi;

    if (VirtualQuery(address, &mbi, sizeof(mbi)) == 0) {

        std::wcout << L"Ошибка VirtualQuery! Неверный адрес.\n";

        return;

    }

    if (!(mbi.State & MEM\_COMMIT)) {

        std::wcout << L"Ошибка: Память не выделена! Используйте функцию выделения памяти.\n";

        return;

    }

    if (!(mbi.Protect & (PAGE\_READWRITE | PAGE\_EXECUTE\_READWRITE))) {

        std::wcout << L"Ошибка: Память защищена от записи! Измените защиту памяти.\n";

        return;

    }

    std::wcout << L"Введите строку (максимум 255 символов): ";

    std::wcin.ignore();  // Игнорируем предыдущие символы в буфере

    std::wcin.getline(data, 256);

    wcscpy((wchar\_t\*)address, data);

    std::wcout << L"Данные успешно записаны в память по адресу: " << address << L"\n";

}

//---------------------------------------------------------------------------//

void PM() {

    void\* address;

    DWORD oldProtect;

    int protectType;

    std::wcout << L"Введите адрес региона памяти (HEX): ";

    std::wcin >> std::hex >> address;

    std::wcout << L"Выберите уровень защиты:\n"

               << L"1 - Только чтение\n"

               << L"2 - Чтение и запись\n"

               << L"3 - Выполнение\n"

               << L"4 - Нет доступа\n"

               << L"Выбор: ";

    std::wcin >> protectType;

    DWORD newProtect;

    switch (protectType) {

        case 1: newProtect = PAGE\_READONLY; break;

        case 2: newProtect = PAGE\_READWRITE; break;

        case 3: newProtect = PAGE\_EXECUTE; break;

        case 4: newProtect = PAGE\_NOACCESS; break;

        default: std::wcout << L"Неверный выбор!\n"; return;

    }

    if (VirtualProtect(address, 4096, newProtect, &oldProtect)) {

        std::wcout << L"Защита памяти установлена!\n";

    } else {

        std::wcout << L"Ошибка установки защиты!\n";

    }

}

//---------------------------------------------------------------------------//

void RM(){

    void\* address;

    wchar\_t checkData[256];

    std::wcout << L"Введите адрес для чтения данных(HEX): ";

    std::wcin >> std::hex >> address;

    std::wcin.ignore();  // Очистка буфера ввода после ввода адреса

    wcscpy(checkData, (wchar\_t\*)address);  // Считываем данные обратно

    if ((checkData[0])==L'\0')

    {

        std::wcout << L"Память пуста, попробуйте занести данные по этому адресу:"<<address<< L"\n";

    }

    else

    {

        std::wcout << L"Прочитанные данные из памяти: " << checkData << L"\n";

    }

}

//---------------------------------------------------------------------------//

**Заключение**

В ходе выполнения данной работы было разработано консольное приложение, позволяющее исследовать виртуальное адресное пространство процесса с использованием Win32 API. Были изучены и практически применены функции для работы с памятью:

* **GetSystemInfo** – получение информации о вычислительной системе, включая размер страницы памяти, минимальный и максимальный адреса доступного виртуального адресного пространства, количество процессоров и другие параметры.
* **GlobalMemoryStatus** – определение общего состояния оперативной памяти, включая объем доступной физической памяти, виртуальной памяти и загруженность системы.
* **VirtualQuery** – анализ состояния заданного участка виртуальной памяти, позволяющий определить его атрибуты (свободная область, зарезервированная память, выделенная память и т. д.).
* **VirtualAlloc / VirtualFree** – динамическое резервирование и выделение физической памяти, как в автоматическом режиме, так и с указанием конкретного адреса. Эти функции позволили исследовать разницу между резервированием и выделением памяти.
* **VirtualProtect** – изменение уровня защиты памяти, включая запрет на запись или выполнение кода в указанном регионе памяти, с последующей проверкой установленных атрибутов.

Проведенные эксперименты показали, что виртуальная память в Windows организована в виде страниц фиксированного размера, и ее управление осуществляется через системные вызовы. Работа с выделенной памятью требует осторожности: например, попытка записи в регион памяти без предварительного его выделения может привести к критическим ошибкам. Аналогично, изменение прав доступа через VirtualProtect демонстрирует механизм защиты памяти от нежелательных изменений.

Результаты экспериментов подтверждают важность понимания принципов управления памятью при разработке приложений, особенно тех, которые работают на низком уровне с ресурсами системы. Данная работа позволила глубже разобраться в механизмах работы с виртуальной памятью, что может быть полезно при разработке программного обеспечения, требующего точного контроля за использованием памяти.