**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Операционные системы»**

**Тема: Управление памятью**

|  |  |  |
| --- | --- | --- |
| Студен гр. 2307 |  | Стукен В.А |
| Преподаватель |  | Тимофеев А.В |

Санкт-Петербург

2024

**Оглавление**

[**Цель работы. 3**](#_Toc162987724)

[**Задание 4.1. Исследовать виртуальное адресное пространство процесса. 3**](#_Toc162987725)

[**Указания к выполнению. 3**](#_Toc162987726)

[**Демонстрация работы программы на нескольких наборах вводимых данных. 4**](#_Toc162987727)

[**Теория по первой части. 14**](#_Toc162987728)

[**Выводы по первой части. 15**](#_Toc162987729)

[**Задание на проецируемый файлы. Два приложения – клиент и сервер. 17**](#_Toc162987730)

[**Код программы Сервера(писателя): 18**](#_Toc162987731)

[**Код программы Клиента(читателя): 20**](#_Toc162987732)

[**Выводы по второй части 21**](#_Toc162987733)

[**Источники 22**](#_Toc162987734)

# Цель работы.

Исследовать механизмы управления виртуальной памятью Win32.

# Задание 4.1. Исследовать виртуальное адресное пространство процесса.

## Указания к выполнению.

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.

## Демонстрация работы программы на нескольких наборах вводимых данных.

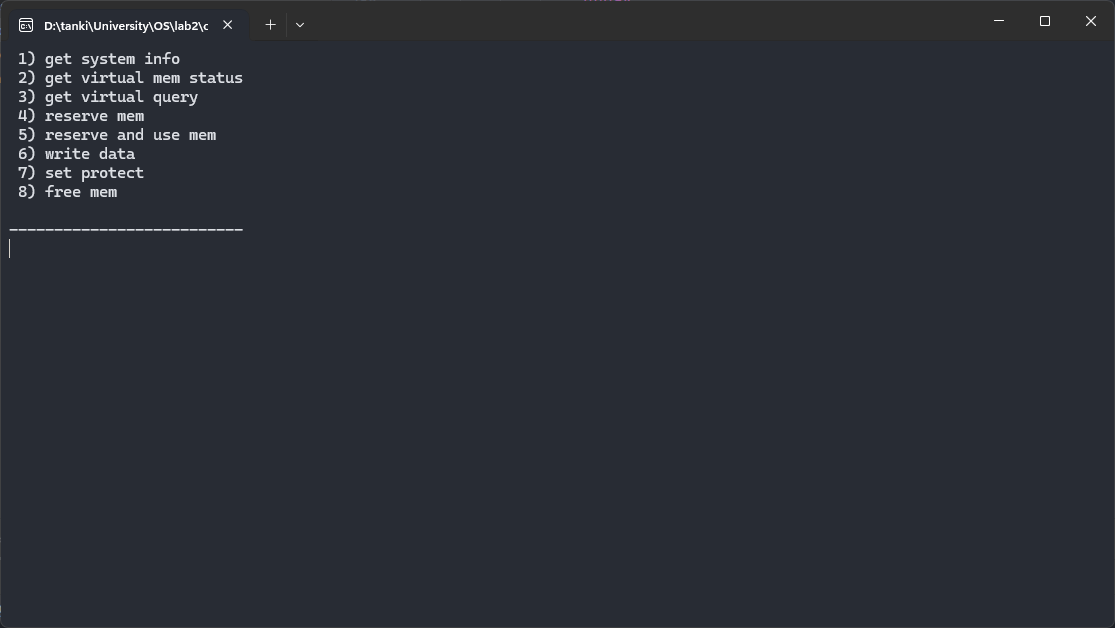


Рисунок 1 Меню программы

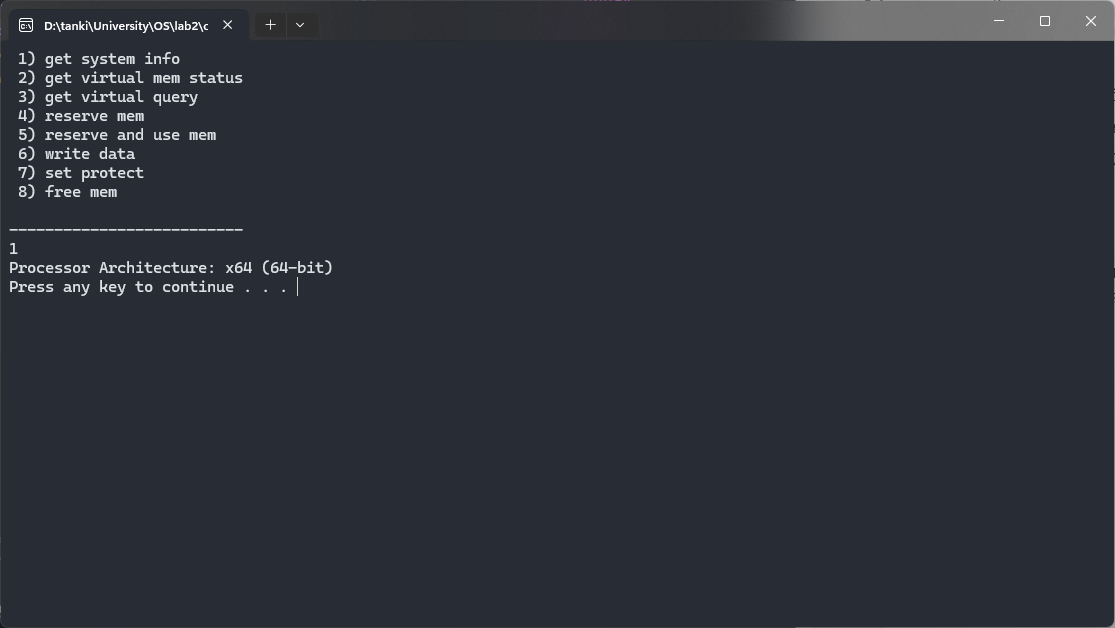


Рисунок 2 Функция get system info

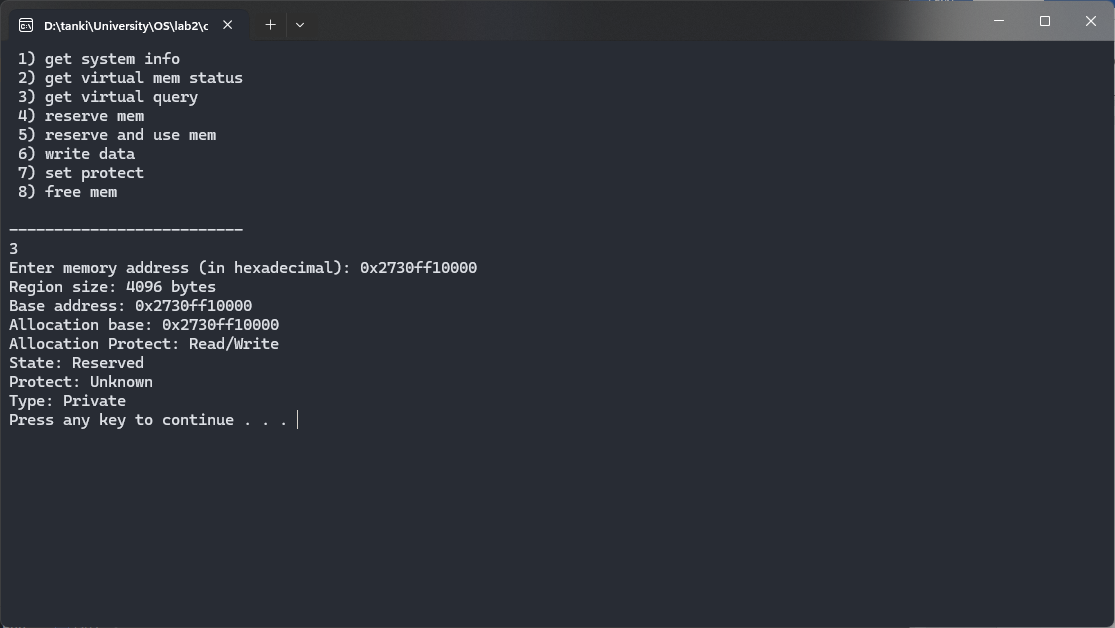


Рисунок 3 функция get virtual query

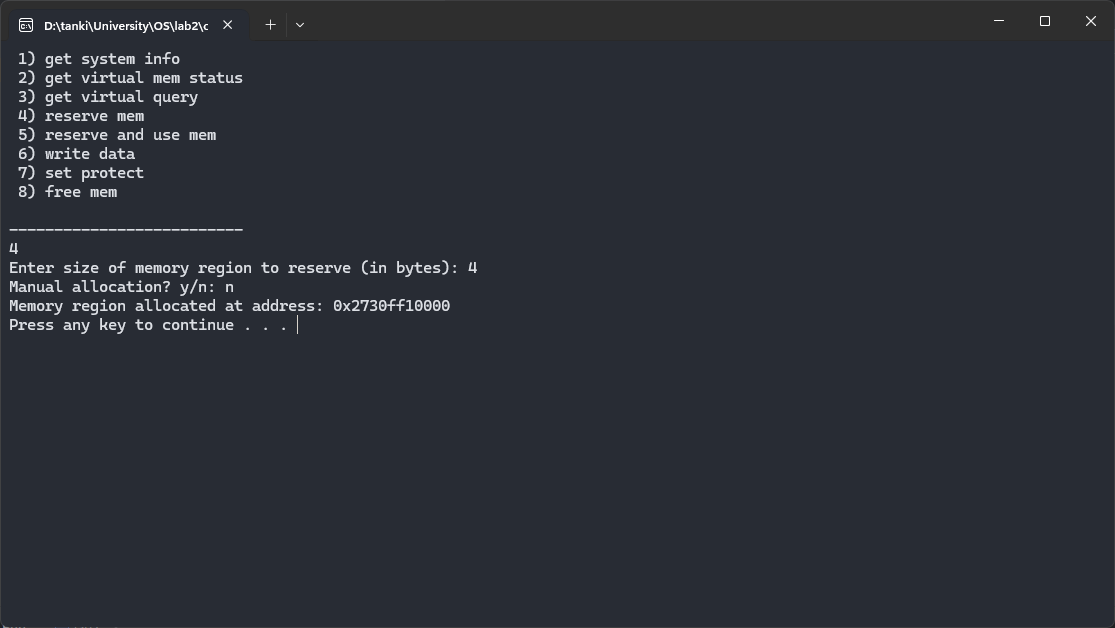


Рисунок 4 функция reserve memory

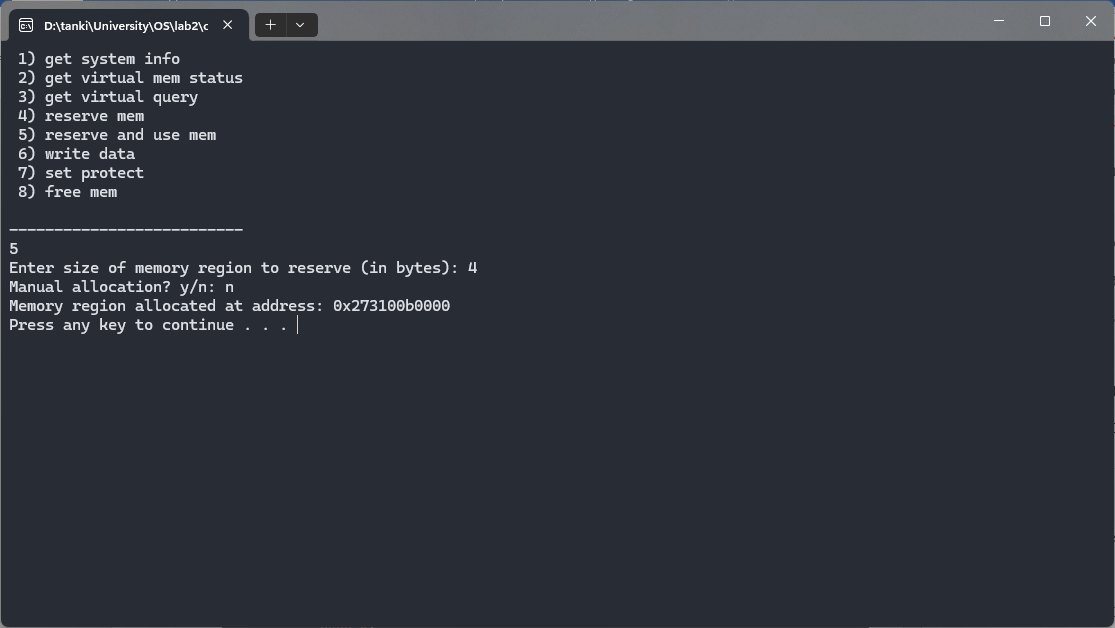


Рисунок 5 функция reserve and use memory

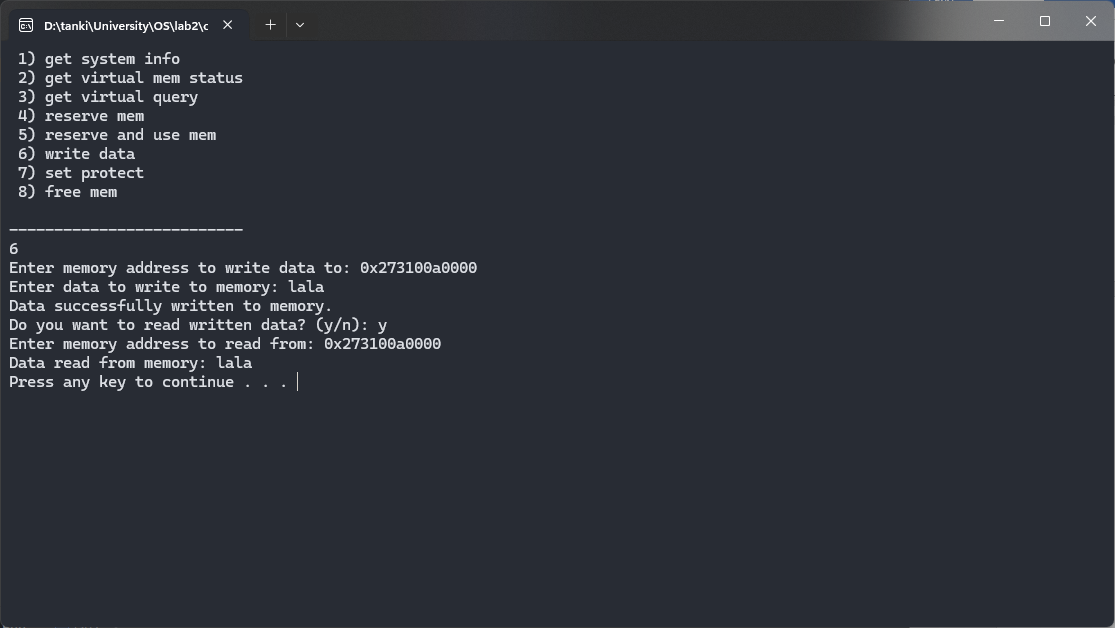


Рисунок 6 функция write data

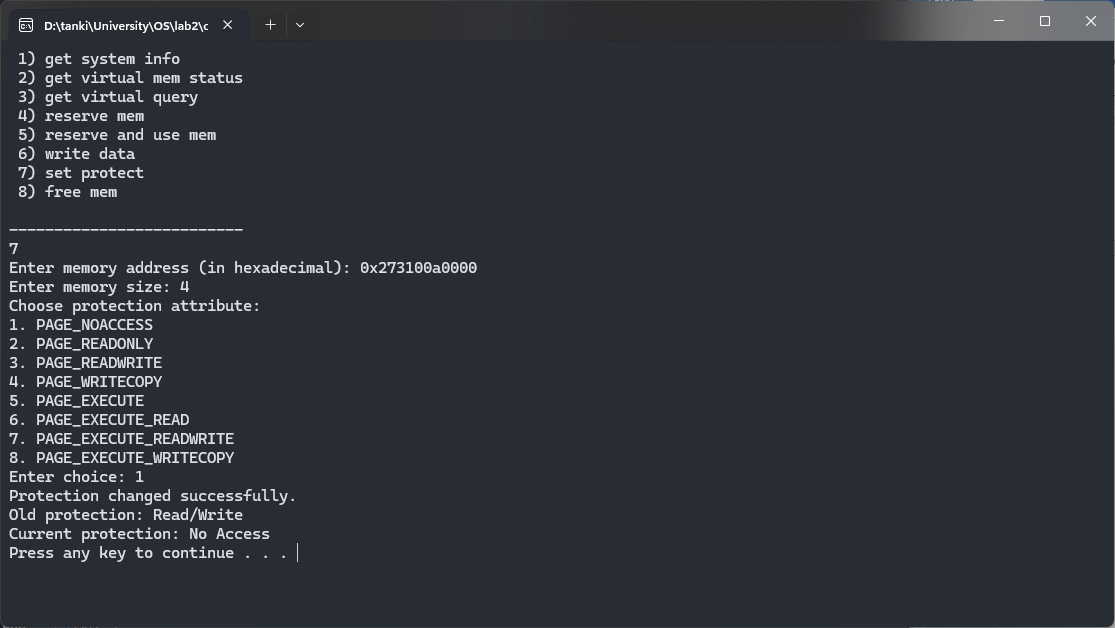


Рисунок 7 функция set protect

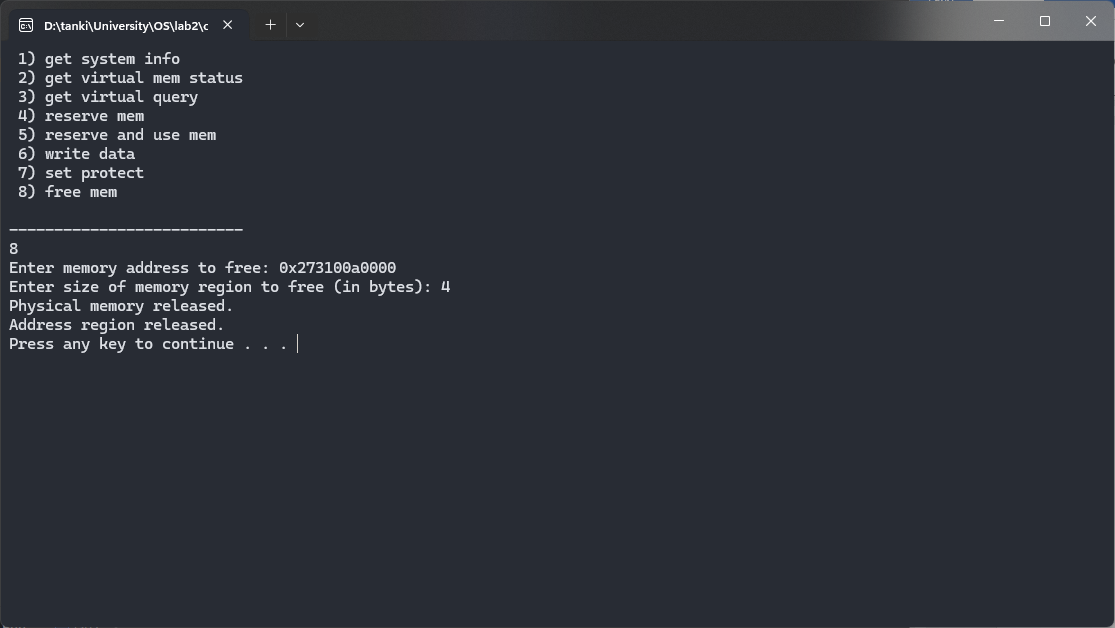


Рисунок 8 функция free memory

1. **Код программы**
2. #include <windows.h>  
   #include <string>  
   #include <iostream>  
   #include <map>  
   #include <vector>  
     
   using namespace std;  
     
   void printMenu() {  
    std::cout  
    << " 1) get system info\n"  
    << " 2) get virtual mem status\n"  
    << " 3) get virtual query\n"  
    << " 4) reserve mem\n"  
    << " 5) reserve and use mem \n"  
    << " 6) write data\n"  
    << " 7) set protect\n"  
    << " 8) free mem\n" << endl;  
    std::cout << "--------------------------\n";  
   }  
     
   // type -> string  
   string TypeToString(DWORD type) {  
    switch (type) {  
    case MEM\_PRIVATE:  
    return "Private";  
    case MEM\_MAPPED:  
    return "Mapped";  
    case MEM\_IMAGE:  
    return "Image";  
    default:  
    return "Unknown";  
    }  
   }  
     
   // protect attribute -> string  
   string ProtectToString(DWORD protect) {  
    switch (protect) {  
    case PAGE\_NOACCESS:  
    return "No Access";  
    case PAGE\_READONLY:  
    return "Read Only";  
    case PAGE\_READWRITE:  
    return "Read/Write";  
    case PAGE\_WRITECOPY:  
    return "Write Copy";  
    case PAGE\_EXECUTE:  
    return "Execute Only";  
    case PAGE\_EXECUTE\_READ:  
    return "Execute/Read";  
    case PAGE\_EXECUTE\_READWRITE:  
    return "Execute/Read/Write";  
    case PAGE\_EXECUTE\_WRITECOPY:  
    return "Execute/Write Copy";  
    default:  
    return "Unknown";  
    }  
   }  
     
   struct ProtectionInfo {  
    int number;  
    std::string description;  
   };  
     
   map<int, DWORD> protectMap = {  
    {1, PAGE\_NOACCESS},  
    {2, PAGE\_READONLY},  
    {3, PAGE\_READWRITE},  
    {4, PAGE\_WRITECOPY},  
    {5, PAGE\_EXECUTE},  
    {6, PAGE\_EXECUTE\_READ},  
    {7, PAGE\_EXECUTE\_READWRITE},  
    {8, PAGE\_EXECUTE\_WRITECOPY}  
   };  
     
   vector<ProtectionInfo> protectionInfoList = {  
    {1, "PAGE\_NOACCESS"},  
    {2, "PAGE\_READONLY"},  
    {3, "PAGE\_READWRITE"},  
    {4, "PAGE\_WRITECOPY"},  
    {5, "PAGE\_EXECUTE"},  
    {6, "PAGE\_EXECUTE\_READ"},  
    {7, "PAGE\_EXECUTE\_READWRITE"},  
    {8, "PAGE\_EXECUTE\_WRITECOPY"}  
   };  
     
   string protectToString(DWORD protect) {  
    switch (protect) {  
    case PAGE\_NOACCESS:  
    return "No Access";  
    case PAGE\_READONLY:  
    return "Read Only";  
    case PAGE\_READWRITE:  
    return "Read/Write";  
    case PAGE\_WRITECOPY:  
    return "Write Copy";  
    case PAGE\_EXECUTE:  
    return "Execute Only";  
    case PAGE\_EXECUTE\_READ:  
    return "Execute/Read";  
    case PAGE\_EXECUTE\_READWRITE:  
    return "Execute/Read/Write";  
    case PAGE\_EXECUTE\_WRITECOPY:  
    return "Execute/Write Copy";  
    default:  
    return "Unknown";  
    }  
   }  
     
   //1  
   void printSystemInfo() {  
    SYSTEM\_INFO systemInfo;  
    GetSystemInfo(&systemInfo);  
     
    std::cout << "Processor Architecture: ";  
    switch (systemInfo.wProcessorArchitecture) {  
    case PROCESSOR\_ARCHITECTURE\_INTEL:  
    std::cout << "x86 (32-bit)" << std::endl;  
    break;  
    case PROCESSOR\_ARCHITECTURE\_AMD64:  
    std::cout << "x64 (64-bit)" << std::endl;  
    break;  
    case PROCESSOR\_ARCHITECTURE\_ARM:  
    std::cout << "ARM" << std::endl;  
    break;  
    default:  
    std::cout << "Unknown" << std::endl;  
    break;  
    }  
   }  
     
   //2  
   void printGlobalMemoryStatus() {  
    MEMORYSTATUSEX memoryStatus;  
    memoryStatus.dwLength = sizeof(memoryStatus);  
    GlobalMemoryStatusEx(&memoryStatus);  
     
    std::cout << "Total physical memory: " << memoryStatus.ullTotalPhys / 1024 / 1024 << " MB ("  
    << memoryStatus.ullTotalPhys << " bytes)" << std::endl;  
    std::cout << "Available physical memory: " << memoryStatus.ullAvailPhys / 1024 / 1024 << " MB ("  
    << memoryStatus.ullAvailPhys << " bytes)" << std::endl;  
    std::cout << "Used memory: " << memoryStatus.ullTotalPhys / 1024 / 1024 - memoryStatus.ullAvailPhys / 1024 / 1024 << " MB"<< std::endl;  
    std::cout << "Total virtual memory: " << memoryStatus.ullTotalVirtual / 1024 / 1024 << " MB ("  
    << memoryStatus.ullTotalVirtual << " bytes)" << std::endl;  
    std::cout << "Available virtual memory: " << memoryStatus.ullAvailVirtual / 1024 / 1024 << " MB ("  
    << memoryStatus.ullAvailVirtual << " bytes)" << std::endl;  
     
     
   }  
     
   //3  
   void determineSiteCondition() {  
    PVOID address;  
    std::cout << "Enter memory address (in hexadecimal): ";  
    std::cin >> address;  
     
    MEMORY\_BASIC\_INFORMATION memoryInfo;  
     
    SIZE\_T result = VirtualQuery(address, &memoryInfo, sizeof(memoryInfo));  
    if (result == sizeof(memoryInfo)) {  
    std::cout << "Region size: " << memoryInfo.RegionSize << " bytes" << std::endl;  
    std::cout << "Base address: " << memoryInfo.BaseAddress << std::endl;  
    std::cout << "Allocation base: " << memoryInfo.AllocationBase << std::endl;  
    std::cout << "Allocation Protect: " << ProtectToString(memoryInfo.AllocationProtect) << std::endl;  
    std::cout << "State: ";  
    switch (memoryInfo.State) {  
    case MEM\_COMMIT:  
    std::cout << "Committed" << std::endl;  
    break;  
    case MEM\_FREE:  
    std::cout << "Free" << std::endl;  
    break;  
    case MEM\_RESERVE:  
    std::cout << "Reserved" << std::endl;  
    break;  
    default:  
    std::cout << "Unknown" << std::endl;  
    break;  
    }  
    std::cout << "Protect: " << ProtectToString(memoryInfo.Protect) << std::endl;  
    std::cout << "Type: " << TypeToString(memoryInfo.Type) << std::endl;  
    } else {  
    std::cerr << "VirtualQuery failed." << std::endl;  
    }  
   }  
     
   //4  
   void reserveVirtualMemory() {  
    LPVOID lpAddress;  
    SIZE\_T dwSize;  
    char choose;  
     
    std::cout << "Enter size of memory region to reserve (in bytes): ";  
    std::cin >> dwSize;  
     
    std::cout << "Manual allocation? y/n: ";  
    std::cin >> choose;  
     
    if (choose == 'y') {  
    std::cout << "Enter memory address to reserve: ";  
    std::cin >> std::hex >> lpAddress;  
    LPVOID lpMemory = VirtualAlloc(lpAddress, dwSize, MEM\_RESERVE, PAGE\_READWRITE);  
    if (lpMemory == nullptr) {  
    std::cerr << "Failed to allocate memory region." << std::endl;  
    return;  
    }  
     
    std::cout << "Memory region allocated at address: " << lpMemory << std::endl;  
    VirtualFree(lpMemory, 0, MEM\_DECOMMIT);  
    } else if (choose == 'n') {  
    LPVOID lpMemory = VirtualAlloc(nullptr, dwSize, MEM\_RESERVE, PAGE\_READWRITE);  
    if (lpMemory == nullptr) {  
    std::cerr << "Failed to allocate memory region." << std::endl;  
    return;  
    }  
     
    std::cout << "Memory region allocated at address: " << lpMemory << std::endl;  
    VirtualFree(lpMemory, 0, MEM\_DECOMMIT);  
    } else {  
    std::cerr << "wrong symbol" << std::endl;  
    }  
   }  
     
   //5  
   void reserveAndUseVirtualMemory() {  
    LPVOID lpAddress;  
    SIZE\_T dwSize;  
    char choose;  
     
    std::cout << "Enter size of memory region to reserve (in bytes): ";  
    std::cin >> dwSize;  
     
    std::cout << "Manual allocation? y/n: ";  
    std::cin >> choose;  
     
    if (choose == 'y' || choose == 'Y') {  
    std::cout << "Enter memory address to reserve: ";  
    std::cin >> std::hex >> lpAddress;  
    LPVOID lpMemory = VirtualAlloc(lpAddress, dwSize, MEM\_COMMIT, PAGE\_READWRITE);  
    if (lpMemory == nullptr) {  
    std::cerr << "Failed to allocate memory region." << std::endl;  
    return;  
    }  
     
    std::cout << "Memory region allocated at address: " << lpMemory << std::endl;  
   // VirtualFree(lpMemory, 0, MEM\_RELEASE);  
    } else if (choose == 'n' || choose == 'N') {  
    LPVOID lpMemory = VirtualAlloc(nullptr, dwSize, MEM\_COMMIT, PAGE\_READWRITE);  
    if (lpMemory == nullptr) {  
    std::cerr << "Failed to allocate memory region." << std::endl;  
    return;  
    }  
     
    std::cout << "Memory region allocated at address: " << lpMemory << std::endl;  
   // VirtualFree(lpMemory, 0, MEM\_RELEASE);  
    } else {  
    std::cerr << "wrong symbol" << std::endl;  
    }  
   }  
     
   void readData(){  
    uintptr\_t address;  
    std::cout << "Enter memory address to read from: ";  
    std::cin >> std::hex >> address;  
     
    HANDLE hProcess = GetCurrentProcess();  
     
    const int bufferSize = 256;  
    char buffer[bufferSize];  
     
    SIZE\_T bytesRead;  
    if (ReadProcessMemory(hProcess, (LPCVOID)address, buffer, bufferSize, &bytesRead) && bytesRead > 0) {  
    std::cout << "Data read from memory: " << std::string(buffer, bytesRead) << std::endl;  
    } else {  
    std::cerr << "Failed to read data from memory." << std::endl;  
    }  
     
    CloseHandle(hProcess);  
   }  
     
   //6  
   void writeData() {  
    uintptr\_t address;  
    std::cout << "Enter memory address to write data to: ";  
    std::cin >> std::hex >> address;  
     
    if (std::cin.fail()) {  
    std::cerr << "Invalid memory address." << std::endl;  
    return;  
    }  
     
    std::string data;  
    std::cout << "Enter data to write to memory: ";  
    std::cin.ignore();  
    std::getline(std::cin, data);  
     
    HANDLE hProcess = GetCurrentProcess();  
     
    auto lpAddress = (LPVOID) address;  
     
    SIZE\_T bytesWritten;  
    if (WriteProcessMemory(hProcess, lpAddress, data.c\_str(), data.size(), &bytesWritten) && bytesWritten == data.size()) {  
    std::cout << "Data successfully written to memory." << std::endl;  
    } else {  
    std::cerr << "Failed to write data to memory." << std::endl;  
    }  
     
    CloseHandle(hProcess);  
    cout<<"Do you want to read written data? (y/n): ";  
    char choose;  
    std::cin >> choose;  
    if (choose =='y') readData();  
   }  
     
   //7  
   void setProtect() {  
    PVOID address;  
    std::cout << "Enter memory address (in hexadecimal): ";  
    std::cin >> std::hex >> address;  
     
    SIZE\_T size;  
    std::cout << "Enter memory size: ";  
    std::cin >> size;  
     
    std::cout << "Choose protection attribute:" << std::endl;  
    for (const auto &info: protectionInfoList) {  
    std::cout << info.number << ". " << info.description << std::endl;  
    }  
     
    int choice;  
    std::cout << "Enter choice: ";  
    std::cin >> choice;  
     
    if (protectMap.find(choice) == protectMap.end()) {  
    std::cerr << "Invalid choice." << std::endl;  
    return;  
    }  
     
    DWORD protect = protectMap[choice];  
    DWORD oldProtect;  
    if (VirtualProtect(address, size, protect, &oldProtect)) {  
    std::cout << "Protection changed successfully." << std::endl;  
    std::cout << "Old protection: " << protectToString(oldProtect) << std::endl;  
    } else {  
    std::cerr << "VirtualProtect failed." << std::endl;  
     
    }  
     
   // Проверяем текущие атрибуты защиты доступа  
    MEMORY\_BASIC\_INFORMATION memoryInfo;  
    SIZE\_T result = VirtualQuery(address, &memoryInfo, sizeof(memoryInfo));  
    if (result != 0) {  
    std::cout << "Current protection: " << protectToString(memoryInfo.Protect) << std::endl;  
    } else {  
    std::cerr << "VirtualQuery failed." << std::endl;  
    return;  
    }  
     
   }  
     
   //8  
   void freeMemory() {  
    LPVOID lpAddress;  
    SIZE\_T dwSize;  
     
    std::cout << "Enter memory address to free: ";  
    std::cin >> std::hex >> lpAddress;  
     
    std::cout << "Enter size of memory region to free (in bytes): ";  
    std::cin >> dwSize;  
     
    VirtualFree(lpAddress, dwSize, MEM\_DECOMMIT);  
    std::cout << "Physical memory released." << std::endl;  
     
    VirtualFree(lpAddress, dwSize, MEM\_RELEASE);  
    std::cout << "Address region released." << std::endl;  
   }  
     
   int main() {  
    int action;  
    bool running = true;  
     
    while (running) {  
    printMenu();  
    std::cin >> action;  
    switch (action) {  
    case 1:  
    printSystemInfo();  
    system("pause");  
    break;  
    case 2:  
    printGlobalMemoryStatus();  
    system("pause");  
    break;  
    case 3:  
    determineSiteCondition();  
    system("pause");  
    break;  
    case 4:  
    reserveVirtualMemory();  
    system("pause");  
    break;  
    case 5:  
    reserveAndUseVirtualMemory();  
    system("pause");  
    break;  
    case 6:  
    writeData();  
    system("pause");  
    break;  
    case 7:  
    setProtect();  
    system("pause");  
    break;  
    case 8:  
    freeMemory();  
    system("pause");  
    break;  
    default:  
    running = false;  
    break;  
    }  
    system("cls");  
    }  
     
    return 0;  
   }

## Теория по первой части.

Виртуальная память — это концепция, которая позволяет уйти от использования физических адресов, используя вместо них виртуальные, и это даёт ряд преимуществ:

Расширение реального адресного пространства. Часть виртуальной памяти может быть вытеснена на жёсткий диск, и это позволяет программам использовать больше оперативной памяти, чем есть на самом деле.

Создание изолированных адресных пространств для различных процессов, что повышает безопасность системы, а также решает проблему привязанности программы к определённым адресам памяти.

Задание различных свойств для разных участков участков памяти. Например, может существовать неизменяемый участок памяти, видный нескольким процессам.  
При этом вся виртуальная память делится на участки памяти постоянного размера, называемые страницами.

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.

## Выводы по первой части.

В ходе выполнения задания было разработано консольное приложение на языке C++ с использованием библиотеки WinAPI для взаимодействия с операционной системой Windows. Приложение предоставляет меню с различными функциями, каждая из которых выполняет определенное действие, используя функции Win32 API.

* Получение информации о вычислительной системе:

Используется функция GetSystemInfo для получения информации о системе, такой как архитектура процессора и размер страницы памяти.

* Определение статуса виртуальной памяти:

Для этого применяется функция GlobalMemoryStatus, которая предоставляет информацию о доступной физической памяти.

* Определение состояния конкретного участка памяти:

Используется функция VirtualQuery, позволяющая получить информацию о состоянии участка памяти по заданному адресу.

* Резервирование региона в автоматическом режиме и в режиме ввода адреса начала региона:

Для этого применяется функция VirtualAlloc, позволяющая выделить регион виртуальной памяти с заданными параметрами.

* Резервирование региона и передача ему физической памяти:

Также используется функция VirtualAlloc, но с определенными флагами, позволяющими выделить регион и назначить ему физическую память.

* Запись данных в ячейки памяти по заданным адресам:

Пользователю предоставляется возможность ввода адреса и данных для записи, после чего используется операция записи.

* Установка защиты доступа для заданного региона памяти и ее проверка:

Функция VirtualProtect позволяет изменить защиту доступа для указанного региона памяти, а также можно выполнить проверку текущих прав доступа.

* Возврат физической памяти и освобождение региона адресного пространства:

Для этого используется функция VirtualFree, которая освобождает регион адресного пространства и возвращает выделенную память.

Приложение было протестировано на нескольких наборах входных данных, результаты работы были успешно протоколированы и проверены на корректность.

В процессе выполнения работы были изучены и применены различные функции Win32 API для работы с памятью и информацией о системе. Это позволило более глубоко понять принципы работы с виртуальной памятью и системными ресурсами в операционной системе Windows.

# Задание на проецируемый файлы. Два приложения – клиент и сервер.

Сервер создает файл на диске и проецирует его в память, далее записывает информацию и ждет,

пока клиент не прочтет, затем отменяет проецирование и удаляет файл.

Клиент открывает файл, проецирует и ждет доступности чтения, затем читает и выводит результат,

в завершении работы отменяет проецирование.

Сделать меню в каждой программе:

 пункты меню Сервера – «выполнить проецирование», «записать данные», «завершить

работу»;

 пункты меню Клиента – «выполнить проецирование», «прочитать данные», «завершить

работу».

## Код программы Сервера(писателя):

//  
// Created by stukenvitalii on 4/1/24.  
//  
#include <fcntl.h>  
#include <unistd.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <cstdio>  
#include <cstring>  
#include <cstdlib>  
#include <iostream>  
  
#define FILENAME "shared\_memory\_file.txt"  
#define FILESIZE sizeof(char) \* 100  
  
void createAndMapFile(int &fd, char \*\*ptr);  
  
void writeToMemory(char \*ptr);  
  
void unmapAndDeleteFile(int fd, char \*ptr);  
  
void printMenu();  
  
int main() {  
 int choice;  
 int fd;  
 char \*ptr = nullptr;  
 do {  
 printMenu();  
 std::cin >> 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(nullptr, FILESIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
 if (\*ptr == MAP\_FAILED) {  
 perror("Error mapping file");  
 close(fd);  
 exit(1);  
 }  
 memset(\*ptr, 0, FILESIZE);  
}  
  
void writeToMemory(char \*ptr) {  
 std::string input;  
 std::cout << "Enter message: ";  
 std::getchar();  
 std::getline(std::cin, input);  
  
 sprintf(ptr, "%s", input.c\_str());  
 printf("Data written to memory.\n");  
}  
  
void unmapAndDeleteFile(int fd, char \*ptr) {  
 munmap(ptr, FILESIZE);  
 close(fd);  
 unlink(FILENAME);  
}  
  
void printMenu(){  
 std::cout << "--------------------------\n";  
 std::cout  
 << " Hello! I'm server app, choose option: \n"  
 << " 1) create file mapping\n"  
 << " 2) write data\n"  
 << " 3) exit\n";  
 std::cout << "--------------------------\n";  
}

## Код программы Клиента(читателя):

//  
// Created by stukenvitalii on 4/1/24.  
//  
#include <iostream>  
#include <fcntl.h>  
#include <unistd.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <cstdio>  
#include <cstdlib>  
#define FILENAME "shared\_memory\_file.txt"  
#define FILESIZE sizeof(char) \* 100  
  
typedef struct MappedFile {  
 char\* ptr;  
 int fd;  
 size\_t size;  
} MappedFile;  
  
MappedFile mapFile();  
  
void readMappedFile(MappedFile mappedFile);  
  
void printMenu();  
  
int main() {  
 int choice;  
 do {  
 printMenu();  
 std::cin >> choice;  
 switch (choice) {  
 case 1:  
 mapFile();  
 break;  
 case 2: {  
 auto mapped = mapFile();  
 readMappedFile(mapped);  
 break;  
 }  
 case 3:  
 printf("Exiting program.\n");  
 break;  
 default:  
 printf("Invalid choice, please try again.\n");  
 }  
 } while (choice != 3);  
 return 0;  
}  
  
MappedFile mapFile() {  
 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(nullptr, st.st\_size, PROT\_READ, MAP\_SHARED, fd,0);  
  
 if (ptr == MAP\_FAILED) {  
 perror("Error mapping file");  
 close(fd);  
 exit(1);  
 }  
  
 MappedFile mappedFile{};  
 mappedFile.fd = fd;  
 mappedFile.ptr = ptr;  
 mappedFile.size = st.st\_size;  
  
 std::cout << "Got mapped file successfully" << std::endl;  
  
 return mappedFile;  
}  
  
void readMappedFile(MappedFile mappedFile) {  
 printf("Client received: %s\n", mappedFile.ptr);  
 munmap(mappedFile.ptr, mappedFile.size);  
 close(mappedFile.fd);  
}  
  
void printMenu() {  
 std::cout << "--------------------------\n";  
 std::cout  
 << " Hello! I'm client app, choose option: \n"  
 << " 1) perform projection\n"  
 << " 2) read data\n"  
 << " 3) exit\n";  
 std::cout << "--------------------------\n";  
}

## Выводы по второй части

Проецируемые файлы представляют собой механизм работы с памятью, при котором файлы отображаются непосредственно в виртуальное адресное пространство программы. Это означает, что страницы в виртуальной памяти программы отображаются на соответствующие физические страницы, занимаемые файлом. Таким образом, доступ к данным в виртуальной памяти осуществляется непосредственно через файл, что позволяет выполнять чтение и запись данных в файл, а также изменять его содержимое, используя логические (виртуальные) адреса.

# Источники

1. Операционные системы: электронные методические указания к лабораторным работам / Сост.: А. В. Тимофеев. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2016.

2. Курс «Операционные системы» в образовательной онлайн-системе Google Класс. Материалы лекций.