**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

# Кафедра Вычислительной техники

**ОТЧЕТ**

# по лабораторной работе № 2.2

**по дисциплине «Операционные системы» Тема: «Управление файловой системой»**

Студент гр. 3311

Пасечный

Л.В.

Преподаватель

Тимофеев

А. В.

Санкт-Петербург

2024

# Введение

**Цель работы:**

Исследовать механизмы управления виртуальной памятью

1)

# Постановка задачи:

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); −

раздельное резервирование региона и передачу ему физической

памяти в автоматическом режиме и в режиме ввода адреса начала

региона (функция Win32 API – VirtualAlloc, VirtualFree); −

одновременное резервирование региона и передача ему физической

памяти в автоматическом режиме и в режиме ввода адреса начала

региона (функция Win32 API – VirtualAlloc, VirtualFree);

− 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). 2. Запустите

приложение и проверьте его работоспособность на нескольких наборах

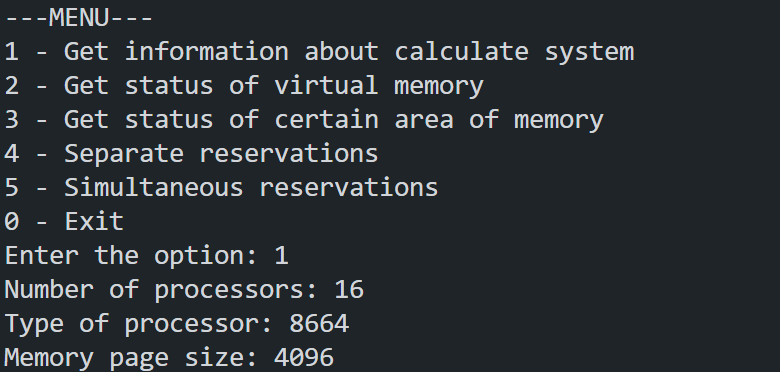
вводимых данных. Запротоколируйте результаты в отчет. Дайте свои

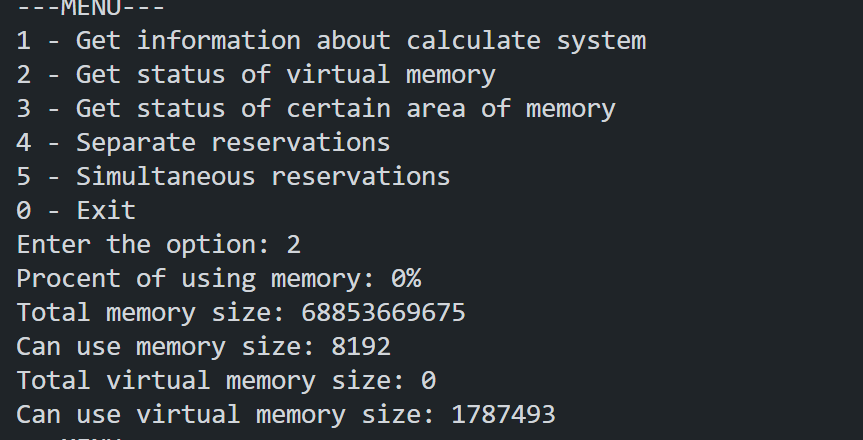
комментарии в отчете относительно выполнения функций Win32 API.

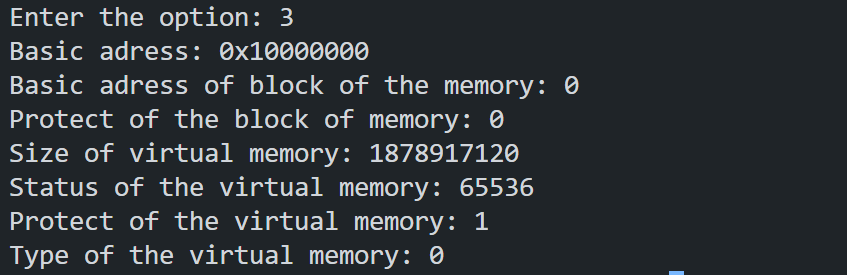
2. Подготовьте итоговый отчет с развернутыми выводами по

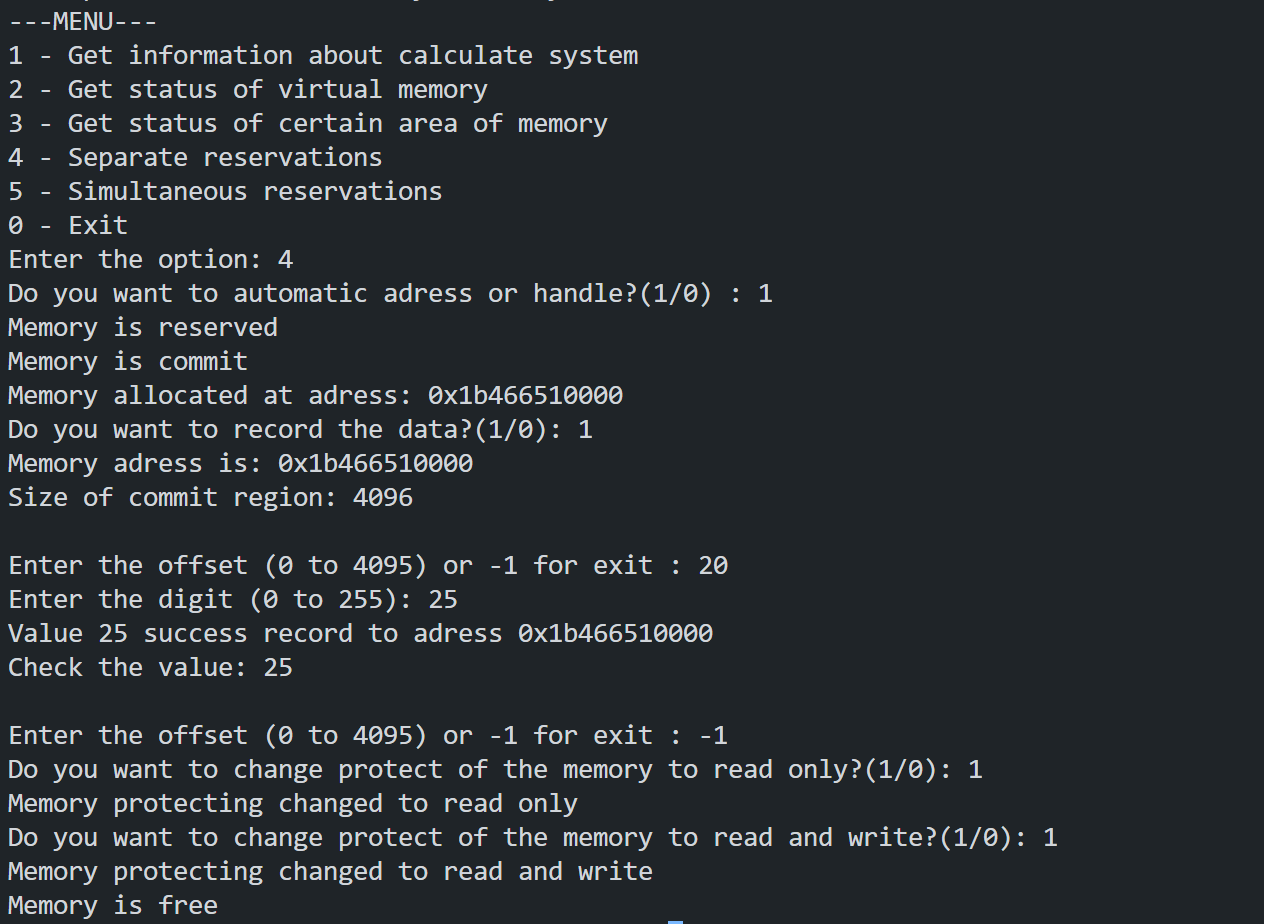
# заданию.

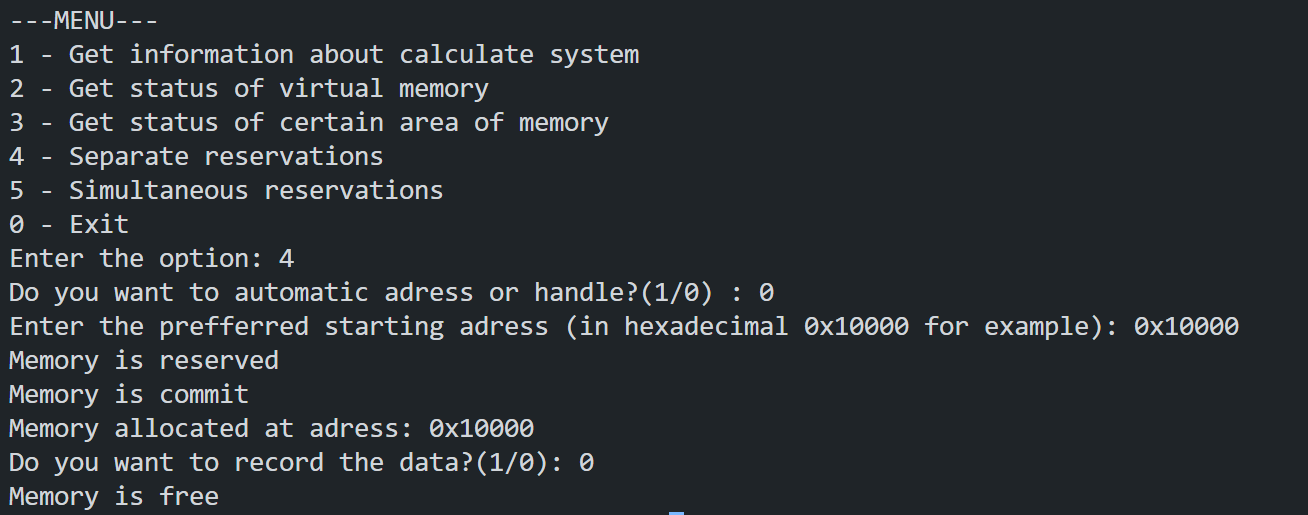
# Результаты:

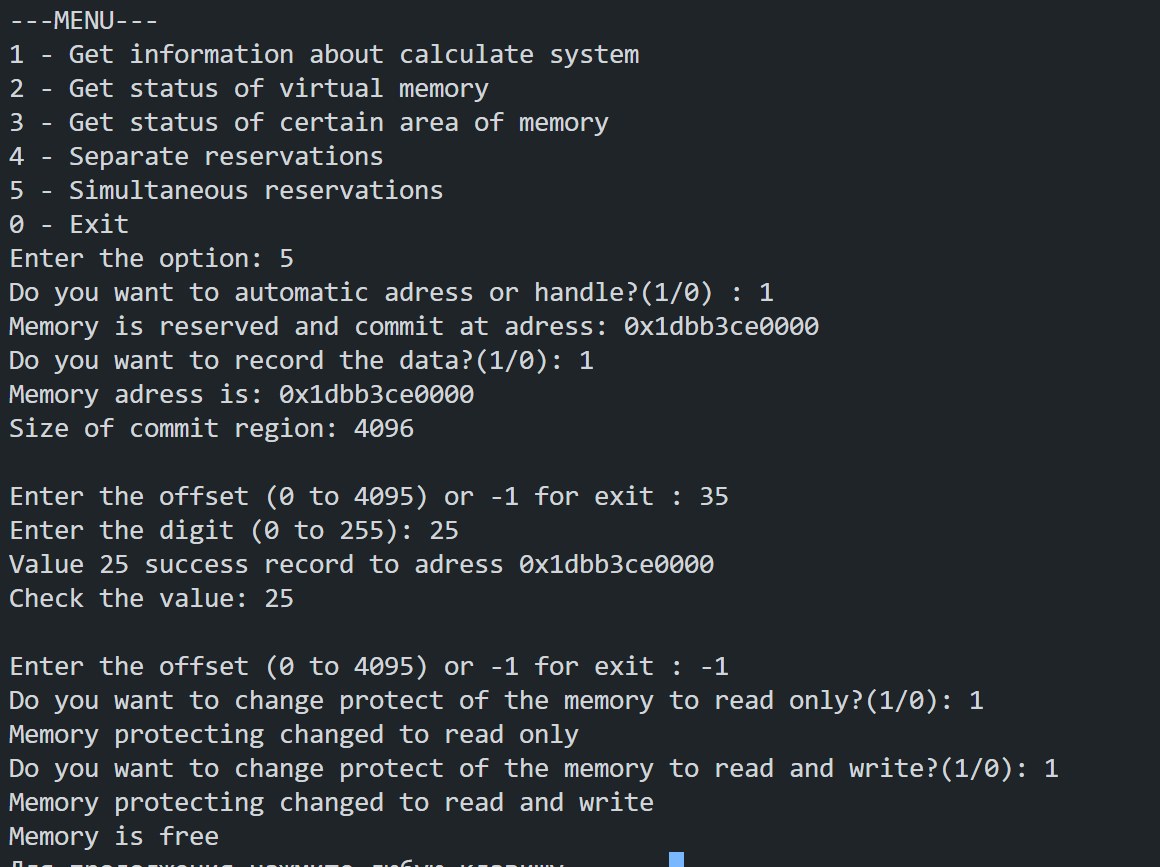












**Заключение**

Программа демонстрирует работу с системной и виртуальной памятью Windows, используя API Win32. Включены функции для получения информации о системе, состоянии памяти, а также для резервирования, выделения, защиты и освобождения участков виртуальной памяти.

# Код программы

# Server:

#include <iostream>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <string.h>

#define FILENAME "shared\_mem.bin"

#define FILESIZE 1024

int fd = -1;

char \*ptr = NULL;

void create\_mapping(){

fd = open(FILENAME, O\_RDWR | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR);

if (fd == -1){

perror("Ошибка при создании файла");

exit(EXIT\_FAILURE);

}

// Установка размеров файла

if (ftruncate(fd, FILESIZE) == -1){

perror("Ошибка установки размеров файла");

exit(EXIT\_FAILURE);

}

// Проецирование файла в память

ptr = (char\*)mmap(NULL, FILESIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (ptr == MAP\_FAILED){

perror("Ошибка проецирования файла");

close(fd);

exit(EXIT\_FAILURE);

}

printf("Файл успешно спроецирован в память\n");

}

void write\_data(){

if (!ptr){

printf("Сначала выполните проецирование!");

return;

}

//Запись данных

sprintf(ptr, "Hello, shared memory!");

printf("Данные записаны в общую память\n");

}

void cleanup(){

if (ptr != NULL){

munmap(ptr, FILESIZE);

ptr = NULL;

}

if (fd == -1){

close(fd);

unlink(FILENAME);

fd = -1;

}

printf("Ресурсы освобождены. Файл удалён.\n");

}

void menu(){

printf("----Menu----\n");

printf("1 - Создать проекцию\n");

printf("2 - Записать данные\n");

printf("3 - Завершить работу\n");

}

int main(){

int choice = 0;

do{

menu();

scanf("%d", &choice);

switch(choice){

case 1:

create\_mapping();

break;

case 2:

write\_data();

break;

case 3:

cleanup();

break;

default:

printf("Попробуйте ещё раз\n");

break;

}

}while (choice != 3);

return 0;

}

**Client:**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/select.h>

#define FILENAME "shared\_mem.bin"

#define FILESIZE 1024

int fd = -1;

char \*ptr = NULL;

void open\_mapping(){

fd = open(FILENAME, O\_RDONLY);

if (fd == -1){

perror("Ошибка открытия файла");

exit(EXIT\_FAILURE);

}

// Проецирование файла

ptr = (char\*)mmap(NULL, FILESIZE, PROT\_READ, MAP\_SHARED, fd, 0);

if (ptr == MAP\_FAILED){

perror("Ошибка проецйирования файла");

close(fd);

exit(EXIT\_FAILURE);

}

printf("Файл успешно спроецирован\n");

}

void read\_data(){

if (!ptr){

printf("Сначала выполните проецирование!\n");

return;

}

//Ожидание доступности данных

fd\_set read\_fds;

FD\_ZERO(&read\_fds);

FD\_SET(fd, &read\_fds);

struct timeval timeout = {5, 0}; // Таймаут 5 секунд

int ready = select(fd + 1, &read\_fds, NULL, NULL, &timeout);

if (ready == -1){

perror("Ошибка select");

exit(EXIT\_FAILURE);

} else if (ready == 0){

printf("Данные недоступны (таймаут)\n");

return;

}

//Чтение данных

printf("Полученные данные: %s\n", ptr);

}

void cleanup(){

if (ptr != NULL){

munmap(ptr, FILESIZE);

ptr = NULL;

}

if (fd == -1){

close(fd);

fd = -1;

}

printf("Ресурсы клиента освобождены\n");

}

void menu(){

printf("---Menu----\n");

printf("1 - Открыть проекцию\n");

printf("2 - Прочитать данные\n");

printf("3 - Выход\n");

}

int main(){

int choice = 0;

do{

menu();

scanf("%d", &choice);

switch(choice){

case 1:

open\_mapping();

break;

case 2:

read\_data();

break;

case 3:

cleanup();

break;

default:

printf("Попробуйте ещё раз!");

break;

}

}while (choice != 3);

return 0;

}

2)

# Постановка задачи:

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.

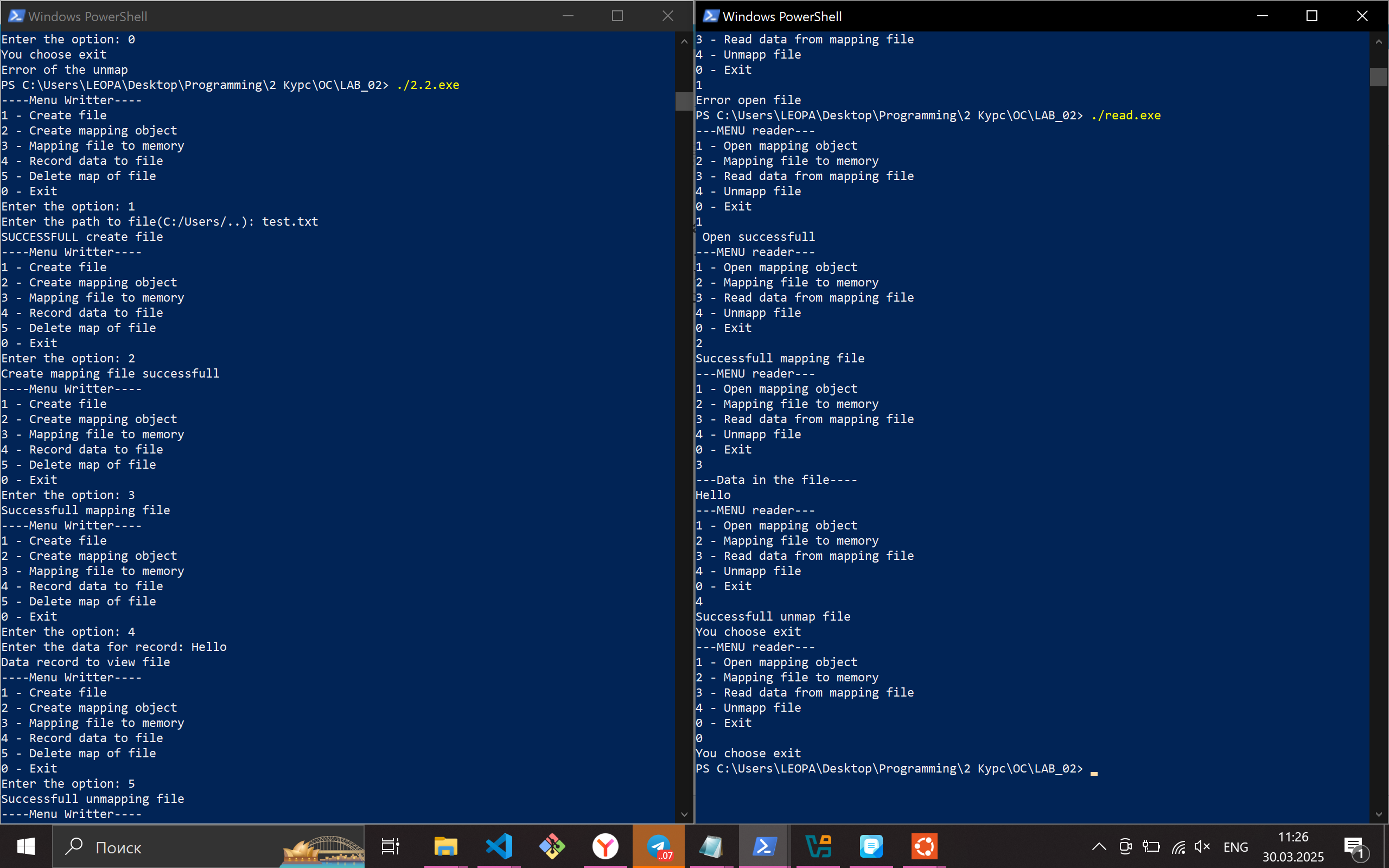
Запротоколируйте результаты в отчет. Дайте свои комментарии в

отчете относительно выполнения функций Win32 API.

3. Подготовьте итоговый отчет с развернутыми выводами по

заданию.

# Результаты:



**Код:**

1. **Писатель**

#include<Windows.h>

#include<iostream>

#include<string>

#define FILE\_SIZE 1024

using namespace std;

HANDLE hFile = INVALID\_HANDLE\_VALUE;

HANDLE hMapping = nullptr;

LPVOID pView = nullptr;

void menu(){

wcout << L"----Menu Writter----" << endl;

wcout << L"1 - Create file" << endl;

wcout << L"2 - Create mapping object" << endl;

wcout << L"3 - Mapping file to memory" << endl;

wcout << L"4 - Record data to file" << endl;

wcout << L"5 - Delete map of file" << endl;

wcout << L"0 - Exit" << endl;

}

void create\_file(){

wstring path;

wcout << "Enter the path to file(C:/Users/..): ";

wcin.ignore();

getline(wcin, path);

hFile = CreateFileW(path.c\_str(),

GENERIC\_READ | GENERIC\_WRITE,

0,

NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

if (hFile == INVALID\_HANDLE\_VALUE){

wcerr << L"Error openning file: " << GetLastError() << endl;

CloseHandle(hFile);

exit(EXIT\_FAILURE);

}

cout << "SUCCESSFULL create file" << endl;

}

void create\_mapping\_file(){

hMapping = CreateFileMappingW(hFile, NULL, PAGE\_READWRITE, 0, FILE\_SIZE, L"MyMappingObj");

if (hMapping == NULL){

cerr << "File mapping error: " << GetLastError() << endl;

exit(EXIT\_FAILURE);

}

cout << "Create mapping file successfull" << endl;

}

void MapViewOfFilefunc(){

if (hMapping == nullptr){

cerr << "ERROR of open hMapping file" << endl;

exit(EXIT\_FAILURE);

}

if (pView != nullptr){

cerr << "This file already view" << endl;

exit(EXIT\_FAILURE);

}

pView = MapViewOfFile(hMapping, FILE\_MAP\_ALL\_ACCESS, 0, 0, FILE\_SIZE);

if (pView == nullptr){

cerr << "ERROR map view if file" << endl;

exit(EXIT\_FAILURE);

}

cout << "Successfull mapping file" << endl;

}

void writeData(){

if (pView == nullptr){

cerr << "File not view" << endl;

return;

}

cout << "Enter the data for record: ";

string input;

cin.ignore();

getline(cin, input);

if (input.length() >= FILE\_SIZE) {

cerr << "Too big input data" << endl;

return;

}

memcpy(pView, input.c\_str(), input.length() + 1);

cout << "Data record to view file" << endl;

}

void unmapView(){

if (pView == nullptr){

cerr << "File not view" << endl;

return;

}

if (UnmapViewOfFile(pView) == 0){

cerr << "Error of the unmap" << endl;

return;

}else{

cout << "Successfull unmapping file" << endl;

pView = nullptr;

}

}

void closeHandle(){

if (pView != nullptr){

unmapView();

}

if (hMapping != nullptr){

CloseHandle(hMapping);

hMapping = nullptr;

}

if (hFile != nullptr){

CloseHandle(hFile);

hFile = nullptr;

}

}

int main(){

int choice;

do{

menu();

cout << "Enter the option: ";

cin >> choice;

switch(choice){

case 1:

create\_file();

break;

case 2:

create\_mapping\_file();

break;

case 3:

MapViewOfFilefunc();

break;

case 4:

writeData();

break;

case 5:

unmapView();

break;

case 0:

cout << "You choose exit" << endl;

break;

default:

cout << "Try again." << endl;

break;

}

}while(choice != 0);

closeHandle();

return 0;

}

1. **Читатель**

#include <windows.h>

#include <iostream>

#include <string>

using namespace std;

#define FILE\_SIZE 1024

HANDLE hFileMapping = nullptr;

LPVOID pView = NULL;

void menu(){

cout << "---MENU reader---" << endl;

cout << "1 - Open mapping object" << endl;

cout << "2 - Mapping file to memory" << endl;

cout << "3 - Read data from mapping file" << endl;

cout << "4 - Unmapp file" << endl;

cout << "0 - Exit" << endl;

}

void OpenFileMappingfunc(){

if (hFileMapping != nullptr){

cerr << "File already open" << endl;

exit(EXIT\_FAILURE);

}

hFileMapping = OpenFileMappingW(

FILE\_MAP\_READ,

FALSE,

L"MyMappingObj"

);

if (hFileMapping == nullptr){

cerr << "Error open file" << endl;

exit(EXIT\_FAILURE);

}else{

cout << " Open successfull" << endl;

}

}

void mapView(){

if (hFileMapping == nullptr){

cerr << "File not mapping" << endl;

exit(EXIT\_FAILURE);

}

if (pView != nullptr){

cerr << "File already view" << endl;

exit(EXIT\_FAILURE);

}

pView = MapViewOfFile(hFileMapping, FILE\_MAP\_READ, 0, 0, FILE\_SIZE);

if (pView == nullptr){

cerr << "Error of view map file" << endl;

exit(EXIT\_FAILURE);

}else{

cout << "Successfull mapping file" << endl;

}

}

void readData(){

if (pView == nullptr){

cerr << "File not view" << endl;

exit(EXIT\_FAILURE);

}

char\* data = static\_cast<char\*>(pView);

cout << "---Data in the file----" << endl;

cout << data << endl;

}

void unmapView() {

if (pView == nullptr){

cerr << "File not view" << endl;

exit(EXIT\_FAILURE);

}

if (UnmapViewOfFile(pView) == 0){

cerr << "Error of file unmapping" << endl;

exit(EXIT\_FAILURE);

}else{

cout << "Successfull unmap file" << endl;

pView = nullptr;

}

}

void closeHandle(){

if (pView != nullptr){

unmapView();

}

if (hFileMapping != nullptr){

CloseHandle(hFileMapping);

hFileMapping = nullptr;

}

}

int main(){

int option;

do{

menu();

cin >> option;

switch(option){

case 1:

OpenFileMappingfunc();

break;

case 2:

mapView();

break;

case 3:

readData();

break;

case 4:

unmapView();

case 0:

cout << "You choose exit" << endl;

break;

default:

cout << "Try again" << endl;

}

}while (option != 0);

closeHandle();

return 0;

}