**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»**

Студентки гр. 3312 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Половникова А.С.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимофеев А.В.

Санкт-Петербург

2025

**Цель работы.**

исследовать механизмы управления виртуальной памятью Win32.

**Задание 2.1.**

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); − 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

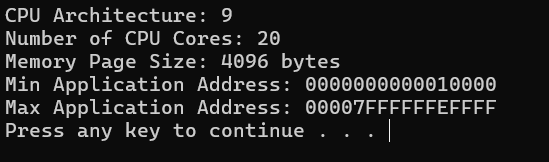
комментарии в отчете относительно выполнения функций Win32 API.

2. Подготовьте итоговый отчет с развернутыми выводами по заданию.

**Отчет о результатах работы функций Win32 API**

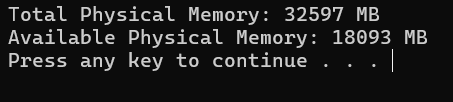
**Вывод системной информации**

Функция *GetSystemInfo* позволяет определить архитектуру процессора, количество ядер, размер страницы памяти, минимальный и максимальный адреса приложения.



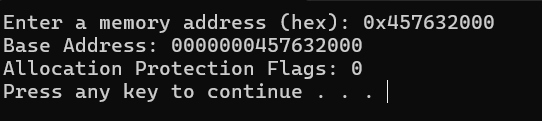
**Вывод статуса памяти**

Функция *GlobalMemoryStatus* выводит общий объем и доступную физическую память. Значения отображаются в мегабайтах.



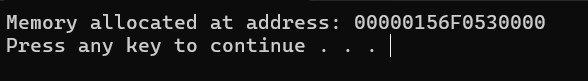
**Запрос информации об области памяти**

Функция *VirtualQuery* позволяет получить информацию об указанной области памяти, включая базовый адрес и защитные флаги.



**Резерв и выделение памяти**

Функция *VirtualAlloc* выделяет и зарезервирует 1 МБ памяти, после чего она освобождается с помощью *VirtualFree*.



**Изменение защиты области памяти**

Функция *VirtualProtect* принимает адрес и меняет его права доступа на PAGE\_READONLY. При успешном выполнении выводит сообщение о успешном обновлении защиты.

**Вывод**

Все функции Win32 API выполняются согласно ожидаемому результату, без ошибок или замечаний.

**Текст программы**

*Main.cpp*

#include <iostream>

#include <windows.h>

void system\_info()

{

SYSTEM\_INFO si;

GetSystemInfo(&si);

std::cout << "CPU Architecture: " << si.wProcessorArchitecture << "\n";

std::cout << "Number of CPU Cores: " << si.dwNumberOfProcessors << "\n";

std::cout << "Memory Page Size: " << si.dwPageSize << " bytes\n";

std::cout << "Min Application Address: " << si.lpMinimumApplicationAddress << "\n";

std::cout << "Max Application Address: " << si.lpMaximumApplicationAddress << "\n";

}

void protect\_memory()

{

void\* address;

DWORD oldProtect;

std::cout << "Enter the memory address to modify protection (hex): ";

std::cin >> address;

if (VirtualProtect(address, 4096, PAGE\_READONLY, &oldProtect))

std::cout << "Memory protection updated successfully!\n";

else

std::cerr << "Error: Failed to update memory protection!\n";

}

void query\_memory\_status()

{

void\* address;

MEMORY\_BASIC\_INFORMATION mbi;

std::cout << "Enter a memory address (hex): ";

std::cin >> address;

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

{

std::cout << "Base Address: " << mbi.BaseAddress << "\n";

std::cout << "Allocation Protection Flags: " << mbi.AllocationProtect << "\n";

}

else

std::cerr << "Error: Memory query failed!\n";

}

void memory\_status()

{

MEMORYSTATUS ms;

GlobalMemoryStatus(&ms);

std::cout << "Total Physical Memory: " << ms.dwTotalPhys / (1024 \* 1024) << " MB\n";

std::cout << "Available Physical Memory: " << ms.dwAvailPhys / (1024 \* 1024) << " MB\n";

}

void reserve\_and\_commit()

{

SIZE\_T size = 1024 \* 1024;

LPVOID mem = VirtualAlloc(NULL, size, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE);

if (mem)

{

std::cout << "Memory allocated at address: " << mem << "\n";

VirtualFree(mem, 0, MEM\_RELEASE);

}

else

std::cerr << "Error: Memory allocation failed!\n";

}

int main()

{

int choice;

do

{

std::cout << "\tMenu:\n";

std::cout << "1. Show System Information\n";

std::cout << "2. Show Memory Status\n";

std::cout << "3. Query Memory Information\n";

std::cout << "4. Allocate and Commit Memory\n";

std::cout << "5. Modify Memory Protection\n";

std::cout << "0. Exit\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

switch (choice)

{

case 1: system("cls"); system\_info(); system("pause"); system("cls"); break;

case 2: system("cls"); memory\_status(); system("pause"); system("cls"); break;

case 3: system("cls"); query\_memory\_status(); system("pause"); system("cls"); break;

case 4: system("cls"); reserve\_and\_commit(); system("pause"); system("cls"); break;

case 5: system("cls"); protect\_memory(); system("pause"); system("cls"); break;

case 0: std::cout << "Exiting the program...\n"; break;

default: std::cout << "Invalid option! Please try again.\n";

}

} while (choice != 0);

return 0;

}