**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

**отчёт   
ЛАБОРАТОРНАЯ РАБОТА № 2**

**по дисциплине «Операционные Системы»**

**Тема: Управление памятью**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 |  | Сапронов К.Д. |
| Преподаватель |  | Тимофеев А.В. |

Санкт-Петербург

2025

**Цель работы:**

Исследовать механизмы управления виртуальной памятью Win32.

**Задание 1.1** Исследование виртуального адресного пространства процесса.

Создать консольное приложение с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которое выполняет:

− получение информации о вычислительной;

− определение статуса виртуальной;

− определение состояния конкретного участка памяти по заданному с клавиатуры;

− раздельное резервирование региона и передачу ему физической памяти в автоматическом режиме и в режиме ввода адреса начала региона;

− одновременное резервирование региона и передача ему физической памяти в автоматическом режиме и в режиме ввода адреса начала региона;

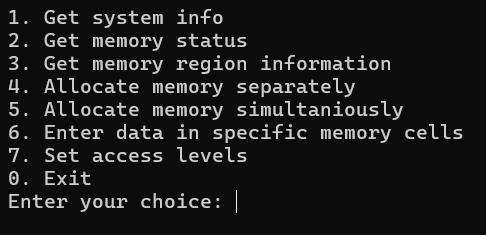
− запись данных в ячейки памяти по заданным с клавиатуры адресам;

− установку защиты доступа для заданного (с клавиатуры) региона памяти и ее проверку.

Запустить приложение и проверить его работоспособность на нескольких наборах вводимых данных. Запротоколировать результаты в отчет. Дать комментарии относительно выполнения функций Win32 API.

**Работа программы:**

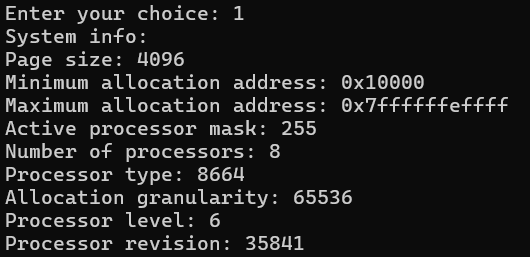
При запуске программы пользователь видит главное меню. В нем может выбрать одну из функций программы (опции 1-7) или выйти из нее (опция 0).



1. **Получение информации о системе**

Функция GetSystemInfo заполняет структуру SYSTEM\_INFO, предоставляя ключевую информацию о системе:

* dwPageSize: Размер страницы памяти. Это минимальная единица, которой оперирует менеджер памяти. При выделении памяти через VirtualAlloc размер всегда округляется вверх до кратного этому значению.
* dwAllocationGranularity: Гранулярность выделения - минимальный адресный блок, который можно зарезервировать.
* lpMinimum/MaximumApplicationAddress: Границы пользовательского адресного пространства.

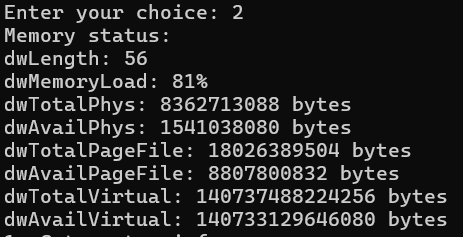


**2. Получение состояния памяти:**

Функция GlobalMemoryStatus возвращает:

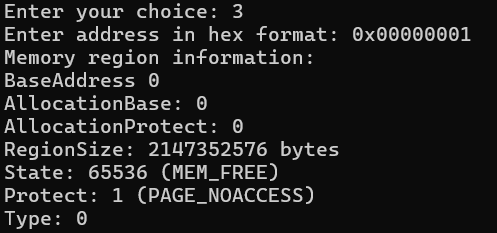
• dwMemoryLoad – процент использования физической памяти

• dwTotalPhys/dwAvailPhys – состояние физической памяти  
• dwTotalVirtual/dwAvailVirtual - состояние виртуальной памяти  
• dwTotalPageFile/dwAvailPageFile – состояние файла подкачки



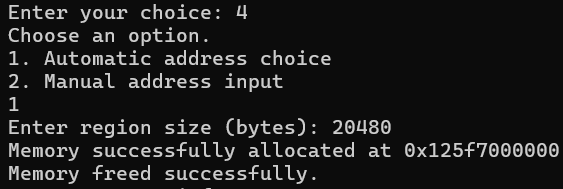
**3. Анализ участка памяти:**

С поиощью VirtualQuery получаем данные о регионе памяти. Ядро хранит базовый адрес, размер, состояние, защиту и тип каждого региона памяти.



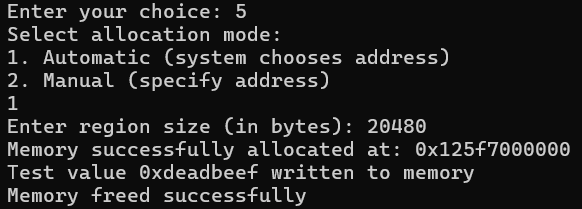
**4. Раздельное резервирование региона и передачу ему физической памяти**

В данном пункте мы сначала выделяем виртуальную память с помощью VirtualAlloc(MEM\_RESERVE), после чего в зарезервированном регионе мы выделяем память физическую или файл подкачки (VirtualAlloc, MEM\_COMMIT).



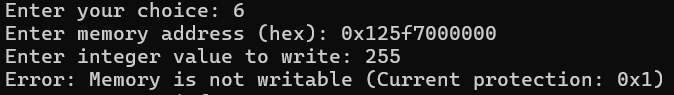
**5. Одновременное резервирование региона и передачу ему физической памяти**

В данном пункте, в отличие от предыдущего, память резервируется и выделяется одновременно (MEM\_RESERVE | MEM\_COMMIT)



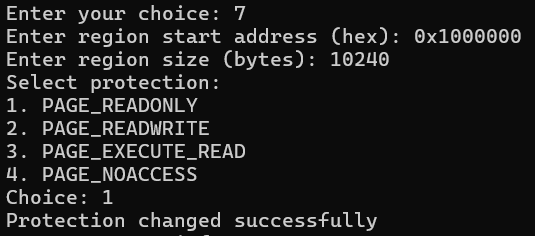
**6. Запись данных в память**

Мы проверяем доступность адреса через VirtualQuery, после чего записываем значение по этому адресу



**7. Установка защиты доступа**

После выбора прав доступа, они применяются к введеному адресу через VirtualProtect



**Выводы по выполнению задания**

В ходе работы были изучены основные механизмы управления виртуальной памятью в Windows с использованием Win32 API. Удалось исследовать структуру адресного пространства процесса, способы резервирования и выделения памяти, а также методы контроля прав доступа.

Программа показала, что виртуальная память в Windows управляется страницами (обычно по 4 КБ), а минимальный размер резервируемого региона составляет 64 КБ. Использование VirtualAlloc позволяет как резервировать память без физического выделения (MEM\_RESERVE), так и сразу выделять ее (MEM\_COMMIT). Раздельный подход полезен для оптимизации памяти, когда требуется заранее зарезервировать большой регион, но выделять его частями.

Также было проверено, как система защищает память: попытка записи в регион без прав PAGE\_READWRITE приводит к ошибке. Функция VirtualProtect позволяет динамически менять уровень доступа, что может использоваться для защиты критических данных или реализации самоизменяющегося кода.

Работа с VirtualQuery подтвердила, что память процесса может находиться в трех состояниях: свободна (MEM\_FREE), зарезервирована (MEM\_RESERVE) или выделена (MEM\_COMMIT). Анализ системной информации (GetSystemInfo, GlobalMemoryStatus) помог понять ограничения адресного пространства и доступные ресурсы.

В целом, Win32 API предоставляет гибкие инструменты для управления памятью, но требует аккуратного использования, чтобы избежать ошибок доступа или утечек. Полученные знания важны для разработки эффективных и безопасных приложений, работающих с большими объемами данных или низкоуровневыми структурами.

**Задание 1.2.** Использование проецируемых файлов для обмена данными между процессами.

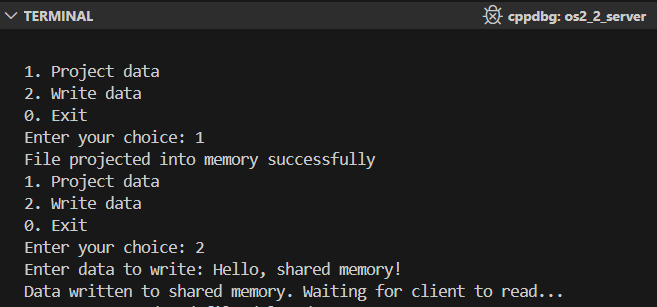
Создать два консольных приложения с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которые выполняют:

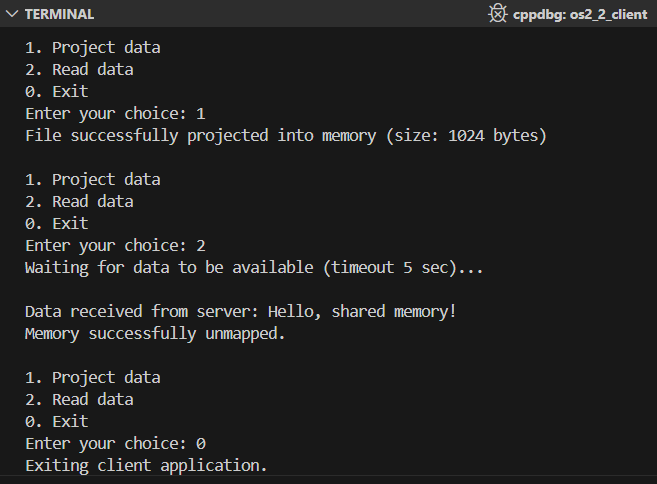
− приложение-писатель создает проецируемый, проецирует фрагмент файла в память, осуществляет ввод данных с клавиатуры и их запись в спроецированный файл;

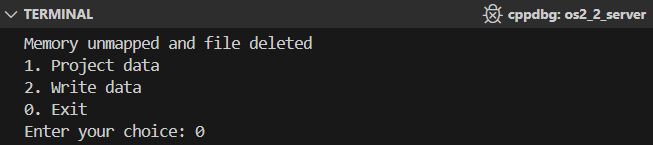
− приложение-читатель открывает проецируемый файл, проецирует фрагмент файла в память, считывает содержимое из спроецированного файла и отображает на экран.

Запустить приложения и проверить обмен данных между процессами, удостовериться в надлежащем выполнении задания. Запротоколировать результаты в отчет.

**Пример выполнения программ.**

****

****

****

**Выводы:**

Исследование механизма проецируемых файлов позволило на практике изучить особенности организации совместного доступа к памяти между независимыми процессами в операционных системах семейства Linux. Ключевым аспектом реализации стало использование системного вызова mmap, который обеспечивает отображение файлов или устройств в адресное пространство процесса. Особенность данного механизма заключается в том, что при использовании флага MAP\_SHARED изменения, вносимые одним процессом, становятся видимыми другим процессам, работающим с тем же отображением, а также сохраняются в исходном файле.

На уровне операционной системы это реализуется через систему виртуальной памяти, где различные виртуальные адреса разных процессов могут быть связаны с одними и теми же физическими страницами памяти. Ядро ОС обеспечивает синхронизацию между виртуальными адресными пространствами процессов и физической памятью, используя таблицы страниц. При этом сохраняется изоляция процессов - каждый из них работает со своими виртуальными адресами, не зная о том, что эти адреса могут быть связаны с общими физическими страницами. Данный механизм позволяет избежать избыточного копирования данных между пространством ядра и пользовательскими процессами, что существенно повышает производительность при работе с большими объемами данных. Однако при этом требуется тщательная синхронизация доступа, особенно в случаях, когда несколько процессов могут одновременно модифицировать общие данные. Также необходимо учитывать требования к выравниванию и размеру отображаемых областей, что связано с особенностями работы подсистемы виртуальной памяти.