

**МИНОБРНАУКИ РОССИИ  
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ  
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
«ЛЭТИ» ИМ. В.И.УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра ВТ**

**ОТЧЕТ  
по лабораторной работе №2  
по дисциплине «Операционные системы»  
Тема: Управление памятью**

Студент гр. 8307

\_\_\_\_\_

Репин С.А.

Преподаватель

\_\_\_\_\_

Тимофеев А.В.

Санкт-Петербург  
2020

## СОДЕРЖАНИЕ

Цель работы	3
Введение	3
1. Исследование виртуального адресного пространство процесса	4
1.1. Получение информации о вычислительной системе . . . . .	4
1.2. Определение статуса виртуальной памяти . . . . .	4
1.3. Определение состояния конкретного участка памяти . . . . .	5
1.4. Резервирование региона памяти . . . . .	6
1.5. Резервирование региона памяти и передача ему физической памяти . . . . .	7
1.6. Запись и чтение данных по определенному адресу . . . . .	9
1.7. Установка защиты доступа . . . . .	10
1.8. Возврат физической памяти и освобождение региона . . . . .	11
2. Использование проецируемых файлов для обмена данными между процессами	13
Вывод	14

## ЦЕЛЬ РАБОТЫ

Исследовать механизмы управления виртуальной памятью Win32.

## ВВЕДЕНИЕ

При выполнении лабораторной работы на языке программирования C стандарта C11 было разработано консольное приложение, управление которым происходит через различные меню, содержащие подпункты и подменю, соответствующие пунктам заданий. Исходный код приложения доступен на GitHub <sup>1</sup>.

Файл	Описание
menu.c	Определение типов и функций для работы с меню
main.c	Точка входа в программу; объявления конкретных меню и переходов между ними
actions.c	Реализация функций непосредственно выполняющих требования заданий (другими словами, callback'и конечных пунктов меню)
error.c	Описание номеров ошибок, а также функции отображения сообщений об ошибках

Таблица 1 Описание файлов в проекте

Сборка проекта производится с помощью Powershell-скрипта *build.ps1* (следует создать папку *build* и запускать скрипт из нее). Также потребуется пакет Build Tools for Visual Studio 2019.

---

<sup>1</sup>[https://github.com/stnrepin/os\\_labs/tree/master/lab2](https://github.com/stnrepin/os_labs/tree/master/lab2)

## 1. ИССЛЕДОВАНИЕ ВИРТУАЛЬНОГО АДРЕСНОГО ПРОСТРАНСТВО ПРОЦЕССА

### 1.1. Получение информации о вычислительной системе

Используя функцию *GetSystemInfo*, программа получает информацию о системе, содержащую такие пункты, как: архитектура процессора, число логических ядер процессора, размер страницы, гранулярность выделения памяти, минимальный (максимальный) доступный для использования адрес памяти и другие.

```
Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 1
Choose action:
1 - System info
2 - Global memory status
3 - Back
[1-3]> 1
Processor arch: x64
Processor level: 6
Processor revision: 0x8e0c
Number of processors: 4
Active provessors: 0 1 2 3
Page size: 4096B
Allocation granularity: 65536B
Minimum address: 0x00000000000010000
Maximum address: 0x00007FFFFFFF
```

Рис. 1.1 Результат выполнения программы

### 1.2. Определение статуса виртуальной памяти

Используя функцию *GlobalMemoryStatus*, программа получает информацию о виртуальной памяти компьютера: процент используемой памяти, общий размер физической памяти, доступный размер физической памяти, общий размер виртуальной памяти, доступный размер виртуальной памяти.

Заметим, что объемы физической и виртуальной памяти сильно отличаются. Размер физической памяти ограничен конфигурацией оборудования (размером ОЗУ, в данном случае), а виртуальной только операционной системой.

```
Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 1
Choose action:
1 - System info
2 - Global memory status
3 - Back
[1-3]> 2
Physical memory in use: 78%
Total physical memory: 4GiB
Available physical memory: 887.56MiB
Committed memory limit: 6.37GiB
Available commitable memory: 1.68GiB
Total virtual memory: 128TiB
Available virtual memory: 128TiB
```

Рис. 1.2 Результат выполнения программы

### 1.3. Определение состояния конкретного участка памяти

Используя функцию *VirtualQuery*, программа получает информацию о состоянии участка памяти, расположенного по указанному адресу. В частности, можно узнать о размере участка памяти, состоянии страниц, доступности страниц.

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 2
Enter the address : 0x00000fffffffffff
Base address: 0x00000FFFFFFFF000
Base address of a range of pages allocated: 0x0000000000000000
Initial protection option:
Region size: 111.95TiB
State of the pages: free
Current protection option: PAGE_NOACCESS
Type of pages: private

```

Рис. 1.3 Результат выполнения программы для адреса 0x00000fffffffffff. Заметим, что память, начиная с адреса 0x00000ffffffff000, доступна для аллокации

#### 1.4. Резервирование региона памяти

Используя функцию *VirtualAlloc*, программа резервирует регион памяти определенного размера по определенному адресу. Для резервирования с автоматическим определением адреса региона памяти в функцию передается адрес равный *NULL*.

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 3
Enter the base address of the region (0 to auto choose): 0x0
Enter the size of the region to allocate (in bytes): 8192
Alloced memory pointer: 0x0000017E57A10000

```

Рис. 1.4 Результат выполнения программы для резервирования участка размером 8192 байта в автоматическом режиме

Попробуем выполнить запись в участок выделенной памяти. Очевидно, это не получится сделать, так как виртуальная память была только лишь зарезервирована, но ей не была выделена физическая память.

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 5
Enter the address (0 to stop): 0x0000017E57A10000
Enter the value to set (hex): 0xab
Error: access violation (108)

```

Рис. 1.5 Программа завершается с ошибкой из-за доступа к не выделенной памяти

Теперь зарезервируем память по адресу 0x00000fffffffffff.

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 3
Enter the base address of the region (0 to auto choose): 0x00000fffffffffff
Enter the size of the region to allocate (in bytes): 8192
Allocated memory pointer: 0x00000FFFFFFF0000

```

Рис. 1.6 Результат выполнения программы для резервирования участка размером 8192 байта

## 1.5. Резервирование региона памяти и передача ему физической памяти

Аналогично предыдущему пункту здесь используется функция *VirtualAlloc*, которой, кроме того, передается параметр *MEM\_COMMIT*, обязывающий ОС передать региону физическую память после резервирования.

Заметим, что программа вернет ошибку, если память перед этим была уже зарезервирована.

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 4
Enter the base address of the region (0 to auto choose): 0x0
Enter the size of the region to allocate (in bytes): 8192
Alloced memory pointer: 0x000002AF42110000

```

Рис. 1.7 Результат выполнения программы для выделения участка памяти размером 8192 байта в автоматическом режиме

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 4
Enter the base address of the region (0 to auto choose): 0x00000fffffffffff
Enter the size of the region to allocate (in bytes): 8192
Alloced memory pointer: 0x0000000000000000
Error: Attempt to access invalid address. (106)

```

Рис. 1.8 Программа выдает ошибку при использовании зарезервированной памяти

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 4
Enter the base address of the region (0 to auto choose): 0x00000fffffffffff
Enter the size of the region to allocate (in bytes): 8192
Alloced memory pointer: 0x00000FFFFFFF0000

```

Рис. 1.9 Результат выполнения программы для выделения участка памяти размером 8192 байта по определенному адресу



```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 2
Enter the address : 0x00000ffffff0000
Base address: 0x00000FFFFFF0000
Base address of a range of pages allocated: 0x00000FFFFFF0000
Initial protection option: PAGE_READWRITE
Region size: 8KiB
State of the pages: committed
Current protection option: PAGE_READWRITE
Type of pages: private

```

Рис. 1.10 Проверка выполнения программы с помощью функции *VirtualQuery*

## 1.6. Запись и чтение данных по определенному адресу

Используя функции *memset*, *memcpy* из стандартной библиотеки C, программа записывает и читает данные по заданным пользователем адресам. Кроме того, используя расширения компилятора Microsoft, программа с помощью конструкции *\_\_try-\_\_except* может определять попытки записи (чтения) по недоступному адресу.

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 5
Enter the address (0 to stop): 0x00000ffffff0001
Enter the value to set (hex): 0xab
Enter the address (0 to stop): 0x00000ffffff0002
Enter the value to set (hex): 0x10
Enter the address (0 to stop): 0x00000ffffff0003
Enter the value to set (hex): 0xfe
Enter the address (0 to stop): 0x0

```

Рис. 1.11 Запись трех байт

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 6
Enter the address (0 to stop): 0x00000ffffff0001
Pointee value: 0xab
Enter the address (0 to stop): 0x00000ffffff0002
Pointee value: 0x10
Enter the address (0 to stop): 0x00000ffffff0003
Pointee value: 0xfe
Enter the address (0 to stop): 0x0

```

Рис. 1.12 Чтение ранее записанных трех байт

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 5
Enter the address (0 to stop): 0x0000017E57A10000
Enter the value to set (hex): 0xab
Error: access violation (108)

```

Рис. 1.13 Программа возвращает ошибку при попытке записи данных в недоступные ячейки памяти

## 1.7. Установка защиты доступа

С помощью функции *VirtualProtect* программа может устанавливать защиту доступа для заданного региона памяти.

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 7
Enter the base address of the region : 0x00000ffffff0000
The region base address: 0x00000FFFFFF0000
The region size: 8KiB
Available protect options:
1 - PAGE_EXECUTE
2 - PAGE_EXECUTE_READ
3 - PAGE_EXECUTE_READWRITE
4 - PAGE_EXECUTE_WRITECOPY
5 - PAGE_NOACCESS
6 - PAGE_READONLY
7 - PAGE_READWRITE
8 - PAGE_WRITECOPY
9 - PAGE_TARGETS_NO_UPDATE
Enter the index of the required options: 5
New page memory-protection option: PAGE_NOACCESS

```

Рис. 1.14 Установка защиты доступа *PAGE\_NOACCESS*

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 5
Enter the address (0 to stop): 0x00000ffffff0001
Enter the value to set (hex): 0xab
Error: access violation (108)

```

Рис. 1.15 Теперь запись данных в ячейки данного региона не удастся

## 1.8. Возврат физической памяти и освобождение региона

Используя функцию *VirtualFree*, программа освобождает регион адресного пространства по заданному адресу и возвращает физическую память.

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 8
Enter the base address of the region : 0x00000ffffff0000
The region base address: 0x00000FFFFFF0000
The region size: 68KiB
Memory freed

```

Рис. 1.16 Освобождение памяти по адресу 0x00000ffffff0000

```

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 2
Enter the address : 0x00000ffffff0000
Base address: 0x00000ffffff0000
Base address of a range of pages allocated: 0x00000ffffff0000
Initial protection option: PAGE_READWRITE
Region size: 68KiB
State of the pages: committed
Current protection option: PAGE_READWRITE
Type of pages: private

Choose action:
1 - System and Memory Info
2 - Get Memory Status
3 - Reserve Memory
4 - Commit Memory
5 - Write Memory
6 - Read Memory
7 - Set Memory Protection
8 - Free
9 - Quit
[1-9]> 2
Enter the address : 0x00000ffffff0000
Base address: 0x00000ffffff0000
Base address of a range of pages allocated: 0x0000000000000000
Initial protection option:
Region size: 111.96TiB
State of the pages: free
Current protection option: PAGE_NOACCESS
Type of pages: private

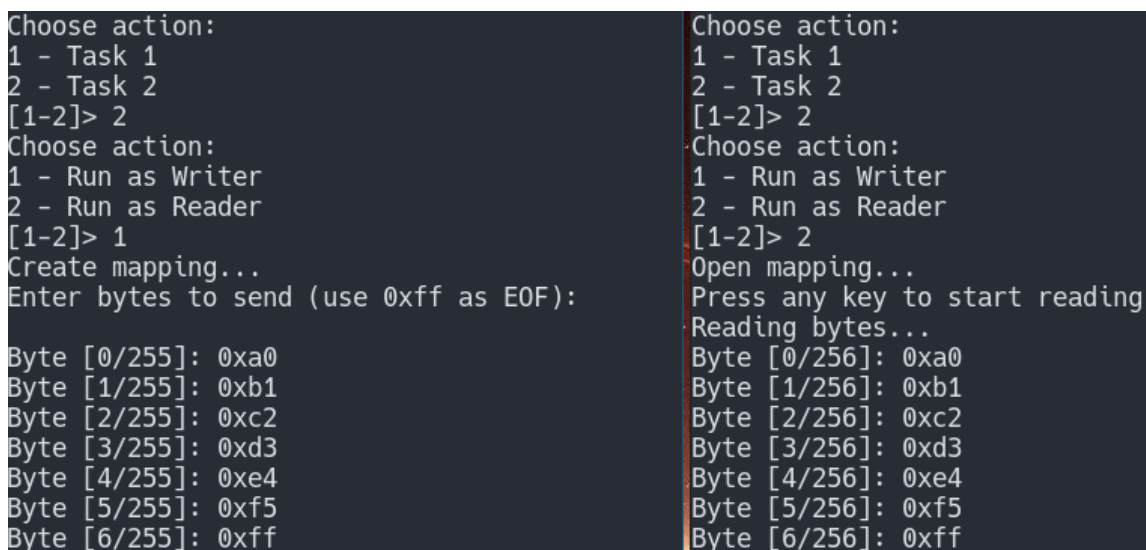
```

Рис. 1.17 Статус региона памяти по адресу 0x00000ffffff0000 до и после освобождения

## 2. ИСПОЛЬЗОВАНИЕ ПРОЕЦИРУЕМЫХ ФАЙЛОВ ДЛЯ ОБМЕНА ДАННЫМИ МЕЖДУ ПРОЦЕССАМИ

Программа может быть использована в двух режимах: писатель (writer) и читатель (reader).

В первом случае, программа создает специальный файл, проецирует его в память и записывает данные в спроецированный файл (функции *CreateFile*, *CreateFileMapping*, *MapViewOfFile*, *UnmapViewOfFile*, *memcpy*). Во втором случае, программа считывает этот файл, также проецирует его в память и выполняет чтение данных (функции *OpenFileMapping*, *MapViewOfFile*, *UnmapViewOfFile*, *memcpy*).



```
Choose action:
1 - Task 1
2 - Task 2
[1-2]> 2
Choose action:
1 - Run as Writer
2 - Run as Reader
[1-2]> 1
Create mapping...
Enter bytes to send (use 0xff as EOF):
Byte [0/255]: 0xa0
Byte [1/255]: 0xb1
Byte [2/255]: 0xc2
Byte [3/255]: 0xd3
Byte [4/255]: 0xe4
Byte [5/255]: 0xf5
Byte [6/255]: 0xff

Choose action:
1 - Task 1
2 - Task 2
[1-2]> 2
Choose action:
1 - Run as Writer
2 - Run as Reader
[1-2]> 2
Open mapping...
Press any key to start reading
Reading bytes...
Byte [0/256]: 0xa0
Byte [1/256]: 0xb1
Byte [2/256]: 0xc2
Byte [3/256]: 0xd3
Byte [4/256]: 0xe4
Byte [5/256]: 0xf5
Byte [6/256]: 0xff
```

Рис. 2.1 Слева программа в режиме писателя передает набор байтов в программу справа, работающую в режиме читателя

## ВЫВОД

В ходе выполнения данной лабораторной работы были получены знания и практические навыки низкоуровневой работы с виртуальной памятью при помощи Windows API. На практике было разобрано устройство виртуальной памяти, ее отличие от физической памяти и ее взаимосвязь с физической памятью. Был изучен подход в организации виртуальной памяти в Windows – страничная организация памяти. Кроме того, теперь известны важные для практики особенности работы с памятью: так, например, память выделяется постранично (то есть нельзя выделить память размером 1 байт), а сам процесс выделения памяти разбивается на два отдельных процесса (резервирование участка виртуальной памяти и непосредственное связывание ее с физической памятью). Также, теперь очевидна польза от использования hugepages-технологии для программ, часто выделяющих память и часто к ней обращающихся.

Во втором задании к лабораторной работе было продемонстрировано важнейшее свойство виртуальной памяти: предоставление унифицированного интерфейса к физической памяти (источники которой могут быть различны, например: ОЗУ, SSD, HDD). В данном случае источником памяти выступает файл из ФС, который средствами Windows API проецируется в память, т.е. данным, хранящимся в файле, ставится в соответствие некоторый диапазон адресов виртуальной памяти. Дальнейшее взаимодействие с файлом происходит также, как и взаимодействие с обычной памятью (чтение/запись данных памяти приводит к чтению/записи данных в файле).

Таким образом, такой подход выступает гибким способом организации межпроцессорного взаимодействия, позволяя использовать файл в качестве связующего элемента между независимыми пространствами виртуальной памяти каждого процесса (один процесс записывает данные по адресу, который принадлежит проецированной памяти, автоматически операционной системой происходит запись в проецированный файл, которая в свою очередь влечет изменение памяти второго процесса по соответствующему адресу).

Кроме того, отображение файла в память дает серьезный выигрыш в производительности операций чтения и записи файла, сокращая количество системных вызовов, уменьшая количество используемой памяти и упрощая взаимодействие с файлом (исчезает необходимость программы в промежуточном буфере для хранения данных файла, так как этим теперь занимается только ОС).