**Лабораторная работа №5**

**Управление памятью**

**Цель работы:** Получение практических навыков в работе с виртуальной памятью, кучами и стеками.

Перед выполнением задания прочитайте его полностью. Схематично представьте себе, что от вас требуется и только после этого выполняйте.

**Темы для предварительного изучения:**

* Утилита VMMap ([ссылка](https://learn.microsoft.com/en-us/sysinternals/downloads/vmmap)).
* Утилита RAMMap ([ссылка](https://learn.microsoft.com/en-us/sysinternals/downloads/rammap)).
* Секции памяти образа PE-файла ([ссылка](https://learn.microsoft.com/en-us/windows/win32/debug/pe-format" \l "special-sections)).

**Постановка задачи для Windows:**

**Приложение Lab-05a:**

Разработайте консольное приложение **Lab-05a** в котором создайте следующее:

* Любую функцию;
* Глобальные инициализированную и неинициализированную переменные;
* Глобальные статические инициализированную и неинициализированную переменные;
* Локальные инициализированную и неинициализированную переменные;
* Локальные статические инициализированную и неинициализированную переменные;
* Объявите функцию main cо стандартными аргументами **argc** и **argv**.

Приложение должно выводить на консоль адреса всех объявленных переменных и функций, а также адреса переменных **argc** и **argv** и перейти в режим ожидания ввода символа на консоль.

С помощью утилиты VMMap найти секции памяти, в которых расположены все эти переменны и функции и заполнить таблицу следующего вида:

|  |  |  |
| --- | --- | --- |
| **Вид переменной** | **Адрес** | **Название секции\*** |
| **global\_init\_numb** | **00007FF69F136000** | **.data** |
| **global\_uninit\_numb** | **00007FF69F1360BC** | **.data** |
| **static\_global\_init\_numb** | **00007FF69F136008** | **.data** |
| **static\_global\_uninit\_numb** | **00007FF69F1360C0** | **.data** |
| **some\_function** | **00007FF69F131000** | **Text** |
| **Argc** | **00000097FC73F8C4** | **Thread Environment Block** |
| **Argv** | **00000097FC73F8C8** | **Thread Environment Block** |
| **local\_init\_numb** | **00000097FC73F8BC** | **Thread Environment Block** |
| **local\_uninit\_numb:** | **00000097FC73F8C0** | **Thread Environment Block** |
| **static\_local\_init\_numb** | **00007FF69F136004** | **.data** |
| **static\_local\_uninit\_numb** | **00007FF69F1360B8** | **.data** |

Примечание! Утилита VMMap поддерживает работу через консоль с выводом структуры памяти процесса в файл. Структура вызова для этого (параметр -64 только в случае, если у вас x64 приложение):

vmmap [-64] -p <PID> <output file>

\*Колонка Details в программе

**Приложение Lab-05b:**

Разработайте консольное приложение **Lab-05b** в котором реализуйте код, который делится на следующие этапы:

* Зарезервировать виртуальной памяти процесса на 256 страниц (размер страницы определить через вызов соответствующей функции) и вывести адрес области памяти на консоль;
* Для второй половины (128 страниц) зарезервированной области выделить физическую память;
* Заполнить данную половину памяти последовательностью целых чисел начиная от 0 с шагом 1;
* Изменить атрибуты защиты страниц памяти на «только для чтения»;
* Освободить выделенную физическую память для 128 виртуальных страниц;
* Освободить 256 страниц виртуальной памяти.

После каждого этапа программа должна ожидать ввода в консоль. В это время требуется найти в программе **VMMap** выделенную область виртуальной памяти и изучить её характеристики. Также на этапах 1, 2, 3 и 5 в программе **RAMMap** проверить выделены ли физические страницы памяти.

В программе **RAMMap** выполняется это на вкладке «Physical Pages», а также используя фильтр внизу окна по значению «Process».

***Вопрос:*** *На каком этапе фактически были выделены физические блоки памяти?*

**Приложение Lab-05с:**

Разработайте консольное приложение **Lab-05с** в котором реализуйте код, который делится на следующие этапы:

* Создание пользовательской кучи со стартовым размером 1 МиБ и максимальным 8 МиБ;
* В цикле выделить 10 блоков памяти из кучи размером 512 КиБ каждый;
* Заполнить каждый блок массивом целых чисел соответствующей длины (512 КиБ / 4 Б);
* В цикле освободить память, выделенную для всех блоков;
* Уничтожить кучу.

Написать функцию **HeapInfo**, которая выводит на консоль информацию о куче, а именно общий размер, размеры областей памяти кучи с указанием адреса начала области, размера области и типа области. Вызов данной функции должен производиться после каждого этапа кроме заполнения массивами (на этапе 2 после каждой итерации).

Для красоты вывода на консоль можно вызывать функцию в связке с паузой следующим образом:

HeapInfo(heap); system("pause & cls");

Попытаться изменить количество выделяемых блоков с 10 до 5 и изменить размер с 512 КиБ до 1 МиБ.

***Вопрос:*** *Объясните результат после попытки изменения количества блоков и их размера. (Задание со звёздочкой)*

**Постановка задачи для Linux:**

**Приложение Lab-05d:**

Разработайте консольное приложение, являющееся полным аналогом **Lab-05а.**

Для заполнения таблицы требуется сделать следующее:

* Запустить процесс;
* Вызвать утилиту «**pmap -X <PID> | awk '{ print $1, $2, $6, $20 }'**» и выписать стартовый адрес вашего образа исполняемого файла (обычно это самая первая строка в таблице)
* Вызвать утилиту «**objdump --adjust-vma <адрес с прошлого шага> -x ./<image-file>**» (здесь вас интересует раздел Sections)

На основе полученной информации из двух утилит найти секции памяти, в которых расположены все эти переменны и функции и заполнить таблицу следующего вида:

|  |  |  |
| --- | --- | --- |
| **Вид переменной** | **Адрес** | **Название секции** |
| global\_initialized | 0x5c059d6f4028 | .data |
| global\_uninitialized | 0x5c059d6f403c | .bss |
| global\_static\_initialized | 0x5c059d6f4030 | .data |
| global\_static\_uninitialized | 0x5c059d6f4040 | .bss |
| local\_initialized | 0x7ffc7839d9dc | stack |
| local\_uninitialized | 0x7ffc7839d9d8 | stack |
| local\_static\_initialized | 0x5c059d6f402c | .data |
| local\_static\_uninitialized | 0x5c059d6f4038 | .bss |
| main | 0x5c059d6f1170 | .text |
| argc | 0x7ffc7839d9e8 | stack |
| argv | 0x7ffc7839d9e0 | stack |

**Приложение Lab-05e:**

Разработайте консольное приложение **Lab-05e** в котором реализуйте код, который делится на следующие этапы:

* Зарезервировать виртуальной памяти процесса на 256 страниц (размер страницы определить через вызов соответствующей функции) и вывести адрес области памяти на консоль;
* Для второй половины (128 страниц) зарезервированной области выделить физическую память (сменить с PROT\_NONE на PROT\_READ и PROT\_WRITE);
* Заполнить данную половину памяти последовательностью целых чисел начиная от 0 с шагом 1;
* Изменить атрибуты защиты страниц памяти на «только для чтения»;
* Освободить 128 виртуальных страниц в которые записан массив;
* Освободить оставшиеся страницы виртуальной памяти.

После каждого этапа программа должна ожидать ввода в консоль. В это время требуется просматривать информацию об областях памяти через утилиту **pmap**. Для удобства можно использовать: **watch pmap -X <PID>**.

Показать и пояснить состояние областей памяти на каждом этапе.

**Приложение Lab-05f:**

Разработайте консольное приложение **Lab-05f** в котором реализуйте код, который делится на следующие этапы:

* В цикле выделить 10 блоков памяти из кучи размером 512 КиБ каждый;
* Заполнить каждый блок массивом целых чисел соответствующей длины (512 КиБ / 4 Б);
* В цикле освободить память, выделенную для всех блоков;
* Уничтожить кучу.

После каждого этапа программа должна ожидать ввода в консоль. В это время требуется просматривать информацию об областях памяти через утилиту **pmap**. Для удобства можно использовать: **watch pmap -X <PID>**.

**Требования к выполненной работе:**

* Разработанные приложения должны быть скомпилированы и протестированы как на Windows, так и на ОС из семейства Linux.
* Для генерации и сборки проекта использовать средство CMake.
* Для компиляции файлов при сборке проекта использовать компилятор Clang.

**Вопросы для контроля:**

1. Что такое физическая память?
2. Что такое логическая память?
3. Что такое виртуальная память?
4. Как устроена виртуальная память?
5. Перечислите и поясните назначение секций памяти процесса.
6. Какие API входят в WinAPI для работы с памятью? В POXIS?
7. Когда лучше использовать виртуальную память напрямую?
8. Когда лучше использовать кучи?