Лабораторная работа № 3

Управление оперативной памятью

Одним из важнейших аппаратных ресурсов, доступ к которым предоставляет операционная система программам, является оперативное запоминающее устройство (ОЗУ), или оперативная память. Динамическое выделение памяти в программах — `GetMem()`, `FreeMem()` и другие в Dephi, **`new`** и **`delete`** в C++ — в конечном счете реализуется обращением к функциям ОС управления памятью.

Функции ОС управления памятью используют для решения ряда задач.

1. Борьба с фрагментацией памяти. Несмотря на то, что механизмы борьбы с фрагментацией заложены в ядро ОС, бывает необходимо заранее резервировать крупные участки памяти под очень большие объекты или под многочисленные мелкие.
2. Управление размещением страниц виртуальной памяти в физическом ОЗУ или на диске в файле подкачки. В критичных к скорости работы случаях приложению требуется указать ОС заранее загрузить некоторые области памяти в ОЗУ для быстрого доступа.
3. Задание специальных свойств областей памяти: защита от чтения и записи, настройка кэширования операционной системой и т. п. Web-браузер, например, может хранить введенный на сайте пароль в защищенной области памяти, чтобы программы-шпионы не могли его считать.

Сама задача управления памятью подробно рассматривается в лекционном курсе.

Указания к выполнению лабораторной работы

При исследовании функций ОС управления памятью полезно знать, какие области памяти используются процессом, как они размещены и какие имеют свойства. Это можно сделать бесплатными программами авторства разработчиков Windows.

Программа VMMap предназначена для наблюдения виртуального адресного пространства (ВАП) процесса. Перед началом анализа необходимо выбрать процесс. Среды разработки (Delphi, Code::Blocks и т. п.) запускают программы как свои дочерние процессы, что не подходит для VMMap. Поэтому анализируемые программы следует запускать из «Проводника».

Данные в памяти изменяются очень часто, поэтому окно VMMap не обновляется автоматически, а нужно это делать кнопкой F5 при необходимости. Если в программе есть

несколько точек исполнения, в которых желательно произвести анализ при помощи VMMap, следует добавить в программу задержки (ожидания нажатия клавиши), чтобы в эти моменты переключаться к VMMap.

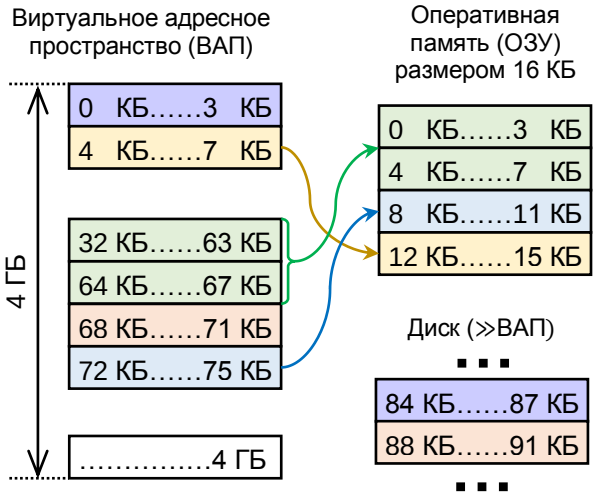
Фрагментацию памяти удобно наблюдать через *Fragmentation View* (пункт меню *View → Fragmentation View*). Ползунок слева позволяет изменять масштаб. При щелчке мышью на область памяти она подсвечивается в списке внизу основного окна VMMap.

Программа RAMMap отображает страничные блоки (области физического ОЗУ), отведенные различным процессам, а также их соответствие страницам ВАП. Указанные сведения полезны при разработке драйверов устройств, взаимодействие с которыми ведется через отведенные области ОЗУ. При анализе производительности прикладных программ также бывает полезно убедиться, что все активно используемые данные размещены ОС в ОЗУ и доступны MMU. В данной ЛР не используется RAMMap.

Задание на лабораторную работу

Вариант 1. Использование виртуальной памяти и её фрагментация

Механизм виртуальной памяти (virtual memory, ВП), или виртуального адресного пространства (virtual address space, ВАП), предназначен для того, чтобы позволить процессам использовать больше памяти, чем физический объем ОЗУ. Вместо физического адреса (ФА) ячейки ОЗУ операционная система предоставляет программе виртуальный адрес (ВА). Когда программа обращается по ВА, ОС при необходимости загружает необходимую область в ОЗУ и преобразует ВА в ФА для доступа к данным. Области ВП, которые не помещаются в ОЗУ, выгружаются временно на диск. Таким образом, в ОЗУ находится только рабочий набор (working set) активно используемых областей памяти, остальное размещается на диске, доступ к которому гораздо медленнее, однако размер существенно больше.



Все современные ОС общего назначения обладают механизмом ВП; в Windows каждому 32-битному процессу предоставляется 4 ГБ памяти, 2 ГБ из которых программа может использовать свободно. Для повышения производительности ВП выделяется страницами (4 КБ на платформе x86); то есть, даже когда программист выделяет 1 байт, ОС отводит под эту область (регион) не менее одной страницы. Работа с виртуальной

памятью реализуется семейством функций Windows API; необходимые в данной ЛР функции также рассмотрены в методическом пособии. Современные ЭВМ включают специальное устройство — блок управления памятью (memory management unit, MMU), обычно в составе ЦП, которые преобразуют ВА в ФА аппаратно, значительно повышая производительность. Задача же операционной системы — подготовить для MMU таблицу страниц и поддерживать её актуальность методами, рассмотренными в лекционном курсе.

Важной проблемой является фрагментация памяти — состояние, когда свободная память имеется только в виде множества небольших областей, и хотя их суммарный объем может быть велик, крупный непрерывный блок выделить невозможно. Предупредить проблемы из-за фрагментации можно, выделив крупный участок памяти заранее. При этом желательно — и возможно — не выделять сразу часть ОЗУ или диска, а только указать ОС, что такой участок понадобится (зарезервировать его).

Задание. Реализовать программу для изучения фрагментации виртуальной памяти.

1. Получить и напечатать общий и доступный объем виртуальной памяти (ВП) при помощи функции `GlobalMemoryStatusEx()`. Пронаблюдать состояние памяти процесса при помощи VMMap.
2. Использовать всю доступную ВП, выделяя функцией `VirtualAlloc()` области размером по 100 МБ, пока это удается. Указатели на выделяемые области сохранять в массив (`std::vector`, `TList` или любой другой).
3. Повторить пункт 1, сравнив доступный объем памяти со 100 МБ.
4. Освободить *каждый второй* из выделенных в пункте 2 блоков при помощи функции `VirtualFree()`.
5. Повторить пункт 1, обратив внимание на объем доступной ВП.
6. Попытаться выделить блок памяти размером ¼ ВП, доступной согласно п. 5. Если пункт выполняется в составе пункта 10, использовать для выделения зарезервированную в пункте 9 область памяти.
7. Освободить оставшиеся выделенные в пункте 2 блоки.
8. Повторить пункт 1, сравнив результаты с полученными в нем ранее.
9. Зарезервировать область памяти того же размера, что и в пункте 6, при помощи функции `VirtualAlloc()` с флагом `MEM_RESERVE`.
10. Повторить пункты 1—6.
11. Определить особенности наблюдавшихся состояний памяти процесса, их различия между собой и объяснить результаты.

Указание. В отчет надлежит включить: вывод рассчитанных в программе значений, описание наблюдений и пояснения к ним (пункт 1 и его повторения).

Вариант 2. Исследование областей виртуальной памяти

Страницы виртуальной памяти имеют ряд атрибутов. Главными являются:

1) состояние страницы (`MEM_*`): свободна (не используется), зарезервирована, загружена в физическую память (ОЗУ или на диск);
2) параметры защиты (`PAGE_*`): разрешено ли чтение, запись или исполнение данных в памяти страницы и т. п.

При нарушении защиты — например, при попытке изменить память, доступную только для чтения, — происходит ошибка доступа к памяти (access violation). С другой стороны, если требуется загрузить в память машинный код и выполнить его, необходимо разрешить в области памяти исполнение кода. Управление защитой страниц выполняется при помощи функции `VirtualProtect()`, запрос параметров защиты — функцией `VirtualQuery()`. Иногда защиту устанавливают в целях отладки: используется специальный распределитель памяти, который выделяет не только запрошенную область памяти, но и сразу после нее — страницу только для чтения. При записи данных за границу выделенной области происходит ошибка доступа — и проблема в программе выявляется.

Задание. Выполнить п. 2 варианта задания к ЛР № 3 по методическому пособию.

Схема выполнения всех вариантов приблизительно одинакова.

1. Получить адреса-границы анализируемой области памяти. Границы всего виртуального адресного пространства (ВАП) процесса можно определить при помощи функции `GetSystemInfo()`. Начальный адрес, если он вводится пользователем, нужно округлить вниз до кратного размеру страницы виртуальной памяти (определяемого `GetSystemInfo()`). Аналогично следует округлять и одиночный адрес, вводимый пользователем.
2. Атрибуты страниц, а также начальные адреса и размеры областей (регионов) памяти можно определить при помощи функции `VirtualQuery()`. Перебор всех страниц (или регионов) можно выполнить, продолжая вызывать функцию `VirtualQuery()` для адресов, смещаемых каждый раз на размер страницы (или региона).

Вариант 3. Использование кучи (heap)

В прикладном программировании термин «куча» (heap) использовался как синоним динамической памяти (dynamic memory). В системном программировании (в Windows) кучей называется заранее зарезервированная область памяти, из которой приложение может выделять (allocate) небольшие блоки в свое пользование. Имеется ряд отличий выделения памяти в куче от выделения виртуальной памяти.

1. Размер выделяемого в куче блока ограничен: менее 512 КБ на 32-битных системах, менее 1 МБ на 64-битных системах. Может быть ограничен и размер самой кучи.
2. Размер областей, выделяемых в виртуальной памяти, всегда кратен размеру страницы (обычно 4 КБ), из кучи же выделяется столько памяти, сколько было запрошено (с небольшим дополнительным расходом). По этой причине использование кучи экономичнее, если выделяется много небольших блоков.
3. Куча расположена в компактной области виртуальной памяти, что снижает общую фрагментацию адресного пространства при использовании кучи (но фрагментация самой кучи может иметь место).
4. Виртуальная память едина для приложения. Однако, приложение может иметь несколько куч; одна создается всегда, остальные — пользователем.

Кучи используют для снижения общей фрагментации виртуальной памяти, ускорения и упрощения работы программы. Ускорение возникает потому, что выделение памяти из кучи требует меньше операций, чем выделение области виртуальной памяти (куча и её служебные структуры меньше по размеру). Упрощение программы возможно следующим образом: создается новая куча, в ней некоторое время выделяются блоки, а затем вся куча уничтожается без необходимости освобождать каждый блок в ней.

Работа с кучей выполняется посредством семейства функций Windows API.

Задание. Реализовать следующую программу для исследования свойств куч.

1. Создать новую кучу с максимальным размером 24 КБ при помощи функции `HeapCreate()`. Изначальный размер кучи можно задать `0`.
2. Полностью использовать кучу, выделяя из нее функцией `HeapAlloc()` блоки случайного размера 32—1024 байт, пока это возможно.
3. При помощи функции `HeapWalk()` перебрать все блоки кучи и определить для выделенных блоков (поле `wFlags` структуры `PROCESS_HEAP_ENTRY` включает флаг `PROCESS_HEAP_ENTRY_BUSY`):
    1) суммарный объем хранимых данных (`cbData`);
    2) суммарные накладные расходы (`cbOverhead`);

3) разность между размером кучи и суммой величин пунктов 1) и 2).

Указание. Здесь и далее рассчитанные значения необходимо печатать.

4. При помощи функции `HeapWalk()` перебрать все блоки кучи, освобождая их с вероятностью ⅓ функцией `HeapFree()`.

   Указание. Текущий блок в цикле можно освобождать только после перехода к следующему.

5. Повторить пункт 3, дополнительно подсчитывая для свободных блоков (поле `wFlags` равно `0`):
   1) суммарный объем;
   2) максимальный размер.
6. Попытаться совершить следующие операции:
   1) Выделить блок размером на 16 байт менее наибольшего свободного блока, затем освободить этот блок (если выделение успешно).
   2) Выделить блок размером на 16 байт менее суммарного свободного объема памяти в куче.

   Наблюдаемые результаты объяснить.
7. Уничтожить созданную кучу при помощи функции `HeapDestroy()`.
8. Создать новую кучу такого же объема, как в пункте 1. Выделять блоки размером 1, 2, 3 и т. д. байт, пока не будет использована вся куча. При помощи функции `HeapWalk()` для каждого выделенного блока напечатать его размер и объем накладных расходов. Построить график зависимости накладных расходов от размера блока, выявить и объяснить закономерность.

   Указание. В C и C++ для корректного вывода поля `cbOverhead` необходимо привести его к типу, например, **`unsigned int`**.
9. Пронаблюдать при помощи VMMap состояние памяти процесса:
   1) перед выполнением п. 1;
   2) после выполнения п. 1;
   3) после выполнения п. 4;
   4) перед выполнением п. 7;
   5) после выполнения п. 7;
   6) после выполнения п. 8.

   Определить изменения, описать и объяснить их.

Указание. В отчет надлежит включить: вывод рассчитанных в программе значений (пункты 3 и 5), описание наблюдений и пояснения к ним (пункты 6, 8 и 9).

Контрольные вопросы

1. Зачем применяется механизм виртуальной памяти, и каков принцип его работы?
2. Что такое страницы виртуальной памяти (ВП) и зачем они применяются на уровне аппаратного обеспечения, ОС и прикладных программ (если применяются)?
3. В чем состоит проблема фрагментации адресного пространства и как она решается на уровне аппаратного обеспечения, ОС и прикладных программ (если решается)?
4. Как при помощи Windows API определить объем имеющейся, свободной и доступной приложению виртуальной памяти? Как соотносятся их размеры, и каковы их наибольшие значения?
5. Какими функциями Windows API выполняется выделение и освобождение областей памяти? Объясните использование каждого параметра этих функций.
6. Какие атрибуты (`PAGE_*` в Windows API) возможны для страницы виртуальной памяти и зачем они могут применяться?
7. Какими функциями Windows API можно определить и изменить атрибуты страниц виртуальной памяти?
8. Как соотносятся область (регион) виртуального адресного пространства и его страница? Можно ли по адресу ячейки определить страницу или регион (если да, то как, если нет, то почему)?
9. Что такое и для чего применяются кучи (heaps) в Windows API? Каковы их ограничения и накладные расходы при использовании? (Подкрепите ответ экспериментальными данными, полученными в ходе работы.)
10. Какие функции и структуры применяются при работе с кучами в Windows API?