Лабораторная работа №8

Архитектура памяти в ОС Windows

Цель работы: научиться использовать функции Win32 API, предназначенные для работы с виртуальным адресным пространством процесса в ОС Windows.

Типы памяти

Взаимосвязь виртуального адресного пространства процесса с физической и внешней памятью представлена на рисунке 4.

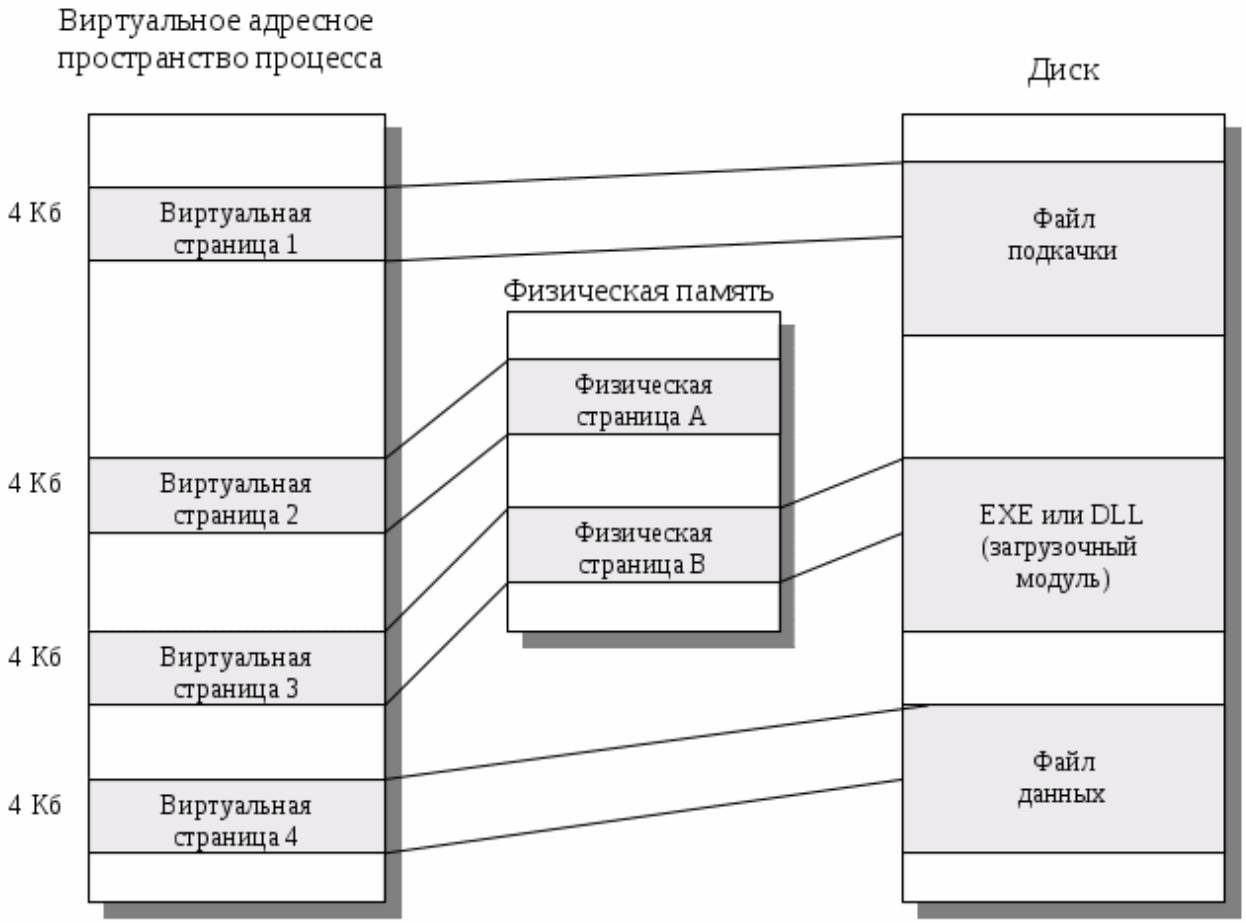

Рисунок 4 – Взаимосвязь виртуального адресного пространства процесса с физической и внешней памятью.

Физическая память

Физическая память – это микросхемы оперативной памяти (RAM), установленные в ПК. Каждый байт физической памяти имеет физический адрес,

который представляет собой число от нуля до числа на единицу меньшего, чем количество байтов физической памяти. Например, ПК с установленными 64 Мб RAM, имеет физические адреса &H00000000-&H04000000 в шестнадцатеричной системе счисления, что в десятичной системе будет 0-67 108 863.

Физическая память (в отличие от файла подкачки и виртуальной памяти) является исполняемой, то есть памятью, из которой можно читать и в которую центральный процессор может посредством системы команд записывать данные.

Виртуальная память

Виртуальная память (virtual memory) – это просто набор чисел, о которых говорят как о виртуальных адресах. Программист может использовать виртуальные адреса, но ОС Windows не способна по этим адресам непосредственно обращаться к данным, поскольку такой адрес не является адресом реального физического запоминающего устройства, как в случае физических адресов и адресов файла подкачки. Для того чтобы код с виртуальными адресами можно было выполнить, такие адреса должны быть отображены на физические адреса, по которым действительно могут храниться коды и данные. Эту операцию выполняет диспетчер виртуальной памяти (Virtual Memory Manager – VMM).

Операционная система Windows обозначает некоторые области виртуальной памяти как области, к которым можно обратиться из программ пользовательского режима. Все остальные области указываются как зарезервированные. Какие области памяти доступны, а какие зарезервированы, зависит от версии операционной системы (Windows 9x или Windows NT).

Страничные блоки памяти

Как известно, наименьший адресуемый блок памяти – байт. Однако самым маленьким блоком памяти, которым оперирует VMM Windows, является страница памяти, называемая также страничным блоком памяти. На ПК с процессорами Intel объем страничного блока равен 4 Кб.

Память файла подкачки

Страничный файл, который называется также файлом подкачки, в Windows находится на жестком диске. Он используется для хранения данных и программ точно так же, как и физическая память, но его объем обычно превышает объем физической памяти. Windows использует файл подкачки (или файлы, их может быть несколько) для хранения информации, которая не помещается в RAM, производя, если нужно, обмен страниц между файлом подкачки и RAM.

Таким образом, диапазон виртуальных адресов скорее согласуется с адресами в файле подкачки, чем с адресами физической памяти. Когда такое согласование достигается, говорят, что виртуальные адреса спроецированы на файл подкачки, или являются проецируемыми на файл подкачки.

Набор виртуальных адресов может проецироваться на физическую память, файл подкачки или любой файл.

Файлы, отображаемые в память

Отображение файла в память – это способ работы с файлами в некоторых операционных системах, при котором всему файлу или некоторой непрерывной его части ставится в соответствие определённый участок памяти (диапазон адресов оперативной памяти). При этом чтение данных из этих адресов фактически приводит к чтению данных из отображенного файла, а запись данных по этим адресам приводит к записи этих данных в файл. Примечательно то, что отображать на память часто можно не только обычные файлы, но и файлы устройств. Любой файл применяется для проецирования виртуальной памяти так же, как для этих целей используется файл подкачки. Фактически единственное назначение файла подкачки - проецирование виртуальной памяти. Поэтому файлы, проецируемые в память подобным образом, называются отображаемыми в память. На предыдущем рисунке изображены именно такие файлы. Соответствующие виртуальные страницы являются спроецированными на файл.

Функция `CreateFileMapping` объявляется так:

```
function CreateFileMapping(hFile: THandle;
lpFileMappingAttributes: PSecurityAttributes; flProtect,
dwMaximumSizeHigh, dwMaximumSizeLow: DWORD; lpName: PChar):
THandle;
stdcall;

function CreateFileMapping; external kernel32 name
'CreateFileMappingA';
```

Она создает объект «отображение файла», используя дескриптор открытого файла, и возвращает дескриптор этого объекта. Дескриптор может использоваться с функцией `MapViewOfFile`, отображающей файл в виртуальную память:

```
function MapViewOfFile(hFileMappingObject: THandle;
dwDesiredAccess: DWORD; dwFileOffsetHigh, dwFileOffsetLow,
dwNumberOfBytesToMap: DWORD): Pointer; stdcall;

function MapViewOfFile; external kernel32 name
'MapViewOfFile';
```

Начальный адрес объекта «отображение файла» в виртуальной памяти возвращает функция `MapViewOfFile`. Можно также сказать, что представление проецируется на файл с дескриптором `hFile`.

Если параметр `hFile`, передаваемый функции `CreateFileMapping`, установлен в `-1`, то объект «отображение файла» (любые представления, созданные на основе этого объекта) проецируем на файл подкачки, а не на заданный файл.

Совместно используемая физическая память

О физической памяти говорят, что она совместно используется (shared), если она отображается на виртуальное адресное пространство нескольких процессов, хотя виртуальные адреса в каждом процессе могут отличаться. Следующий рисунок иллюстрирует это утверждение.

Если файл, такой как DLL, находится в совместно используемой физической памяти, то о нем можно говорить как о совместно используемом.

Одно из преимуществ файлов, отображаемых в память, заключается в том, что их легко использовать совместно. Присвоение имени объекту «отображение файла» делает возможным совместное использование файла несколькими процессами. В этом случае его содержимое отображено на совместно используемую физическую память (см. рис. 5). Возможно также совместное пользование содержимого файла подкачки с помощью механизма отображения файла.

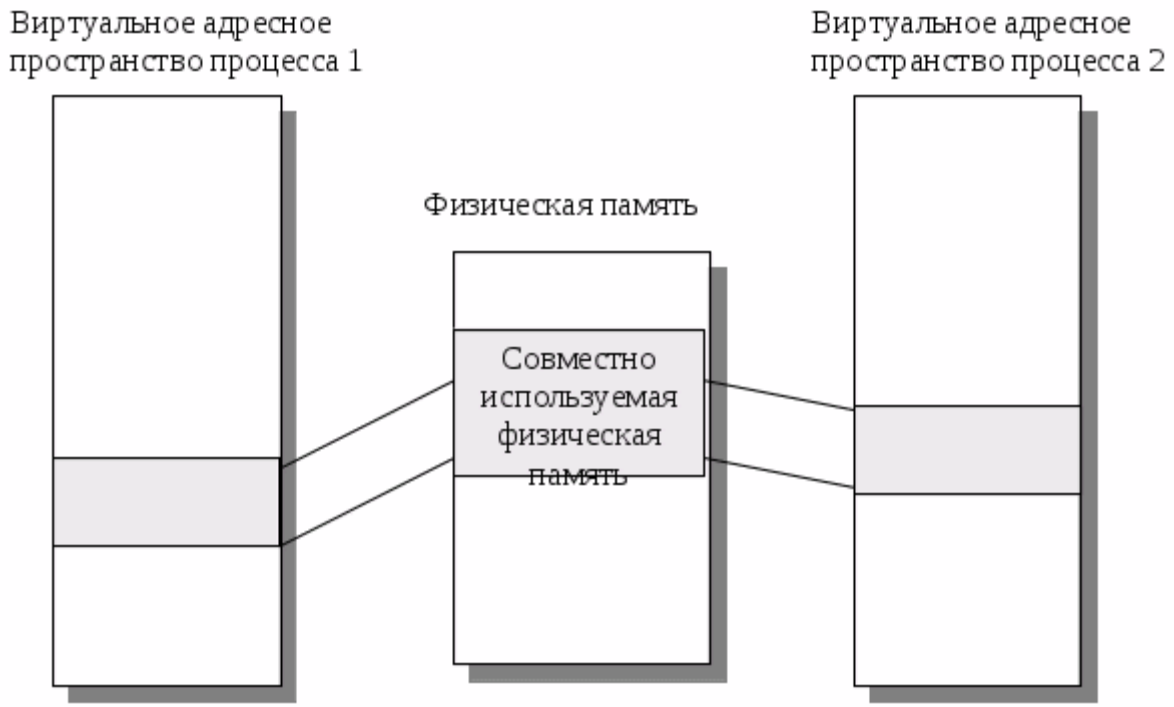

Рисунок 5 – Совместно используемая память

В частности, можно создать такой объект, проецируемый на файл подкачки, просто установив параметр `hFile` функции `CreateFileMapping` в `-1`.

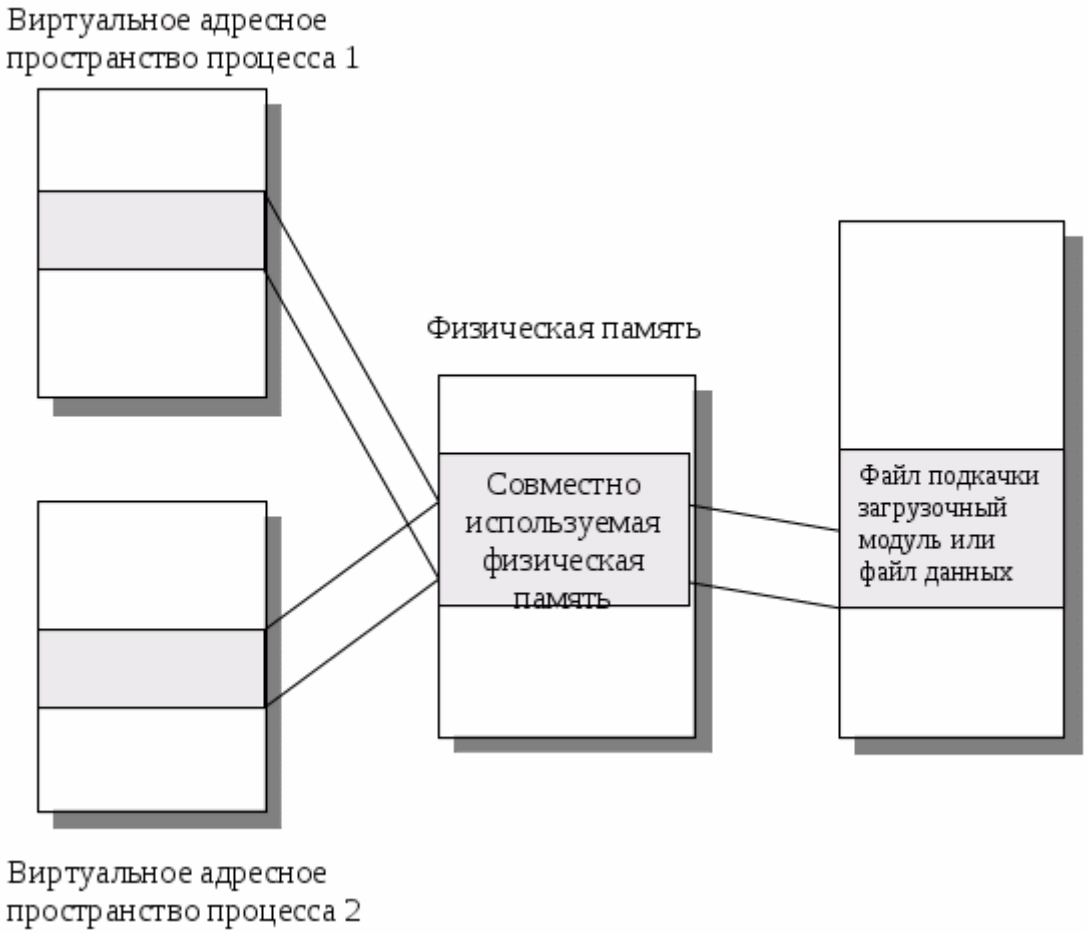

Рисунок 6 – Совместное использование памяти двумя процессами

Адресное пространство процесса

Каждый процесс Win32 получает виртуальное адресное пространство, объем которого равен 4 Гб. Таким образом, код процесса может ссылаться на адреса с &H00000000 по &HFFFFFFFF (или с 0 по 2^{32} - 1 = 4 294 967 295 в десятичной системе счисления). Конечно, так как виртуальные адреса - это просто числа, заявление о том, что каждый процесс получает свое собственное виртуальное адресное пространство, выглядит довольно бессмысленным. Тем не менее, это утверждение должно означать, что Windows не видит никакой взаимосвязи в том, что и процесс A, и процесс B используют один и тот же виртуальный адрес, например &H40000000. В частности, Windows может сопоставить (или не сопоставить) виртуальным адресам каждого процесса разные физические адреса.

Использование адресного пространства в Windows 9x

На рисунке 7 показана общая схема использования адресного пространства процесса в Windows 9x.

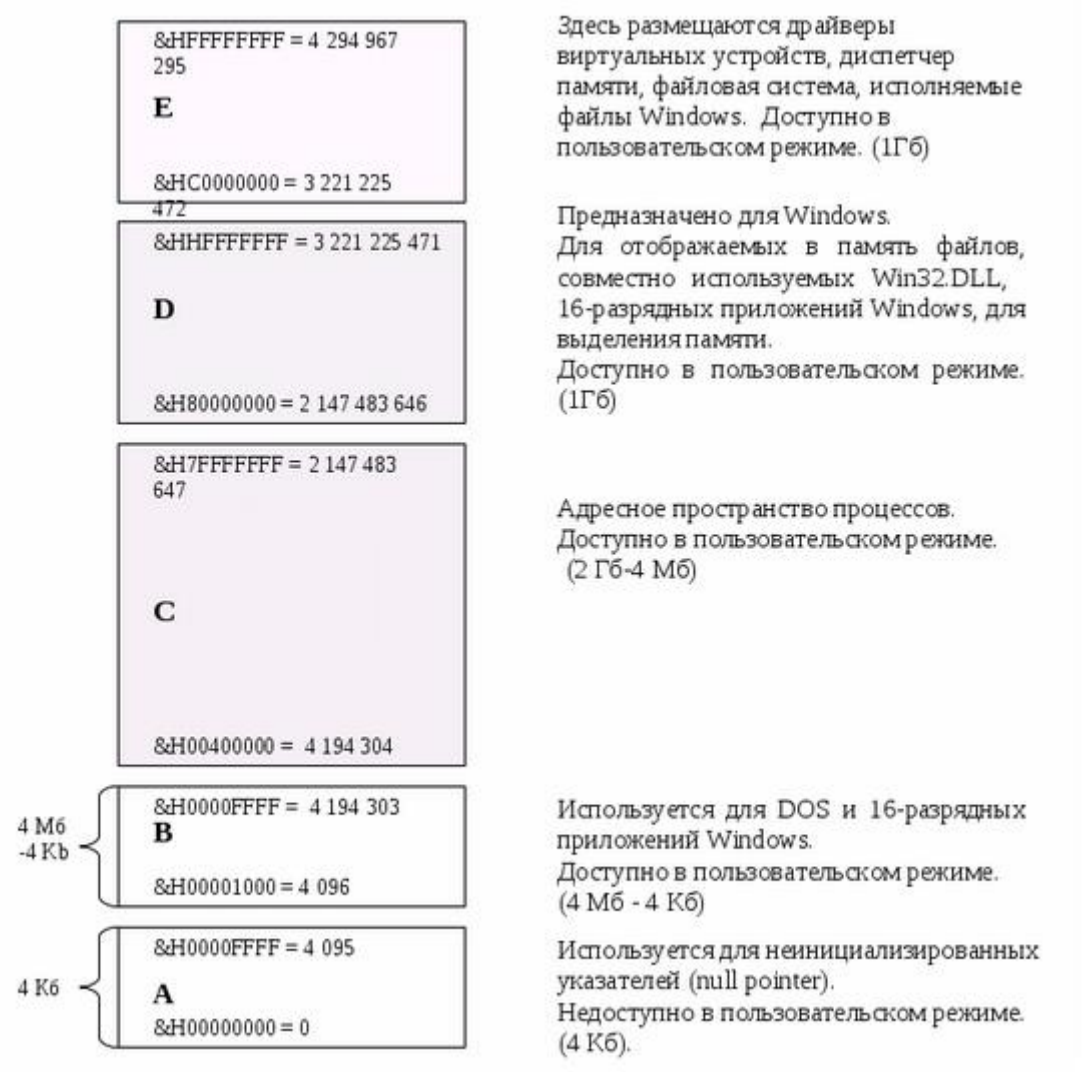

Рисунок 7 – Общая схема использования адресного пространства процесса в Windows 9x

1) **Область A.** Как следует из рис. 7, Windows 9x резервирует область A, объем которой всего лишь 4 Кб, для того же, что и Windows NT первые 64 Кб памяти, - с целью предупреждения о нулевых указателях. Эта область защищена, и попытка обращения к ней из программы пользовательского режима приводит к ошибке нарушения доступа.

2) **Область B.** Данная область памяти используется для поддержания совместимости с приложениями DOS и 16-разрядными приложениями Windows. Несмотря на потенциальную доступность, она не должна использоваться для программирования.
3) **Область C** – это адресное пространство, используемое прикладными программами и их DLL. Здесь размещаются также и модули Windows. Например, если приложению требуется управляющий элемент OCX, его модуль будет находиться в этой области.
4) **Область D.** Windows 9x отображает системные DLL Win32 (KERNEL32.DLL, USER32.DLL и т.д.) в это адресное пространство. Данные файлы используются совместно, то есть несколько процессов могут обращаться к единственной копии такого файла в физической памяти. Область D доступна для программ пользовательского режима (однако размещать их здесь не рекомендуется).
5) **Область E.** Данная область также содержит совместно используемые файлы Windows, такие как исполнительная система Windows и ядро, драйверы виртуальных устройств, файловая система, программы управления памятью. Она также доступна для программ пользовательского режима.

Распределение виртуальной памяти

Каждая страница виртуального адресного пространства может находиться в одном из трех состояний:

– Reserved (зарезервирована) – страница зарезервирована для использования;
– Committed (передана) – для данной виртуальной страницы выделена физическая память в файле подкачки или в файле, отображаемом в память;
– Free (свободна) – данная страница не зарезервирована и не передана, и поэтому в данный момент она недоступна для процесса.

Виртуальная память может быть зарезервирована или передана с помощью вызова API-функции VirtualAlloc:

```
LPVOID VirtualAlloc(
LPVOID IpAddress, //Адрес резервируемой или выделяемой
области.
DWORD dwSise, //Объем области.
DWORD flAllocationType, // Тип распределения.
DWORD flProtect // Тип защиты от доступа.
);
```

Параметр `flAllocationType` может принимать значения следующих констант (помимо других);

– `MEM_RESERVE` – параметр, резервирующий область виртуального адресного пространства процесса без выделения физической памяти. Тем не менее, память может быть выделена при следующем вызове этой же функции;

– `MEM_COMMIT` – параметр, выделяющий физическую память в оперативной памяти или в файле подкачки на диске для заданного зарезервированною набора страниц.

Эти две константы могут объединяться для того, чтобы зарезервировать и выделить память одной операцией.

Разделение процедур резервирования и передачи памяти имеет некоторые преимущества. Например, резервирование памяти является очень полезной процедурой с точки зрения практичности. Если приложению требуется большой объем памяти, можно зарезервировать всю память, а выделить только ту часть, которая нужна в данный момент, раздвигая, таким образом, временные рамки более трудоемкой операции выделения физической памяти.

Windows тоже использует этот подход, когда выделяет память под стек каждого вновь создаваемого потока. Система резервирует 1 Мб виртуальной памяти под стек каждого потока, но выделяет первоначально только две страницы (8 Кб).

Защита памяти

Параметр `flProtect` функции `virtualAlloc` используется для задания типа защиты от доступа, соответствующего вновь выделенной (committed) виртуальной памяти (это не относится к резервируемой памяти). Существуют следующие методы защиты:

- `PAGE_READONLY` присваивает доступ «только для чтения» выделенной виртуальной памяти;
- `PAGE_READWRITE` назначает доступ «чтение-запись» выделенной виртуальной памяти;
- `PAGE_WRITECOPY` устанавливает доступ «запись копированием» (copy-onwrite) выделенной виртуальной памяти.
- `PAGE_EXECUTE` разрешает доступ «выполнение» выделенной виртуальной памяти. Тем не менее, любая попытка чтения - записи этой памяти приведет к нарушению доступа;
- `PAGE_EXECUTE_READ` назначает доступ «выполнение» и «чтение»;
- `PAGE_EXECUTE_READWRITE` разрешает доступ «выполнение», «чтение» и «запись»;
- `PAGE_EXECUTE_WRITECOPY` присваивает доступ «выполнение», «чтение» и «запись копированием»;
- `PAGE_NOACCESS` запрещает все виды доступа к выделенной виртуальной памяти.

Любые из этих значений, за исключением `PAGE_NOACCESS`, могут комбинироваться при помощи логического оператора OR со следующими двумя флагами:

- `PAGE_GUARD` определяет помеченные страницы как защищенные (guard page). При любой попытке обращения к защищенной странице система возбуждает исключительную ситуацию `STATUS_GUARD_PAGE` и снимает с данной страницы статус защищенной. Таким образом, защищенные страницы предупреждают только о первом обращении к ним;

– PAGE_NOCACHES запрещает кэширование выделенной памяти.

Следует объяснить, что такое доступ «запись копированием». Допустим, некоторая страница физической памяти совместно используется двумя процессами. Если она помечена как «только для чтения», то два процесса без проблем могу совместно пользоваться этой страницей. Однако возможны ситуации, когда каждому процессу требуется разрешить запись в эту память, но без воздействия на другой процесс. После установки защиты «запись копированием» при попытке записи в совместно используемую страницу система создаст ее копию специально для процесса, которому нужно осуществить запись. Таким образом, данная страница перестает быть совместно используемой, а представление ее данных в других процессах остается неизменным.

Необходимо отметить, что атрибуты защиты страницы могут быть изменены с помощью API-функции VirtualProtect.

Гранулярность при распределении памяти

Если параметр IpAddress не является кратным 64 Кб, то система округляет указанный адрес в меньшую сторону до ближайшего числа, кратного 64 Кб. Windows всегда выравнивает начальный адрес виртуальной памяти на границу гранулярности распределения, которая является числом кратным 64 Кб (при использовании процессоров Intel). Другими словами, начальный адрес любого блока зарезервированной памяти представляет собой число кратное 64 Кб.

Кроме того, объем выделяемой памяти всегда кратен объему системной страницы, то есть 4 Кб. Поэтому функция VirtualAlloc будет округлять любое запрашиваемое количество байтов в большую сторону до ближайшего числа, кратного объему страницы.

Дескриптор виртуальных адресов

Система отслеживает, какие из виртуальных страниц являются зарезервированными, при помощи структуры, называемой дескриптором

виртуальных адресов (Virtual Address Descriptor - VAD). Другого способа определения не существует.

Управление виртуальной памятью

Виртуальный адрес не указывает на подлинное месторасположение объекта в физической памяти, вместо этого системой поддерживается карта страниц (page map) для каждого процесса, с помощью этой структуры производится трансляция виртуальных адресов в соответствующие физические.

MMU

MMU (Memory Management Unit) – часть операционной системы, занимающаяся управлением виртуальной памятью. Диспетчер MMU отображает виртуальные адреса в адресном пространстве процесса на физические страницы памяти компьютера.

MMU скрывает от потоков процесса физическую организацию памяти, чтобы гарантировать, что поток может получать доступ только к собственному адресному пространству, но не к пространству других процессов. Таким образом, с точки зрения потока, организация памяти ее процесса выглядит гораздо проще, чем реальное размещение страниц в физической памяти.

Виртуальная память позволяет операционной системе выделять процессам больше памяти, чем физически установлено на компьютере. Это достигается за счет использования дискового пространства для расширения памяти и обмена кодом между физической памятью и диском по мере надобности. Виртуальная память отображается на адреса физической.

На рисунке 8 показан процесс преобразования при отображении виртуальных адресов в физические. Он называется попаданием в (физическую) страницу (page hit).

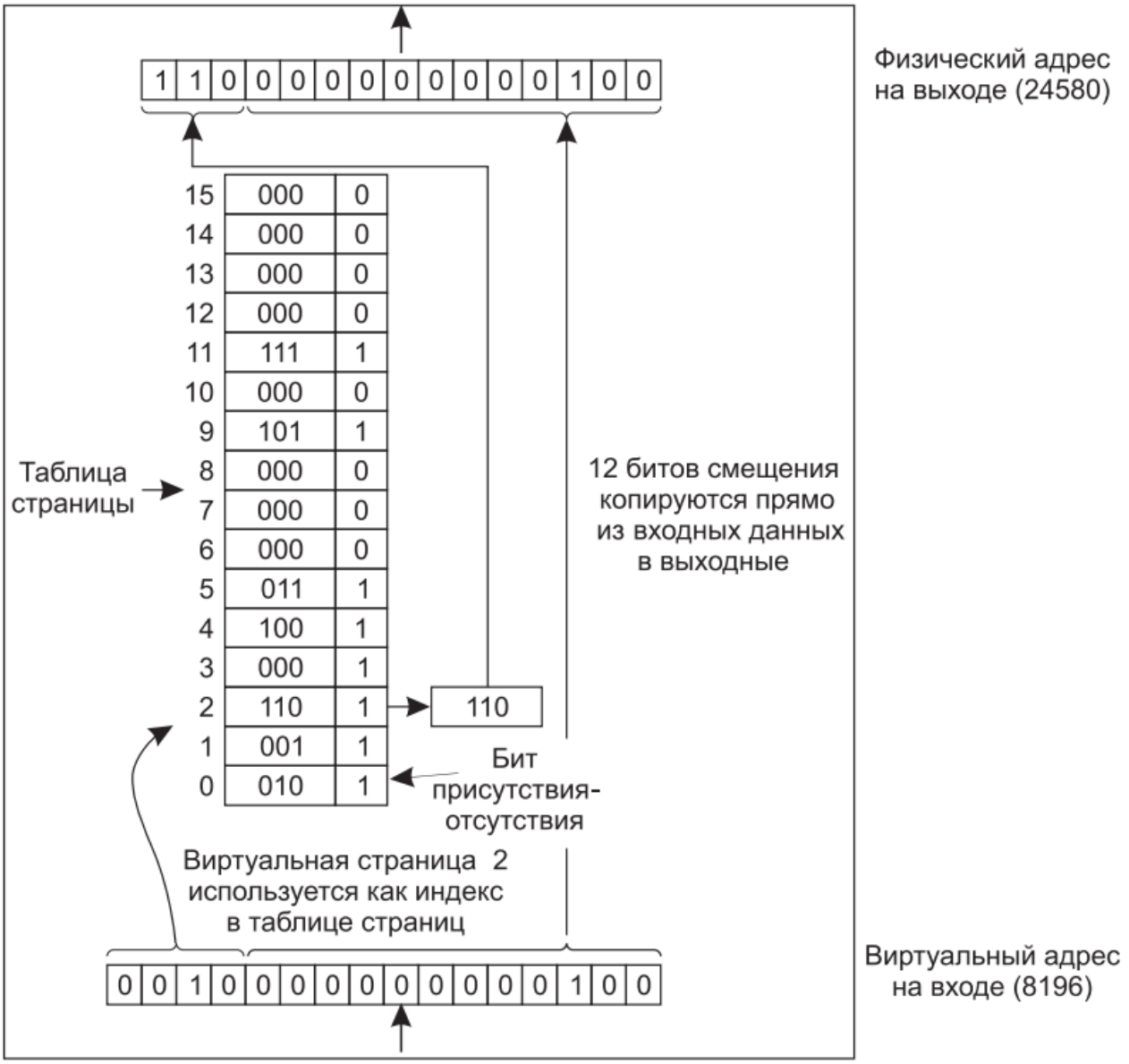

Рисунок 8 – Преобразование виртуального адреса в физический

В целях повышения эффективности Диспетчер памяти осуществляет выделение памяти в два этапа: на первом этапе выполняется резервирование памяти, а на втором – фактическая ее передача процессу. Переданная память (committed memory) является частью файла подкачки, представляющего собой файл на жестком диске, используемый для записи содержимого страниц, откачанных из памяти. Зарезервированная память (reserved memory) представляет собой набор адресов виртуальной памяти, выделенных по запросу некоторого потока (thread). Сначала происходит резервирование виртуальной памяти, и только после этого – ее передача. Фактически процедура

резервирования памяти поддерживает непрерывность выделенного процессу адресного пространства, которое затем потребляется по мере надобности.

Все виртуальные адреса делятся на три части. Самая левая часть (биты 22-31) содержит индекс каталога страниц процесса. Windows поддерживает отдельный каталог страниц для каждого процесса. Его адрес хранится в одном из регистров центрального процессора, который называется CR3. (В операцию переключения задач входит переведение CR3 в состояние, когда он указывает на каталог страниц процесса, на который осуществляя переключение.) Каталог страниц одержит 1024 четырехбайтовых элемента.

Windows поддерживает для каждого процесса совокупность таблиц страниц (page table). Каждый элемент каталога страниц содержит уникальный номер. Поэтому Windows поддерживает до 1024 таблиц страниц. (В действительности таблицы страниц создаются только при попытке обращения к данным или коду конкретному виртуальному адресу, а не когда выделяется виртуальная память).

Следующая часть виртуального адреса (биты 12-21) используется в качестве индекса в таблице страниц, соответствующей выбранному элементу каталога страниц. Каждый элемент таблицы, соответствующий указанному индексу, содержит в 20 старших разрядах номер страничного блока, который задает конкретный страничный блок физической памяти.

Третья, и последняя, часть виртуального адреса (биты 0-11) представляет собой смещение в данном страничном блоке. Сочетание номера страничного блока и смещения дают в совокупности адрес физической памяти.

Так как каждая таблица страниц состоит из 1024 элементов и количество таблиц равно 1024, общее количество страничных блоков, которое можно определить, таким образом, будет 1024 x 1024 = 210 x 210 = 220. Так как каждый страничный блок имеет объем 4 Кб = 4 x 210 байт, то теоретический предел физического адресного пространства будет 4 x 230 = 4 Гб.

Выделение памяти процессу означает выделение ее в файле подкачки.

У этой довольно сложной схемы преобразования есть несколько важных преимуществ. Одно из них - очень небольшой объем страничных блоков, которые легко могут быть размещены в памяти. Гораздо легче найти непрерывный блок памяти размером 4 Кб, чем, скажем, 64 Кб.

Но основное преимущество заключается в том, что адреса виртуальной памяти двух процессов могут быть сознательно преобразованы в разные или в одни и те же физические адреса.

Предположим, что Process1 и Process2 обращаются в программе к одному и тому же виртуальному адресу. При преобразовании виртуальных адресов в физические для каждого из процессов используются их собственные каталоги страниц. Поэтому, хотя индексы в каталогах страниц одинаковы и в том, и в другом случаях, они все же представляют собой индексы из разных каталогов. Таким способом VMM может гарантировать, что виртуальные адреса каждого процесса будут преобразованы в разные физические адреса.

С другой стороны, VMM может также дать гарантию, что виртуальные адреса двух процессов, независимо от того являются ли они одинаковыми или нет, будут преобразованы в один и тот же физический адрес. Один из способов добиться этого - установить соответствующий элемент в обоих каталогах страниц на одну и ту же таблицу страниц и, следовательно, на один и тот же страничный блок. Таким образом, процессы могут совместно использовать физическую память.

Каталог и таблицы системных страниц

Нужно также упомянуть, что Windows поддерживает каталог системных страниц (system page directory) для работы с виртуальной памятью, зарезервированной Windows, так же, как и соответствующую совокупность таблиц системных страниц.

Совместно используемые страницы

Ситуация с совместно используемой физической памятью является значительно более сложной. Следует отметить, что VMM использует концепцию, называемую прототипированием элементов таблицы страниц. Идея заключается в том, что обычные элементы таблицы каждого из совместно использующих память процессов указывают не на физическую память, а на общий прототип элемента таблицы страниц. А тот, в свою очередь, может ссылаться на совместно используемую физическую память.

Два процесса могут совместно использовать объект «проецируемый файл». При этом, при помощи функции `MapViewOfFile` каждый процесс проецирует этот объект на свое адресное пространство и используют эту часть адресного пространства как разделяемую область данных. Общий механизм таков: один процесс создает объект «проецируемый файл» с помощью функции `CreateFileMapping` и порождает другой процесс, передавая ему в наследство описатель этого объекта. Дочерний процесс может пользоваться этим описателем наравне с родительским. Проблема состоит только в том, как сообщить дочернему процессу, какой из переданных ему в наследство описателей является описателем «проецируемого файла». Это можно сделать любым способом. Например, передачей параметров при запуске процесса, через переменные среды, передачей сообщения в главное окно процесса и так далее. Общая область данных может быть создана не только путем проецирования файла, но и путем проецирования части файла подкачки. Для этого в функцию `CreateFileMapping` необходимо передать в качестве параметра не описатель ранее открытого файла, а `-1`. В этом случае необходимо задать размеры выделяемой области. Кроме того, в параметре `lpName` можно задать имя объекта, которое является глобальным в системе. Если это имя задается в системе впервые, то процессу выделяется новая область данных, а если имя было уже задано, то именованная область данных предоставляется для совместного использования.

Если один процесс меняет разделяемую область данных, то она меняется и для другого процесса. Операционная система обеспечивает **когерентность** разделяемой области данных для всех процессов. Но для обеспечения когерентности процессы должны работать с одним объектом «проецируемый файл», а не с одним файлом.

Рабочие наборы

Каждая страница виртуального адресного пространства процесса объемом 4 Гб существует в одном из трех состояний - свободном (free), зарезервированном (reserved) или переданном (committed). Теперь можно также сказать, что каждая переданная страница (committed page) является или действительной, или недействительной. Совокупность действительных страниц, то есть спроецированных на физическую память, называют рабочим наборам (working set) процесса. Рабочий набор постоянно меняется по мере того, как страницы подкачиваются в память или, выполняется обратное действие.

Системный рабочий набор (system working set) характеризует виртуальные страницы системной памяти, которые в данный момент отображены на физическую память.

Размер рабочего набора процесса ограничен теми установками, которые определяет Windows в зависимости от объема физической памяти. Эти значения приведены в следующей таблице 14.

Таблица 14 – Размер рабочего набора процесса.

Модель памяти	**Объем памяти**	**Минимальный размер рабочего набора процесса**	**Максимальный размер рабочего набора процесса**
Small	<=19 Мб	20 страниц (80 Кб)	45 страниц (180 Кб)
Medium	20-32 Мб	30 страниц (120 Кб)	145 страниц (580 Кб)
Large	>=33 Мб	50 страниц (200 Кб)	345 страниц(1380Кб)

Эти пределы могут быть изменены с помощью API-функции:

```
SetProcessWorkingSetSize:
BOOL SetProcessWorkingSetSize(
```

```
HANDLE hProcess, // Открытый дескриптор интересующего
процесса.
DWORD dwMinimumWorkingSetSize,
// Задает мин. размер рабочего набора в байтах.
DWORD dwMaximumWorkingSetSize
// Задает максимальный размер рабочего набора в байтах.
);
```

Присвоение каждому из двух параметров размера значения `-1` приведет к тому, что функция сожмет размер рабочего набора до 0 и тем самым временно удалит данный процесс из физической памяти.

Действительный размер рабочего набора процесса может изменяться во времени, так как Windows увеличивает рабочий набор, если замечает, что у процесса большое количество страничных промахов.

Таблица 14 – Пределы размера системного рабочего набора.

Модель памяти	**Объем памяти**	**Минимальный размер рабочего набора процесса**	**Максимальный размер рабочего набора процесса**
Small	<=19 Мб	388 страниц (1,5 Мб)	500 страниц (180 Мб)
Medium	20-32 Мб	688 страниц (2,7 Мб)	1150 страниц (580 Мб)
Large	>=33 Мб	1188 страниц (4,6 Мб)	2050 страниц (1380 Мб)

База данных страничных блоков

Windows фиксирует состояние каждой физической страницы памяти в структуре данных, называемой базой данных страничных блоков (Page Frame Database). Каждая физическая страница может находиться в одном из восьми различных стояний:

- активная, или действительная (active, valid). Страница в текущий момент отображается на виртуальную память, входя, таким образом, в рабочий набор страниц;
- переходная (transition). Страница в процессе перехода к активному состоянию;

– резервная (standby). Страница только что вышла из состояния «активная», но осталась неизменной;
– измененная (modified). Страница вышла из состояния «активная». Ее содержание, пока она находилась в указанном состоянии, было изменено, но еще не записано на диск;
– измененная незаписанная (modified no write). Страница находится в состоянии «измененная», но особо помечена как страница, содержимое которой не сброшено на диск. Используется драйверами файловой системы Windows;
– свободная (free). Страница свободна, но содержит произвольные записи и, следовательно, не может использоваться процессом;
– обнуленная (zeroed). Страница свободна и инициализирована нулями потоком нулевой страницы. Может быть выделена процессу;
– плохая (bad). В странице были отмечены ошибки четности или какие-то другие аппаратные ошибки, поэтому она не должна использоваться.

Кучи памяти в 32-разрядной Windows

При создании процесса Windows назначает ему кучу по умолчанию (default heap), то есть изначально резервирует область виртуальной памяти объемом 1 Мб. Тем не менее, при необходимости система будет регулировать размер кучи, которая используется самой Windows для различных целей.

API-функция GetProcessHeap используется для получения дескриптора кучи. При помощи функции HeapCreate, возвращающей дескриптор кучи, программист может создавать дополнительные кучи.

Есть несколько причин создавать дополнительные кучи вместо того, чтобы использовать кучу по умолчанию. Например, те кучи, которые предназначены для конкретных задач, часто оказываются более эффективными. Кроме того, ошибки записи данных в кучу, память для которой выделена из специализированной кучи, не затронут данных других куч. Наконец, выделение памяти из специализированной кучи в общем случае будет означать, что данные

в памяти упакованы более плотно друг к другу, а это может уменьшить потребность в загрузке страниц из файла подкачки. Следует также упомянуть, что доступ к куче упорядочен (serialized), то есть система заставляет каждый поток, пытающийся обратиться к памяти кучи, дожидаться своей очереди, пока другие потоки не закончат производимые операции. Следовательно, только один поток в каждый момент времени может выделять или освобождать память кучи во избежание неприятных конфликтов.

16-разрядная Windows поддерживает и глобальную, и локальную кучи. Соответственно в данной системе реализованы функции `GlobalAlloc` и `LocalAlloc`. Они выполняются, но не очень эффективны, поэтому следует избегать их применения в Win32. Однако их все-таки приходится использовать для некоторых целей, таких как создание окна просмотра буфера обмена.

Функции работы с кучей

Кучи (heaps) – динамически распределяемые области данных.

Для работы с кучами используются следующие функции:

- `GetProcessHeap` возвращает дескриптор кучи процесса по умолчанию;
- `GetProcessHeaps` возвращает список дескрипторов всех куч, используемых в данный момент процессом;
- `HeapAlloc` выделяет блок памяти из заданной кучи;
- `HeapCompact` дефрагментирует кучу, объединяя свободные блоки. Может также освобождать неиспользуемые страницы памяти кучи;
- `HeapCreate` создает новую кучу в адресном пространстве процесса;
- `HeapDestroy` удаляет заданную кучу;
- `HeapFree` освобождает предварительно выделенные блоки памяти кучи;
- `HeapLock` блокирует кучу, при использовании данной функции только один поток имеет к ней доступ. Другие потоки, запрашивающие доступ, переводятся в состояние ожидания до тех пор, пока поток, владеющий

кучей, не разблокирует ее. Это одна из форм синхронизации потоков, то есть тот прием, которым система реализует упорядоченность доступа;

- `HeapReAlloc` перераспределяет блоки памяти кучи. Используется для изменения размера блока;
- `Heapsize` возвращает размер выделенного блока памяти кучи;
- `HeapUnlock` разблокирует кучу, которая до этого была заблокирована функцией `HeapLock`;
- `HeapValidate` проверяет пригодность кучи (или отдельного ее блока), если имеются ли какие-либо повреждения;
- `HeapWalk` позволяет программисту просматривать содержимое кучи. Обычно используется при отладке.

Отображения виртуальной памяти

Функция Win32 API VirtualQuery может использоваться для получения информации о состоянии адресов виртуальной памяти. Синтаксис ее таков:

```
DWORD VirtualQuery(
LPCVOID IpAddress,  // Адрес области.
PMEMORY_BASICONFORMATION IpBuffer, // Адрес информационного
буфера
DWORD dwLength // Размер буфера
);
```

Используется также функция `VirtualQueryEx`, расширенная версия `VirtualQuery`, которая позволяет получать информацию о внешних виртуальных адресных пространствах:

```
DWORD VirtualQueryEx(
HANDLE hProcess     // Дескриптор процесса
LPCVOID IpAddress, // Адрес области
MEMORY_BASIC_INFORMATION IpBuffer, // Адрес информационного
буфера
DWORD dwLength // Размер буфера
);
```

Параметр `hProcess` - это дескриптор процесса. Параметр `IpAddress` - это начальный адрес для записи результирующих данных, который будет округляться в меньшую сторону до ближайшего кратного размеру страницы (4 Кб). Обе функции возвращают информацию в следующую структуру.

```
Struct MEMORY_BASlC_INFORMATION {
PVOID BaseAdciress; // Базовый адрес области
PVOID AllocationBase; // Базовый адрес выделенной области
DWORD AllocationProtect; // Первоначальная защита от доступа
DWORD RegionSize; // Размер области в байтах
DWORD State; // Передана зарезервирована, свободна
DWORD Protect; // Текущая защита от доступа
DWORD Type; // Тип страниц
}
```

Чтобы понять принцип действия членов этой структуры, необходимо знать о назначении данной функции. Чтобы сделать определение более понятным, назовем страницу, которой принадлежит адрес `IpAddress`, заданной (specified) Следующий рисунок поможет разобраться в новой терминологии.

Рисунок 9 – Отображение виртуальной памяти

Функция VirtualQueryEx всегда заполняет следующие члены структуры MEMORY_BASIC_INFORMATION:

- BaseAddress, которая возвращает базовый адрес заданной страницы;
- RegionSize, представляющая собой количество байтов от начала заданной страницы до вершины заданной области.

Если страница, содержащая адрес IpAddress, свободна (не зарезервирована и не передана), член структуры State содержит символьную константу MEM_FREE. Остальные члены (кроме BaseAddress и RegionSize) не имеют значения.

Если страница, содержащая адрес IpAddress, не свободна, функция определяет выделенную область (allocation region), то есть область виртуальной памяти, которая включает заданную страницу и была, первоначально выделена с помощью вызова функции VirtualAlloc.

Начиная с базового адреса заданной страницы, функция последовательно просматривает все страницы выделенной области, проверяя, совпадают ли их типы выделения (allocation type) и защиты (protection type) с аналогичными типами заданной страницы. Совокупность всех совпадающих упорядоченных страниц представляет собой заданную область. К ней относятся значения структуры MEMORY_BASIC_INFORMATION. Страница считается совпадающей с заданной страницей, если она удовлетворяет двум следующим условиям:

- страница имеет тот же тип выделения, что и первоначальная страница, в соответствии со следующими значениями флага: MEM_COMMIT, MEM_RESERVE, MEM_FREE, MEM_PRIVATE, MEM_MAPPED или MEM_IMAGE;
- страница имеет тот же тип зашиты, что и первоначальная страница, в соответствии со следующими значениями флага: PAGE_READONLY, PAGE_READWRITE, PAGE_NOACCESS, PAGE_WRITECOPY, PAGE_EXECUTE, PAGE_EXECUTE_READ, PAGE_EXECUTE_READWRITE, PAGE_EXECUTE_WRITECOPY, PAGE_GUARD или PAGE_NOCACHE.

Рассмотрим остальные члены структуры MEMORY_BASIC_INFORMATION:

- `AllocationBase` - базовый адрес выделенной области;
- `AllocationProtect` - первоначальный тип защиты выделенной области;
- `State` - одно из трех значений: `MEM_FREE`, `MEM_RESERVE` или `MEM_COMMIT`. Относится к заданной области;
- `Protect` - текущий тип защиты заданной области;
- `Type` - одно из трех значений: `MEM_IMAGE, MEM_MAPPED` или `MEM_PRIVATE`. Относится к заданной области. Эти константы имеют следующий смысл: `MEM_IMAGE` указывает, что область отображена на файл образа задачи (image file), то есть на загрузочный; `MEM_MAPPED` указывает, что область отображена на не загрузочный отображаемый в память файл (например, файл данных); `MEM__PRIVATE` указывает, что область используется одним процессом, а не совместно.

Запуск исполняемых файлов и динамически связываемых библиотек

При исполнении функции `CreateProcess` система обращается к VMM для выполнения следующих действий:

1) Создать адресное пространство процесса (размером 4Gb)
2) Резервировать в адресном пространстве процесса регион размером, достаточным для размещения исполняемого файла. Начальный адрес региона определяется в заголовке EXE-модуля. Обычно он равен `0x00400000`, но может быть изменен при построении файла параметром /BASE компоновщика.
3) Отобразить исполняемый файл на зарезервированное адресное пространство. Тем самым VMM распределяет физические страницы не из файла подкачки, а непосредственно из EXE-модуля.
4) Таким же образом отобразить на адресное пространство процесса необходимые ему динамически связываемые библиотеки. Информация о необходимых библиотеках находится в заголовке EXE-модуля. Желательное расположение региона адресов описано внутри библиотеки. Visual C++, например, устанавливает по умолчанию адрес `0x10000000`. Этот адрес может

так же изменяться параметром /BASE компоновщика. Если при загрузке выясняется, что данный регион занят, то система попытается переместить библиотеку в другой регион адресов, на основе настроечной информации, содержащейся в DLL-модуле. Однако эта операция снижает эффективность системы и, кроме того, если настроечная информация удалена при компоновке библиотеки параметром /FIXED, то загрузка становится вообще невозможной.

При одновременном запуске нескольких приложений Win32® отображает один и тот же исполняемый файл и библиотеки на адресные пространства различных процессов. При этом возникает проблема независимого использования процессами статических переменных и областей данных. Кроме того, изменение данных исполняющейся программой не должно приводить к изменению EXE-файла. Win32 откладывает решение этой проблемы на максимально возможный срок (Lazy Evaluation). При этом используется классический механизм отложенного копирования (copy-on-write — копирование при попытке записи). Все страницы адресного пространства процесса получают атрибут защиты PAGE_WRITECOPY. При попытке записи в такую страницу возникает исключение нарушения защиты и VMM копирует страницу для обратившегося процесса. В дальнейшем эта страница будет выгружаться в файл подкачки. После копирования происходит рестарт команды, вызвавшей исключение.

Практическое задание

а) Работа с блоком адресов в адресном пространстве процесса

1) Зарезервируйте блок адресов и выделите для него физическую память с помощью функции `VirtualAlloc()`.
2) Проверьте правильность работы функции, определив, что адрес зарезервированного блока ненулевой.
3) Освободите физическую память, выделенную для региона.
4) Освободите зарезервированный блок адресов с помощью `VirtualFree()`.
5) Проверьте успешность работы функции.

Задание, для получения дополнительных баллов

1) Вручную проверьте сколько памяти на вашем ПК.
2) Зарезервируйте блок виртуальной памяти с помощью переменной `size = 128K`. Заполните его каким-нибудь значениями (`i=1…N` или с помощью функции `Rand()`). Можно выводить индикатор, когда заполняется блок в 4-8 Кб. Используйте специальную переменную для учета времени, которое понадобилось, чтобы заполнить блок, и выводите ее на дисплей (Указание: воспользуйтесь функцией `GetTickCount()`). После выполнения задания освободите блок.
3) Увеличьте размер выделяемой памяти `size` вдвое, пока `size <= 1-2-3` Гб, затем повторите п.2.
4) Цель задания – найти время, когда включится механизм подкачки (начинается работа с диском). При этом время работы должно вырасти как минимум на порядок. Определите при каком значении переменной `size` это произойдет. Замечание: на ПК должно быть достаточно свободного места на жестком диске!

б) Работа с файлом, проецируемым в память

1) Создайте файл и запишите в него последовательность из нескольких символов.
2) Создайте объект типа файл с помощь функции `CreateFile()`.
3) Создайте объект типа проецируемый файл, функция `CreateFileMapping()`.
4) Спроецируйте его на массив типа `char`, функция `MapViewOfFile()`.
5) Прочитайте содержимое массива, измените его содержимое.
6) Снимите проекцию с помощью `UnmapViewOfFile()`.
7) Уничтожьте объекты типа файл и проецируемый файл с помощью `CloseHandle()`.
8) Просмотрите содержимое файла.

в) Организация взаимодействия двух процессов через общую область данных.

1) Создайте новый проект.
2) Создайте объект типа проецируемый файл, но вместо указателя на объект файл используйте константу `INVALID_HANDLE_VALUE`, обязательно укажите размер региона, параметру `lpName` присвойте значение, которое станет его уникальным идентификатором в системе. Тем самым вы часть файла подкачки сделали доступным для проецирования в память и при этом задали ей глобальное имя в системе.
3) Спроецируйте её на массив типа `char`.
4) Задайте цикл обработки нажатия клавиш, который бы позволял считывать и записывать в массив символы либо строки.
5) Выполните вывод на экран содержимого проецируемого файла.
6) Снимите проекцию.
7) Уничтожьте объект типа проецируемый файл.
8) Создайте исполняемый файл и запустите два экземпляра программы.
9) Второй экземпляр при попытке создать регион файла подкачки, проецируемый в память обнаружит, что имя уже используется системой, поэтому он не создаст новый регион, а откроет существующий, в результате один и тот же регион файла подкачки будет спроецирован на адресные пространства двух процессов.
10) Выполните обмен данными.

Контрольные вопросы

1) Опишите общую схему использования адресного пространства процесса и назначение соответствующих областей.
2) Что такое менеджер виртуальной памяти?
3) В чем отличие виртуальной памяти от физической? Нарисуйте схему преобразования виртуального адреса в физический.
4) Каким образом распределяется виртуальная память?

5) Каким образом можно защитить память от доступа?
6) Как происходит работа приложений с виртуальной памятью?
7) Что означает термин «гранулярность» при распределении памяти?
8) Дайте определение дескриптора виртуальных адресов.
9) Какие функции выполняет MMU?
10) Как происходит совместное использование страниц?
11) Рабочие наборы: определение, назначение, границы.
12) Для чего нужна база данных страничных блоков? В каких состояниях может находится страница?
13) Запуск исполняемых файлов и динамически связываемых библиотек.
14) Файлы данных, проецируемые в память.
15) Взаимодействие процессов через общую область данных. Когерентность.
16) Дайте определение кучи. Какие функции позволяют работать с кучами?