Лабораторная работа № 4

Архитектура памяти Windows

Цель работы: получение практических навыков по использованию Win32 API для исследования памяти Windows

Типы памяти

В Windows реализована система виртуальной памяти, основанная на модели плоского (линейного) адресного пространства, которая создает для каждого процесса иллюзию огромного закрытого адресного пространства. Виртуальная память формирует логическое представление памяти, которое может не соответствовать физической структуре. Во время выполнения диспетчер памяти —при поддержке оборудования —транслирует (отображает) виртуальные адреса в физические, где фактически хранятся данные. Управляя защитой и отображением адресов, ОС может предотвращать столкновения отдельных процессов или перезапись данных ОС.

Так как в большинстве систем объем физической памяти гораздо меньше суммарного объема виртуальной памяти, используемой работающими процессами, диспетчер памяти переносит (выгружает) часть содержимого памяти на диск. Выгрузка данных на диск освобождает физическую память, чтобы она могла использоваться для других процессов или самой ОС. Когда поток обращается по виртуальному адресу, выгруженному на диск, диспетчер виртуальной памяти подгружает информацию обратно в память с диска.

На рис. 1.1 продемонстрировано использование виртуальной памяти двумя процессами; части памяти отображаются в физическую память (ОЗУ), а другие части выгружаются на диск. Обратите внимание на то, что непрерывные блоки виртуальной памяти могут отображаться в несмежные блоки физической памяти. Эти блоки называются страницами, а их размер по умолчанию составляет 4 Кбайт.

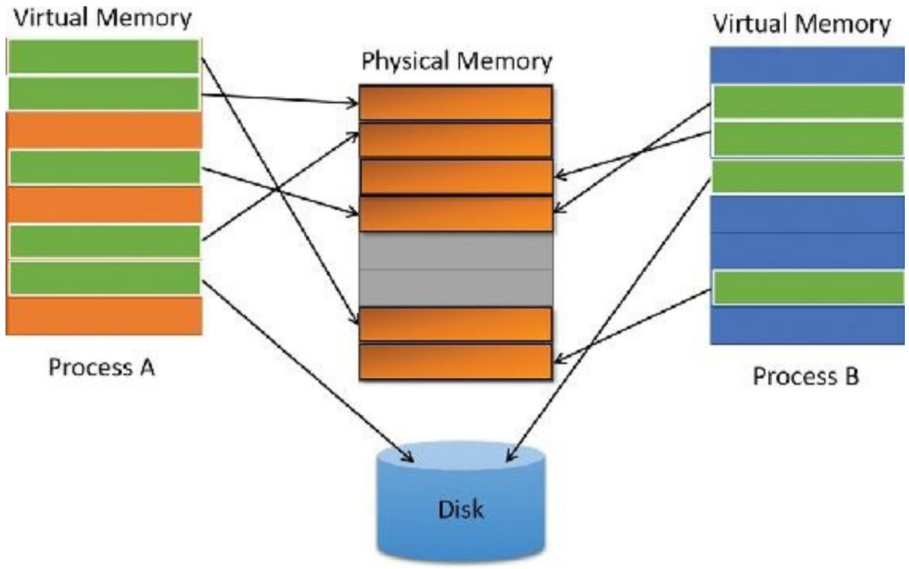


Рис. 1.1

Размер виртуального адресного пространства изменяется в зависимости от аппаратной платформы. В 32-разрядных платформах x86 общее виртуальное адресное пространство ограничено теоретическим максимумом 4 Гбайт. По умолчанию Windows выделяет нижнюю половину адресного пространства (адреса от 0x80000000 до OxFFFFFFFF) процессам для их собственной закрытой памяти, а верхнюю половину (адреса от 0x80000000 до OxFFFFFFFF) —для собственного защищенного использования памяти ОС. Отображения нижней половины

изменяются в соответствии с виртуальным адресным пространством текущего процесса, но отображения верхней половины (в большинстве случаев) всегда состоят из виртуальной памяти ОС. Windows поддерживает параметры времени загрузки, которые предоставляют процессам, выполняющим особым образом помеченные программы, возможность использовать до 3 Гбайт закрытого адресного пространства, оставляя 1 Гбайт для ОС. На рис. 1.2 изображены две типичные структуры виртуального адресного пространства, поддерживаемые 32-разрядными версиями Windows.

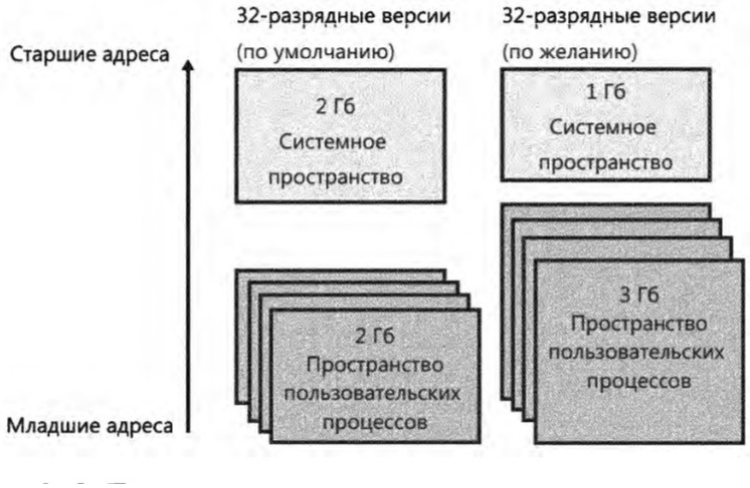


Рис. 1.2. Типичная структура виртуального адресного пространства в 32-разрядных версиях Windows

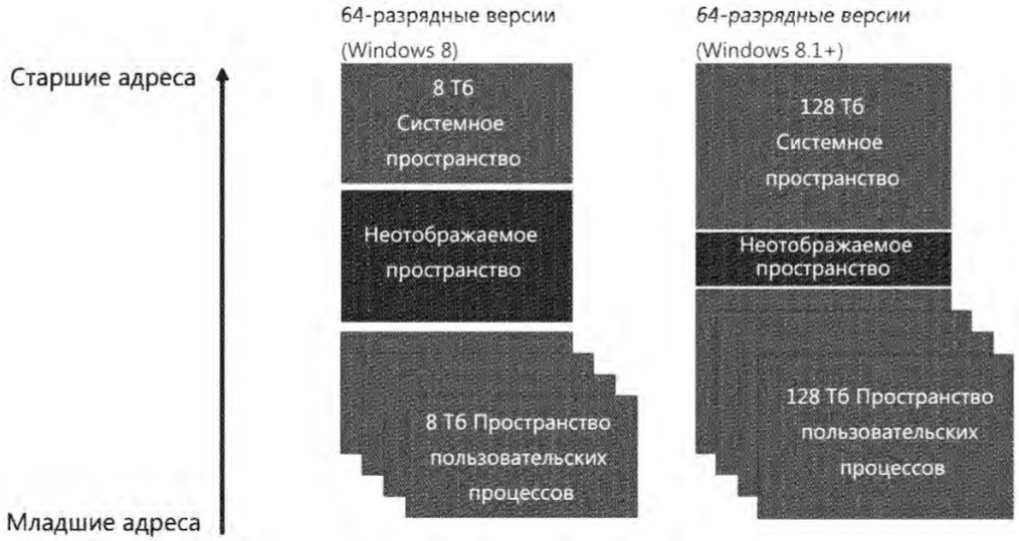


Рис. 1.2. Типичная структура виртуального адресного пространства в 64-разрядных версиях Windows

Физическая память

Физическая память - это реальные микросхемы *RAM*, установленные в компьютере. 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 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 Windows VMM, является страница памяти, называемая также страничным блоком памяти. На компьютерах 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.

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 ни какой взаимосвязи в том, что и процесс А, и процесс В используют один и тот же виртуальный адрес, например &H40000000. В частности, Windows может сопоставить (или не сопоставить) виртуальным адресам каждого процесса разные физические адреса.

Структура адресных пространств x86

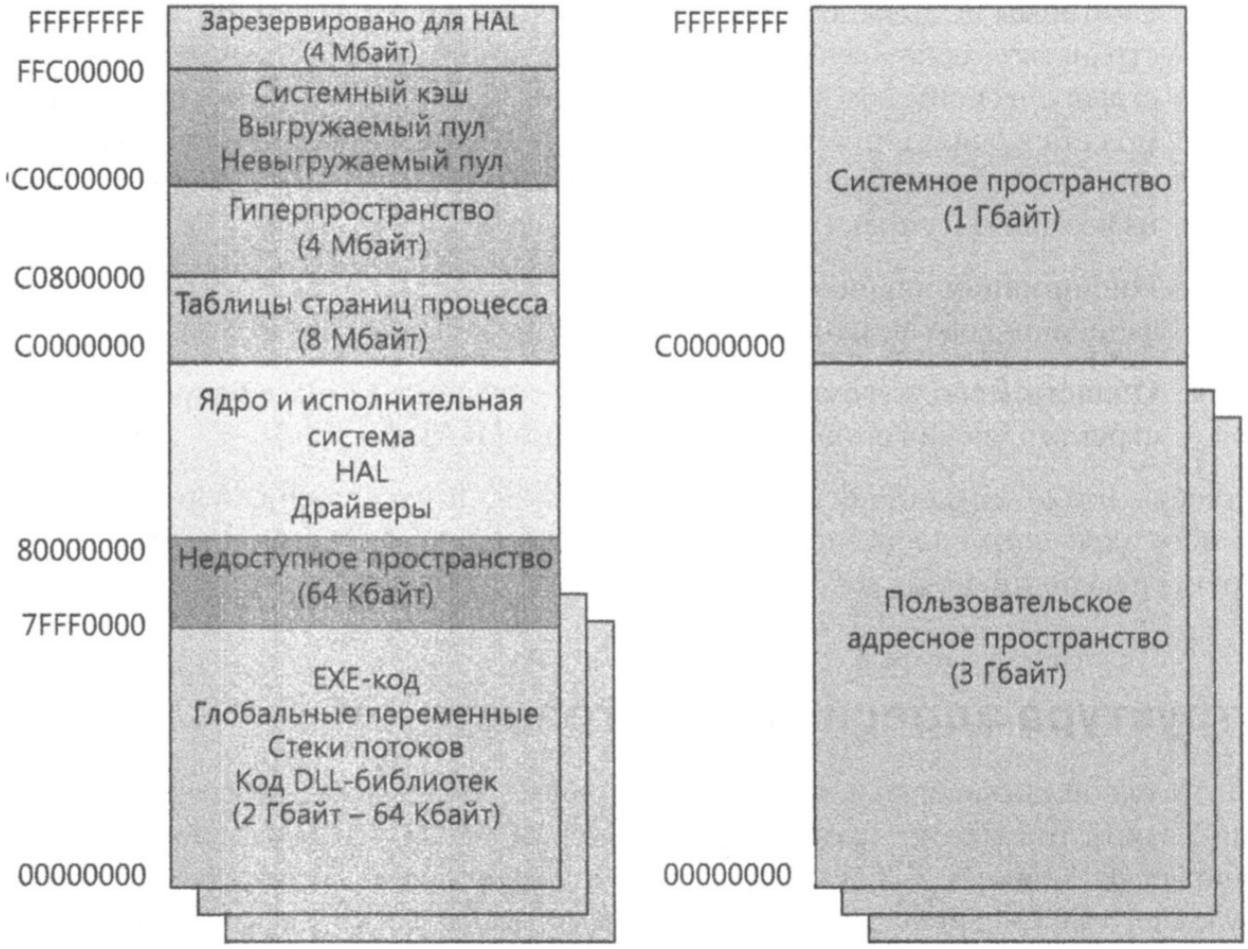


Рис. 2. Структура виртуальных адресных пространств на платформе x86

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 Virtual Protect.

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.

Пример использования функции GlobalMemoryStatus

API-функция GlobalMemoryStatus, записывающаяся таким образом:

в Delphi:

```
procedure GlobalMemoryStatus(var lpBuffer: TMemoryStatus); stdcall;
procedure GlobalMemoryStatus; external kernel32 name 'GlobalMemoryStatus';
```

выводит множество данных, имеющих отношение к памяти, в составе следующей структуры:

```
struct_MEMORYSTATUS {
  DWORD dwLength;        // Размер структуры MEMORYSTATUS.
  DWORD dwMernoryLoad;   // Процент используемой памяти.
  DWORD dwTotalPhys;     // Количество байтов физической памяти.
  DWORD dwAvailPhys;     // Количество свободных байтов физической памяти.
  DWORD dwTotalPageFile;   // Размер в байтах файла подкачки.
  DWORD dwAvailPageFile; // Количество свободных байтов файла подкачки.
  DWORD dwTotalVirtual;    // Количество байтов адресного пространства,
                           // доступного пользователю.
  DWORD dwAvailvirtual;    // Количество свободных байтов памяти,
                           // доступных пользователю.
End Type
```

В Delphi:

```
_MEMORYSTATUS = record
  dwLength: DWORD;
  dwMemoryLoad: DWORD;
  dwTotalPhys: DWORD;
  dwAvailPhys: DWORD;
  dwTotalPageFile: DWORD;
  dwAvailPageFile: DWORD;
```

```
 dwTotalVirtual: DWORD;
dwAvailVirtual:  DWORD;
end;
```

Управление виртуальной памятью

Рассмотрим, как диспетчер виртуальной памяти Windows преобразует адреса виртуальной памяти и физические.

Преобразование виртуальных адресов в физические: попадание

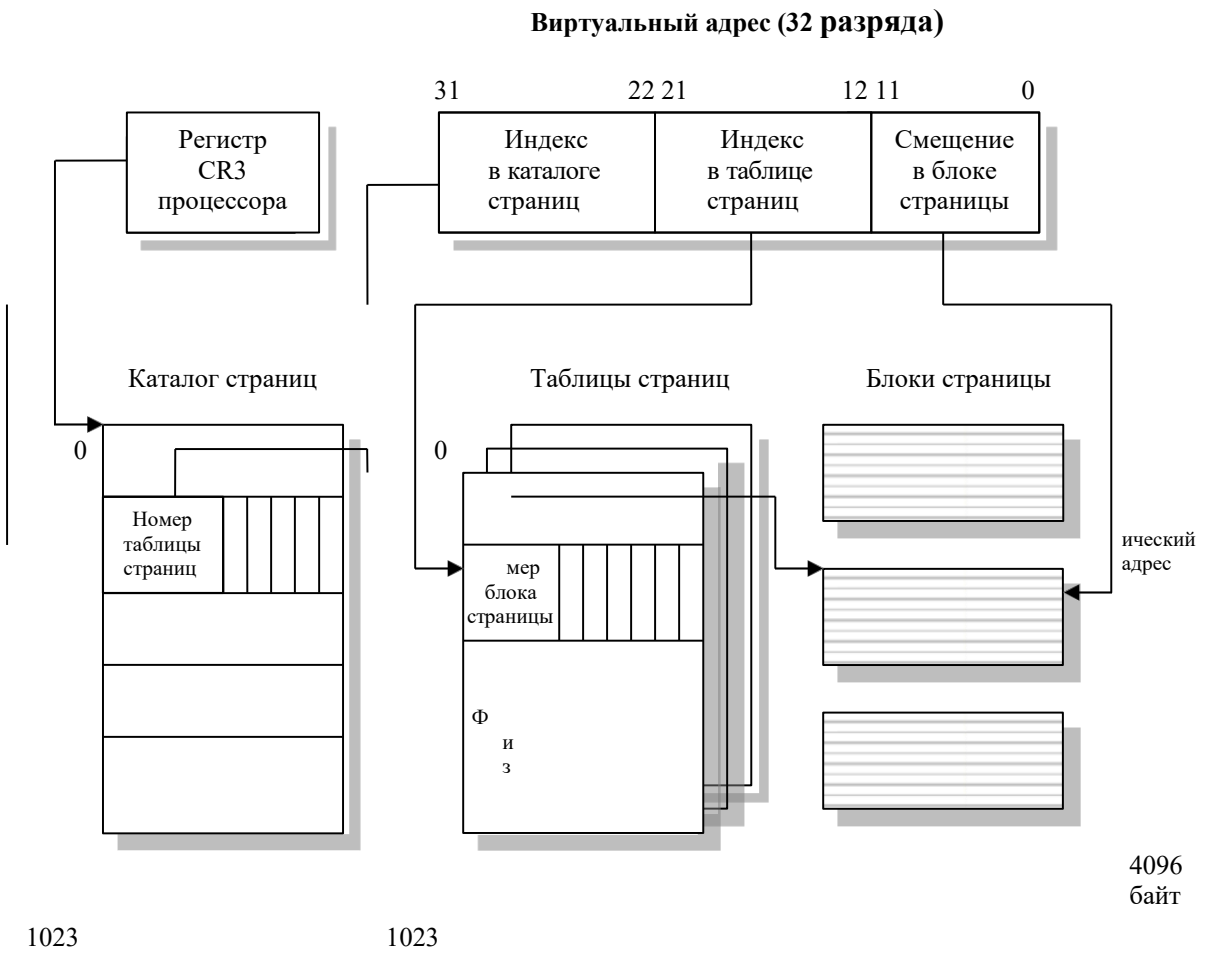
На рисунке показан процесс преобразования при отображении виртуальных адресов в физические. Он называется попаданием в (физическую) страницу (*page hit*).

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 = 2^{10} x 2^{10} = 2^{20}. Так как каждый страничный блок имеет объем 4 Кб = 4 x 2^{10} байт, то теоретический предел физического адресного пространства будет 4 x 2^{30} = 4 Гб.

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, не будем углубляться в детали, а отметим только, 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.

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.

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

Пределы размера системного рабочего набора приведены в следующей таблице.

Модель памяти	Объем памяти	Минимальный размер рабочего набора процесса	Максимальный размер рабочего набора процесса
Small	<=19Мб	388страниц(1,5 Мб)	500 страниц (2,0 Мб)
Medium	20-32 Мб	688 страниц (2,7 Мб)	1150 страниц (4,5 Мб)
Large	>= 32 Мб	1188 страниц (4,6 Мб)	2050 страниц (8 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

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;
- *HeapCreat*е 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.

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.

СОДЕРЖАНИЕ ОТЧЕТА

1. Наименование лабораторной работы, ее цель.

2. Разработанное программное обеспечение для приложения, которое:

   - выдает информацию, получаемую при использовании API *GlobalMemoryStatus.* При выводе информации использовать диаграммы.

   - Составляет карту виртуальной памяти для любого процесса.

3. Примеры разработанных приложений (результаты и тексты программ).