Управление процессами, потоками и памятью в ОС Windows. Часть 2

Лекция

Ревизия: 0.2

Последняя модификация: Vlad Kovtun

Дата последней модификации: 03.04.2015 14:59:00

История изменений

15.09.2010 – Версия 0.1. Первичный документ. Ковтун В.Ю.

08.09.2014 – Версия 0.2. Общая редакция. Ковтун В.Ю.

Содержание

Лекция 11. Управление процессами, потоками и памятью в ОС Windows. Часть 2

Вопросы

1. Введение в диспетчер памяти.
2. Сервисы диспетчера памяти.
3. Системные пулы памяти.
4. Структуры виртуального адресного пространства.
5. Трансляция адресов.
6. Обработка ошибок страниц.
7. Дескрипторы виртуальных адресов.
8. Объекты-разделы.
9. Рабочие наборы.

Введение в диспетчер памяти

Введение

По умолчанию виртуальный размер процесса в 32-разрядной Windows — 2 Гб. Если образ помечен как поддерживающий большое адресное пространство и система загружается со специальным ключом (/3gb в файле boot.ini), 32-разрядный процесс может занимать до 3 Гб в 32-разрядной Windows и до 4 Гб в 64-разрядной. Размер виртуального адресного пространства процесса в 64-разрядной Windows составляет 7152 Гб на платформе IA64 и 8192 Гб на платформе x64.

Максимальный объем физической памяти, поддерживаемый Windows, варьируется от 2 до 1024 Гб в зависимости от версии и редакции Windows. Так как виртуальное адресное пространство может быть больше или меньше объема физической памяти в компьютере, диспетчер управления памятью решает две главные задачи.

Трансляция, или проецирование (mapping), виртуального адресного пространства процесса на физическую память. Это позволяет ссылаться на корректные адреса физической памяти, когда потоки, выполняемые в контексте процесса, читают и записывают в его виртуальном адресном пространстве. Физически резидентное подмножество виртуального адресного пространства процесса называется **рабочим набором** (working set).

Подкачка части содержимого памяти на диск, когда потоки или системный код пытаются задействовать больший объем физической памяти, чем тот, который имеется в наличии, и загрузка страниц обратно в физическую память по мере необходимости.

Кроме управления виртуальной памятью диспетчер памяти предоставляет базовый набор сервисов, на которые опираются различные подсистемы окружения Windows. К этим сервисам относится:

- **поддержка файлов, проецируемых в память** (memory-mapped files) [их внутреннее название — **объекты-разделы** (section objects)],
- поддержка памяти, копируемой при записи,
- поддержка приложений, использующих большие разреженные адресные пространства.

Диспетчер памяти также позволяет процессу выделять и использовать большие объемы физической памяти, чем можно спроецировать на виртуальное адресное пространство процесса (например, в 32-разрядных системах, в которых установлено более 4 Гб физической памяти).

Компоненты диспетчера памяти

Диспетчер памяти является частью исполнительной системы Windows, содержится в файле Ntoskrnl.exe и включает следующие компоненты:

Набор сервисов исполнительной системы для выделения, освобождения и управления виртуальной памятью; большинство этих сервисов доступно через Windows API или интерфейсы драйверов устройств режима ядра.

Обработчики ловушек трансляции недействительных адресов (translation-not-valid) и нарушений доступа для разрешения аппаратно обнаруживаемых исключений, связанных с управлением памятью, а также загрузки в физическую память необходимых процессу страниц.

Несколько ключевых компонентов, работающих в контексте шести различных системных потоков режима ядра:

Диспетчер рабочих наборов (working set manager) с приоритетом 16. Диспетчер настройки баланса (системный поток, создаваемый ядром) вызывает его раз в секунду или при уменьшении объема свободной памяти ниже определенного порогового значения. Он реализует общие правила управления памятью, например усечение рабочего набора, старение и запись модифицированных страниц.

Поток загрузки и выгрузки стеков (process/stack swapper) с приоритетом 23. Выгружает (outswapping) и загружает (inswapping) стеки процесса и потока. При необходимости операций со страничным файлом этот поток пробуждается диспетчером рабочих наборов и кодом ядра, отвечающим за планирование.

Подсистема записи модифицированных страниц (modified page writer) с приоритетом 17. Записывает измененные страницы, зарегистрированные в списке модифицированных страниц, обратно в соответствующие страничные файлы. Этот поток пробуждается, когда возникает необходимость в уменьшении размера списка модифицированных страниц.

Подсистема записи спроецированных страниц (mapped page writer) с приоритетом 17. Записывает измененные страницы спроецированных файлов на диск. Пробуждается, когда нужно уменьшить размер списка модифицированных страниц или когда страницы модифицированных файлов находятся в этом списке более 5 минут. Этот второй поток записи модифицированных страниц требуется потому, что он может генерировать ошибки страниц, в результате которых выдаются запросы на свободные страницы.

Поток сегмента разыменования (dereference segment thread) с приоритетом 18. Отвечает за уменьшение размеров системного кэша и изменение размеров страничного файла.

Поток обнуления страниц (zero page thread) с приоритетом 0. Заполняет нулями страницы, зарегистрированные в списке свободных страниц, (В некоторых случаях обнуление памяти выполняется более скоростной функцией `MiZeroInParallel`.).

Внутренняя синхронизация

Как и другие компоненты исполнительной системы Windows**, диспетчер памяти полностью** реентерабелен и поддерживает одновременное выполнение в multi-CPU системах, управляя тем, как потоки захватывают ресурсы. С этой целью диспетчер памяти контролирует доступ к собственным структурам данным, используя внутренние механизмы синхронизации, например спин-блокировку и ресурсы исполнительной системы.

Диспетчер памяти должен синхронизировать доступ к общесистемным ресурсам, как:

- база данных **номеров фреймов страниц** (PFN) (контроль через спин-блокировку),
- объекты-разделы,
- системный рабочий набор (контроль через спин-блокировку с заталкиванием указателя),
- страничные файлы (контроль через объекты «мьютекс»).

В Windows XP и Windows Server 2003 ряд таких блокировок был либо удален, либо оптимизирован, что позволило резко снизить вероятность конкуренции.

К другим операциям, в которых больше не используется захват блокировок, относятся:

- контроль квот на пулы подкачиваемой и неподкачиваемой памяти,
- управление передачей страниц,
- выделение и проецирование физической памяти через функции поддержки AWE (Address Windowing Extensions).

Эти изменения особенно важны в multi-CPU системах, где они позволили уменьшить частоту блокировки диспетчера памяти на период модификации со стороны другого CPU какой-либо глобальной структуры или вообще исключить такую блокировку.

Конфигурирование диспетчера памяти

Как и большинство компонентов Windows, диспетчер памяти старается автоматически оптимизировать работу систем различных масштабов и конфигураций при разных уровнях загруженности. Некоторые стандартные настройки можно изменить через параметры в разделе реестра `HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemoryManagement`, но, как правило, они оптимальны в большинстве случаев.

Многие пороговые значения и лимиты, от которых зависит политика принятия решений диспетчером памяти, вычисляются в период загрузки системы на основе доступной памяти и типа продукта (Windows 2000 Professional, Windows XP Professional и Windows XP Home Edition оптимизируется для интерактивного использования в качестве персональной системы, а системы Windows Server — для поддержки серверных приложений). Эти значения записываются в различные переменные ядра и впоследствии используются диспетчером памяти. Некоторые из них можно найти поиском в Ntoskrnl.exe глобальных переменных с именами, которые начинаются с `Mm` и содержат слово «maximum» или «minimum».

ВНИМАНИЕ. Не изменяйте значения этих переменных. Как показывают результаты тестирования, автоматически вычисляемые значения обеспечивают оптимальное быстродействие. Их модификация может привести к непредсказуемым последствиям вплоть до зависания и даже краха.

Сервисы диспетчера памяти

Диспетчер памяти предоставляет набор системных сервисов для выделения и освобождения виртуальной памяти, разделения памяти между процессами, проецирования файлов в память, сброса виртуальных страниц на диск, получения информации о диапазоне виртуальных страниц, изменения атрибутов защиты виртуальных страниц и блокировки в памяти.

Как и другие сервисы исполнительной системы, сервисы управления памятью требуют при вызове передачи описателя того процесса, с виртуальной памятью которого будут проводиться операции. Таким образом, вызывающая программа может управлять как собственной памятью, так и памятью других процессов (при наличии соответствующих прав). Например, если один процесс порождает другой, у первого по умолчанию остается право на манипуляции с виртуальной памятью второго. Впоследствии родительский процесс может выделять и освобождать память, считывать и записывать в нее данные через сервисы управления виртуальной памятью, передавая им в качестве аргумента описатель дочернего процесса. Подсистемы используют эту возможность для управления памятью своих клиентских процессов; она же является ключевой для реализации отладчиков, так как им нужен доступ к памяти отлаживаемого процесса для чтения и записи.

Большинство этих сервисов предоставляется через Windows API. В него входят три группы прикладных функций управления памятью:

- для операций со страницами виртуальной памяти (`Virtualxxx`),
- **проецирования файлов в память** (`CreateFileMapping`, `MapViemOfFile`),
- **управления кучами** (`Heapxxx`, а также функции из старых версий интерфейса — `Localxxx` и `Globalxxx`).

Диспетчер памяти поддерживает такие сервисы, как выделение и освобождение физической памяти, блокировка страниц в физической памяти для передачи данных другим компонентам исполнительной системы режима ядра и драйверам устройств через DMA. Имена этих функций начинаются с префикса `Mm`. Кроме того, существуют процедуры поддержки исполнительной системы, имена которых начинаются с `Ex`. Не являясь частью диспетчера памяти в строгом смысле этого слова, они применяются для выделения и освобождения памяти из системных куч (пулов подкачиваемой и неподкачиваемой памяти), а также для манипуляций с ассоциативными списками.

Большие и малые страницы

Виртуальное адресное пространство делится на единицы, называемые **страницами**. Это вызвано тем, что аппаратный блок управления памятью транслирует виртуальные адреса в физические по страницам. Поэтому страница — **наименьшая единица защиты на аппаратном уровне**. (Различные параметры защиты страниц описываются в разделе «Защита памяти» далее.) Страницы бывают двух размеров: малого и большого. Реальный размер зависит от аппаратной платформы (см. Таблица 1).

Таблица 1. Размер страницы

Архитектура	Размер малой страницы, кб	Размер большой страницы, Мб
x32	4	4 (в PAE системах 2)
x64	4	2
IA64	8	16

Преимущество больших страниц — скорость трансляции адресов для ссылок на другие данные в большой странице. Дело в том, что первая ссылка на любой байт внутри большой страницы заставляет аппаратный **ассоциативный буфер трансляции** (translation look-aside buffer, TLB) (см. раздел «Ассоциативный буфер трансляции» далее) загружать в свой кэш информацию, необходимую для трансляции ссылок на любые другие байты в этой большой странице. При использовании малых страниц для того же диапазона виртуальных адресов требуется больше элементов TLB, что заставляет чаще обновлять элементы по мере трансляции новых виртуальных адресов. А это в свою очередь требует чаще обращаться к структурам таблиц страниц при ссылках на виртуальные адреса, выходящие за пределы данной малой страницы. TLB — очень маленький кэш, и поэтому большие страницы обеспечивают более эффективное использование этого ограниченного ресурса.

Чтобы задействовать **преимущества больших страниц** в системах с достаточным объемом памяти, Windows проецирует на такие страницы базовые образы ОС (Ntoskrnl.exe и Hal.dll) и базовые системные данные. Windows также автоматически проецирует на большие страницы запросы объемного ввода-вывода (драйверы устройств вызывают `MmMapIoSpace`), если запрос удовлетворяет длине и выравниванию для большой страницы. Наконец, Windows разрешает приложениям проецировать на такие страницы свои образы, закрытые области памяти и разделы, поддерживаемые страничным файлом (pagefile-backed sections). (См. описание флага `MEM_LARGEPAGE` функции `VirtualAlloc`.) Можете указать, чтобы и другие драйверы устройств проецировались на большие страницы, добавив многострочный параметр реестра `HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\MemoryManagement\LargePageDrivers` и задав имена драйверов как отдельные строки с нулем в конце.

Один из побочных эффектов применения больших страниц заключается в следующем. Так как аппаратная защита памяти оперирует страницами как наименьшей единицей, то, если на большой странице содержатся код только для чтения и данные для записи/чтения, она должна быть помечена как доступная для чтения и записи, т.е. код станет открытым для записи. А значит, **драйверы устройств или другой код режима ядра мог бы в результате скрытой ошибки модифицировать код ОС или драйверов, изначально предполагавшийся только для чтения, и не вызвать нарушения доступа к памяти**. Однако при использовании малых страниц для проецирования ядра части NTOSKRNL.EXE и HAL.DLL только для чтения будут спроецированы именно как страницы только для чтения. Хотя это снижает эффективность трансляции адресов, зато при попытке драйвера устройства (или другого кода режима ядра) модифицировать доступную только для чтения часть ОС произойдет немедленный крах с указанием на неверную инструкцию.

ПРИМЕЧАНИЕ. Если подозреваете, что источник ваших проблем связан с повреждением кода ядра, включите Driver Verifier — это автоматически отключит использование больших страниц.

Резервирование и передача страниц

Страницы в адресном пространстве процесса могут быть **свободными** (free), **зарезервированными** (reserved) или **переданными** (committed). Приложения могут **резервировать** (reserve) адресное пространство и **передавать память** (commit) зарезервированным страницам по мере необходимости. Резервировать страницы и передавать им память можно одним вызовом. Эти сервисы предоставляются через Windows-функции `VirtualAlloc` и `VirtualAllocEx`.

Резервирование адресного пространства позволяет потоку резервировать диапазон виртуальных адресов для последующего использования. Попытка доступа к зарезервированной памяти влечет за собой нарушение доступа, так как ее страницы не спроецированы на физическую память.

При попытке доступа, адреса переданных страниц, в конечном счете, транслируются в допустимые адреса страниц физической памяти. Переданные страницы могут быть закрытыми (не предназначенными для разделения с другими процессами) или спроецированными на представление объекта-раздела (на которое в свою очередь могут проецировать страницы другие процессы).

Закрытые страницы процесса, к которым еще не было обращения, создаются при первой попытке доступа как обнуленные. Закрытые переданные страницы могут впоследствии записываться ОС в страничный файл (в зависимости от текущей ситуации). Такие страницы недоступны другим процессам, если только они не используют функции `ReadProcessMemory` или `WriteProcessMemory`. Если переданные страницы спроецированы на часть проецируемого файла, их скорее всего придется загрузить с диска при условии, что они не были считаны раньше из-за обращения к ним того же или другого процесса, на который спроецирован этот файл.

Страницы записываются на диск по обычной процедуре записи модифицированных страниц, которые перемещаются из рабочего набора процесса в список модифицированных страниц и в конечном счете на диск (о рабочих наборах и списке модифицированных страниц — чуть позже). Страницы проецируемого файла можно сбросить на диск явным вызовом функции `FlushViewOfFile`.

Для **возврата страниц** (decommitting) и/или **освобождения** виртуальной памяти предназначена функция `VirtualFree` или `VirtualFreeEx`. Различия между возвратом и освобождением страниц такие же, как между резервированием и передачей: возвращенная память все еще зарезервирована, тогда как освобожденная память действительно свободна и не является ни переданной, ни зарезервированной.

Такой двухэтапный процесс (резервирование и передача) помогает снизить нагрузку на память, откладывая передачу страниц до реальной необходимости в них. **Резервирование памяти** — операция относительно быстрая и не требующая большого количества ресурсов, поскольку в данном случае не расходуется ни физическая память (драгоценный системный ресурс), ни квота процесса на ресурсы страничного файла (число страниц, передаваемых процессу из страничного файла). При этом нужно создать или обновить лишь сравнительно небольшие внутренние структуры данных, отражающие состояние адресного пространства процесса.

Резервирование памяти с последующей ее передачей особенно эффективно для приложений, нуждающихся в потенциально большой и непрерывной области виртуальной памяти: зарезервировав требуемое адресное пространство, они могут передавать ему страницы порциями, по мере необходимости. Эта методика применяется и для организации стека пользовательского режима для каждого потока. Такой стек резервируется при создании потока. По умолчанию стеку передается только начальная страница, а следующая страница просто помечается как **сторожевая** (guard page).

Блокировка памяти

В целом, принятие решений о том, какие страницы следует оставить в физической памяти, лучше сохранить за диспетчером памяти. Однако в особых обстоятельствах можно подкорректировать работу диспетчера памяти. Существует два способа блокировки страниц в памяти.

Windows-приложения могут блокировать страницы в рабочем наборе своего процесса через функцию `VirtualLock`. Максимальное число страниц, которые процесс может блокировать, равно минимальному размеру его рабочего набора за вычетом восьми страниц. Следовательно, если процессу нужно блокировать большее число страниц, он может увеличить минимальный размер своего рабочего набора вызовом функции `SetProcessWorkingSetSize`.

Драйверы устройств могут вызывать функции режима ядра `MmProbeAndLockPages`, `MmLockPagableCodeSection` и `MmLockPagableSectionByHandle`. Блокированные страницы остаются в памяти до снятия блокировки. Хотя число блокируемых страниц не ограничивается, драйвер не может блокировать их больше, чем это позволяет счетчик доступных резидентных страниц.

Гранулярность выделения памяти

Windows **выравнивает** начало каждого региона зарезервированного адресного пространства в соответствии с **гранулярностью выделения памяти** (allocation granularity). Это значение можно получить через Windows-функцию `GetSystemInfo`. В

настоящее время оно равно 64 Кб. Такая величина выбрана из соображений поддержки будущих CPU с большим размером страниц памяти (до 64 Кб) или виртуально индексируемых кэшей (virtually indexed caches), требующих общесистемного выравнивания между физическими и виртуальными страницами (physical-to-virtual page alignment). Благодаря этому уменьшается риск возможных изменений, которые придется вносить в приложения, полагающиеся на определенную гранулярность выделения памяти. (Это ограничение не относится к коду Windows режима ядра — используемая им гранулярность выделения памяти равна одной странице.)

Windows также добивается, чтобы размер и базовый адрес зарезервированного региона адресного пространства всегда был кратен размеру страницы. Например, системы типа x86 используют страницы размером 4 Кб, и, если вы попытаетесь зарезервировать 18 Кб памяти, на самом деле будет зарезервировано 20 Кб. А если вы укажете базовый адрес 3 Кб для 18-килобайтного региона, то на самом деле будет зарезервировано 24 Кб.

Разделяемая память и проецируемые файлы

Разделяемой (shared memory) называется память, видимая более чем одному процессу или присутствующая в виртуальном адресном пространстве более чем одного процесса. Например, если два процесса используют одну и ту же DLL, есть смысл загрузить ее код в физическую память лишь один раз и сделать ее доступной всем процессам, проецирующим эту DLL (Рис. 1).

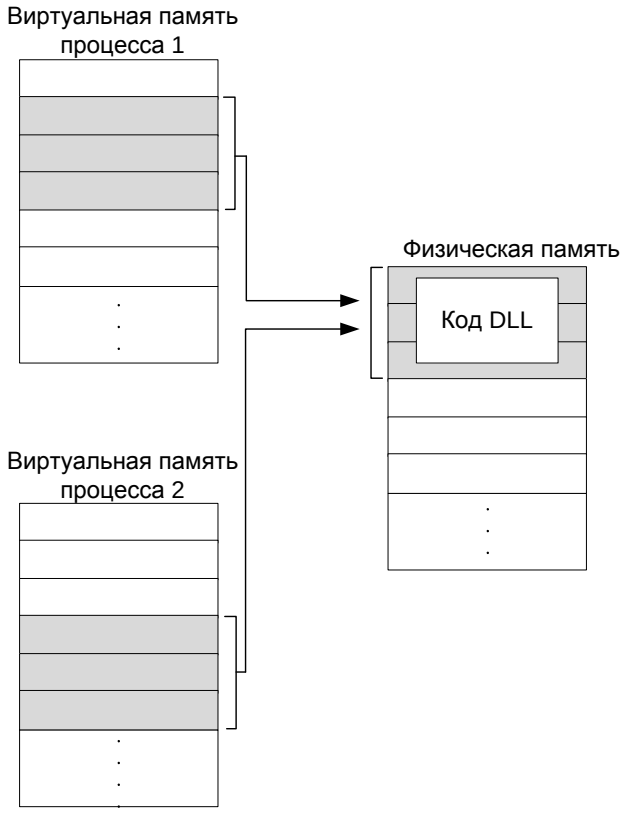

Рис. 1. Разделение памяти процессами

Каждый процесс поддерживает **закрытые области памяти** для хранения собственных данных, но программные инструкции и страницы немодифицируемых данных в принципе можно использовать совместно с другими процессами. Такой вид разделения реализуется автоматически, поскольку страницы кода в исполняемых образах проецируются с атрибутом «только для выполнения», а страницы, доступные для записи, — с атрибутом «копирование при записи» (copy-on-write).

Для реализации разделяемой памяти используются примитивы диспетчера памяти, объекты «раздел», которые в Windows XP называются **объектами «проекция файла»** (file mapping objects).

Этот фундаментальный примитив диспетчера памяти применяется для проецирования виртуальных адресов в основной памяти, страничном файле или любых других файлах, к которым приложение хочет обращаться так, будто они находятся в памяти. Раздел

может быть открыт как одним процессом, так и несколькими; иначе говоря, объекты «раздел» вовсе не обязательно представляют разделяемую память.

Объект «раздел» может быть связан с открытым файлом на диске (который в этом случае называется **проецируемым**) или с переданной памятью (для ее разделения). Разделы, проецируемые на переданную память, называются **разделами**, **поддерживаемыми страничными файлами** (page file backed sections), так как при нехватке памяти их страницы перемещаются в страничный файл. (Однако Windows может работать без страничного файла, и тогда эти разделы «поддерживаются» физической памятью.) Разделяемые переданные страницы, как и любые другие страницы, видимые в пользовательском режиме, всегда обнуляются при первом обращении к ним.

Для создания объекта «раздел» используется Windows-функция `CreateFileMapping`, которой передается описатель проецируемого файла (или `INVALID_HANDLE_VALUE` в случае раздела, поддерживаемого страничным файлом), а также необязательные имя и дескриптор защиты. Если разделу присвоено имя, его может открыть другой процесс вызовом `OpenFileMapping`. Кроме того, можно предоставить доступ к объектам «раздел» через наследование описателей (определив при открытии или создании описателя, что он является наследуемым) или их дублирование (с помощью `DuplicateHandle`). Драйверы также могут манипулировать объектами «раздел» через функции `ZwOpenSection`, `ZwMapViewOfSection` и `ZwUnmapViewOfSection`.

Объект «раздел» может ссылаться на файлы, длина которых намного превышает размер адресного пространства процесса. (Если раздел поддерживается страничным файлом, в нем должно быть достаточно места для размещения всего раздела.) Используя очень большой объект «раздел», процесс может проецировать лишь необходимую ему часть этого объекта, которая называется **представлением** (view) и создается вызовом функции `MapViewOfFile` с указанием проецируемого диапазона. Это позволяет процессам экономить адресное пространство, так как на память проецируется только представление объекта «раздел».

Windows-приложения могут использовать проецирование файлов для упрощения ввода-вывода в файлы на диске, просто делая их доступными в своем адресном пространстве.

Защита памяти

Windows обеспечивает защиту памяти, предотвращая случайную или преднамеренную порчу пользовательскими процессами данных в адресном пространстве системы или других процессов. В Windows предусмотрено **четыре основных способа защиты памяти**.

Доступ ко всем общесистемным структурам данных и пулам памяти, используемым системными компонентами режима ядра, возможен лишь из режима ядра — у потоков пользовательского режима нет доступа к соответствующим страницам. Когда поток пользовательского режима пытается обратиться к одной из таких страниц, CPU генерирует исключение, и диспетчер памяти сообщает потоку о нарушении доступа.

У каждого процесса имеется индивидуальное закрытое адресное пространство, защищенное от доступа потоков других процессов, исключение составляют те случаи, когда процесс разделяет какие-либо страницы с другими процессами или когда у другого процесса есть права на доступ к объекту «процесс» для чтения и/или записи, что позволяет ему использовать функции `ReadProcessMemory` и `WriteProcessMemory`. Как только поток ссылается на какой-нибудь адрес, аппаратные средства поддержки виртуальной памяти совместно с диспетчером памяти перехватывают это обращение и транслируют виртуальный адрес в физический. Контролируя трансляцию виртуальных адресов, Windows гарантирует, что потоки одного процесса не получат несанкционированного доступа к страницам другого процесса.

Кроме косвенной защиты, обеспечиваемой трансляцией виртуальных адресов в физические, все CPU, поддерживаемые Windows, предоставляют ту или иную форму аппаратной защиты памяти (например, доступ для чтения и записи, только для чтения и т.д.); конкретные механизмы такой защиты зависят от архитектуры CPU. Скажем, страницы кода в адресном пространстве процесса помечаются атрибутом «только для чтения», что защищает их от изменения пользовательскими потоками.

Атрибуты защиты памяти, определенные в Windows API, перечислены в Таблица 2 (см. также документацию на функции `VirtualProtect`, `VirtualProtectEx`, `VirtualQuery` и `VirtualQueryEx`).

Таблица 2. Атрибуты защиты памяти, определенные в Windows API

Атрибут	Описание
PAGE_NOACCESS	Любая попытка чтения, записи или выполнения кода на этой странице вызывает нарушение доступа
PAGE_READONLY	Любая попытка записи (или выполнения кода на CPU без поддержки запрета на выполнение) на этой странице вызывает нарушение доступа, но чтение разрешено
PAGE_READWRITE	Страница доступна для чтения и записи никакие действия не вызывают нарушения доступа
PAGE_EXECUTE	Любая попытка записи на этой странице вызывает нарушение доступа, но выполнение кода (и чтения на всех существующих CPU разрешено
PAGE_EXECUTEREAD	Любая попытка записи на этой странице вызывает нарушение доступа, но чтение и выполнение разрешено
PAGE_EXECUTE_READWRITE	Страница доступна для чтения, записи и выполнения — никакие действия не вызывают нарушения доступа
PAGE_WRITECOPY	Любая попытка записи на этой странице заставляет систему выдать процессу закрытую копию данной страницы; при попытке выполнения кода на этой странице возникает нарушение доступа
PAGE_EXECUTE_WRITECOPY	Любая попытка записи на этой странице заставляет систему выдать процессу закрытую копию данной страницы. Чтение и выполнение кода на этой странице разрешено (в этом случае копия страницы не создается)
PAGE_GUARD	Любая попытка чтения или записи сторожевой страницы вызывает исключение EXCEPTION_GUARD_PAGE, и она перестает быть сторожевой; этот атрибут можно комбинировать с любым другим атрибутом, кроме PAGE_NOACCESS
PAGE_NOCACHE	Используется некэшируемая физическая память, что, как правило, не рекомендуется делать. Имеет смысл для драйверов устройств, например, для проецирования буфера видеокадра без кэширования
PAGE_WRITECOMBINE	Разрешает комбинированную запись на страницу памяти. В этом случае CPU может кэшировать запросы на запись в память для большей производительности. Например, при наличии нескольких запросов на запись по одному адресу возможна запись лишь по последнему запросу

Поток может менять атрибуты защиты виртуальных страниц раздела (на уровне отдельных страниц), если это не противоречит разрешениям, указанным в ACL для данного раздела. Так, диспетчер памяти позволит потоку сменить атрибут страниц общего раздела «только для чтения» на «копирование при записи», но запретит его изменение на атрибут «для чтения и записи». Копирование при записи разрешается потому, что не влияет на другие процессы, тоже использующие эти данные.

Запрет на выполнение

Предотвращением выполнения данных (data execution prevention, DEP), означает, что попытка передачи управления какой-либо инструкции на странице, помеченной атрибутом «запрет на выполнение», приведет к нарушению доступа к памяти. Благодаря этому блокируются попытки определенных типов вирусов воспользоваться ошибками в ОС, которые иначе позволили бы выполнить код, размещенный на странице данных. Попытка выполнить код на странице, помеченной атрибутом «запрет на выполнение», в режиме ядра вызывает крах системы с кодом `ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY`. Если такая же попытка предпринимается в пользовательском режиме, то генерируется исключение `STATUS_ACCESS_VIOLATION` (`0xc0000005`); оно доставляется потоку, в котором была эта недопустимая ссылка. Если процесс выделяет память, которая должна быть исполняемой, то при вызове функций, отвечающих за выделение памяти, он обязан явно указать для соответствующих страниц флаг `PAGE_EXECUTE`, `PAGE_EXECUTE_READ`, `PAGE_EXECUTEREADWRITE` или `PAGE_EXECUTE_WRITECOPY`.

В 64-разрядных версиях Windows атрибут защиты «запрет на выполнение» всегда применяется ко всем 64-разрядным программам и драйверам устройств, и его нельзя отключить. Поддержка такой защиты для 32-разрядных программ зависит от конфигурационных параметров системы. В 64-разрядной Windows защита от выполнения применяется к стекам потоков (как режима ядра, так и пользовательского режима), к страницам пользовательского режима, не помеченным явно как исполняемые, к пулу подкачиваемой памяти ядра и к сеансовому пулу ядра. Однако в 32-разрядной Windows защита от выполнения применяется только к стекам потоков и страницам пользовательского режима.

Копирование при записи

Защита страницы типа «копирование при записи» — механизм оптимизации, используемый диспетчером памяти для экономии физической памяти. Когда процесс проецирует копируемое при записи представление объекта «раздел» со страницами, доступными для чтения и записи, диспетчер памяти — вместо того чтобы создавать закрытую копию этих страниц в момент проецирования представления — откладывает создание копии до тех пор, пока не закончится запись в них. Эта методика используется и всеми современными UNIX-системами. На Рис. 2 показана ситуация, когда два процесса совместно используют три страницы, каждая из которых помечена как копируемая при записи, но ни один из процессов еще не пытался их модифицировать.

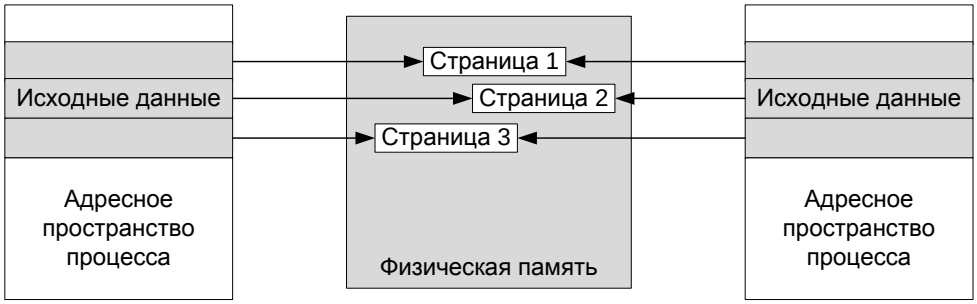

Рис. 2. Состояние страниц до копирования при записи

Если поток любого из этих процессов что-то записывает на такую страницу, генерируется исключение, связанное с управлением памятью. Обнаружив, что запись ведется на страницу с атрибутом «копирование при записи», диспетчер памяти, вместо того чтобы сообщить о нарушении доступа, выделяет в физической памяти новую страницу, доступную для чтения и записи, копирует в нее содержимое исходной страницы, обновляет соответствующую информацию о страницах, проецируемых на данный процесс, и закрывает исключение. В результате команда, вызвавшая исключение, выполняется повторно, и операция записи проходит успешно. Но, как показано на Рис. 3, новая страница теперь является личной собственностью процесса, инициировавшего запись, и не видима другим процессам, совместно использующим страницу с атрибутом «копирование при записи». Каждый процесс, что-либо записывающий на эту разделяемую страницу, получает в свое распоряжение ее закрытую копию.

Одно из применений копирования при записи — поддержка точек прерываний для отладчиков. Например, по умолчанию страницы кода доступны только для выполнения. Если программист при отладке программы устанавливает точку прерывания, отладчик должен добавить в код программы соответствующую команду. Для этого он сначала меняет атрибут защиты страницы на `PAGE_EXECUTE_READWRITE`, а затем модифицирует поток команд. Поскольку страница кода является частью проецируемого раздела, диспетчер памяти создает закрытую копию для процесса с установленной точкой прерывания, тогда как другие процессы по-прежнему используют исходную страницу кода.

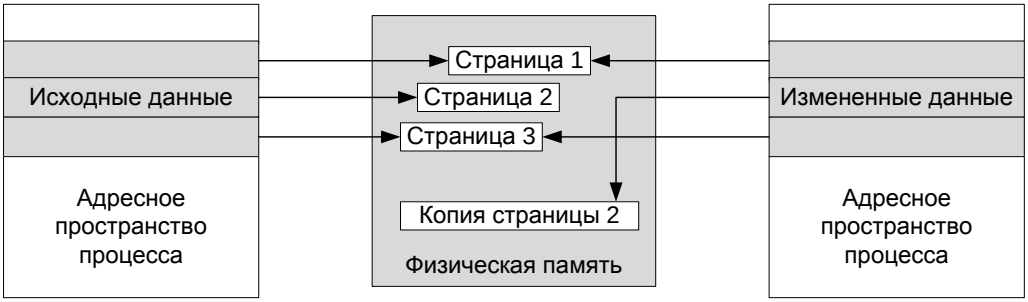

Рис. 3. Состояние страниц после копирования

Копирование при записи может служить примером **алгоритма отложенной оценки** (lazy evaluation), который диспетчер памяти применяет при любой возможности. В таких алгоритмах операции, чреватые большими издержками, не выполняются до тех пор, пока не станут абсолютно необходимыми, — если операция так и не понадобится, никаких издержек вообще не будет.

Подсистема POSIX использует преимущества копирования при записи в реализации функции `fork`. Как правило, если UNIX-приложения вызывают `fork` для создания другого процесса, то первое, что делает новый процесс, — обращается к функции `exec` для повторной инициализации адресного пространства исполняемой программы. Вместо копирования всего адресного пространства при вызове `fork` новый процесс использует страницы родительского процесса, помечая их как копируемые при записи. Если дочерний процесс что-то записывает на эти страницы, он получает их закрытую копию. В ином случае оба процесса продолжают разделять страницы без копирования. Так или иначе диспетчер памяти копирует лишь те страницы, на которые процесс пытается что-то записать, а не все содержимое адресного пространства.

Оценить частоту срабатывания механизма копирования при записи можно с помощью счетчика Memory: Write Copies/Sec (Память: Запись копий страниц/сек).

Многие приложения выделяют память небольшими блоками (менее 64 Кб — минимума, поддерживаемого функциями типа `VirtualAlloc`). Выделение столь большой области (64 Кб) для сравнительно малого блока весьма неоптимально с точки зрения использования памяти и производительности. Для устранения этой проблемы в Windows имеется компонент — **диспетчер куч** (heap manager), который управляет распределением памяти внутри больших областей, зарезервированных с помощью функций, выделяющих память в соответствии с гранулярностью страниц. Гранулярность выделения памяти в диспетчере куч сравнительно мала: 8 байтов в 32-разрядных системах и 16 байтов в 64-разрядных. **Диспетчер куч обеспечивает** оптимальное использование памяти и производительность при выделении таких небольших блоков памяти. Функции диспетчера куч локализованы в двух местах: в `Ntdll.dll` и `Ntoskrnl.exe`. API-функции подсистем (вроде API-функций Windows-куч) вызывают функции из `Ntdll`, а компоненты исполнительной системы и драйверы устройств — из `Ntnskml`. Родные интерфейсы доступны только внутренним компонентам Windows и драйверам устройств режима ядра Документированный интерфейс Windows API для куч (функции с префиксом `Heap`) представляют собой тонкие оболочки, которые вызывают родные функции из Ntdll.dll. Кроме того, для поддержки устаревших Windows-приложений предназначены унаследованные API-функции (с префиксом `Local` или `Global`). К наиболее часто используемым Windows-функциям куч относятся:

`HeapCreate` или `HeapDestroy` — соответственно создает или удаляет кучу. При создании кучи можно указать начальные размеры зарезервированной и переданной памяти;

`HeapAlloc` — выделяет блок памяти из кучи;

`HeapFree` — освобождает блок, ранее выделенный через `HeapAlloc`;

`HeapReAlloc` — увеличивает или уменьшает размер уже выделенного блока;

`HeapLock` и `HeapUnlock` — управляют взаимным исключением (mutual exclusion) операций, связанных с обращением к куче;

`HeapWalk` — перечисляет записи и области в куче.

Address Windowing Extensions

Чтобы 32-разрядный процесс мог получить доступ к большему объему физической памяти, Windows поддерживает набор функций под общим названием **Address Windowing Extensions** (AWE). Так, в системе под управлением Windows 2000 Advanced Server с 8 Гб физической памяти серверное приложение базы данных может с помощью AWE использовать под кэш базы данных до 6 Гб памяти.

Выделение и использование памяти через функции AWE осуществляется в три этапа.

1. Выделение физической памяти.

2. Создание региона виртуального адресного пространства — окна, на которое будут проецироваться представления физической памяти.

3. Проецирование на окно представлений физической памяти.

Для выделения физической памяти приложение вызывает Windows-функцию `AllocateUserPhysicalPages`. (Эта функция требует, чтобы у пользователя была привилегия `Lock Pages In Memory`.). Затем приложение обращается к Windows-функции `VirtualAlloc` с флагом `MEM_PHYSICAL`, чтобы создать окно в закрытой части адресного пространства процесса, на которое проецируется (частично или полностью) ранее выделенная физическая память. Память, выделенная через AWE, может быть использована почти всеми функциями Windows API (например, функции Microsoft DirectX ее не поддерживают).

Если приложение создает в своем адресном пространстве окно размером 256 Мб и выделяет 4 Гб физической памяти (в системе с объемом физической памяти более 4 Гб), то оно получает доступ к любой части физической памяти, проецируя ее на это окно через Windows-функции `MapUserPhysicalPages` или `MapUserPhysicalPagesScatter`. Размер физической памяти, единовременно доступный приложению при такой схеме выделения, определяется размером окна в виртуальном адресном пространстве. На Рис. 4 показано AWE-окно в адресном пространстве серверного приложения, на которое проецируется регион физической памяти, предварительно выделенный через `AllocateUserPhysiculPuges`.

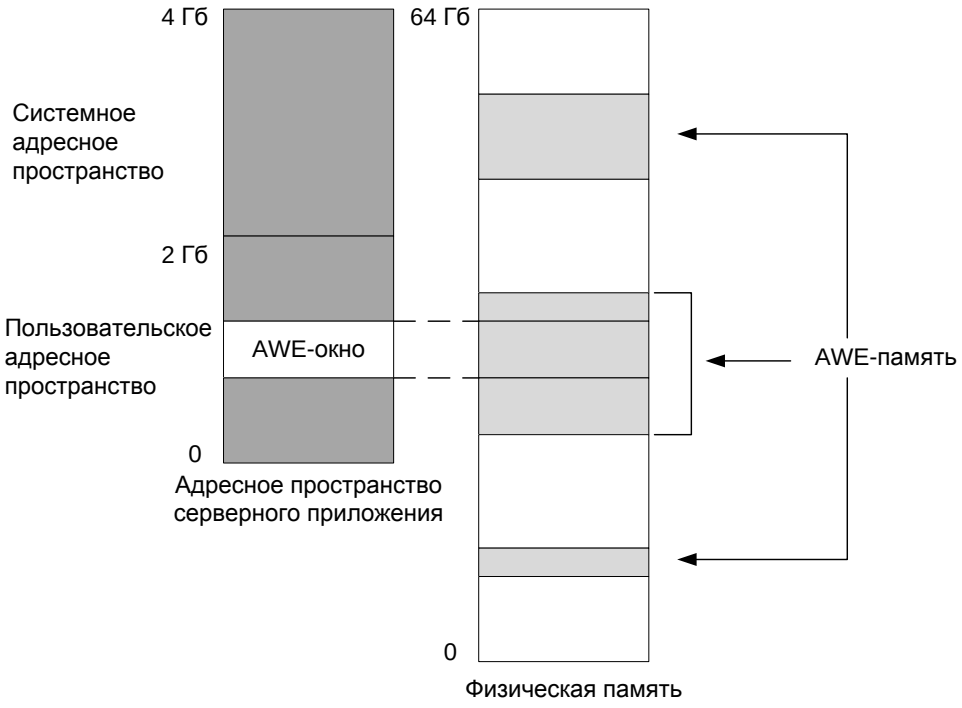

Рис. 4. Использование AWE для проецирования физической памяти

AWE-функции имеются во всех выпусках Windows и доступны независимо от объема физической памяти в системе. Однако AWE наиболее полезен в системах с объемом физической памяти не менее 2 Гб, поскольку тогда этот механизм — единственное средство для прямого использования более чем 2 Гб памяти 32-разрядным процессом. Еще одно его применение — защита. Так как AWE-память никогда не выгружается на диск, данные в этой памяти никогда не имеют копии в страничном файле, а значит, никто не сумеет просмотреть их, загрузив компьютер с помощью альтернативной ОС.

Теперь несколько слов об ограничениях, налагаемых на память, которая выделяется и проецируется с помощью AWE-функций.

Страницы такой памяти нельзя разделять между процессами.

Одну и ту же физическую страницу нельзя проецировать по более чем одному виртуальному адресу в рамках одного процесса.

В более старых версиях Windows страницы такой памяти могут иметь единственный атрибут защиты — «для чтения и записи». В Windows Server 2003 Service Pack 1 и выше, также поддерживаются атрибуты «нет доступа» и «только для чтения».

Системные пулы памяти

При инициализации системы диспетчер памяти создает два типа динамических пулов памяти, используемых компонентами режима ядра для выделения системной памяти.

Пул неподкачиваемой памяти (nonpaged pool). Состоит из диапазонов системных виртуальных адресов, которые всегда присутствуют в физической памяти и доступны в любой момент (при любом IRQL и из контекста любого процесса) без генерации ошибок страниц. Одна из причин существования такого пула — невозможность обработки ошибок страниц при IRQL уровня «DPC/dispatch» и выше.

Пул подкачиваемой памяти (paged pool). Регион виртуальной памяти в системном пространстве, содержимое которого система может выгружать в страничный файл и загружать из него. Драйверы, не требующие доступа к памяти при IRQL уровня «DPC/dispatch» и выше, могут использовать память из этого пула. Он доступен из контекста любого процесса. Оба пула находятся в системном адресном пространстве и проецируются на виртуальное адресное пространство любого процесса. Исполнительная система предоставляет функции для выделения и освобождения памяти в этих пулах (см. описание функций, чьи имена начинаются с `ExAllocatePool`, в Windows DDK).

В single-CPU системах создается три пула подкачиваемой памяти, а в multi-CPU — пять. Наличие нескольких подкачиваемых пулов уменьшает частоту блокировки системного кода при одновременных обращениях нескольких потоков к процедурам управления пулами. Начальный размер подкачиваемого и неподкачиваемого пулов зависит от объема физической памяти и может при необходимости расти до максимального значения, вычисляемого в период загрузки системы.

ПРИМЕЧАНИЕ. Будущие выпуски Windows, возможно, будут поддерживать пулы динамических размеров, а значит, лимита на максимальный размер больше не будет.

Настройка размеров пулов

`NonPagedPoolSize` и `PagedPoolSize` в разделе реестра `HKLM\SYSTEM\CurrentControlSet`. Чтобы установить другие начальные размеры этих пулов, измените значения параметров `\Control\SessionManager\MemoryManagement` с 0 (при этом система сама вычисляет размеры) на нужные величины (в байтах). Нельзя превысить предельные значения, перечисленные в Таблица 3. Значение `0xFFFFFFFF` для `PagedPoolSize` указывает, что выбран наибольший из возможных размеров, однако увеличение пула подкачиваемой памяти будет происходить за счет записей системной таблицы страниц (page table entries, PTF).

Таблица 3. Максимальные размеры пулов

Тип пула	Максимальный размер в 32-х разрядных системах	Максимальный размер в 64-х разрядных системах
Неподкачиваемый	256 Мб (128 Мб, если был задан загрузочный параметр /3GB)	128 Гб

Подкачиваемый	491,875 Мб (в Windows 7000 и Windows XP); 650 Мб (Windows Server 2003)	128 Гб

Мониторинг использования пулов

Объект Memory (память) предоставляет отдельные счетчики размеров пулов неподкачиваемой и подкачиваемой памяти (как для виртуальной, так и для физической частей). Кроме того, утилита Poolmon из Windows Support Tools сообщает детальную информацию об использовании этих пулов. Для просмотра такой информации нужно включить внутренний параметр `EnablePoolTagging` (который всегда включен в проверочных версиях, а также в Windows Server 2003, где его вообще нельзя выключить). Чтобы включить данный параметр, запустите утилиту `Gflags` из Windows Support Tools, Platform SDK или DDK и выберите переключатель `EnablePoolTagging`.

Ассоциативные списки

Windows поддерживает **механизм быстрого выделения памяти — ассоциативные списки** (look-aside lists). Главное различие между пулом и ассоциативным списком в том, что из пула можно выделять блоки памяти различного размера, а из ассоциативного списка — только фиксированные. Хотя пулы обеспечивают более высокую гибкость, ассоциативные списки работают быстрее, так как не используют спин-блокировку и не заставляют систему подбирать подходящую область свободной памяти, в которой мог бы уместиться текущий выделяемый блок.

Функции `ExInitializeNPagedLookasideList` и `ExInitializePagedLookasideList` (документированные в DDK) позволяют компонентам исполнительной системы и драйверам устройств создавать ассоциативные списки, размеры которых кратны размерам наиболее часто используемых структур данных. Для минимизации издержек, связанных с синхронизацией в multi-CPU системах, некоторые компоненты исполнительной системы, в том числе диспетчер ввода-вывода, диспетчер кэша и диспетчер объектов, создают отдельные для каждого CPU ассоциативные списки, из которых выделяется память под часто используемые структуры данных. Сама исполнительная система создает для каждого CPU универсальные ассоциативные списки подкачиваемой и неподкачиваемой памяти с гранулярностью выделения в 256 байтов или менее.

Структуры виртуального адресного пространства

Здесь описываются компоненты в пользовательском и системном адресных пространствах, а также специфика адресных пространств в 32- и 64-разряд-ных системах. Эта информация поможет вам понять ограничения на виртуальную память для процессов и системы на обеих платформах.

На виртуальное адресное пространство в Windows проецируются три основных вида данных:

- код и данные, принадлежащие процессу,
- код и данные, принадлежащие сеансу,
- общесистемные код и данные.

Каждому процессу выделяется собственное адресное пространство, недоступное другим процессам (если только у них нет разрешения на открытие процесса с правами доступа для чтения и записи). Потоки внутри процесса никогда не получают доступа к виртуальным адресам вне адресного пространства своего процесса, если только не проецируют данные на раздел общей памяти и/или не используют специальные функции, позволяющие обращаться к адресному пространству другого процесса. Сведения о виртуальном адресном пространстве процесса хранятся в **таблицах страниц** (page tables). Таблицы страниц размещаются на страницах памяти, доступных только в режиме ядра, поэтому пользовательские потоки в процессе не могут модифицировать структуру адресного пространства своего процесса.

В системах с поддержкой нескольких сеансов (Windows 2000 Server с установленной службой Terminal Services, Windows XP и Windows Server 2003) пространство сеанса содержит информацию, глобальную для каждого сеанса. **Сеанс** (session) состоит из процессов и других системных объектов (вроде WindowStation, рабочих столов и окон). Эти объекты представляют сеанс единственного пользователя, который

зарегистрировался на рабочей станции. У каждого сеанса есть своя область пула подкачиваемой памяти, используемая подсистемой Windows (`Win32k.sys`) для выделения памяти под сеансовые GUI-структуры данных. Кроме того, каждый сеанс получает свою копию процесса подсистемы Windows (`Csrss.exe`) и `Winlogon.exe`. За создание новых сеансов отвечает процесс диспетчера сеансов (`Smss.exe`). Его задачи включают загрузку сеансовых копий `Win32k.sys` и создание специфических для сеанса экземпляров процессов Csrss и Winlogon, а также пространства имен диспетчера объектов.

Для виртуализации сеансов все общие для сеанса структуры данных проецируются на область системного пространства, которая называется **пространством сеанса** (session space). При создании процесса этот диапазон адресов проецируется на страницы, принадлежащие тому сеансу, к которому относится данный процесс. Размер области для проецируемых представлений в пространстве сеанса можно настраивать, используя параметры в разделе реестра `HKLM\System\CurrentControlSet\Control\SessionManager\MemoryManagement`. (В 32-разрядных системах эти параметры игнорируются при загрузке системы с параметром /3GB.)

Наконец, системное пространство содержит глобальные код и структуры данных ОС, видимые каждому процессу. Системное пространство состоит из следующих компонентов:

Системный код. Содержит образ ОС, HAL и драйверы устройств, используемые для загрузки системы.

Представления, проецируемые системой. Сюда проецируются `Win32k.sys`, загружаемая часть подсистемы Windows режима ядра, а также используемые ею графические драйверы режима ядра.

Гиперпространство. Особая область, применяемая для проецирования списка рабочего набора процесса и временного проецирования других физических страниц для таких операций, как обнуление страницы из списка свободных страниц (если список обнуленных страниц пуст и нужна обнуленная страница), подготовка адресного пространства при создании нового процесса и объявление недействительными PTE в других таблицах страниц (например, при удалении страницы из списка простаивающих страниц).

Список системного рабочего набора. Структуры данных списка рабочего набора, описывающие системный рабочий набор.

Системный кэш. Виртуальное адресное пространство, применяемое для проецирования файлов, открытых в системном кэше.

Пул подкачиваемой памяти. Системная куча подкачиваемой памяти.

Элементы системной таблицы страниц (PTE). Пул системных PTE используемых для проецирования таких системных страниц, как пространство ввода-вывода, стеки ядра и списки дескрипторов памяти. Можно узнать сколько системных PTE доступно, проверив значение счетчика Memory: Free System Page Table Entries (Память: Свободных элементов таблицы страниц) в оснастке Performance (Производительность).

Пул неподкачиваемой памяти. Системная куча неподкачиваемой памяти, обычно состоящая из двух частей, которые располагаются внизу и вверху системного пространства.

Данные аварийного дампа. Область, зарезервированная для записи информации о состоянии системы на момент краха.

Область, используемая HAL. Область, зарезервированная под структуры, специфичные для HAL

ПРИМЕЧАНИЕ. Внутреннее название системного рабочего набора — **рабочий набор системного кэша** (system cache working set). Однако этот термин неудачен, так как в системный рабочий набор входит не только кэш, но и пул подкачиваемой памяти, подкачиваемые системные код и данные, а также подкачиваемые код и данные драйверов.

Структуры пользовательского адресного пространства на платформе x86

По умолчанию каждый пользовательский процесс в 32-разрядной версии Windows располагает собственным адресным пространством размером до Гб; остальные 2 Гб забирает себе ОС. Windows 2000 Advanced Server, Windows 2000 Datacenter Server,

Windows XP Service Pack 2 и выше, а также Windows Server 2003 (все версии) поддерживают загрузочный параметр (ключ /3GB в Boot.ini), позволяющий создавать пользовательские адресные пространства размером по 3 Гб. Windows XP и Windows Server 2003 поддерживают дополнительный ключ (/USERVA), который дает возможность задавать размер пользовательского адресного пространства между 2 и 3 Гб (значение указывается в мегабайтах). Структуры этих двух адресных пространств показаны на Рис. 5.

Поддержка возможности расширения пользовательского адресного пространства для 32-разрядного процесса за пределы 2 Gb введена как временное решение для поддержки приложений вроде серверов баз данных, которым для хранения данных требуется больше памяти, чем возможно в 2 Gb адресном пространстве. Но лучше, конечно, пользоваться уже рассмотренными AWE-функциями.

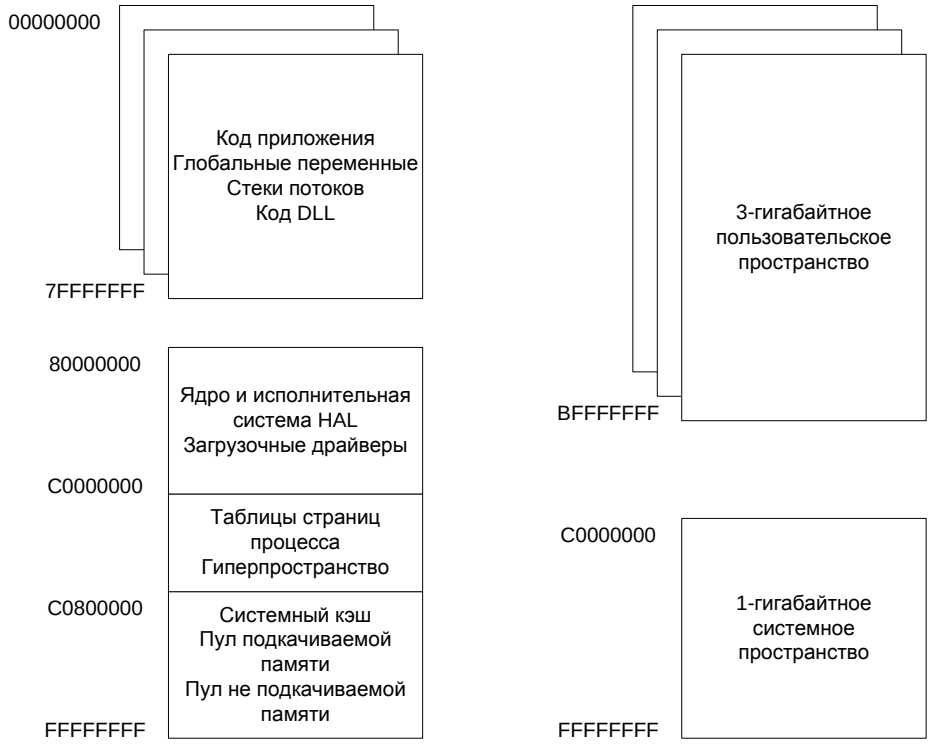

Рис. 5. Структуры виртуального адресного пространства в системах типа x86

Для расширения адресного пространства процесса за пределы 2 Gb в заголовке образа должен быть указан флаг `IMAGE_FILE_LARGE_ADDRESS_AWARE`. Иначе Windows резервирует это дополнительное пространство, и виртуальные адреса выше `0x7FFFFFFF` становятся недоступны приложению. (Так делается, чтобы избежать краха приложения, не способного работать с этими адресами) Этот флаг можно задать ключом компоновщика `/LARGEADDRESSAWARE` при сборке исполняемого файла. Данный флаг не действует при запуске приложения в системе с 2-гигабайтным адресным пространством для пользовательских процессов. (Если вы загрузите любую версию Windows Server с параметром /3GB, размер системного пространства уменьшится до 1 Гб, но пользовательское пространство все равно останется 2 Gb, даже несмотря на поддержку запускаемой программой большого адресного пространства.)

Несколько системных образов помечаются как поддерживающие большие адресные пространства, благодаря чему они могут использовать преимущества систем, работающих с такими пространствами. К их числу относятся:

- Lsass.exe — подсистема локальной аутентификации;
- Inetinfo.exe — Internet Information Services (IIS);
- Chkdsk.exe — утилита Check Disk;
- Dllhst3g.exe — специальная версия Dllhost.exe (для COM+ приложений).

Наконец, поскольку по умолчанию память, выделяемая через `VirtualAlloc`, начинается с младших адресов (если только процесс не выделяет очень много виртуальной памяти или не имеет очень сильно фрагментированного виртуального адресного пространства), она никогда не достигает самых старших адресов. Поэтому при тестировании можно указать, что выделение памяти должно начинаться со старших

адресов. Для этого добавьте в реестр DWORD-параметр `HKLM\System\CurrentControlSet\Control\SessionManager\MemoryManagement\AllocationPreference` и присвойте ему значение `0x100000`.

Структуры системного адресного пространства на платформе x86

Детально описано в [3].

Структуры 64-разрядных адресных пространств

Теоретически 64-рязрядное виртуальное адресное пространство может быть до 16 экзабайтов (18 446 744 073 709 551616 байтов, или примерно 17,2 миллиарда гигабайтов). В отличие от 32-разрядного адресного пространства на платформ x8б, где по умолчанию оно делится на две равные части (половина для процесса и половина для системы), 64-разрядное адресное пространство делится на ряд регионов разного размера, компоненты которого концептуально совпадают с порциями пользовательского, системного и сеансового пространств. Размер этих регионов отражает лимиты текущей реализации, которые могут быть расширены в будущих выпусках.

Детальные структуры адресных пространств IA64 и x64 различаются незначительно. Структуру адресного пространства для IA64 см. на Рис. 6, а для x64 — на Рис. 7.

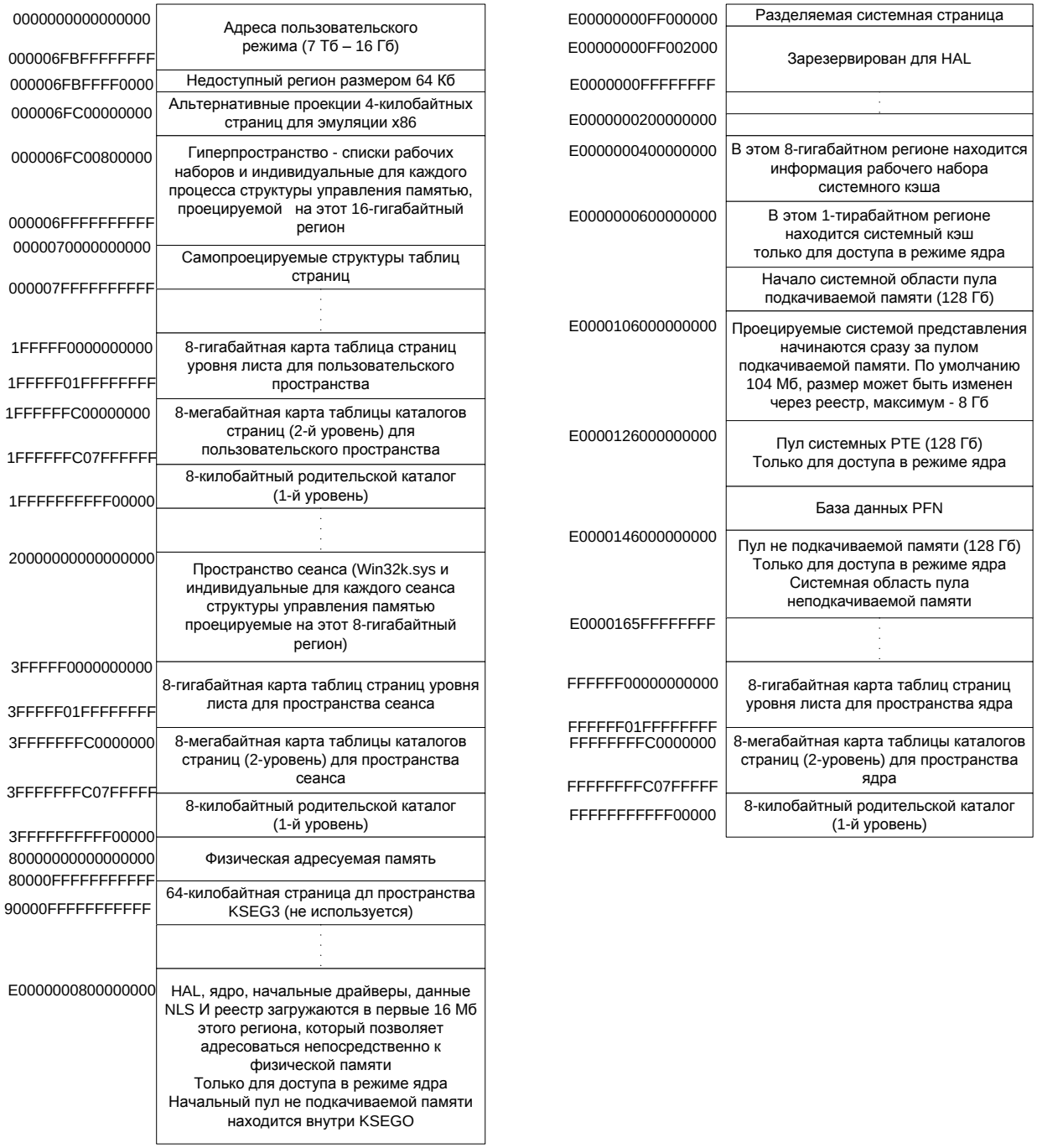

Рис. 6. Структура адресного пространства на платформе IA64

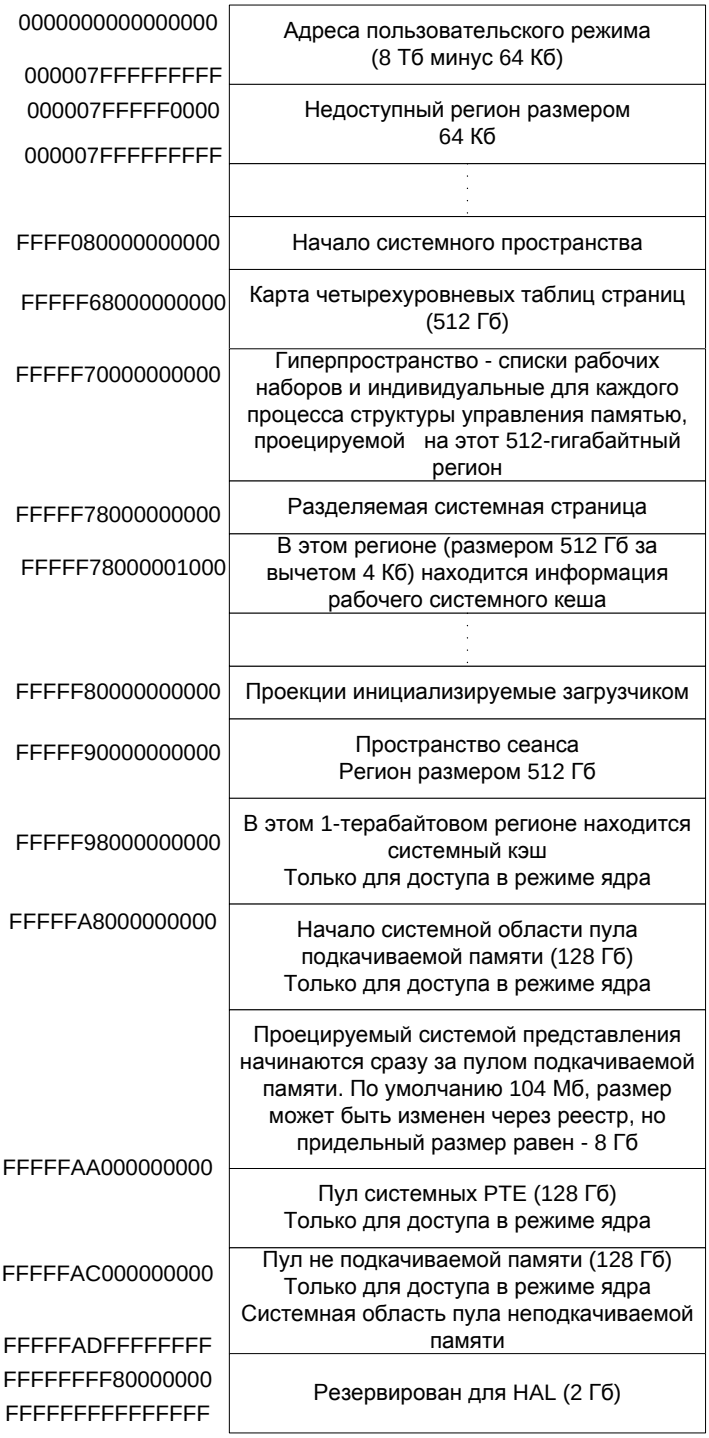

Рис. 7. Структура адресного пространства на платформе x64

Трансляция адресов

Трансляция виртуальных адресов на платформе x86

С помощью структур данных (таблиц страниц), создаваемых и поддерживаемых диспетчером памяти, CPU транслирует виртуальные адреса в физические. Каждый виртуальный адрес сопоставлен со структурой системного пространства, которая называется **элементом таблицы страниц** (page table entry, PTE) и содержит физический адрес, соответствующий виртуальному. Например, на Рис. 8 показаны три последовательно расположенные виртуальные страницы, проецируемые на три разрозненные физические страницы (платформа x86).

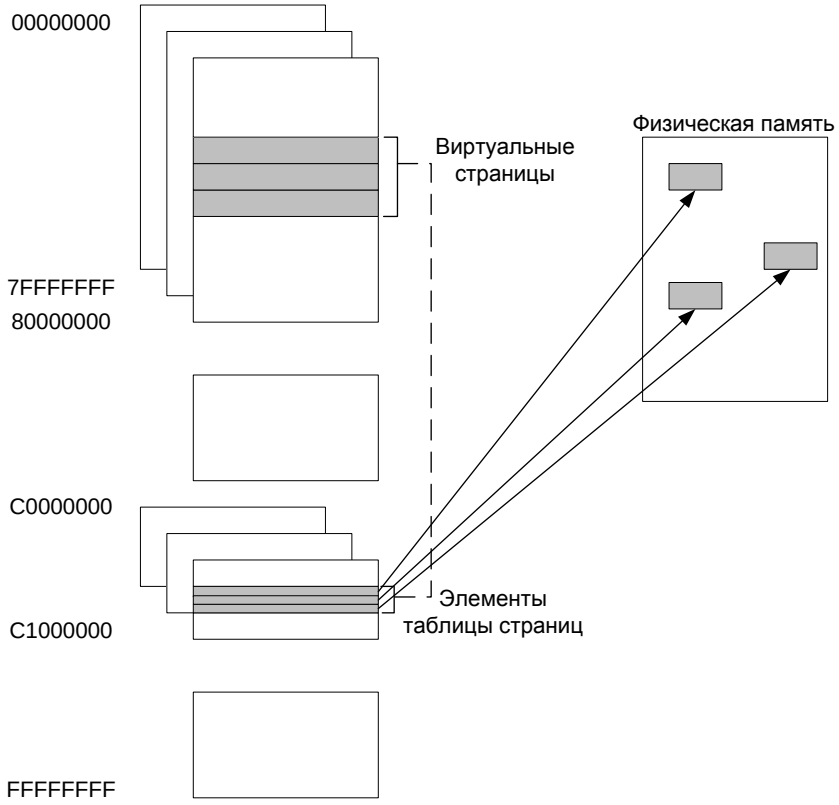

Рис. 8. Трансляция виртуальных адресов в физические, платформа x86

По умолчанию в x86-системе Windows для трансляции виртуальных адресов в физические использует двухуровневую таблицу страниц (x86-системы, работающие с PAE-версией ядра, используют трехуровневую таблицу страниц). 32-разрядный виртуальный адрес интерпретируется как совокупность трех элементов:

- индекса каталога страниц,
- индекса таблицы страниц,
- индекса байта.

Они применяются в качестве указателей в структурах, описывающих проекции страниц (Рис. 9). Размеры страницы и PTE определяет размеры каталога страниц и полей индекса таблицы страниц. Так, в x86-системах длина индекса байта составляет 12 битов, поскольку размер страницы равен 4096 байтов (т.е. 2^{12}).

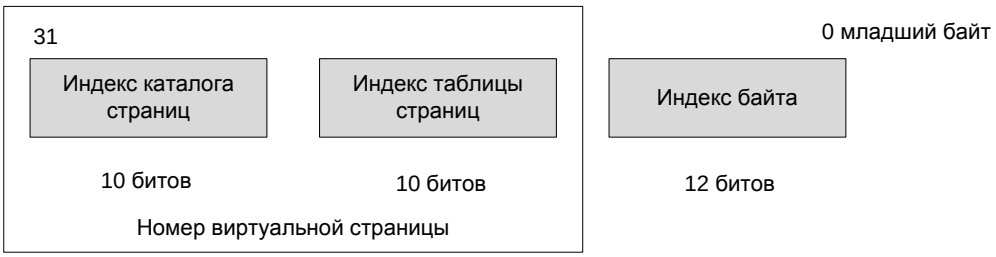

Рис. 9. Элементы 32-разрядного виртуального адреса в x86-системах

Индекс каталога страниц (page directory index) применяется для поиска таблицы страниц, содержащей PTE для данного виртуального адреса. С помощью **индекса таблицы страниц** (page table index) осуществляется поиск PTE, который, как уже говорилось, содержит физический адрес, по которому проецируется виртуальная страница. **Индекс байта** (byte index) позволяет найти конкретный адрес на физической странице. Взаимосвязи этих трех величин и их использование для трансляции виртуальных адресов в физические показаны на Рис. 10.

При трансляции виртуального адреса выполняются следующие операции.

1. **Аппаратные средства управления памятью** находят каталог страниц текущего процесса. При каждом переключении контекста процесса эти средства получают адрес

каталога страниц нового процесса. Обычно ОС записывает этот адрес в специальный регистр CPU.

2. **Индекс каталога страниц** используется как указатель для поиска элемента каталога страниц (page directory entry, PDE), который определяет местонахождение таблицы страниц, нужной для трансляции виртуального адреса. PDE содержит **номер фрейма страницы** (page frame number, PFN) таблицы страниц (если она находится в памяти; однако такие таблицы могут выгружаться в страничный файл).

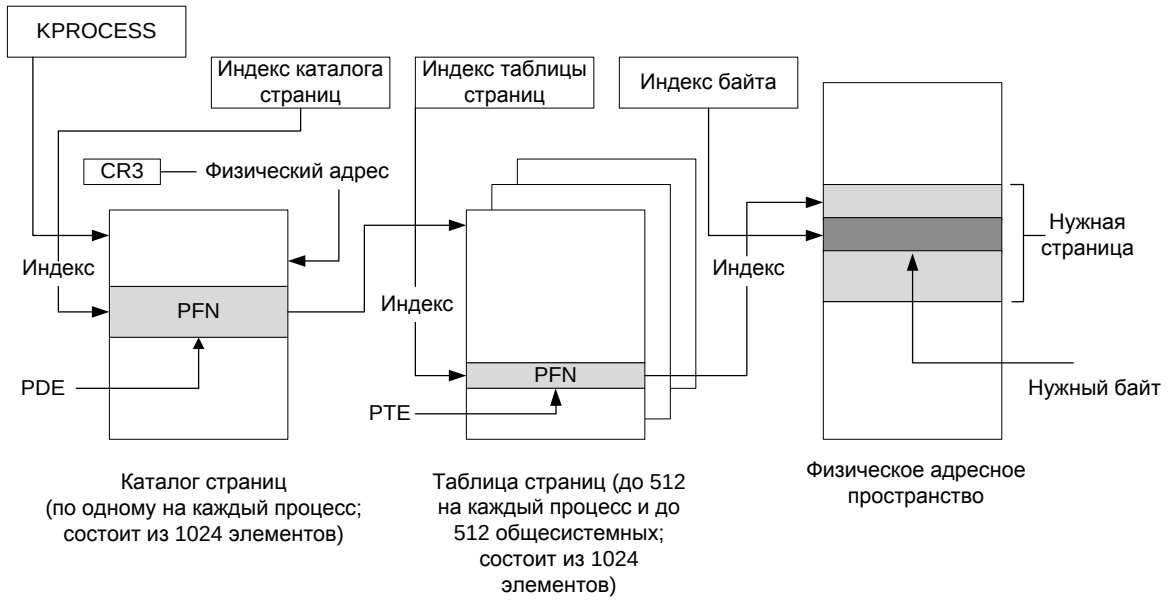

Рис. 10. Трансляция виртуального адреса в x86 системах

3. **Индекс таблицы страниц** используется как указатель для поиска PTE, который определяет местонахождение требуемой виртуальной страницы.

4. На основе PTE отыскивается страница. Если она действительна, то содержит PFN соответствующей страницы физической памяти. Если PTE сообщает, что страница недействительна, обработчик ошибок подсистемы управления памятью пытается найти страницу и сделать ее действительной. Если сделать страницу действительной не удалось (например, из-за ошибки защиты), обработчик ошибок генерирует нарушение доступа или вызывает переход в состояние отладки.

5. Если PTE указывает на действительную страницу, для поиска адреса нужных данных на физической странице используется индекс байта. Ознакомившись с общей картиной, перейдем к детальному рассмотрению структуры каталогов страниц, таблиц страниц и PTE.

Ассоциативный буфер трансляции

Как известно, трансляция каждого адреса требует двух операций поиска: **сначала нужно найти подходящую таблицу страниц в каталоге страниц**, затем — **элемент в этой таблице**. Поскольку выполнение этих двух операций при каждом обращении по виртуальному адресу могло бы снизить быстродействие системы до неприемлемого уровня, большинство CPU кэшируют транслируемые адреса, в результате чего необходимость в повторной трансляции при обращении к тем же адресам отпадает. CPU поддерживает такой кэш в виде массива ассоциативной памяти, называемого **ассоциативным буфером трансляции** (translation look-aside buffer, TLB). Ассоциативная память вроде TLB представляет собой вектор, ячейки которого можно считывать и сразу сравнивать с целевым значением. В случае TLB вектор содержит сопоставления физических и виртуальных адресов для недавно использовавшихся страниц, а также атрибуты защиты каждой страницы, как показано на Рис. 11. Каждый элемент TLB похож на элемент кэша, в метке которого хранятся компоненты виртуального адреса, а в поле данных — номер физической страницы, атрибуты защиты, битовый флаг Valid и, как правило, битовый флаг Dirty. Эти флаги отражают состояние страницы, которой соответствует кэшированный PTE. Если в PTE установлен битовый флаг Global (используется для страниц системного пространства, глобально видимых всем процессам), то при переключениях контекста элемент TLB не объявляется недействительным.

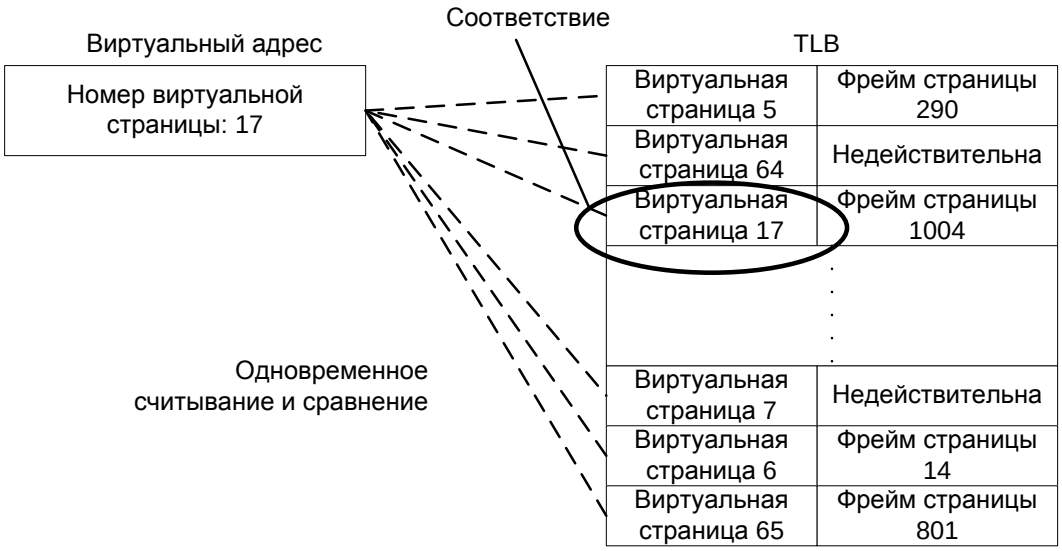

Рис. 11. Применение ассоциативного буфера трансляции

Часто используемым виртуальным адресам обычно соответствуют элементы в TLB, который обеспечивает чрезвычайно быструю трансляцию виртуальных адресов в физические, а в результате и быстрый доступ к памяти. Если виртуального адреса в TLB нет, он все еще может быть в памяти, но для его поиска понадобится несколько обращений к памяти, что увеличит время доступа. Если виртуальный адрес оказался в страничном файле или если диспетчер памяти изменил его PTE, диспетчер памяти должен явно объявить соответствующий элемент TLB недействительным. Если процесс повторно обращается к нему, генерируется ошибка страницы, нужная страница загружается обратно в память и для нее вновь создается элемент TLB.

Диспетчер памяти по возможности обрабатывает аппаратные и программные PTE одинаково.

Для архитектур IA64 и x64 используются немного иные подходы, они детально описаны в [3].

Physical Address Extension (PAE)

При работе CPU в режиме PAE (режим проецирования памяти Physical Address Extension) блок управления памятью (memory management unit, MMU) разделяет виртуальные адреса на 4 поля (Рис. 12).

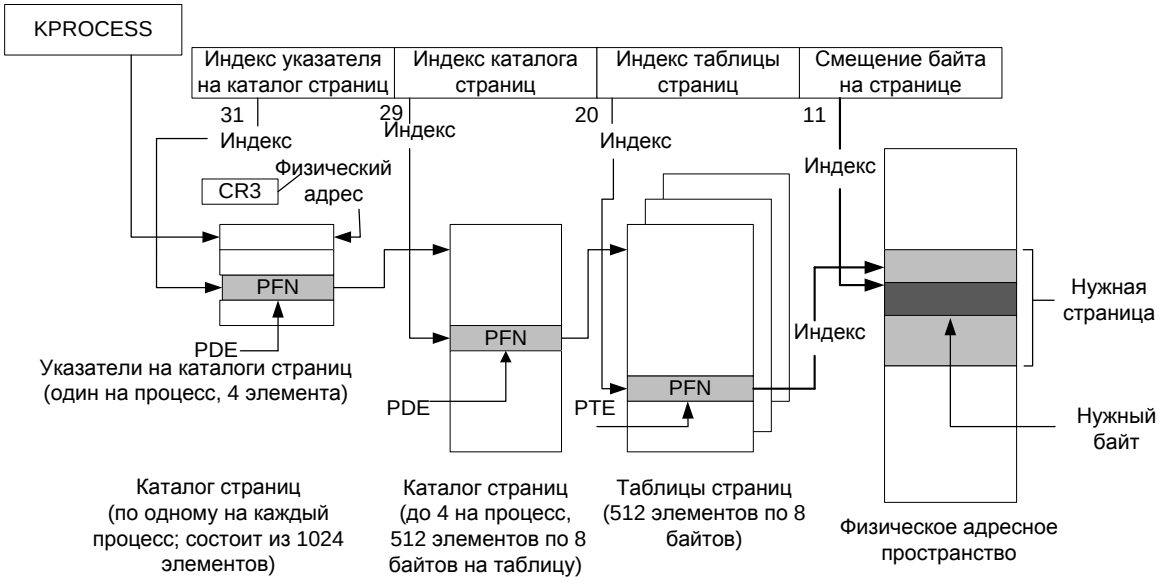

Рис. 12. Проецирование страниц по механизму PAE

Трансляция виртуальных адресов на платформе x64

64-разрядная Windows на платформе x64 применяет четырехуровневую схему таблиц страниц. У каждого процесса имеется расширенный каталог страниц верхнего уровня (называемый картой страниц уровня 4), содержащий 512 указателей на структуру третьего уровня - **родительский каталог страниц**. Каждый родительский каталог страниц хранит 512 указателей на каталоги страниц второго уровня, а те содержат по 512 указателей на индивидуальные таблицы страниц. Наконец, таблицы страниц (в каждой из которых 512 PTE) указывают на страницы в памяти. В текущих реализациях архитектуры x64 размер виртуальных адресов ограничен 48 битами.

Обработка ошибок страниц

Если битовый флаг Valid в PTE сброшен, это значит, что нужная страница по какой-либо причине сейчас недоступна процессу. Расскажем о типах недействительных PTE и о том, как разрешаются ссылки на такие PTE.

ПРИМЕЧАНИЕ Детально рассматриваются только PTE на 32-разрядной платформе x86. PTE для 64-разрядных систем содержат аналогичную информацию.

При ссылке на недействительную страницу возникает ошибка страницы (page fault), и обработчик ловушки ядра перенаправляет ее обработчику `MmAccessFault` диспетчера памяти. Последняя функция, выполняемая в контексте вызвавшего ошибку потока, предпринимает попытку ее разрешения (если это возможно) или генерирует соответствующее исключение. Причины таких ошибок перечислены в [3].

Операции ввода-вывода, связанные с подкачкой страниц

Такие операции ввода-вывода происходят в результате запроса на чтение страничного или проецируемого файла из-за ошибки страницы. Кроме того, поскольку в страничный файл могут помещаться и таблицы страниц, обработка ошибки страницы в случае таблицы страниц может повлечь за собой новые ошибки страниц.

Операции ввода-вывода, связанные с подкачкой, являются синхронными, т.е. поток ждет завершения подобной операции на каком-либо событии и она не может быть прервана вызовом асинхронной процедуры (APC). Для идентификации ввода-вывода как связанною с подкачкой подсистема подкачки страниц (pager) вызывает функцию запроса ввода-вывода, указывая специальный модификатор. По завершении операции подсистема ввода-вывода освобождает событие. Это пробуждает подсистему подкачки страниц, и она продолжает свою работу.

В ходе операции ввода-вывода, связанной с подкачкой, поток, который вызвал ошибку страницы, не владеет критичными синхронизирующими объектами, используемыми при управлении памятью. Другие потоки того же процесса могут вызывать функции управления виртуальной памятью и обрабатывать ошибки страниц в ходе операции ввода-вывода, связанной с подкачкой. Однако подсистема подкачки страниц должна уметь выходить из некоторых ситуаций, которые могут возникать на момент завершения такой операции:

- другой поток в том же или другом процессе вызывает ошибку той же страницы, из-за чего происходит конфликт ошибок страницы (см. следующий раздел);
- страница удалена из виртуального адресного пространства и перепроецирована;
- сменился атрибут защиты страницы;
- ошибка относится к прототипному PTE, а страница, которая проецирует этот PTE, отсутствует в рабочем наборе.

Подсистема подкачки страниц выходит из таких ситуаций следующим образом. Перед запросом на операцию ввода-вывода, связанную с подкачкой, она сохраняет в стеке ядра потока статусную информацию, что позволяет после выполнения запроса распознать возникновение одной из перечисленных выше ситуаций и при необходимости отбросить ошибку страницы, не делая эту страницу действительной. Если команда, вызвавшая ошибку страницы, выдается повторно, вновь активизируется подсистема подкачки страниц, и PTE вычисляется заново.

Конфликты ошибок страницы

Конфликт ошибок страницы (collided page fault) возникает, когда другой поток или процесс вызывает ошибку страницы, уже обрабатываемой в данный момент из-за предыдущей ошибки того же типа. Подсистема подкачки страниц распознает и оптимальным образом разрешает такие конфликты, поскольку они нередки в системах

с поддержкой многопоточности. Если другой поток или процесс вызывает ошибку той же страницы, подсистема подкачки страниц обнаруживает конфликт ошибок страницы, отмечая при этом, что страница находится в переходном состоянии и что она сейчас считывается. (Эта информация извлекается из элемента базы данных PFN.) Далее подсистема подкачки страниц переходит в ожидание на событии, указанном в элементе базы данных PFN. Это событие было инициализировано потоком, вызвавшим первую ошибку страницы.

По завершении операции ввода-вывода событие переходит в свободное состояние. Первый поток, захвативший блокировку базы данных PFN, отвечает за заключительные операции, связанные с подкачкой. К ним относятся проверка статуса операции ввода-вывода (чтобы убедиться в ее успешном завершении), сброс бита «в процессе чтения» в базе данных PFN и обновление PTE.

Когда следующие потоки захватывают блокировку базы данных PFN для завершения обработки конфликтующих ошибок страницы, сброшенный бит «в процессе чтения» сообщает подсистеме подкачки страниц, что начальное обновление закончено, и она проверяет флаг ошибок в элементе базы данных PFN. Если этот флаг установлен, PTE не обновляется, и в потоке, вызвавшем ошибку страницы, генерируется исключение «in-page error» (ошибка в процессе загрузки страницы).

Страничные файлы

Страничные файлы (page files) предназначены для хранения модифицированных страниц, которые используются каким-то процессом, но должны быть выгружены из памяти на диск. Пространство в страничном файле резервируется, когда происходит начальная передача страниц, но реальные участки страничного файла не выбираются до тех пор, пока страницы не выгружаются на диск. Важно отметить, что система накладывает ограничение на число передаваемых закрытых страниц. Поэтому значение счетчика производительности Process: Page File Bytes на самом деле отражает суммарный объем закрытой памяти, переданной процессам. Соответствующие страницы могут находиться в страничном файле (частично или целиком) или, напротив, в физической памяти.

Диспетчер памяти отслеживает использование закрытой переданной памяти на глобальном уровне и по каждому процессу отдельно (в виде квоты страничного файла). И вновь эти данные отражают не размер использованного пространства в страничном файле, а объем переданной закрытой памяти. Соответствующие счетчики увеличиваются при передаче виртуальных адресов, требующих новых закрытых физических страниц.

При загрузке системы процесс диспетчера сеансов считывает список страничных файлов, которые он должен открыть. Этот список хранится в параметре реестра `HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\Memory Management\PagingFiles`. Этот многострочный параметр содержит имя, минимальный и максимальный размеры каждого страничного файла. Windows поддерживает до 16 страничных файлов. Страничные файлы нельзя удалить во время работы системы, так как процесс System открывает описатель каждого страничного файла. Тот факт, что страничные файлы открываются системой, объясняет, почему встроенное средство дефрагментации не в состоянии дефрагментировать страничный файл в процессе работы системы. Для дефрагментации страничного файла используйте бесплатную утилиту `Pagedefrag` с сайта www.sysinternals.com. В ней применяется тот же подход, что и в других сторонних утилитах дефрагментации: она запускает свой процесс дефрагментации на самом раннем этапе загрузки системы, еще до открытия страничных файлов диспетчером сеансов.

Поскольку страничный файл содержит части виртуальной памяти процессов и ядра, для большей безопасности его можно настроить на очистку при выключении системы. Для этого установите параметр реестра `HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\MemoryManagement\ClearPageFileAtShutdown` в 1. Иначе в страничном файле останутся те данные, которые были выгружены в него к моменту выключения системы. И к этим данным сможет обратиться любой, кто получит физический доступ к компьютеру.

Дескрипторы виртуальных адресов

Момент загрузки страниц в память диспетчер памяти определяет, используя алгоритм **подкачки по требованию** (demand-paging algorithm). Страница загружается с диска,

если поток, обращаясь к ней, вызывает ошибку страницы. Подобно копированию при записи, подкачка по требованию является одной из форм отложенной оценки (lazy evaluation) выполнения операции только при ее абсолютной необходимости.

Диспетчер памяти использует отложенную оценку не только при загрузке страниц в память, но и при формировании таблиц, описывающих новые страницы. Например, когда поток передает память большой области виртуальной памяти с помощью `VirtualAHoc`, диспетчер памяти мог бы немедленно создать таблицы страниц, необходимые для доступа ко всему диапазону выделенной памяти.

При использовании алгоритма отложенной оценки выделение даже больших блоков памяти происходит очень быстро. Когда поток выделяет память, диспетчер памяти должен соответственно отреагировать. Для этого диспетчер памяти поддерживает набор структур данных, которые позволяют вести учет зарезервированных и свободных виртуальных адресов в адресном пространстве процесса. Эти структуры данных называются **дескрипторами виртуальных адресов** (virtual address descriptors, VAD). Для каждого процесса диспетчер памяти поддерживает свой набор VAD, описывающий состояние адресного пространства этого процесса. Для большей эффективности поиска VAD организованы в виде двоичного дерева с автоматической балансировкой. В Windows Server 2003 реализован **алгоритм дерева AVL** (это первые буквы фамилий его разработчиков — Adelson-Velskii и Landis), который обеспечивает более эффективную балансировку VAD-дерева, а это уменьшает среднее число операций сравнения при поиске VAD, соответствующего некоему виртуальному адресу.

Когда процесс резервирует адресное пространство или проецирует представление раздела, диспетчер памяти создает VAD для хранения информации из запроса на выделение — диапазона резервируемых адресов, его типа (разделяемый или закрытый), возможности наследования содержимого диапазона дочерними процессами, атрибутов защиты, установленных для страниц этого диапазона.

При первом обращении потока по какому-либо адресу диспетчер памяти должен создать PTE страницы, содержащей данный адрес. Для этого он находит VAD, чей диапазон включает нужный адрес, и использует его информацию для заполнения PTE. Если адрес выпадает из диапазонов VAD или находится в зарезервированном, но не переданном диапазоне адресов, диспетчер памяти узнает, что поток не выделил память до попытки ее использования, и генерирует нарушение доступа.

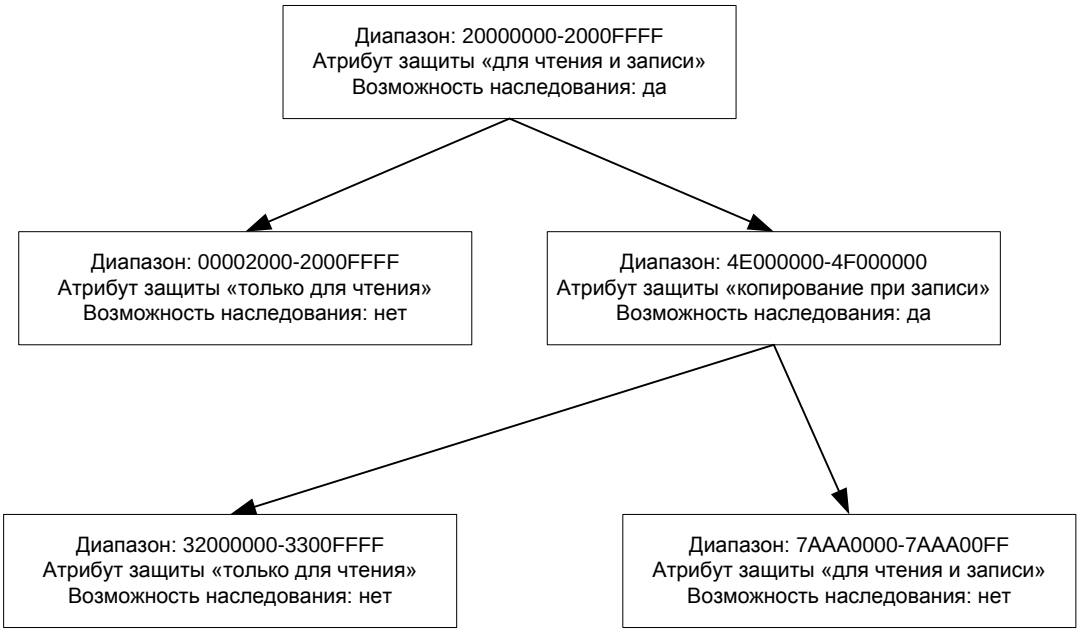

Рис. 13. Дескрипторы виртуальных адресов

Объекты-разделы

Объект «раздел» (section object), в подсистеме Windows **называемый объектом «проекция файла»** (file mapping object), представляет блок памяти, доступный двум и более процессам для совместного использования. Объект-раздел можно проецировать на страничный файл или другой файл на диске.

Исполнительная система использует разделы для загрузки исполняемых образов в память, а диспетчер кэша — для доступа к данным в кэшированном файле. Объекты «раздел» также позволяют проецировать файлы на адресные пространства процессов. При этом можно обращаться к файлу как к большому массиву, проецируя разные представления объекта-раздела и выполняя операции чтения-записи в памяти, а не в самом файле, — такие операции называются **вводам-выводам в проецируемые файлы** (mapped file I/O). Если программа обратится к недействительной странице (отсутствующей в физической памяти), возникнет ошибка страницы, и диспетчер памяти автоматически загрузит эту страницу в память из проецируемого файла. Если программа модифицирует страницу, диспетчер памяти сохранит изменения в файле в процессе обычных операций, связанных с подкачкой.

Как и другие объекты, разделы создаются и уничтожаются диспетчером объектов. Он создает и инициализирует заголовок объекта «раздел», а диспетчер памяти определяет тело этого объекта. Диспетчер памяти также реализует сервисы, через которые потоки пользовательского режима могут получать и изменять атрибуты, хранящиеся в теле объекта «раздел». Структура объекта «раздел» показана на Рис. 14.

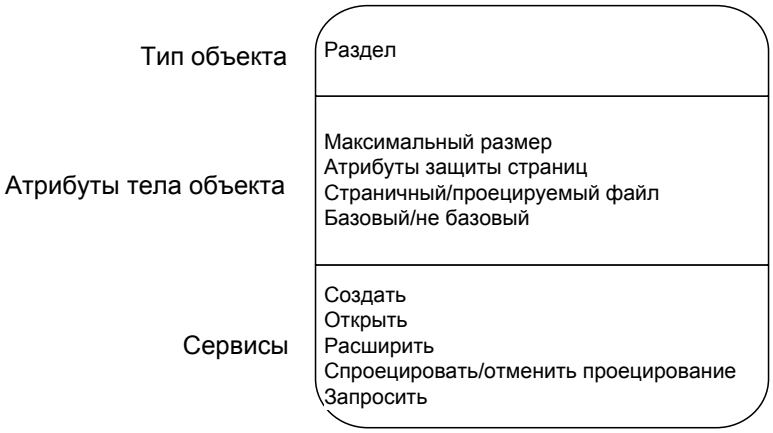

Рис. 14. Объект-раздел

Рис. 15. Внутренние структуры объекта-раздел

Рабочие-наборы

Сосредоточимся на виртуальной части Windows-процесса — таблицах страниц, PTE и VAD. Как известно, подмножество виртуальных страниц, резидентных в физической памяти, называется **рабочим набором** (working set). Существует три вида рабочих наборов:

- **процесса** — содержит страницы, на которые ссылаются его потоки;

- **системы** — содержит резидентное подмножество подкачиваемого системного кода (например, Ntoskrnl.exe и драйверов), пула подкачиваемой памяти и системного кэша;
- **сеанса** — в системах с включенной службой Terminal Services каждый сеанс получает свой рабочий набор. Он содержит резидентное подмножество специфичных для сеанса структур данных режима ядра, создаваемых частью подсистемы Windows, которая работает в режиме ядра (Win32k.sys), пула подкачиваемой памяти сеанса, представлений, проецируемых в сеансе, и других драйверов устройств, проецируемых на пространство сеанса.

Прежде чем детально рассматривать каждый тип рабочего набора, обсудим общие правила выбора страниц, загружаемых в память, и определения срока их пребывания в физической памяти.

Подкачка по требованию

Диспетчер памяти Windows использует алгоритм подкачки по требованию с кластеризацией. Когда поток вызывает ошибку страницы, диспетчер памяти загружает не только страницу, при обращении к которой возникла ошибка, но и несколько предшествующих и/или последующих страниц. Эта стратегия обеспечивает минимизацию числа операций ввода-вывода, связанных с подкачкой. Поскольку программы (особенно большие) в любой момент времени обычно выполняются в небольших областях своего адресного пространства, загрузка виртуальных страниц кластерами уменьшает число операций чтения с диска. При ошибках, связанных со ссылками на страницы данных в образах, размер кластера равен 3, в остальных случаях — 7.

Однако политика подкачки по требованию может привести к тому, что какой-то процесс будет вызывать очень много ошибок страниц в момент начала выполнения его потоков или позднее при возобновлении их выполнения. Для оптимизации запуска процесса (и системы) в Windows XP и Windows Server 2003 введен **механизм интеллектуальной предвыборки** (intelligent prefetch engine), также называемый **средством логической предвыборки** (logical prcfetcher).

Средство логической предвыборки

В ходе типичной загрузки системы или приложения порядок ошибок страниц таков, что некоторые страницы запрашиваются из одной части файла, затем из совсем другой его части, потом из другого файла и т.д. Такие скачкообразные переходы значительно замедляют каждую операцию доступа, и, как показывает анализ, время поиска на диске становится доминирующим фактором, который негативно сказывается на скорости загрузки системы и приложений. Предварительная выборка целого пакета страниц позволяет упорядочить операции доступа без лишнего «рыскания» по диску и тем самым ускорить запуск системы и приложений. Нужные страницы могут быть известны заранее благодаря высокой степени корреляции операций доступа при загрузках системы или запусках приложений.

Средство предвыборки, впервые появившееся в Windows XP, пытается ускорить загрузку системы и запуск приложений, отслеживая данные и код, к которым происходит обращение при этих процессах, и используя полученную информацию при последующих загрузке системы и запуске приложений для заблаговременного считывания необходимых кода и данных. Когда средство предвыборки активно, диспетчер памяти уведомляет код средства предвыборки в ядре об ошибках страниц — как аппаратных (требующих чтения данных с диска), так и программных (требующих простого добавления данных, которые уже находятся в памяти, в рабочий набор процесса).

Собрав трассировочную информацию об ошибках страниц, организованную в виде списка обращений к файлу метаданных NTFS MFT (Master File Table), а также списка ссылок на файлы и каталоги, код средства предвыборки, работающий в режиме ядра, уведомляет компонент предвыборки в службе Task Scheduler (Планировщик заданий) (`\Windows\System32\Schedsvc.dll`) и с этой целью переводит в свободное состояние объект-событие с именем `PrefetchTracesReady`.

Правила размещения

Когда поток вызывает ошибку страницы, диспетчер памяти должен также определить, в каком участке физической памяти следует разместить виртуальную страницу. При этом

он руководствуется **правилами размещения** (placement policy). Выбирая фреймы страниц, Windows учитывает размер кэшей CPU и стремится свести нагрузку на них к минимуму.

Если на момент появления ошибки страницы физическая память заполнена, выбирается страница, подлежащая выгрузке на диск для освобождения памяти под новую страницу. Этот выбор осуществляется по правилам замены (replacement policy). При этом действуют два общих правила замены: LRU (least recently used) и FIFO (first in, first out). **Алгоритм LRU** (также известный как алгоритм часов и реализованный в большинстве версий UNIX) требует от подсистемы виртуальной памяти следить за тем, когда используется страница в памяти. Страница, не использовавшаяся в течение самого длительного времени, удаляется из рабочего набора. **Алгоритм FIFO** работает проще: он выгружает из физической памяти страницу, которая находилась там дольше всего независимо от частоты ее использования.

Правила замены страниц могут быть глобальными или локальными. Глобальные правила позволяют использовать для обработки ошибки страницы любой фрейм страниц независимо от того, принадлежит ли он другому процессу. Например, в результате применения глобальных правил замены с применением алгоритма FIFO будет найдена и выгружена на диск страница, находившаяся в памяти наибольшее время, а локальные правила замены ограничат сферу поиска самой старой страницей из набора, который принадлежит процессу, вызвавшему ошибку страницы. Таким образом, глобальные правила замены делают процессы уязвимыми от поведения других процессов, и одно сбойное приложение может негативно отразиться на всей ОС.

В Windows реализована комбинация локальных и глобальных правил замены. Когда размер рабочего набора достигает своего лимита и/или появляется необходимость его усечения из-за нехватки физической памяти, диспетчер памяти удаляет из рабочих наборов ровно столько страниц, сколько ему нужно освободить.

Управление рабочими наборами

Все процессы начинают свой жизненный цикл с максимальным и минимальным размерами рабочего набора по умолчанию — 50 и 345 страниц соответственно, Хотя это мало что дает, эти значения по умолчанию можно изменить для конкретного процесса через Windows-функцию `SetProcessWorkingSetSize`, но для этого нужна привилегия `Increase Scheduling Priority`. Однако, если только не укажете процессу использовать жесткие лимиты на рабочий набор (новшество Windows Server 2003), эти лимиты игнорируются в том смысле, что диспетчер памяти разрешит процессу расширение за установленный максимум при наличии интенсивной подкачки страниц и достаточного объема свободной памяти (либо, напротив, уменьшит его рабочий набор ниже минимума при отсутствии подкачки страниц и при высокой потребности системы в физической памяти). Хотя в Windows 2000 степень расширения процесса за максимальную границу рабочего набора увязывалась с вероятностью его усечения, в Windows XP это решение принимается исключительно на основе того, к скольким страницам обращался процесс.

В Windows Server 2003 жесткие лимиты на размеры рабочего набора могут быть заданы вызовом функции `SetProcessWorkingSetSizeEx` с флагом `QUOTA_LIMITS_HARDWS_ENABLE`. Этой функцией пользуется, например, Windows System Resource Manager (WSRM).

Максимальный размер рабочего набора не может превышать общесистемный максимум, вычисленный при инициализации системы и хранящийся в переменной ядра `MmMaximwnWorkingSetSize`. Это значение представляет собой число страниц, доступных на момент вычислений (суммарный размер списков обнуленных, свободных и простаивающих страниц), за вычетом 512 страниц. Однако существуют жесткие верхние лимиты на размеры рабочих наборов — они перечислены в Таблица 4.

Таблица 4. Верхний лимит на максимальные размеры рабочих наборов

Версия Windows	Максимальный размер рабочего набора
x86-версии Windows 2000, Windows XP, Windows XP SP1, Windows Server 2003	1984 Мб
x86-версии Windows XP SP2, Windows Server 2003 SP1	2047,9 Мб

x8б-версии Windows, загружаемые с ключом /3GB	3008 Мб
IA64	7152 Гб
x64	8192 Гб

Когда возникает ошибка страницы, система проверяет лимиты рабочего набора процесса и объем свободной памяти. Если условия позволяют, диспетчер памяти разрешает процессу увеличить размер своего рабочего набора до максимума. Но, если памяти мало, Windows предпочитает заменять страницы в рабочем наборе, а не добавлять в него новые.

Хотя Windows пытается поддерживать достаточный объем доступной памяти, записывая измененные страницы на диск, при слишком быстрой генерации модифицированных страниц понадобится больше свободной памяти. Поэтому, когда свободной физической памяти становится мало, вызывается диспетчер рабочих наборов (working set manager), который выполняется в контексте системного потока диспетчера настройки баланса и инициирует автоматическое усечение рабочего набора для увеличения объема доступной в системе свободной памяти.

Диспетчер рабочих наборов принимает решения об усечении каких-либо рабочих наборов, исходя из объема доступной памяти. Если памяти достаточно, он подсчитывает, сколько страниц можно при необходимости изъять из рабочего набора. Как только такая необходимость появится, он уменьшит рабочие наборы, размер которых превышает минимальный. Диспетчер рабочих наборов динамически регулирует частоту проверки рабочих наборов и оптимальным образом упорядочивает список процессов — кандидатов на усечение рабочего набора. Например, первыми проверяются процессы со множеством страниц, к которым не было недавних обращений; часто простаивающие процессы большего размера являются более вероятными кандидатами, чем реже простаивающие процессы меньшего размера; процессы активного приложения рассматриваются в последнюю очередь и т.д.

Определив, что размеры рабочих наборов процессов превышают минимальные значения, диспетчер ищет страницы, которые можно удалить из их рабочих наборов и сделать доступными для использования в других целях. Если свободной памяти по-прежнему не хватает, диспетчер продолжает удалять страницы из рабочих наборов процессов, пока в системе не появится минимальное количество свободных страниц.

Диспетчер настройки баланса и подсистема загрузки-выгрузки

Расширение и усечение рабочего набора выполняется в контексте системного потока **диспетчера настройки баланса** (balance set manager) (процедура `KeBalanceSetManager`). Его поток создается при инициализации системы. Хотя с технической точки зрения диспетчер настройки баланса входит в состав ядра, для анализа и регулировки рабочих наборов он обращается к диспетчеру рабочих наборов.

Диспетчер настройки баланса ждет на двух объектах «событие»: один из них освобождается по сигналу таймера, срабатывающего раз в секунду, а другой представляет собой внутреннее событие диспетчера рабочих наборов, освобождаемое диспетчером памяти, когда возникает необходимость в изменении рабочих наборов. Например, если в системе слишком часто генерируются ошибки страниц или список свободных страниц слишком мал, диспетчер памяти пробуждает диспетчер настройки баланса, а тот вызывает диспетчер рабочих наборов для усечения таких наборов. Если свободной памяти много, диспетчер рабочих наборов разрешает процессам, часто вызывающим ошибки страниц, постепенно увеличивать размеры своих рабочих наборов, подкачивая в память страницы, при обращении к которым возникали ошибки. Однако рабочие наборы расширяются лишь по мере необходимости.

Диспетчер настройки баланса, пробуждаемый в результате срабатывания таймера, выполняет следующие операции.

1. При каждом четвертом пробуждении из-за срабатывания таймера освобождает событие, которое активизирует системный поток, выполняющий процедуру `KeSwapProcessOrStack` — подсистему загрузки-выгрузки (swapper).

2. Проверяет ассоциативные списки и регулирует глубину их вложения, если это необходимо (для ускорения доступа, снижения нагрузки на пул и уменьшения его фрагментации).

3. Ищет потоки, чей приоритет может быть повышен из-за нехватки времени CPU.

4. Вызывает диспетчер рабочих наборов (имеющий собственные внутренние счетчики, которые определяют, когда и насколько агрессивно следует проводить усечение рабочих наборов).

Подсистема загрузки-выгрузки пробуждается и планировщиком, если стек ядра потока, подлежащего выполнению, или весь процесс выгружен в страничный файл. Подсистема загрузки-выгрузки ищет потоки, которые находились в состоянии ожидания в течение 7 секунд (Windows 2000) или 15 секунд (Windows XP и Windows Server 2003). Если такой поток есть, подсистема загрузки-выгрузки переводит его стек ядра в переходное состояние (перемещая соответствующие страницы в список модифицированных или простаивающих страниц). Здесь действует принцип «если поток столько времени ждал, подождет и еще». Когда из памяти удаляется стек ядра последнего потока процесса, этот процесс помечается как полностью выгруженный. Вот почему у долго простаивавших процессов (например, у Winlogon после вашего входа в систему) может быть нулевой размер рабочих наборов

Системный рабочий набор

Подкачиваемый код и данные ОС тоже управляются как единый **системный рабочий набор** (system working set). В системный рабочий набор могут входить страницы пяти видов:

- системного кэша;
- пула подкачиваемой памяти;
- подкачиваемого кода и данных Ntoskrnl.exe;
- подкачиваемого кода и данных драйверов устройств;
- проецируемых системой представлений.

Выяснить размер системного рабочего набора и пяти его элементов можно с помощью счетчиков производительности или системных переменных.

Узнать интенсивность подкачки страниц в системном рабочем наборе позволяет счетчик Memory: Cache Faults/Sec (Память: Ошибок кэш-памяти/ сек), который сообщает число ошибок страниц, генерируемых в системном рабочем наборе (как аппаратных, так и программных).

Литература

1. Э. Таненбаум. Современные операционные системы. 2-ое изд. –СПб.: Питер, 2002. – 1040 с.

2. Э. Таненбаум, А. Вудхалл. Операционные системы: разработка и реализация. Классика CS. –СПб.: Питер, 2006. –576 с.

3. М. Руссинович, Д. Соломон. Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP, Windows 2000. Мастер-класс. / Пер. с англ. -4-е изд. –М.: Издательско-торговый дом «Русская редакция»; СПб.: Питер; 2005. -992 с.

4. Microsoft Development Network. URL: http://msdn.com