1. Управление процессами в ОС Windows.

В WRK за управление процессами отвечает диспетчер процессов (base\ntos\ps), а многие важные структуры данных описаны в заголовочных файлах base\ntos\inc\ps.h и base\ntos\inc\ke.h.

Процесс (process) – это программа (пользовательская или системная) в ходе выполнения.

Блок KPROCESS, входящий в блок EPROCESS, и РЕВ (process execution block), на который указывает EPROCESS, содержат дополнительные сведения об объекте «процесс». Блок KPROCESS, иногда называемый блоком управления процессом (process control block, PCB), показан на Рис. 3. Он содержит базовую информацию, нужную ядру Windows для планирования потоков.

Заголовок диспетчера

Время в режиме ядра

Элемент списка

Inswap

/

Outswap

(

«

загружен

/

выгружен

»

)

Спин

-

блокировка процесса

Привязка к процессорам

Счетчик резидентского стека ядра

Базовый приоритет процесса

Квант

,

выделяемый потокам по

умолчанию

Состояние процесса

Зародыш потоков

Флаг отключения динамического

повышения приоритета

Каталог страниц процесса

KTHREAD

Время в пользовательском режиме

. . .

Рис. 3. Блок управления процессом исполнительной системой

1. ОС Windows: потоки (внутреннее устройство потоков, структуры данных, переменные ядра, сопутствующие функции, создание потоков).

Процесс может содержать один или несколько потоков (thread) – объектов, которым операционная система предоставляет процессорное время. Сам по себе процесс не выполняется – выполняются его потоки. Таким образом, машинные команды, записанные в исполняемом файле, выполняются на процессоре в составе потока. Если потоков несколько, они могут выполняться одновременно.

Замечание. "Одновременное" (или "параллельное") выполнение потоков подразумевает одну из двух принципиально разных ситуаций, зависящих от количества процессоров (ядер) на компьютере. В том случае, если имеется всего один процессор с одним ядром, в один момент времени может выполняться только один поток. Однако операционная система может быстро переключать процессор с выполнения одного потока на другой и вследствие высокой частоты процессоров у пользователя возникает иллюзия одновременной работы нескольких программ. Такая ситуация называется псевдопараллельное выполнение потоков. Если в компьютере установлен многоядерный процессор или количество процессоров больше одного, то возможно истинно параллельное или просто параллельное выполнение потоков.

Формируется структура данных ETHREAD, стек и контекст потока, генерируется идентификатор потока. Поток создается при помощи функции NtCreateThread, определенной в файле base\ntos\ps\create.c, (строка 117), которая вызывает функцию PspCreateThread (тот же файл, строка 295). При этом выполняются следующие действия:

• создается объект ETHREAD (строка 370).

• Заполняются поля структуры ETHREAD, связанные с процессом-владельцем, – указатель на структуру EPROCESS (ThreadsProcess) и идентификатор процесса (Cid.UniqueProcess) (строки 396 и 398).

• Генерируется уникальный идентификатор потока (функция ExCreateHandle) и сохраняется в поле Cid.UniqueThread структуры EPROCESS (строки 400–402).

• Заполняются стартовые адреса потока, системный (StartAddress) и пользовательский (Win32StartAddress) (строки 468-476).

• Инициализируются поля структуры KTHREAD при помощи вызова функции KeInitThread (строки 490–498 для потока пользовательского режима и 514–522 для потока режима ядра).

• Функция KeStartThread заполняет остальные поля структуры ETHREAD и вставляет поток в список потоков процесса (строка 564).

• Если при вызове функции PspCreateThread установлен флаг CreateSuspended ("Приостановлен") поток переводится в состояние ожидания (функция KeSuspendThread, строка 660); иначе вызывается функция KeReadyThread (строка 809), которая ставит поток в очередь готовых к выполнению потоков (см. лекцию 9 "Планирование потоков").

Рождение потока

Жизненный цикл потока начинается при его создании программой. Запрос на его создание в конечном счете поступает исполнительной системе Windows, где диспетчер процессов выделяет память для объекта «поток» и вызывает ядро для инициализации блока потока ядра. Ниже перечислены основные этапы создания потока Windowsфункцией CreateThread (которая находится в Kernel32.dll).

1. CreateThread создает стек пользовательского режима в адресном пространстве процесса.

2. CreateThread инициализирует аппаратный контекст потока, специфичный для конкретной архитектуры CPU.

3. Для создания объекта «поток» исполнительной системы вызывается NtCreateThread. Он создается в приостановленном состоянии.

4. CreateThread уведомляет подсистему Windows о создании нового потока, и та выполняет некоторые подготовительные операции.

5. Вызвавшему коду возвращаются описатель и идентификатор потока

(сгенерированный на этапе 3).

6. Выполнение потока возобновляется, и ему может быть выделено время CPU, если только он не был создан с флагом CREATE\_SUSPENDED. Перед вызовом по пользовательскому стартовому адресу поток выполняет операции, описанные в разделе «Этап 3: создание первичного потока, его стека и контекста» ранее в этой главе.

1. Планирование потоков: уровни приоритета процессов и потоков, состояния потоков.

Если *операционная система* поддерживает многопоточность, она может распределять *процессорное время* либо между процессами, либо между потоками. В операционной системе *Windows* *процессор* предоставляется потокам, иначе говоря, осуществляется планирование на уровне потоков.

Таким образом, если один процесс имеет пять потоков, а второй – десять, то первый процесс будет занимать *процессор* в два раза больше времени, чем второй (при условии, конечно, что все потоки имеют равный приоритет и выполняют примерно одинаковую работу).

**Алгоритмы планирования**

Существуют разные алгоритмы планирования. Рассмотрим основные виды.

**1. Вытесняющие/невытесняющие алгоритмы.**

В случае вытесняющего алгоритма операционная система в любой момент времени может прервать выполнение текущего потока и переключить процессор на другой поток. В невытесняющих алгоритмах поток, которому предоставлен процессор, только сам решает, когда передать управление операционной системе.

**2. Алгоритмы с квантованием.**

Каждому потоку предоставляется квант времени, в течение которого поток может выполняться на процессоре. По истечении кванта операционная система переключает процессор на следующий поток в очереди. Квант обычно равен целому числу интервалов системного таймера1.

**3. Алгоритмы с приоритетами.**

Каждому потоку назначается приоритет (priority) – целое число, обозначающее степень привилегированности потока. Операционная система при наличии нескольких готовых к выполнению потоков выбирает из них поток с наибольшим приоритетом.

В Windows реализован смешанный алгоритм планирования – вытесняющий, на основе квантования и приоритетов.

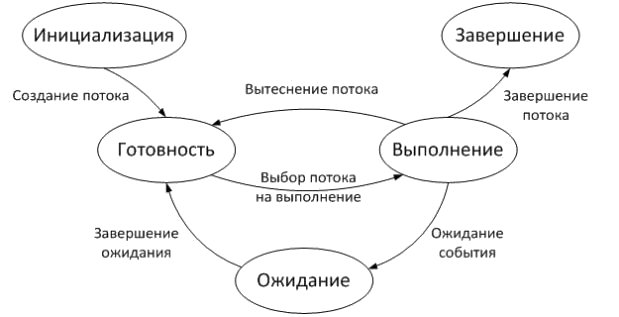
**Состояния потоков**

За время своего существования поток может находиться в нескольких состояниях. Перечислим основные состояния:

1. Готовность (Ready) – поток готов к выполнению и ждет своей очереди занять процессор.
2. Выполнение (Running) – поток выполняется на процессоре.
3. Ожидание (Waiting) – поток не может выполняться, поскольку ждет наступление некоторого события (например, завершения операции ввода-вывода или сообщения от другого потока)

Кроме основных существует ещё несколько состояний – Инициализация (Init), Завершение (Terminate), Состояние простоя (Standby), Переходное состояние (Transition), Состояние отложенной готовности (Deferred ready). Подробнее о них можно узнать в [[5](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.5); [2](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.2)].

На [рис.9.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16576?page=1#image.9.1) показаны основные состояния потока, возможные переходы между состояниями и условия переходов.



**Рис. 9.1.**Состояния потока

### **Уровни приоритета**

Как показано на Рис. 10, в Windows предусмотрено 32 уровня приоритета — от 0 до 31. Эти значения группируются так:

* шестнадцать уровней **реального времени** (16-31);
* пятнадцать **варьируемых (динамических)** уровней (1—15);
* один **системный уровень** (0), зарезервированный для потока обнуления страниц (zero page thread).



Рис. 10. Уровни приоритета потоков

Уровни приоритета потока назначаются с учетом двух разных точек зрений Windows API и ядра Windows. Windows API сначала упорядочивает процессы по классам приоритета, назначенным при их создании [Real-time (реального времени), High (высокий), Above Normal (выше обычного), Normal (обычный), Below Normal (ниже обычного) и Idle (простаивающий)], а затем — по относительному приоритету индивидуальных потоков в рамках этих процессов [Time-critical (критичный по времени), Highest (наивысший), Above-normal (выше обычного), Normal (обычный), Below-normal (ниже обычного), Lowest (наименьший) и Idle (простаивающий)].

Базовый приоритет каждого потока в Windows API устанавливается, исходя из класса приоритета его процесса и относительного приоритета самого потока. Связь между приоритетами Windows API и внутренними приоритетами ядра Windows (в числовой форме) показана на Рис. 11.

Если у процесса только одно значение приоритета (базовое), то у каждого потока их два: текущее и базовое. Решения, связанные с планированием, принимаются на основе текущего приоритета. В определенных обстоятельствах система может на короткое время повышать приоритеты потоков в динамическом диапазоне (1-15). Windows никогда не изменяет приоритеты потоков в диапазоне реального времени (16-31), поэтому у таких потоков базовый приоритет идентичен текущему.



Рис. 11. Взаимосвязь приоритетов в ядре и Windows API

Начальный базовый приоритет потока наследуется от базового приоритета процесса, а тот наследует его от родительского процесса. Это поведение можно изменить при вызове Windows-функции CreateProcess или команды START. Приоритет процесса можно изменить и после его создания, используя функцию SetPriorityClass или различные утилиты, предоставляющие доступ к этой функции через UI, например диспетчер задач и Process Explorer. В частности, можете понизить приоритет процесса, интенсивно использующего время CPU, чтобы он не мешал обычным операциям в системе. Смена приоритета процесса влечет за собой смену приоритетов всех его потоков, но их относительные приоритеты остаются прежними. Но изменение приоритетов индивидуальных потоков внутри процесса обычно не имеет смысла, потому что вы не знаете, чем именно занимается каждый из них (если только сами не пишете программу или не располагаете исходным кодом); так что изменение относительных приоритетов потоков может привести к неадекватному поведению этого приложения.

Обычно базовый приоритет процесса (а значит, и базовый приоритет первичного потока) по умолчанию равен значению из середины диапазонов приоритетов процессов (24, 13, 10, 8, 6 или 4). Однако базовый приоритет некоторых системных процессов (например, диспетчера сеансов, контроллера сервисов и сервера локальной аутентификации) несколько превышает значение по умолчанию для класса Normal (8). Более высокий базовый приоритет по умолчанию обеспечивает запуск потоков этих процессов с приоритетом выше 8. Чтобы изменить свой начальный базовый приоритет, такие системные процессы используют внутреннюю функцию

NtSetInformationProcess.

1. Кванты времени, выделяемые потокам, управление величиной кванта.

**Кванты**

В Windows имеется два базовых размера кванта – 2 интервала системного таймера и 12 интервалов. Если квант времени короткий, то потоки будут переключаться быстрее и "отзывчивость" (responsiveness) системы улучшится – это важное свойство для пользователя, поэтому в клиентских системах Windows по умолчанию используются короткие кванты. При этом производительность системы в целом снижается, поскольку потоки не будут успевать выполнять свои задачи в течение выделенного кванта, а частые переключения создадут высокие накладные расходы (служебные операции системы при смене потоков). Вследствие этого в серверных версиях Windows по умолчанию применяются длинные кванты.

Длительность интервала системного таймера (в сотнях наносекунд) хранится в переменной KeMaximumIncrement (для x86 – файл base\ntos\ex\i386\splocks.asm, строка 140; для x64 – файл base\ntos\ex\amd64\hifreqlk.asm, строка 147) и устанавливается функцией KeSetTimeIncrement (файл base\ntos\ke\miscc.c, строка 711 на основе значения, предоставляемого HAL.

Каждый процесс хранит величину кванта в поле QuantumReset структуры KPROCESS (файл base\ntos\inc\ke.h, строка 1029). Значение в этом поле равно количеству интервалов таймера, умноженному на 3. Например, для длинных квантов (12 интервалов) значение QuantumReset будет равно 36. Таким образом, при каждом срабатывании таймера (возникает прерывание) система уменьшает квант выполняющегося потока на 3 единицы.

Умножение на три введено для того чтобы можно было в разной степени уменьшать квант в двух различных ситуациях – срабатывании таймера (квант уменьшается на 3 единицы) и выходе из состояния ожидания (квант уменьшается на единицу). Уменьшение кванта при выходе потока из состояния ожидания применяется чтобы избежать ситуации бесконечно выполняющегося потока: если при каждом срабатывании таймера поток находится в состоянии ожидания, а при выходе из ожидания значение кванта не изменяется, то теоретически поток может выполняться бесконечно. Поэтому при выходе из состояния ожидания текущее значение его кванта уменьшается на единицу.

Значение кванта может быть изменено пользователем. Например, на Windows 7 нужно проделать следующее: Компьютер – Свойства – Дополнительные параметры системы – вкладка "Дополнительно" – раздел "Быстродействие" – Параметры – вкладка "Дополнительно" – раздел "Распределение времени процессора". Можно выбрать короткие кванты ("Оптимизировать работу программ") или длинные ("Оптимизировать работу служб, работающих в фоновом режиме") ([рис.9.2](https://www.intuit.ru/studies/courses/10471/1078/lecture/16576?page=1#image.9.2)).

1. Сценарии планирования выполнения потоков, переключение контекста, поток простоя.

**Алгоритм планирования в Windows**

В Windows отсутствует единый модуль, отвечающий за планирование потоков. Алгоритм планирования реализуется несколькими процедурами ядра, совокупность которых называется диспетчером ядра (kernel’s dispatcher).

Для хранения данных, необходимых для планирования, предназначена база данных диспетчера ядра, которая является частью структуры KPRCB (Kernel Processor Control Block), описанной в файле base\ntos\inc\i386.h (строка 1073). Эта структура создается для каждого процессора, присутствующего в системе. Структура KPRCB содержит следующие поля, требуемые для планирования:

* CurrentThread – указатель на текущий выполняющийся поток;
* NextThread – указатель на следующий поток для выполнения;
* IdleThread – указатель на поток простоя;
* DispatcherReadyListHead – массив списков, содержащих указатели на потоки, готовые к выполнению. Количество элементов массива совпадает с количеством уровней приоритета в системе (32), т. е. для каждого приоритета поддерживается своя очередь потоков в состоянии готовности;
* ReadySummary – 32 битное число, каждый из разрядов которого отвечает за один уровень приоритета. Единица в N-ом разряде означает, что очередь готовых к выполнению потоков с приоритетом N не пустая. Это поле используется для ускорения поиска при выборе потока для выполнения.

Выбор потока с максимальным приоритетом из массива DispatcherReadyListHead с использованием поля ReadySummary осуществляется функцией KiSelectReadyThread (файл base\ntos\ke\ki.h, строка 3550).

Рассмотрим основные ситуации, возникающие при планировании потоков.

**1. Выбор потока на выполнение.**

Просматривается очередь готовых к выполнению потоков (сначала поле ReadySummary, затем, когда определена непустая очередь с максимальным приоритетом, поле DispatcherReadyListHead) и выбирается первый поток в очереди с наибольшим приоритетом, которому для выполнения предоставляется квант времени ([рис.9.4](https://www.intuit.ru/studies/courses/10471/1078/lecture/16576?page=2#image.9.4)).

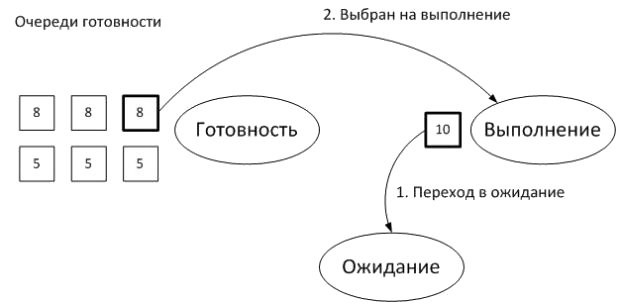


**Рис. 9.4.**Выбор потока для выполнения (квадратами обозначены потоки, числами – их приоритеты)

**2. Переход выполняющегося потока в состояние ожидания.**

Выполняющийся поток вызывает одну из функций ожидания (см. MSDN – Wait Functions [[10](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.10)]) и освобождает процессор. Его квант времени не истек и сохраняется за потоком, но при выходе из состояния ожидания уменьшается на единицу (см. параграф "Кванты" этой лекции).

Диспетчер ядра выбирает на выполнение первый поток из очереди с наибольшим приоритетом ([рис.9.5](https://www.intuit.ru/studies/courses/10471/1078/lecture/16576?page=2#image.9.5)).



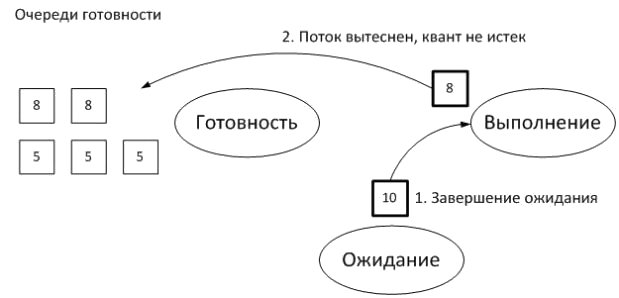
**Рис. 9.5.**Переход потока в состояние ожидания

**3. Вытеснение потоком с большим приоритетом.**

Во время выполнения поток может быть вытеснен при появлении потока с большим приоритетом. Такая ситуация может возникнуть по следующим причинам:

* поток с большим приоритетом завершил ожидание ([рис.9.6](https://www.intuit.ru/studies/courses/10471/1078/lecture/16576?page=2#image.9.6));
* приоритет потока в очереди готовности динамически увеличился (см. далее в этой лекции);
* в системе создан поток с большим приоритетом.

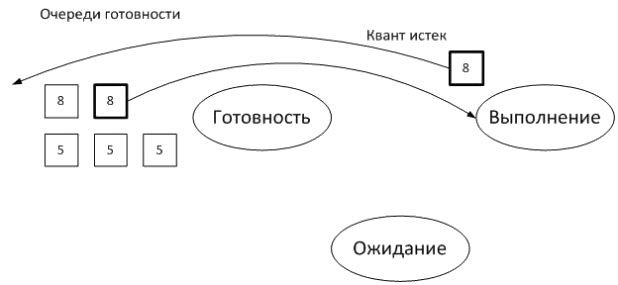
В любом случае выполняющийся поток вытесняется, помещается в начало очереди готовности с соответствующим приоритетом; при этом неистраченная часть кванта остается за потоком.



**Рис. 9.6.**Вытеснение потока

**4. Завершение кванта времени**

Когда квант времени, предоставленный потоку, истекает, операционная система проверяет, есть ли в очереди готовности поток с таким же приоритетом или выше. Если есть, то поток помещается в конец соответствующей очереди готовности и новый поток выбирается на выполнение ([рис.9.7](https://www.intuit.ru/studies/courses/10471/1078/lecture/16576?page=2#image.9.7)). Если такие потоки отсутствуют, выполняющемуся потоку может быть предоставлен новый квант времени.



**Динамическое повышение приоритета**

Если бы операционная система осуществляла планирование потоков только на основе выше рассмотренных ситуаций, большинство потоков с низким приоритетом вообще никогда не выполнялись бы – диспетчер ядра все время выбирал бы потоки с наивысшим приоритетом.

Чтобы дать всем потокам шанс на выполнение операционная система применяет механизм динамического повышения приоритета (Priority Boosts), который работает в следующих случаях:

* возникает событие диспетчера ядра;
* завершается операции ввода/вывода;
* происходит событие пользовательского интерфейса;
* поток слишком долго ожидает ресурс;
* поток слишком долго ожидает своей очереди на выполнение.

Замечание. Никогда не повышаются приоритеты потоков реального времени (16–31).

1. Особенности планирования потоков для многопроцессорных систем.

В однопроцессорной системе алгоритм планирования относительно прост: всегда выполняется поток с наивысшим приоритетом, готовый к выполнению. В многопроцессорной системе планирование усложняется, так как Windows пытается подключать поток к наиболее оптимальному для него CPU, учитывая предпочтительный и предыдущий CPU для этого потока, а также конфигурацию многопроцессорной системы. Поэтому, хотя Windows пытается подключать готовые к выполнению потоки с наивысшим приоритетом ко всем доступным CPUs, она гарантирует лишь то, что на одном из процессоров будет работать (единственный) поток с наивысшим приоритетом.

Прежде чем описывать специфические алгоритмы, позволяющие выбирать, какие потоки, когда и на каком процессоре будут выполняться, давайте рассмотрим дополнительную информацию, используемую Windows для отслеживания состояния потоков и CPUs как в обычных многопроцессорных системах, так и в двух новых типах таких систем, поддерживаемых Windows, — в системах с физическими CPUs, поддерживающими логические (hyperthreaded systems), и NUMA.

1. Диспетчер памяти ОС Windows: компоненты диспетчера, синхронизация, конфигурирование.

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 хб4.

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-notvalid) 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 спинблокировку),

• 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