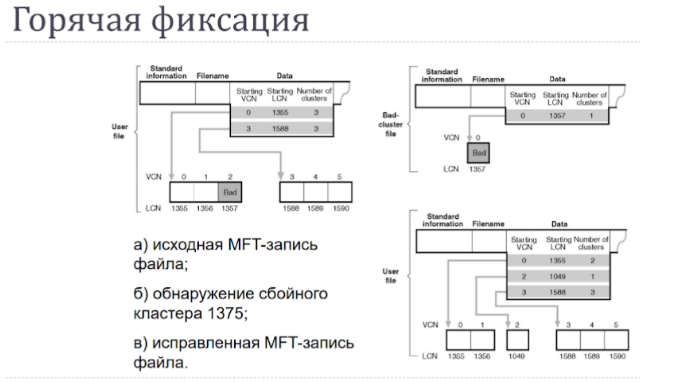
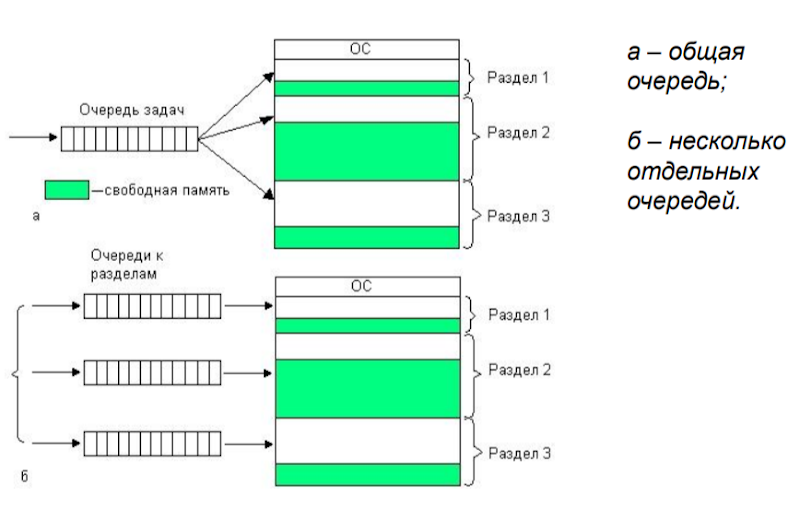
**10.Сжатие файлов в NTFS. Защита целостности данных в NTFS.**

Как происходит сжатие, почему именно по 16 кластеров, алгоритм для разреженных файлов, алгоритм Lempel-Ziv. Разреженные применяются в лог-файле. Горячая фиксация, механизм транзакций



**11.Методы распределения памяти без использования внешней памяти**

Разделение фиксированными разделами (не обязательно равными) – выбор подходящего под процесс раздела

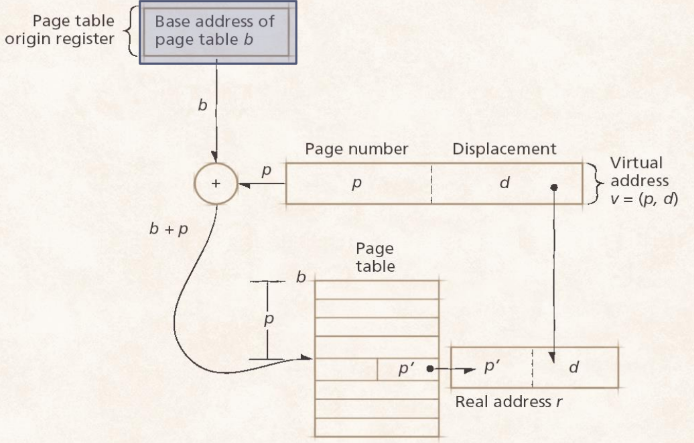


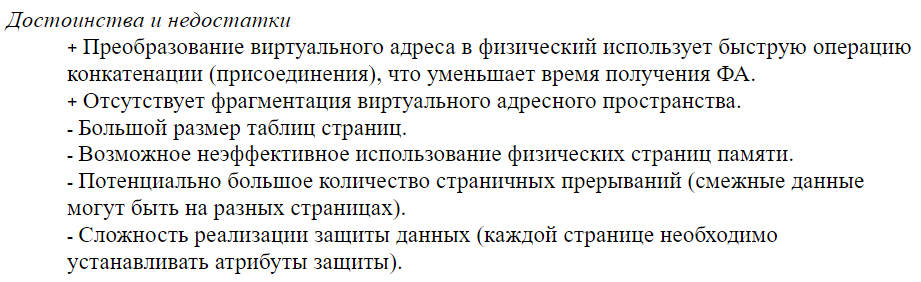
Распределение переменными разделами (заранее не делим на разделы, а выделяем поступившей задаче. плохо – фрагментация памяти). First fir, Best fit, Worst fit.

Перемещаемы разделы – сжатие, борьба с фрагментацией

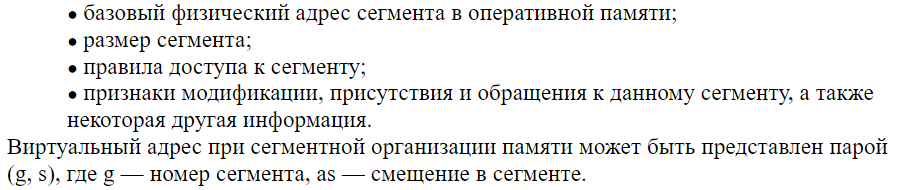
**12.Методы распределения памяти с использованием внешней памяти.**

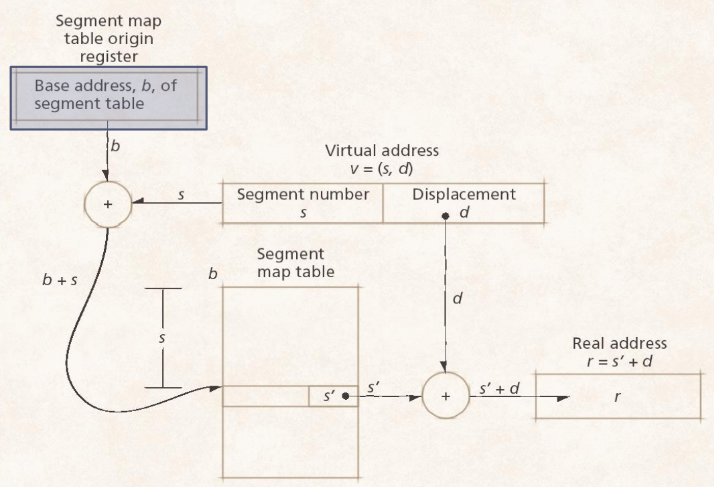
Страничное распределение. Таблица страниц – номер физ. страницы, признак присутствия, признак модификации, признак обращения, физ. и вирт. смещения равны между собой, таблицы в ОП, таблица страниц загружается в контекст процесса. Как преобразуется ВА в ФА, страничное прерывание (загрузка или своппинг)



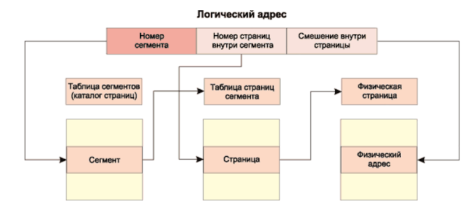


Сегментное распределение. Таблица сегментов





Сегментно-страничное распределение – в сегменте своя таблица страниц



**13.Стратегии управления виртуальной памятью (свопинг).**

Свопиинг == замещение

Стратегии: выборка (по запросу, с упреждением), размещение, замещение

Алгоритмы замещения: **оптимальный** (невозможный), **FIFO**, **FIFO2** – проверяется признак присутствия, если 1 – то обнуляем и ставим в конец очереди, **LRU** – дольше всех неиспользованные страницы, нужен признак обращения и счетчик обращений, **LRU2** – признак достоверности == эмуляция признака обращения, **NRU (clock)** – кольцевой список, сбрасывается признак обращения, **NFU** – программный счетчик на каждую страницу, наим. знач. счетчика, нельзя забыть страницу, можно если модифицировать.

Стратегии выделения физических страниц процессам: фиксированное для каждого процесса, переменное, изменяется во время жизни процесса.

Области действия алг. замещения: локальные (внутри АП процесса), глобальные (можно вытеснять из другого процесса)

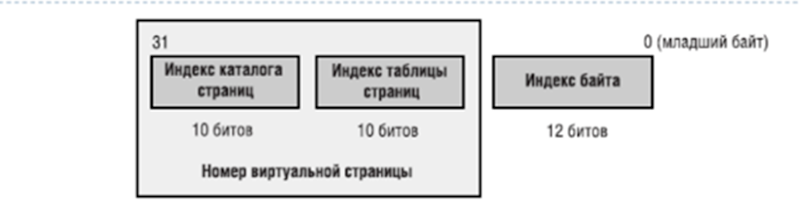
Трешинг – подкачка дольше, чем использование. Решение – использование локальных алгоритмов замещения.

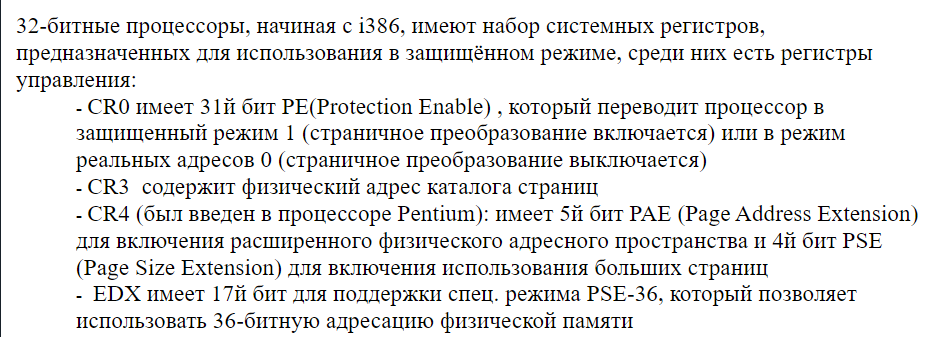
**14.Архитектура памяти MS Windows 2000+. Менеджер ВП. Виртуальное АП. Двухуровневое страничное преобразование. Большие страницы.**

ВП, разделение на пользовательскую и для нужд ОС



Функции и задачи WMM

В винде многоуровневое страничное распределение. Каждому процессу свой массив указателей на таблицы страниц.



Дальше пиздец

**15. хз че это**

1. Динамическое TLS

4 байт, функции апи, количество ячеек ограничено

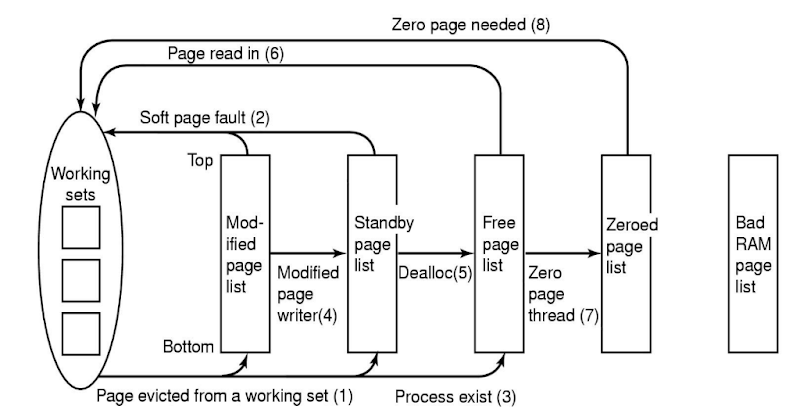
* TlsAlloc () - резервирует ячейку и возвращает ее индекс, TLS\_OUT\_OF\_INDEXES - если массив ячеек весь занят
* TlsSetValue () - записывает данные в ячейку с индексом, который был возвращен TlsAlloc
* TlsGetValue () - получает данные по индексу TlsAlloc
* TlsFree () - очищает ячейку по индексу TlsAlloc

2. Статическое

Все неограниченно ничего не надо, \_\_declspec(thread)

**16.Реализация свопинга в MS Windows 2000+.**

Состояния страниц: Valid/Active/Present, Standby, Modified, Zeroed, Bad



# **17.Организация «статической» виртуальной памяти в MS Windows 2000+. Блоки адресов. Состояния блоков адресов.**

Дескрипторы виртуальных адресов – набор структур данных, с помощью которых ведется учёт зарезервированных и свободных диапазонов ВАП.

Двоичное дерева с автоматической балансировкой

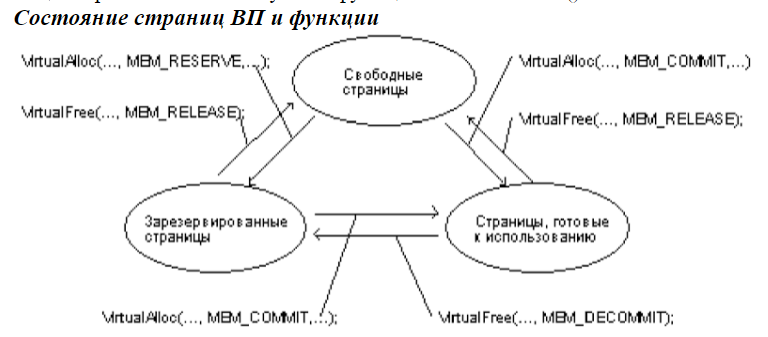
Начальный адрес блока должен быть выровнен на значение **dwAllocationGranularity** (обычно равно 64K), при необходимости адрес округляется «вниз». Размер блока должен быть кратен размеру страницы **dwPageSize**, при необходимости округляется «вверх».

Блок выделен/зврезервирован/свободен

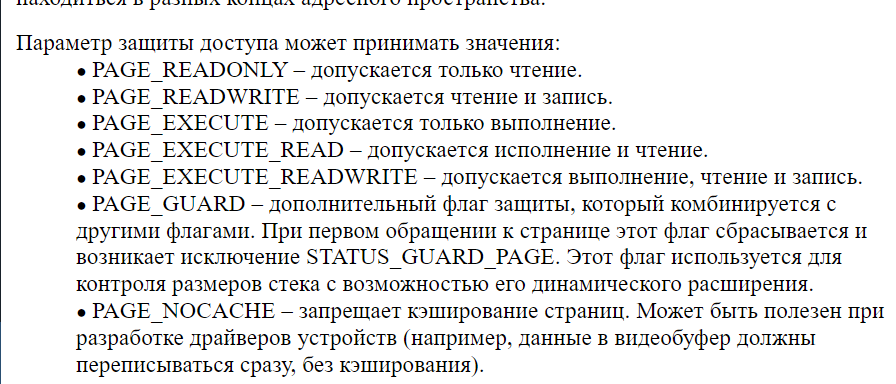
**VirtualAlloc (), VirtualFree (), (ex)** для произвольного процесса

**VirtualLock () и VirtualUnlock ()** – блок на своппинг

**VirtualProtect () и VirtualProtectEx ()** – изменение атрибутов защиты региона



* MEM\_COMMIT – отображает ранее зарезервированный блок адресов на физическую память или файл подкачки, выделяя при этом память. Может комбинироваться с флагом MEM\_RESERVE для одновременного резервирования и выделения.
* MEM\_TOP\_DOWN – выделяет память по наибольшему возможному адресу. Имеет смысл только при lpAddress = NULL.
* MEM\_RELEASE – освободить зарезервированный регион. При использовании этого флага параметр dwSize должен быть равен нулю.

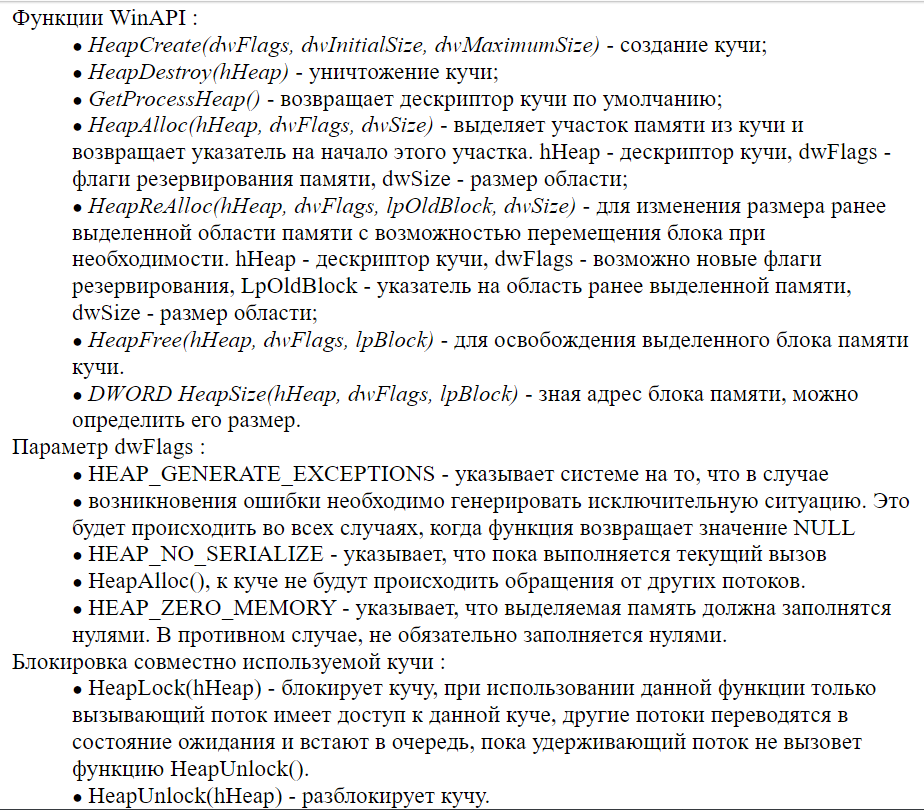


С помощью VirtualAlloc можно выделять большие страницы, для этого в типе выделения нужно передать параметр MEM\_LARGE\_PAGES, но перед этим нужно получить на это привилегии для процесса с помощью AdjustTokenPrivileges().

***VirtualQuery() и VirtualQueryEx()*** - состояние региона адресов.

**18.Организация «динамической» виртуальной памяти в MS Windows 2000+. Назначение и преимущества по сравнению с кучами ANSI C. Функции Win32 API.**

Куча – 1 Мб по умолчанию. malloc, realloc, calloc… - в ANSI C



Элементы разных структур разного размера лучше размещать в разных кучах

**- для повышения эффективности управления памятью**

Подкачка только элементов структуры, а не всего рабочего множества – **уменьшение рабочего множества процесса**

Куча для каждого потока – без гонки данных **– для повышения производительности многопоточных приложений**

**Быстрое освобождение всей памяти в куче**

**Все потоки равноправны по памяти и не могут получить больше, чем распределено для кучи**

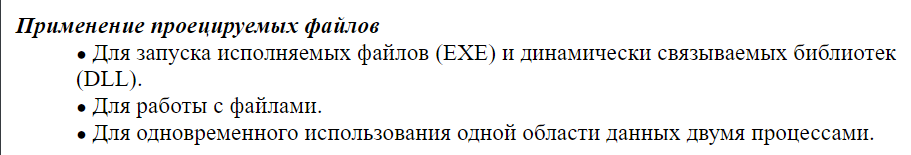
Использование HeapAlloc() с GetProcessHeap() вместо malloc() позволяет нам передавать динамически выделенные объекты между модулями, то есть имеется возможность освобождать память из другого модуля без повреждения кучи – гибкость

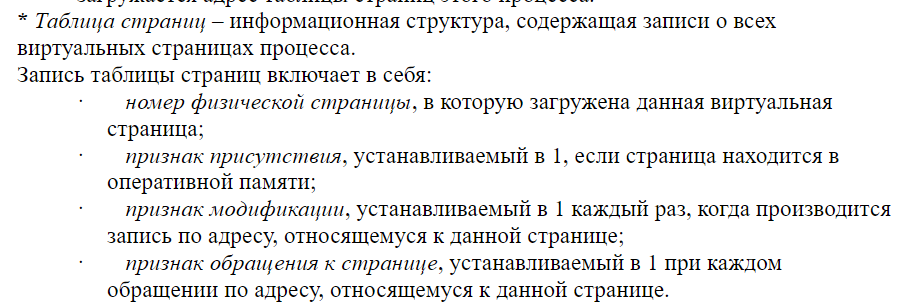
**19.Проецируемые файлы в MS Windows 2000+, назначение и использование. Функции Win32 API.**

***Проецируемый файл*** - это сегмент виртуальной памяти, которому было назначено побайтовое соответствие с сегментом физической памяти. Этот ресурс обычно представляет собой файл, физически присутствующий на диске.

Использование проецируемых файлов стирает для программиста грань между оперативной и дисковой памятью.

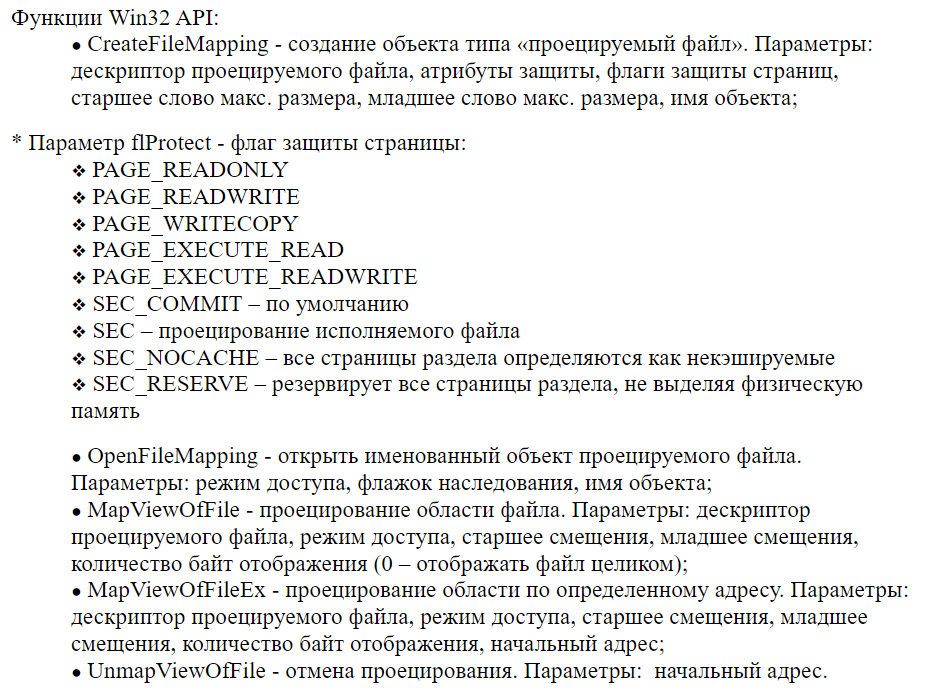
Управляет ими менеджер виртуальной памяти (между кэшем и ОП – процессор). Только на локальном компутере.





Как ВА преобразуется в ФА

Как проецируется файл



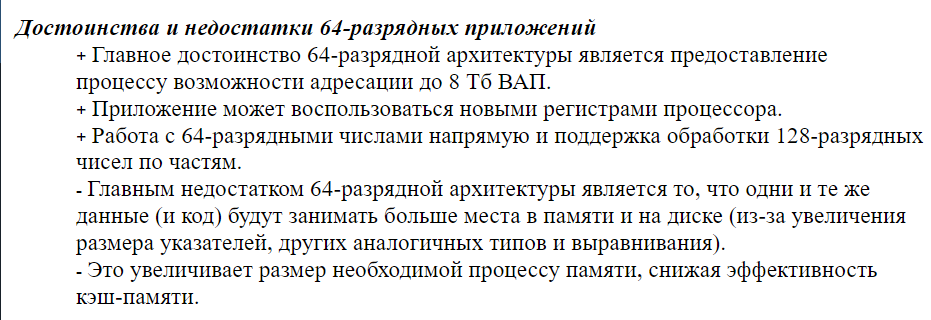
# **20.Особенности управления памятью в 64-разрядных операционных системах MS Windows. Страничное преобразование.**

***Архитектура AMD64***

* 40-разрядные физические адреса, 48-разрядные виртуальные адреса;
* шестнадцать 64-разрядных целочисленных регистров;
* шестнадцать 128-разрядных регистров SSE/SSE2/SSE3;

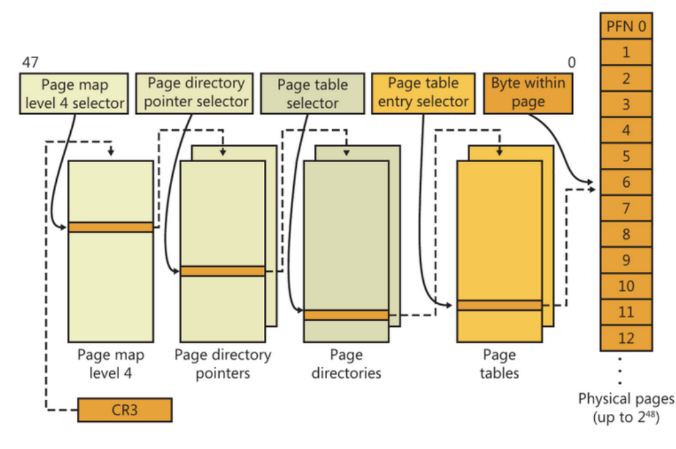
Теоретически 64-разрядные процессы используют 64- битные указатели, так что их максимальное ВАП равно 2^64 = 16 экзабайтам.

Однако в текущих реализациях архитектуры x64 размер виртуальных адресов ограничен 48 битами.



WoW64 (Windows-on-Windows 64-bit)

4 уровня страниц

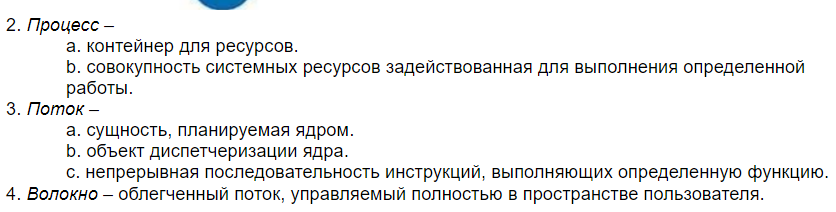


Большие страницы 2 Мб – только 2, 3, 4 уровни

Сверхбольшие 1 Гб – 3, 4

# **21.Объекты управления центральным процессором и объединения ресурсов в MS Windows 2000+. Атрибуты процессов и потоков.**

Задание (с виндовс 2000) – набор процессов с общими квотами и лимитами (так как недоступна иерархия процессов)



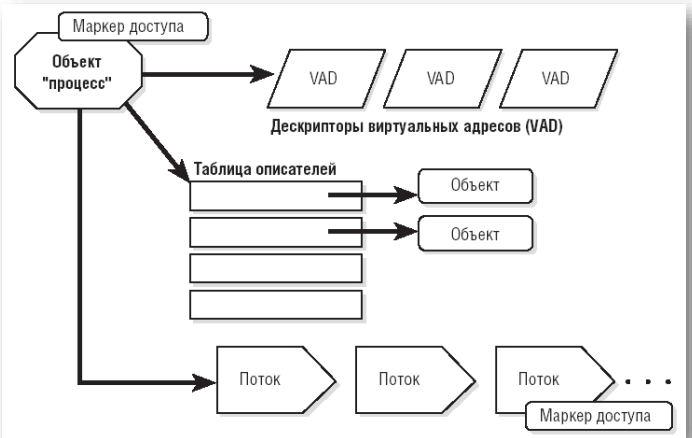
И процесс и поток – объекты

Ресурсы процесса: ВАП, файлы, семафоры и другое shit, хотя бы один поток

Атрибуты процесса: ИД, ВАП, исполняемая программа, таблица открытых дескрипторов объектов, дескриптор безопасности, маркер доступа, базовый приоритет, список потоков, пределы квот, счетчики производительности.

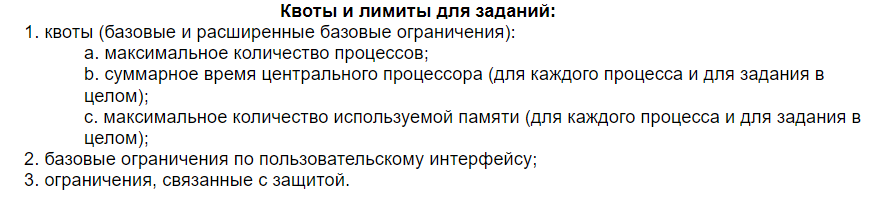
Атрибуты потока: ИД клиента, контекст, стек для ядра и стек для пользовательского режима, TLS, дескриптор безопасности, динамический приоритет, базовый приоритет, процессорная совместимость (маска), номер идеального процессора, номер прошлого процессора, счетчик приостановок, счетчики производительности.

Потоки не имеют своего ВАП.



У потока по умолчанию маркер доступа родителя, но он может получить свой.

Seed – идеальный процессор, +1 при новом потоке.



Волокна (fibers, или облегченные потоки) введены в Windows 2000 для переноса существующих серверных приложений из UNIX.   Единовременно поток будет выполнять код **лишь одного волокна вне зависимости от числа процессорных ядер**.

Управление волокнами:

* Выполняющееся волокно **может уступить управление** другому волокну, указав его адрес при вызове функции SwitchToFiber ().
* Отдельное волокно **может получить свои данные**, назначенные ему функцией CreateFiber (), обратившись к функции GetFiberData ().
* Волокно **может идентифицировать себя** при помощи функции GetCurrentFiber ().
* Функция DeleteFiber ()**уничтожает волокно**и все относящиеся к нему данные.

**22.Общие принципы диспетчеризация (планирования загрузки) в MS Windows 2000+. Классы приоритетов. Относительные приоритеты. Динамическое изменение приоритетов.**

Принципы диспетчеризации в винде 2000+:

1. Вытесняющая многозадачность
2. Следующий поток выбирается по приоритеты
3. Балансировка нагрузки между процессорами

HAL устанавливает продолжительность тика. 10 мс у однопроцессорных x86, 15 мс у многопроцессорных.

1 тик – 3 условные единицы

У польз. версий винды квант потока 2 тика, у серверных 12

Параметр реестра Win32PrioritySeparation – три двухбитных поля в диапазоне

0x00000000 - 0x0000003F

Первые два: длительность интервала,

00 или 11: короткие интервалы на клиентских редакциях Windows и длинные интервалы на Windows Server

01: Longer - длинные интервалы

10: Shorter - короткие интервалы

Вторые два: будет ли активным процессам (foreground) выделяться больше времени, чем фоновым (background)

00 или 11: на клиентских редакциях Windows активным процессам отводится больше времени, на серверных - одинаково

01: Variable - активным процессам отводится больше времени (переменная длительность квантов)

10: Fixed - равное количество квантов и для фоновых, и для активных процессов (фиксированная длительность квантов)

Третьи два: (Ratio) определяет соотношение между количеством квантов для активных процессов и количеством квантов для фоновых процессов (1:1, 2:1 или 3:1), имеет значение только при BB = 01b (Variable):

00: равное количество квантов для фоновых и активных процессов (1:1)

01: соотношение 2:1 (активным процессам отводится в 2 раза больше квантов, чем фоновым)

10 или 11: соотношение 3:1



Фоновый режим – нулевые колонки

Активный – в зависимости от третьих двух бит Win32PrioritySeparation

Классы приоритетов (базовый приоритет):

1. **Real-time (24)** – потоки в этом процессе обязаны немедленно реагировать на события, обеспечивая выполнение критических по времени **задач. Такие потоки вытесняют даже компоненты ОС**. Процессы этого класса нельзя запустить, если пользователь не имеет привилегии **Increase Scheduling Priority**.
2. **High (13)** – потоки в этом процессе тоже должны немедленно реагировать на события, обеспечивая выполнение критических по времени задач. **Task Manager**
3. **Above Normal (10)** – класс приоритета, промежуточный между normal и high
4. **Normal (8)** – потоки в этом процессе не предъявляют особых требований к выделению им процессорного времени **(99% приложений)**.
5. **Below Normal (6)** – класс приоритета, промежуточный между normal и idle.
6. **Idle (4)** – потоки в этом процессе выполняются, когда система не занята другой работой. **Фоновые утилиты, экранные заставки…**

Относительный приоритет:

1. **Idle** – поток выполняется с приоритетом 16 в классе real-time и с приоритетом 1 в других класса.
2. **Lowest** –поток выполняется с приоритетом на два уровня **ниже**обычного для данного класса приоритетов.
3. **Below normal** – поток выполняется с приоритетом на один уровень ниже обычного для данного класса.
4. **Normal** – поток выполняется с обычным приоритетом процесса для данного класса.
5. **Above normal** – поток выполняется с приоритетом на один уровень **выше**обычного для данного класса.
6. **Highest** – Поток выполняется с приоритетом на два уровня выше обычного для данного класса
7. **Time-critical** – поток выполняется с приоритетом 31 в классе real-time и с приоритетом 15 в других классах

Базовый уровень приоритета потока – уровень приоритета, получаемый комбинацией относительного приоритета потока и класса приоритета процесса, которому принадлежит данный поток. 1-15 может измениться, 15+ не может.

Принципы адаптивного планирования:

Приоритет потоков, которые перешли в состояние ожидания, не использовав полностью выделенный им квант, повышается.

Если поток полностью исчерпал свой квант, то его приоритет понижается на некоторую величину.

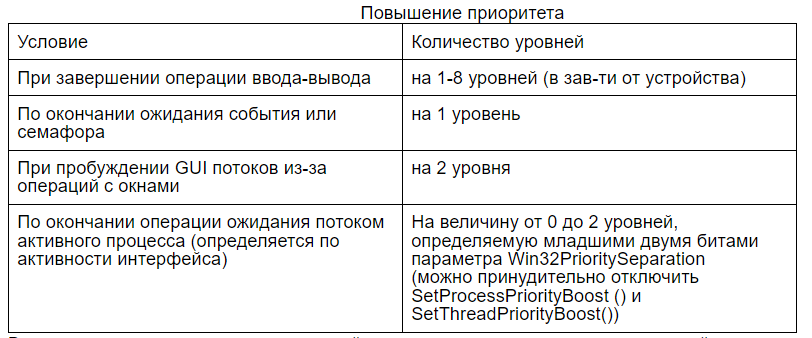
Приоритет не изменяется, если поток вытеснен более приоритетным потоком.

Повышение приоритета для «голодающих» потоков.

Голодание потоков – проблема полной недоступности ресурсов определенному готовому потоку в связи с тем, что процессор занимают потоки с более высоким приоритетом.

Обнаружив такой поток, не выполняемый на протяжении уже 3-4 секунд, ОС выполняет:

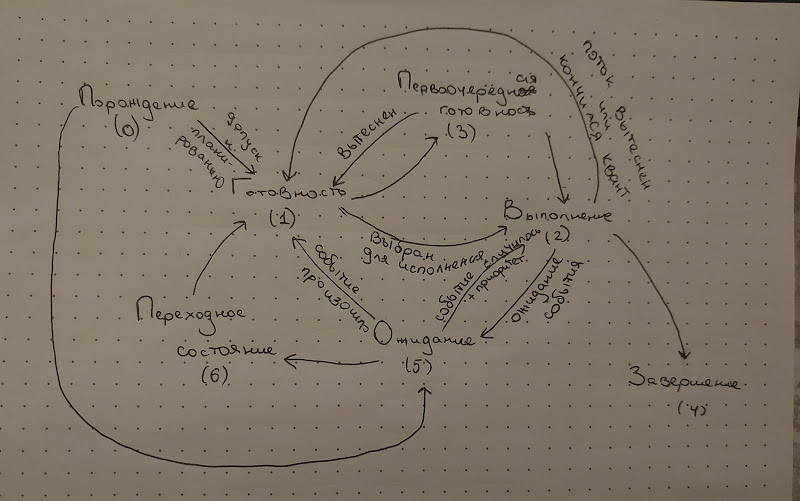
1. поднимает приоритет «голодающего» потока до 15;
2. выделяет ему удвоенный квант времени;
3. по истечении удвоенного кванта времени потоку немедленно возвращается его базовый приоритет.



# **23.Граф состояний потоков в MS Windows 2000+. Поток простоя. Принципы адаптивного планирования.**

Своя очередь для каждого уровня приоритета (0-31). Сводка готовности – маска из 32 бит, указывает на присутствие 1 или более готовых потока в очереди с данными уровнем приоритета.

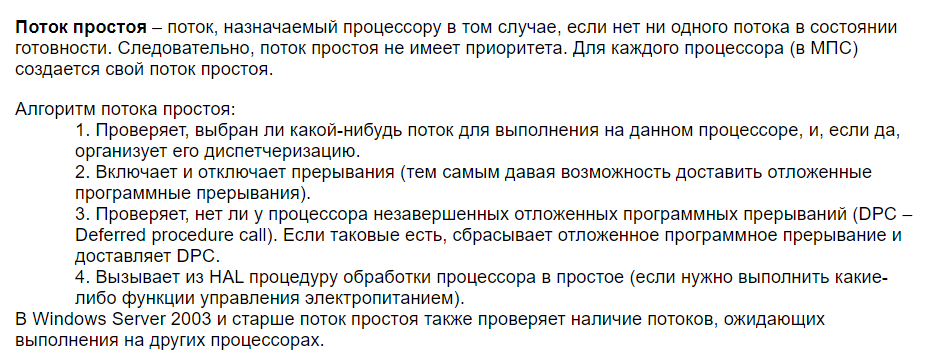
Первоочередная готовность, выполнение, ожидание, переходное состояние, завершение.



Условия освобождения потоком процессора:

1. Исчерпан квант и в очереди есть потоки с равным уровнем приоритета.
2. Появляется более приоритетный готовый поток.
3. Поток снижает свой приоритет, в очереди оказывается более приоритетный поток.
4. Поток сам добровольно переходит в очередь готовых (Sleep (0), SwitchToThread())
5. Поток добровольно блокируется, уходя в состояние ожидания (Sleep(n),  WaitForSingleObject(), SuspendThread(), синхронный ввод/вывод и тд)
6. Поток добровольно исключает текущий процессор из маски совместиости (привязки) (для МПС).
7. Поток завершает свое выполнение.

**Поток простоя** – поток, назначаемый процессору в том случае, если нет ни одного потока в состоянии готовности. Следовательно, поток простоя не имеет приоритета. Для каждого процессора (в МПС) создается свой поток простоя.



# **24.Граф состояний потоков в MS Windows 2000+. Особенности планирования в многопроцессорных системах. Особенности планирования в ОС MS Windows Vista и Server 2008.**

Графы из 23 вопроса

Планирование в многопроцессорных системах:

1. Если есть простаивающие процессоры: пытаемся подключить к простаивающему, если их несколько – выбираем по табличке, после выбора переводим поток в standby
2. Если нет простаивающих – ставим на идеальный, если его нет в маске привязки – выбираем с наибольшим номером. Если идеальный есть в маске, но занят – сравниваем приоритеты. Если приоритет нового потока выше – вытесняет и выполняется. Иначе – он следующий в очереди на этот процессор.

Табличка выбора:

### Windows 2000

1. **При наличии нескольких простаивающих процессоров**:
   * Предпочтение отдается сначала идеальному процессору, затем предыдущему, и наконец текущему процессору.
2. **Если все эти процессоры заняты**:
   * Выбирается первый простаивающий процессор, соответствующий маске привязки, в порядке убывания их номеров.

### Windows XP и Windows 2003+

1. **Выбор процессора из разрешенных маской привязки**:
   * Если текущий процессор разрешен маской привязки, поток планируется на текущем процессоре.
2. **Алгоритм выбора при отсутствии подходящего текущего процессора**:
   * **NUMA**: Если в узле идеального процессора есть простаивающие процессоры, выбрать из них.
   * **HT (гиперпоточность)**: Если есть физический процессор с всеми простаивающими логическими процессорами, выбрать из них.
   * Если текущий процессор в оставшемся списке простаивающих, выбрать его.
   * Если нет, выбрать простаивающий логический процессор на физическом процессоре с идеальным процессором.
   * Исключить процессоры в состоянии сна, если это не оставляет пустой список.
3. **Заключительный выбор**:
   * Поток подключается к процессору с наименьшим номером из оставшегося списка.

Выбор потока для процессора:

1. Windows 2000 и XP:  
   Из очереди готовых выбирается поток с наивысшим приоритетом, для которого выполняется одно из дополнительных условий:
2. поток уже выполнялся в прошлый раз на данном процессоре;
3. данный процессор должен быть идеальным для этого потока;
4. поток провел в состоянии Ready более трех тактов системного таймера;
5. поток имеет приоритет не менее 24.
6. Windows 2003+:  
   Т.к. у каждого процессора собственный список потоков в состоянии готовности, то по окончании выполнения текущего потока процессор проверяет эту очередь.  
   Если очередь оказывается пуста, то подключается поток простоя, который сканирует очереди готовых потоков на других процессорах и ищут поток, который можно было бы забрать на данный процессор.  
   *Пометка: в NUMA-системах (Non-Uniform Memory Architecture) поток простоя проверяет процессоры сначала в своем узле, а потом в других узлах.*

**Windows Vista и Server 2008:**

Каждые 10-15 мс все версии виндовс выполняют подрограмму прерывания интервального таймера, чтобы обновить статистику загрузки процессора потоком.

Проблема – считается, что поток выполнялся все время, хотя было прерывание, и может быть снят с выполнения.

Поэтому в этих версиях винды введи **счетчик циклов процессора** (в современных процессорах для этого есть регистр). Просчитывается, сколько прерываний может быть во время выполнения потока, и они не засчитываются в его время работы.

**25. Функции Win32 API создания и завершения процессов и потоков, управления потоками. Функции Win32 API создания и завершения потоков**

* *CreateProcess()* - создание процесса
* *CreateThread()* - создание потока

Обязательный параметр - адрес на структуру SECURITY\_ATTRIBUTES, Большинство объектов Windows именованные.

**Создание процесса:**

**Вызов** CreateProcess (), создание объекта ядра процесс (счетчик пользователей = 1), создание ВАП и загрузка кода, создание объекта поток (счетчик 1), первичный поток начинает выполнять main. Вернется TRUE если все ок.

***Параметры CreateProcess***(Господи…)

* Параметры pszApplicationName и pszCommandLine. Эти параметры определяют имя исполняемого файла, которым будет пользоваться новый процесс, и командную строку, передаваемую этому процессу.
* Параметры psaProcess, psaThread. Параметры psaProcess и psaThread позволяют определить нужные атрибуты защиты для объектов «процесс» и «поток» соответственно. Если передать NULL, то система будет использовать дескрипторы защиты по умолчанию.
* Параметр blnheritHandles позволяет разрешить (TRUE) дочернему процессу наследовать дескрипторы объектов родительского процесса с правами доступа как у оригиналов.
* Параметр fdwCreate определяет класс приоритета процесса и флаги , влияющие на то, как именно создается новый процесс. Флаги комбинируются булевым оператором OR.
* Параметр pvEnvironment указывает на блок памяти, хранящий строки переменных окружения, которыми будет пользоваться новый процесс. Если его значение – NULL, то дочерний процесс наследует строки переменных окружения от родительского процесса.
* Параметр pszCurDir позволяет родительскому процессу установить текущие диск и каталог для дочернего процесса. Если его значение – NULL, рабочий каталог нового процесса будет тем же, что и у приложения, его породившего.
* Параметр psiStartlnfo указывает на структуру STARTUPINFO. Элементы структуры STARTUPINFO используются Windows-функциями при создании нового процесса.
* Параметр ppiProclnfo указывает на структуру типа PROCESS\_INFORMATION, которую функция CreateProcess инициализирует в ходе своего выполнения.

***Параметр fdwCreate 🙁***

* Флаги DEBUG\_PROCESS и DEBUG\_ONLY\_THIS\_PROCESS позволяют родительскому процессу проводить отладку дочернего процесса, а также всех процессов, которые последним могут быть порождены (DEBUG\_PROCESS).
* Флаг CREATE\_SUSPENDED позволяет создать процесс и в то же время приостановить его первичный поток. Это позволяет родительскому процессу модифицировать содержимое памяти в адресном пространстве дочернего, изменить приоритет и т.д. Для разрешения выполнения приостановленного потока следует использовать функцию ResumeThread ().
* Флаг DETACHED\_PROCESS блокирует доступ дочернего процесса к консольному окну родительского процесса и сообщает системе, что вывод следует перенаправить в новое консольное окно.
* Флаг CREATE\_NEW\_CONSOLE приводит к созданию нового консольного окна для нового процесса.
* Флаг CREATE\_NO\_WINDOW позволяет создать процесс без пользовательского интерфейса.
* Флаг CREATE\_BREAKAWAY\_FROM\_JOB позволяет процессу, включенному в задание, создать новый процесс, отделенный от этого задания.

**Завершение:**

* входная функция первичного потока возвращает управление (рекомендуемый способ);
* один из потоков процесса вызывает функцию ExitProcess () (нежелательный способ);
* поток другого процесса вызывает функцию TerminateProcess () (тоже нежелательно);
* все потоки процесса сами «умирают» (большая редкость).

Явный вызов ExitProcess () и TerminateProcess () – распространенная ошибка, которая мешает правильной очистке ресурсов.

ExitProcess - завершает процесс с кодом завершения fuExitCode (один из потоков ее вызывает)

TerminateProcess () - завершить произвольный процесс с дескриптором hProcess с кодом завершения fuExitCode (асинхронная)

Чтобы узнать момент завершения процесса - WaitForSingleObject ()

***Управление динамическими приоритетами потоков процесса***

BOOL SetProcessPriorityBoost(

HANDLE hProcess, // дескриптор процесса

BOOL DisablePriorityBoost // состояние форсированного приоритета

);

BOOL GetProcessPriorityBoost(

HANDLE hProcess, // дескриптор процесса

PBOOL pDisablePriorityBoost // состояние форсированного приоритета

);

***Функция создания потока***

HANDLE CreateThread (

PSECURITY\_ATTRIBUTES psa, // атрибуты защиты потока

SIZE\_T cbStack, // размер начального стека

PTHREAD\_START\_ROUTINE pfnStartAddr, // адрес функции потока

PVOID pvParam, // аргументы для вызова функции потока

DWORD dwCreate, // параметры создания потока

PDWORD pdwThreadID // идентификатор потока

);

***Параметры CreateThread***

* Параметр psa позволяет задать атрибуты защиты для создаваемого потока. Если передать NULL, то система будет использовать дескриптор защиты по умолчанию. Чтобы дочерние процессы могли наследовать дескриптор этого потока, определите структуру SECURI TY\_ATTRIBUTES и инициализируйте ее элемент hlnheritHandle значением TRUE.
* Параметр cbStack определяет, какую часть виртуального адресного пространства процесса поток сможет использовать под свой стек.
* Параметр pfnStartAddr определяет адрес функции потока, с которой должен будет начать работу создаваемый поток, а параметр pvParam идентичен параметру рvРаrаm функции потока.
* Параметр fdwCreate принимает одно из двух значений: 0 (исполнение потока начинается немедленно) или CREATE\_SUSPENDED (поток приостанавливается).
* Параметр pdwThreadlD – это адрес для возврата функцией CreateThread () идентификатора нового потока.

***Функция CreateRemoteThread***

Функция *CreateRemoteThread*() создает поток, который запускается в виртуальном адресном пространстве другого процесса с дескриптором hProcess.

HANDLE CreateRemoteThread (

HANDLE hProcess, // дескриптор процесса

LPSECURITY\_ATTRIBUTES lpThreadAttributes, // дескриптор защиты

SIZE\_T dwStackSize,// размер начального стека

LPTHREAD\_START\_ROUTINE lpStartAddress, // адрес функции потока

LPVOID lpParameter, // аргументы для вызова функции потока

DWORD dwCreationFlags, // параметры создания потока

LPDWORD lpThreadId // идентификатор потока

);

***Установка приоритета потока***

BOOL SetThreadPriority(

HANDLE hThread, // дескриптор потока

int nPriority // уровень приоритета потока

);

Функция SetThreadPriority () дает возможность установки базового уровня приоритета потока относительно класса приоритета его процесса.

***Относительные приоритеты потоков***THREAD\_PRIORITY\_ABOVE\_NORMAL  THREAD\_PRIORITY\_BELOW\_NORMAL  THREAD\_PRIORITY\_HIGHEST  THREAD\_PRIORITY\_IDLE  THREAD\_PRIORITY\_LOWEST  THREAD\_PRIORITY\_NORMAL  THREAD\_PRIORITY\_TIME\_CRITICAL

***Завершение потока***

Поток можно завершить четырьмя способами:

* функция потока возвращает управление путем вызова оператора return (рекомендуемый способ);
* поток самоуничтожается вызовом функции ExitThread () (нежелательный способ);
* один из потоков данного или стороннего процесса вызывает функцию TerminateThread () (нежелательный способ);
* завершается процесс, содержащий данный поток (тоже нежелательно).

Принудительное завершение потока с помощью функций ExitThread () и TerminateThread () нежелательно, т.к. процесс продолжает работать, но при этом весьма вероятна утечка памяти или других ресурсов, которые были запрошены закрытым потоком.

***Функция ExitThread***

Для принудительного завершения потока в рамках процесса-владельца

необходимо вызвать функцию: ExitThread (DWORD dwExitCоde);

Через параметр dwExitCode Вы передаете код завершения потока.

***Функция TerminateThread***

Вызов этой функции позволяет завершить практически любой поток:

TerminateThread(HANDLE hThread, DWORD dwExitCode);

В параметр dwExitCode помещается код завершения потока.

После вызова этой функции поток с дескриптором hThread завершается, а счетчик пользователей, соответствующего ему экземпляра объекта ядра,

уменьшится на 1.

***Управление динамическими приоритетами потока***

BOOL SetThreadPriorityBoost(

HANDLE hThread, // дескриптор потока

BOOL DisablePriorityBoost // состояние форсирования приоритета

);

BOOL GetThreadPriorityBoost(

HANDLE hThread, // дескриптор потока

PBOOL pDisablePriorityBoost // состояние форсажа приоритета );

**26.Критические секции и состязания. Семафоры и мьютексы. Задача о читателях и писателях. Взаимная блокировка (тупики).**

Критические данные – общие переменные, которые изменяются двумя или более потоками в состязательном режиме.

Неэффективный способ решения – запрет прерываний на время нахождения в КС.

***Условия взаимного исключения гонок со стороны процессов***

* В каждый момент времени в критических секциях, связанными с одними критическими данными, не должно находиться более одного процесса.
* Процесс вне критической секции не может блокировать другие процессы.
* Должна быть невозможна ситуация, когда процесс вечно ждет попадания в критическую секцию.
* В программе не должно быть предположений о скорости ее выполнения или количестве процессоров.

Семафор – кол-во одновременно находящихся потоков, +1 -1 атомарные операции, -1 = P(S) когда освободился, +1 = V(S) когда зашел поток.

*CreateSemaphore* с параметрами: атрибуты защиты, начальное значение счетчика семафора, максимальное значение счетчика, имя объекта

*OpenSemaphore* с параметрами: режим доступа, флаг наследования, имя объект

* SEMAPHORE\_ALL\_ACCESS – полный доступ (разрешены все возможные виды доступа)
* SEMAPHORE\_MODIFY\_STATE – определяет возможность изменения значение счетчика семафора функцией ReleaseSemaphore()
* SYNCHRONIZE – позволяет использовать дескриптор объекта семафора в любой из функций ожидания

Для прохождения через семафор (захвата семафора) необходимо использовать функции WaitForSingleObject () или WaitForMultipleObject ().

ReleaseSemaphore () +1 : дескриптор семафора, значение инкремента, адрес переменной для записи предыдущего значения семафора.

Мьютекс – имеет одного владельца.

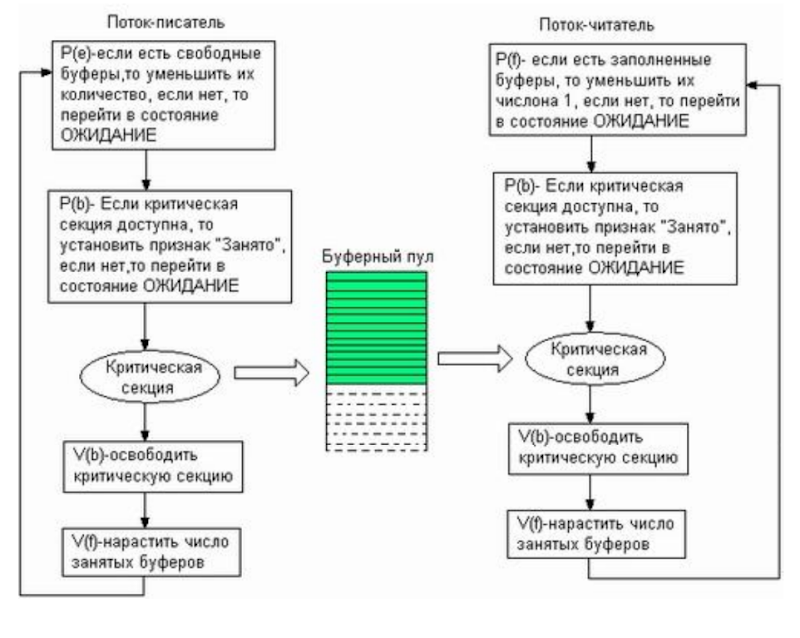
CreateMutex - создание. Параметры: атрибуты защиты, начальное состояние, имя объекта

OpenMutex- открытие. Параметры: режим доступа, флаг наследования, имя объекта

* MUTEX \_ALL\_ACCESS – полный доступ (разрешены все возможные виды доступа)
* MUTEX \_MODIFY\_STATE – определяет возможность изменения значение счетчика мьютекса функцией ReleaseMutex ()
* SYNCHRONIZE – позволяет использовать дескриптор объекта мьютекса в любой из функций ожидания

Задача о читателях и писателях:

Пусть буферный пул состоит из N буферов, каждый из которых может содержать одну запись.



Тупик/deadlock - есколько процессов находятся в состоянии бесконечного ожидания ресурсов, захваченных этими процессами.

***Условия возникновения тупиков***

* потоки требуют предоставления им права монопольного управления ресурсами, которые им выделяются (условие взаимоисключения);
* потоки удерживают за собой ресурсы, уже выделенные им, ожидая в тоже время выделения дополнительных ресурсов (условие ожидания ресурсов);
* ресурсы нельзя отобрать у потоков, удерживающих их, пока эти ресурсы не будут использованы для завершения работы (условие не перераспределяемости);
* существует кольцевая цепь потоков, в которой каждый поток удерживает за собой один или более ресурсов, требующихся следующему потоку цепи (условие кругового ожидания).

**27.Синхронизация потоков с использованием объектов ядра MS Windows 2000+. Таймеры ожидания, мьютексы. Функции Win32 API.**

Простейшей формой коммуникации потоков является синхронизация.

Занят/свободен

Процессы, потоки, задания, файлы, консольный ввод, уведомления об изменении файлов, события, таймеры, семафоры, мьютексы

Объект-поток находится в состоянии «занят» все время существования, но устанавливается системой в состояние «свободен», когда его выполнение завершается.

Аналогично, ядро устанавливает процесс в состояние «свободен», когда завершился его последний поток.

В противоположность этому, объект – таймер «срабатывает» через заданное время (по истечении этого времени ядро устанавливает объект – таймер в состояние «свободен»).

Потоки спят, пока ожидаемые ими объекты заняты (флажок опущен). Как только объект освободился (флажок поднят), спящий поток замечает это, просыпается просыпается и возобновляет выполнение.

Wait-функции позволяют потоку в любой момент приостановиться и ждать освобождения одного или нескольких объектов ядра.

***Таймеры ожидания***

Таймеры ожидания (waitable timers) – это объекты ядра, которые самостоятельно переходят в свободное состояние в определенное время или через регулярные промежутки времени.

Объекты «таймеры ожидания» всегда создаются в занятом состоянии («0»).

***Создание и открытие таймера ожидания***

CreateWaitableTimer - создание. Параметры: атрибуты защиты, тип сброса таймера(TRUE – ручной), имя объекта

OpenWaitableTimer - открытие. Параметры: режим доступа, флаг наследования, имя объекта.

* TIMER\_ALL\_ACCESS – полный доступ (разрешены все возможные виды доступа)
* TIMER\_MODIFY\_STATE – определяет возможность изменения состояния таймера функциями SetWaitableTimer () и CancelWaitableTimer ()
* SYNCHRONIZE – позволяет использовать дескриптор объекта таймера в любой из функций ожидания

***Отмена действия таймера***

Для отмены действия таймера ожидания следует использовать функцию CancelWaitableTimer (), после ее применения таймер не сработает до следующего вызова SetWaitableTimer ().

Для переустановки текущих параметров таймера ожидания нет необходимости вызывать CancelWaitableTimer (), каждый вызов SetWaitableTimer () автоматически отменяет предыдущие настройки перед установкой новых.

**28. События**

Событие – самая примитивная разновидность объектов синхронизации, которая просто уведомляют об окончании какой-либо операции.

События содержат счетчик числа пользователей (как и все объекты ядра) и две булевы переменные: одна сообщает тип данного объекта-события, другая – его состояние (свободен или занят).

Объекты-события бывают двух типов: со сбросом вручную (manual-reset events) и с автосбросом (auto-reset events).

События с «ручным» сбросом нужно применять, если событие ждут несколько потоков. Только этот тип события позволяет выполнить синхронизацию нескольких потоков от одного события.

***Применение «событий»***

Объекты-события обычно используют в том случае, когда какой-то поток выполняет инициализацию, а затем сигнализирует другому потоку, что тот может продолжить работу.

Инициализирующий поток переводит объект «событие» в занятое состояние и приступает к своим операциям.

Закончив, он сбрасывает событие в свободное состояние. Тогда другой поток, который ждал перехода события в свободное состояние, пробуждается и вновь становится планируемым.

***Создание объекта типа «событие»***

Функция *CreateEvent*с параметрами: атрибуты защиты, тип сброса, начальное состояние, имя объекта.

Параметр тип сброса сообщает системе, хотите Вы создать событие со сбросом вручную (TRUE) или с автосбросом (FALSE).  Параметр начальное состояние определяет начальное состояние события – свободное (TRUE) или занятое (FALSE).

***Совместное использование объекта «события» процессами***

Если объект «событие» успешно создан, то CreateEvent () возвращает дескриптор объекта специфичный для конкретного процесса.

Потоки из других процессов могут получить доступ к этому объекту:

1) наследованием описателя с применением функции DuplicateHandle () (универсальный способ);

2) вызовом OpenEvent () c передачей в параметре lpName имени, совпадающего с указанным в аналогичном параметре функции CreateEvent().

***Открытие объекта типа «событие»***

Функция OpenEvent с параметрами: режим доступа, флаг наследования, имя объекта.

Функция OpenEvent () возвращает дескриптор существующего именованного объекта события. Вызывающий процесс может использовать возвращаемый данной функцией дескриптор в качестве аргумента любой функции, использующей дескриптор объекта события, при условии соблюдения ограничений, налагаемых значением аргумента режим доступа.

***Режимы доступа к «событиям»***

* EVENT\_ALL\_ACCESS – полный доступ (разрешены все возможные виды доступа)
* EVENT\_MODIFY\_STATE – обеспечивает возможность использования дескриптора объекта события в функциях SetEvent () и ResetEvent (), изменяющих состояние объекта события
* SYNCHRONIZE – позволяет использовать дескриптор объекта события в любой из функций ожидания

***Управление «событиями»***

* Перевод события в свободное состояние: BOOL SetEvent (HANDLE hEvent);
* Перевод события в занятое состояние: BOOL ResetEvent (HANDLE hEvent);
* Освобождение события и перевод его обратно в занятое состояние: BOOL PulseEvent (HANDLE hEvent);

Функция *PulseEvent()* устанавливает событие и тут же переводит его обратно в сброшенное состояние, это равнозначно последовательному вызову *SetEvent*() и *ResetEvent*().

Если *PulseEvent*() вызывается для события со сбросом вручную, то все потоки, ожидающие этот объект, получают управление.

При вызове *PulseEvent*() для события с автосбросом пробуждается только один из ждущих потоков. А если ни один из потоков не ждет объект-событие, вызов функции не дает никакого эффекта.

**29. Синхронизация потоков с использованием объектов ядра MS Windows 2000+. Организация асинхронной работы с файлами в Win32 API**

***Асинхронный ввод-вывод   + перекрывающийся***

Для организации асинхронной работы с файлами необходимо при вызове функции CreateFile () установить флаг FILE\_FLAG\_OVERLAPPED в параметре dwFlagsAndAttributes.

После этого функции ReadFile () и WriteFile () будут работать асинхронно, т.е. только запускать операции ввода вывода и не ожидать их завершения.  Структура данных OVERLAPPED, на которую указывает параметр lpOverlapped, должна оставаться допустимой для длительной операции чтения. Она не должна быть переменной, которая может выйти из области действия, пока происходит операция чтения файла.

***Функции файлового ввода-вывода и их параметры***

Функции: ReadFile, WriteFile

Параметры:

* hFile – дескриптор файла;
* lpBuffer – адрес буфера, в который будет производиться чтение/запись;
* nNumberOfBytes… – количество байт, которые необходимо прочитать/записать;
* lpNumberOfBytes… – адрес переменной, в которой будет размещено количество реально прочитанных/записанных байт;
* lpOverlapped – указатель на структуру OVERLAPPED, управляющую асинхронным вводом выводом.

***Структура перекрывающегося асинхронного в/в***

typedef struct \_OVERLAPPED {

DWORD Internal; //Используется операционной системой. //Хранит статус завершения операции.

DWORD InternalHigh; //Используется ОС. //Хранит количество переданных байт.

DWORD Offset; //Позиция в файле, начиная с которой необходимо //операцию чтения (записи).

DWORD OffsetHigh;//Количество байт для передачи.

HANDLE hEvent; //Описатель события, которое произойдет при завершении //операции чтения (записи).

} OVERLAPPED;

**Варианты организации асинхронного в/в**

1. Создать событие и передать его дескриптор в ReadFile () или WriteFile () в качестве элемента hEvent структуры OVERLAPPED.
2. Объект – сам файл, не поддерживается перекрывающийся в/в
3. С использованием ReadFileEx () и WriteFileEx (). В качестве дополнительного параметра в эти функции передается адрес функции завершения (APC –asynchronous procedure call), которая будет вызываться всякий раз при завершении операции ввода/вывода.

***Функция Sleep***

Функция приостанавливает поток на dwMilliseconds миллисекунд.  Особенности выполнения функции Sleep ():

* поток добровольно отказывается от остатка кванта времени;
* система приостанавливает поток на период, примерно равный заданному;
* Вы можете вообще запретить планировать поток, передав в качестве dwMilliseconds значение INFINITE (-1);
* Вы можете вызвать Sleep и передать в качестве dwMilliseconds ноль. В этом случае поток будет вытеснен с процессора и помещен в очередь ожидания. Однако поток снова будет запущен, если нет других готовых потоков с тем же приоритетом.

***Функция SleepEx***

Функция приостанавливает выполнения потока до наступления события ввода/вывода или на время.

Отличия выполнения от функции Sleep ():

* если параметр bAlertable = FALSE, то функция ведет себя аналогично Sleep ();
* если параметр bAlertable = TRUE, и этот поток переходит в ожидание оповещения и может продолжить выполнение после срабатывания вызова APC или истечение времени блокировки;
* функция возвращает значение WAIT\_IO\_COMPLETION, если завершение произошло в результате срабатывания вызова APC.

**30.Атомарные операции и lockless программирование. Критические секции в Win32 API.**

*Состояние гонки*—состояние системы, при котором результат выполнения операций зависит от того, в какой последовательности будут выполняться отдельные процессы в этой системе, но управлять этой последовательностью нет возможности.

***Lockless программирование***

*Lockless программирование* – разработка неблокирующих многопоточных приложений. Отказ от использования блокирующих примитивов типа мьютексов и даже критических секций для доступа к разделяемым данным.

Достоинство – повышенная производительность многопоточных приложений на многоядерных процессорах.

***Виды атомарных операций***

* Все инструкции вида Операция Регистр-Регистр можно считать атомарными, так как регистры за пределами вычислительного процессорного ядра не видны.
* Загрузка данных из памяти по выровненному адресу в регистр общего назначения.
* Сохранение данных из регистра общего назначения в память по выровненному адресу.
* Специальные операции для атомарной работы (например, cmpxchg).
* Многие команды вида Чтение-Модификация-Запись могут быть сделаны искусственно атомарными с помощью операции блокировки шины (префикс lock).

***Реализация атомарных операций в Windows 2000+***

* Для увеличения значения целочисленных переменных – InterlockedIncrement, InterlockedIncrement64.
* Для уменьшения значения целочисленных переменных – InterlockedDecrement, InterlockedDecrement64.
* Для изменения значений целочисленных переменных – InterlockedExchange, InterlockedExchange64, InterlockedExchangeAdd, InterlockedExchangePointer.
* Для изменения значений целочисленных переменных со сравнением – InterlockedCompareExchange, InterlockedCompareExchangePointer.

Критическая секция

***Функции Win32 API для работы с критическими секциями***

* InitializeCriticalSection – создание КС;
* DeleteCriticalSection – освобождение ресурсов КС;
* EnterCriticalSection – вход в КС;
* LeaveCriticalSection – освобождение КС;
* TryEnterCriticalSection – проверка КС.

Эти функции имеют в качестве параметра предварительно проинициализированную структуру типа CRITICAL\_SECTION.

***Структура типа CRITICAL\_SECTION***

typedef struct \_RTL\_CRITICAL\_SECTION {

PRTL\_CRITICAL\_SECTION\_DEBUG DebugInfo; // исп. ОС

LONG LockCount; // счетчик блокировок

LONG RecursionCount; // счетчик повторного захвата

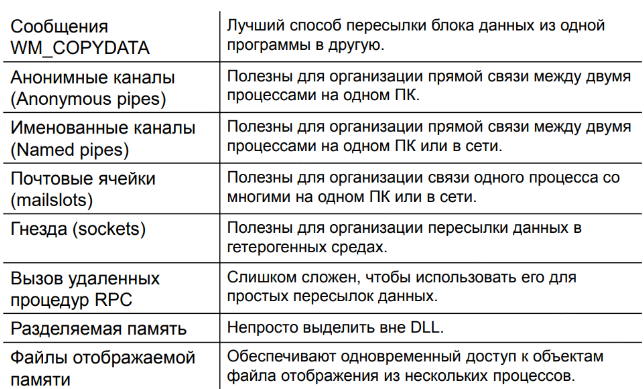
HANDLE OwningThread; // ID потока, владеющего секцией

HANDLE LockSemaphore; // дескриптор события

ULONG\_PTR SpinCount; // кол-во холостых циклов перед вызовом ядра

} RTL\_CRITICAL\_SECTION, \*PRTL\_CRITICAL\_SECTION;

**31.Передача информации в MS Windows 2000+. Анонимные каналы. Функции Win32 API.**



Что такое канал. В канале FIFO

Анонимные каналы

Главная цель – служить каналом между родительским и дочерним процессом или между дочерними процессами.

Функция создания канала CreatePipe с параметрами: дескриптор конца чтения канала, дескриптор конца записи канала, атрибуты защиты, размер буфера канала (0 – по умолчанию)

Функция ReadFile () – чтение из канала

Функция WriteFile () – запись в канал

***Пример использования анонимного канала***

1. Создаем анонимный канал с помощью CreatePipe ().
2. Подготавливаем строку параметров для дочернего процесса.
3. Вызываем CreateProcess (), чтобы загрузить дочернее приложение.  Закрываем дескриптор записи канала. Это необходимо, так для родительского процесса этот дескриптор не нужен, а канал не будет работать, если открыть более чем один дескриптор записи.
4. Теперь вы можете читать данные с помощью ReadFile (). Вы должны последовательно вызывать ReadFile (), пока она не возвратит ноль, что будет означать, что больше данных нет.
5. После окончания работы закроем дескриптор чтения канала.

С помощью функции PeekNamedPipe () процесс-читатель может также определить, имеются ли в канале данные без их удаления.

**32.Передача информации в MS Windows 2000+. Именованные каналы. Функции Win32 API.**

Named Pipe File System является виртуальной файловой системой, которая управляет именованными каналами.

Каналы named pipes относятся к классу файловых объектов API Win32.

* Именованный канал может быть установлен между процессами разных компьютеров, объединенных сетью.
* Именованный канал может быть однонаправленным или двунаправленным (дуплексным).
* Именованный канал поддерживает асинхронный обмен.

Формат имени канала: \\.\pipe\[path]pipename

Именованный канал создается с помощью функции *CreateNamedPipe*

Параметры создания канала:

* lpName – имя именованного канала;
* dwOpenMode – направление передачи данных (PIPE\_ACCESS\_DUPLEX, PIPE\_ACCESS\_INBOUND, PIPE\_ACCESS\_OUTBOUND);
* dwPipeMode – режим передачи данных;
* nMaxInstances – максимальное количество каналов с данным именем, которые могут открыть клиенты (обычно от 1 до 255);
* nOutBufferSize и nInBufferSize – размер буферов на отправку и прием (со стороны сервера), 0 – размер по умолчанию;
* nDefaultTimeout – время ожидания по умолчанию для функции WaitNamedPipe ();
* lpSecurityAttributes – указатель на структуру с атрибутами защиты создаваемого объекта.

***Режимы передачи***

* PIPE\_TYPE\_BYTE, PIPE\_TYPE\_MESSAGE – данные записываются в канал как поток байт или как сообщения
* PIPE\_READMODE\_BYTE, PIPE\_READMODE\_MESSAGE – данные считываются из канала как поток байт или как сообщения
* PIPE\_WAIT, PIPE\_NOWAIT – обмен происходит в блокирующем или неблокирующем режиме

Описать клиент, сервер

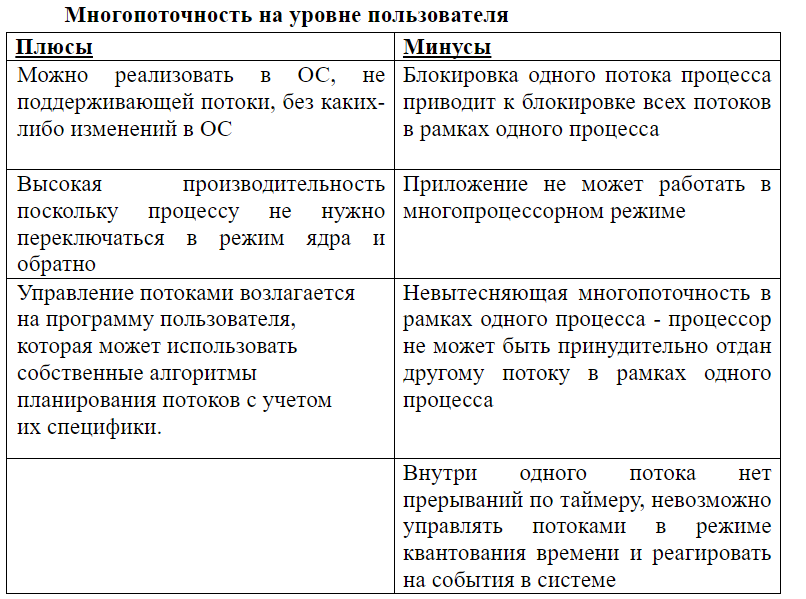
Можно создать несколько экземпляров канала (один сервер на несколько клиентов)

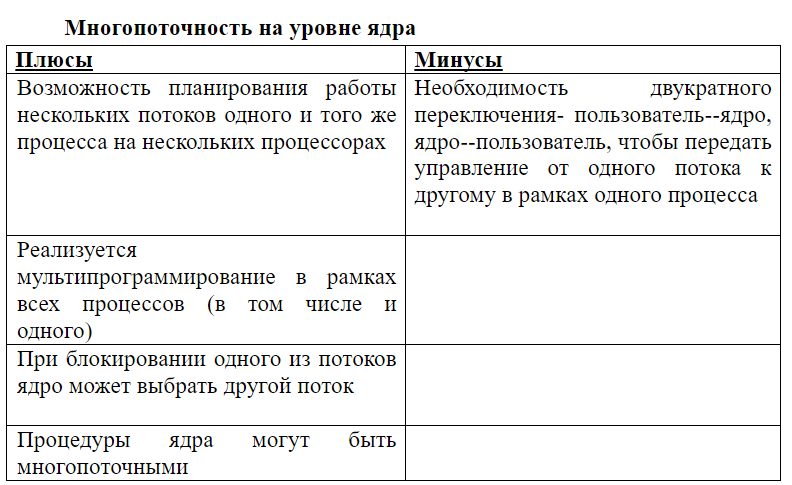
WaitNamedPipe () – гарантированное соединение с клиентом. Параметры - имя канала, интервал ожидания

**1. Определение ОС. Функции ОС. Процессы и потоки. Виды многопоточности**

ОС, Критерий эффективности ОС, ВС, Функции ОС, Типы аппаратных ресурсов

Процесс, поток, мультипрограммирование, 3 состояния процесса(потока), Диспетчеризация потоков, Виды многопоточности





2. Классификация ОС. История развития. Тенденции развития современных процессоров