Основы ядерной архитектуры ОС Windows

Часть 1

Применение знаний ядра ОС

- Средства защиты информации – системы шифрования сетевого трафика, защита данных на диске (виртуальные диски, фильтр-драйверы, защищенные файловые системы)
- Вредоносное ПО
- Средства анализа в самом широком смысле (антивирусы, отладчики, системные утилиты)
- В первую очередь знание ядерной архитектуры важно для аналитика

Применение знаний при разработке инфраструктуры анализа бинарного кода

- Драйвер получения информации: список процессов, модулей и копии адресных пространств процессов
- Алгоритмы в составе инфраструктуры анализа, позволяющие выявлять: переключение процессов/потоков/адресных пространств, обмен данными между адресными пространствами, механизмы межпоточной синхронизации и др.

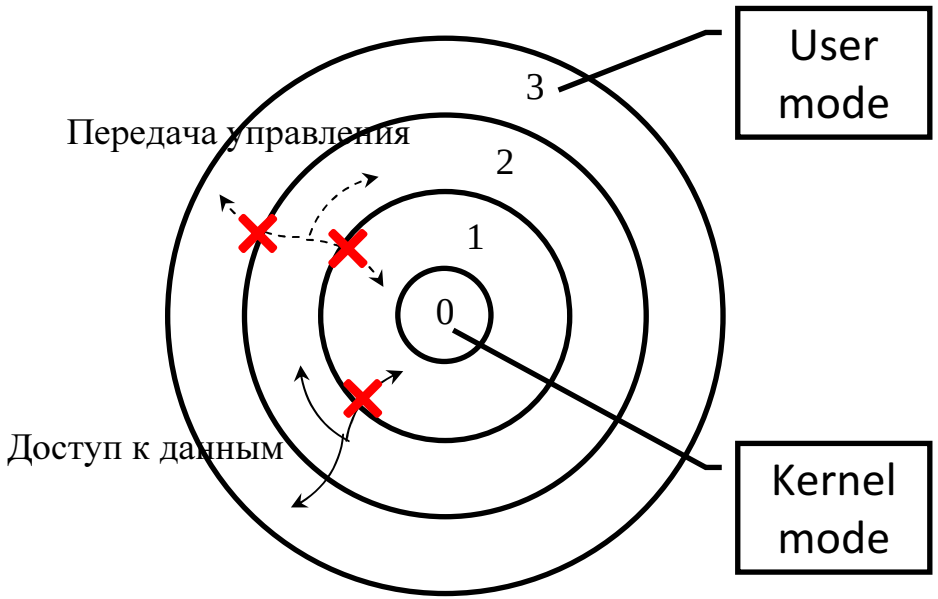
Основные понятия

- **Процессы**, **потоки** и **адресные пространства** в ОС Windows: взаимосвязь с процессорной архитектурой

- Архитектура процессора в защищенном режиме предоставляет для использования разработчикам ОС некоторые механизмы:
- Защита памяти («кольца защиты»)
- Сегментация
- Страничная организация памяти
- Многозадачность (task switching)

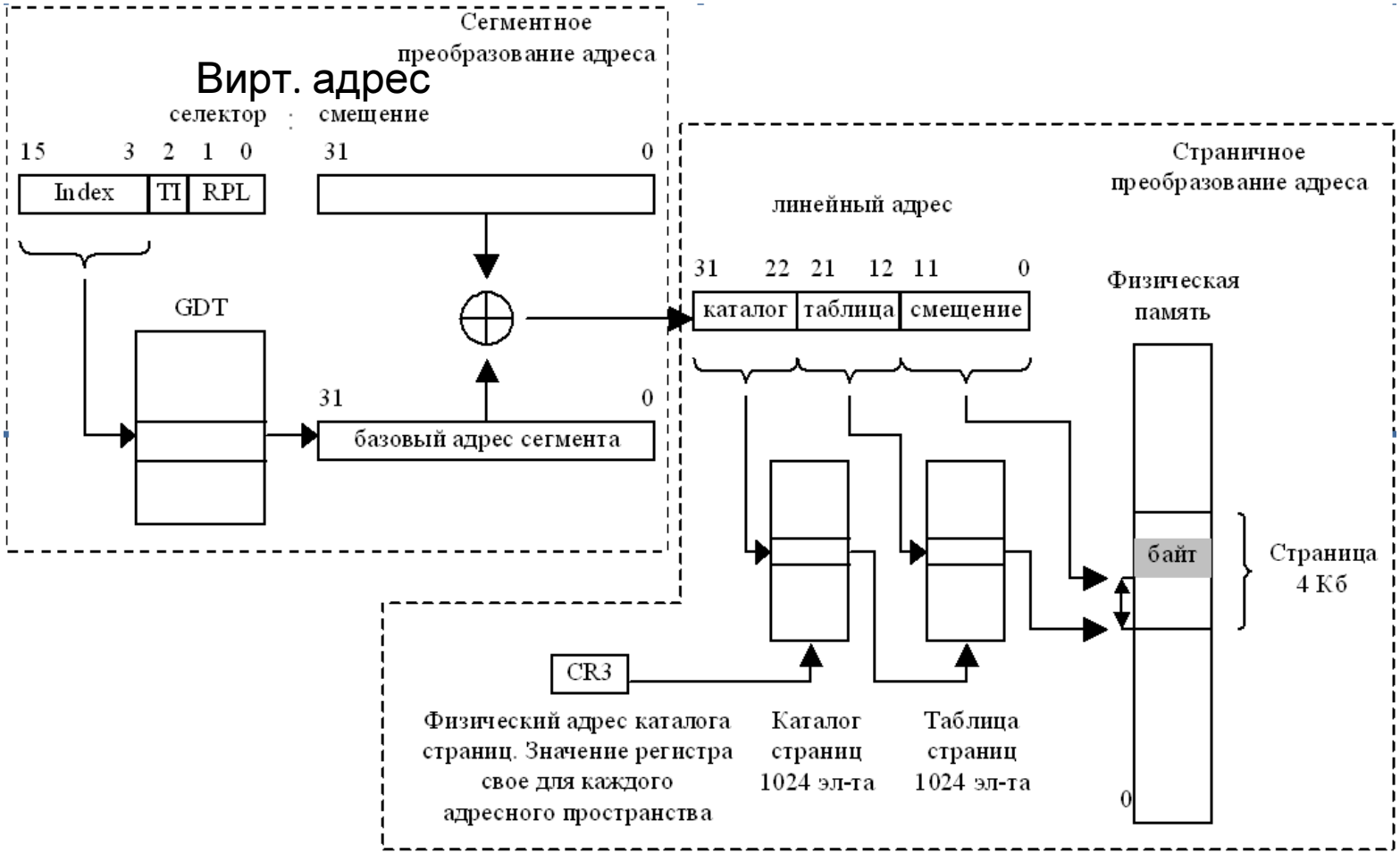
- особенности использования этих процессорных механизмов во многом определяют архитектуру ОС, в особенности рассматриваемые понятия: термины процесс, поток и адресное пространство имеют смысл только в рамках конкретной ОС

Защита памяти по привилегиям («кольца защиты»)



Важной особенностью реализации механизма колец защиты является то, что для каждого кольца защиты хранится своя копия стековых регистров (ESP), и при переключении кольца защиты аппаратура процессора осуществляет замену текущего значения ESP на сохраненную копию, соответствующую новому кольцу защиты.

Особенности организации памяти в Windows



ОС NT, хотя и использует селекторы, но использует их в минимальной степени.
NT реализует плоскую 32-разрядную модель памяти с размером линейного адресного пространства 4 Гб (2^{32} байт).
В NT определено 11 селекторов, из которых нас будут интересовать всего 4:

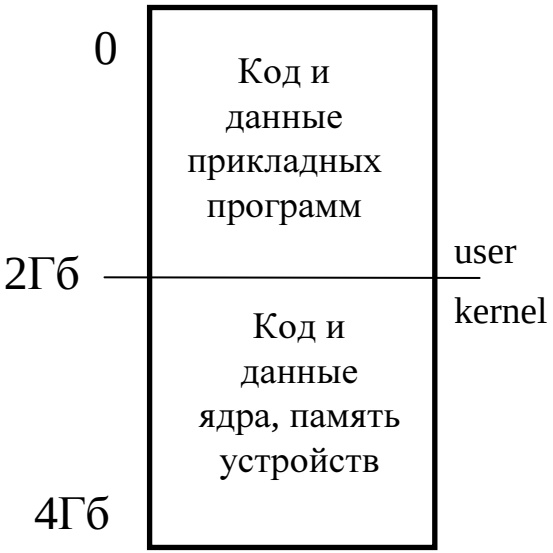
Селектор Hex (bin)	Назначение (RPL)	база	предел	DPL	тип
08 (001000)	Code32 (kernel)	00000000	FFFFFFFF	0 (kernel)	RE
10 (010000)	Data32 (kernel)	00000000	FFFFFFFF	0 (kernel)	RW
1b (011011)	Code32 (user)	00000000	FFFFFFFF	3 (user)	RE
23 (100011)	Data32 (user)	00000000	FFFFFFFF	3 (user)	RW

Эти 4 селектора позволяют адресовать все 4Гб линейного адресного пространства, разница только в режиме доступа.
Первые 2 селектора имеют DPL=0 и используются драйверами и системными компонентами для доступа к системному коду, данным и стеку.
Вторые 2 селектора используются кодом пользовательского режима для доступа к коду, данным и стеку пользовательского режима. Эти селекторы являются константами для ОС NT.
Сегментное преобразование пары селектор:смещение дает 32-битный линейный адрес, для всех рассматриваемых селекторов совпадающее со значением смещения виртуального адреса.

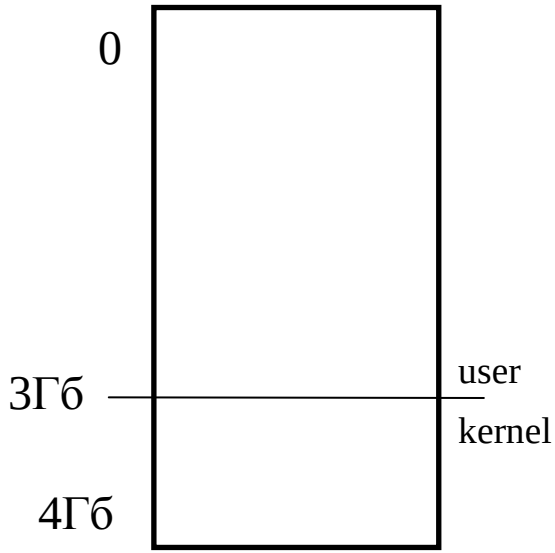
- Реальная защита памяти по привилегиям, как и реализация понятия адресное пространство, осуществляется посредством механизма страничной организации памяти:
- Каждый контекст памяти (адресное пространство процесса) описывается своим каталогом страниц. Физический адрес текущего каталога страниц содержится в регистре CR3
- элемент таблицы страниц содержит бит, указывающий на **возможность доступа к странице из пользовательского режима** (0 - доступ только ядру, 1 - доступ пользовательскому режиму). При этом **все страницы доступны из режима ядра**. Таким образом, механизм защиты страниц использует всего 2 уровня привилегий - 0 (ядро) и 3 (пользователь). С помощью этого механизма адресное пространство делится на 2 части: системную и пользовательскую.
- элемент таблицы страниц содержит бит, указывающий на **возможность записи в соответствующую страницу памяти** (0 - доступ только на чтение, 1 - чтение/запись). Чтение возможно всегда, когда разрешен доступ по привилегиям.
- В некоторых режимах элемент таблицы страниц содержит бит, **определяющий соответствующую страницу как данные** (запрет на выполнение кода с данной страницы)

Виртуальное адресное пространство процесса

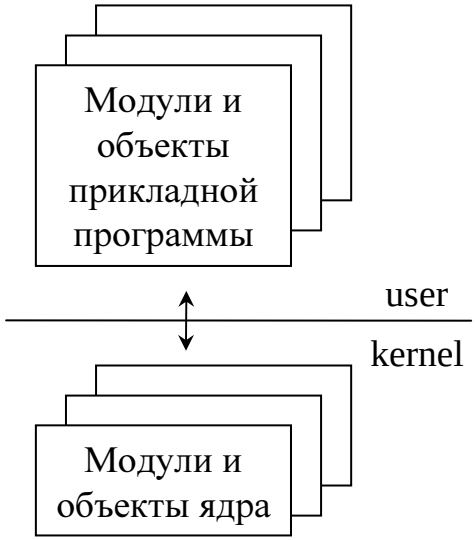
Адресное пространство в 32 разрядном режиме



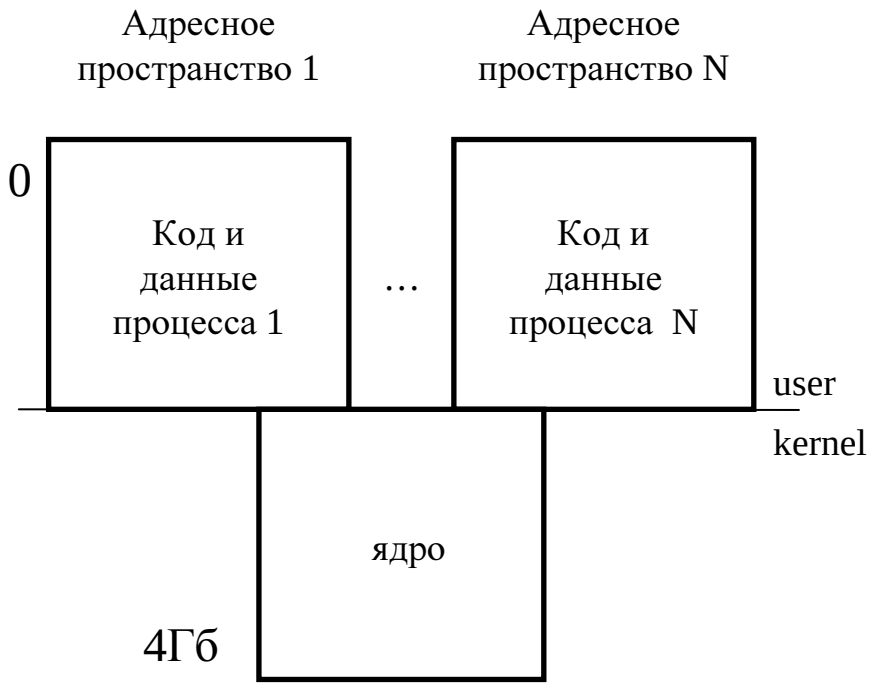
Границу между пользовательским и системным диапазоном можно сдвинуть



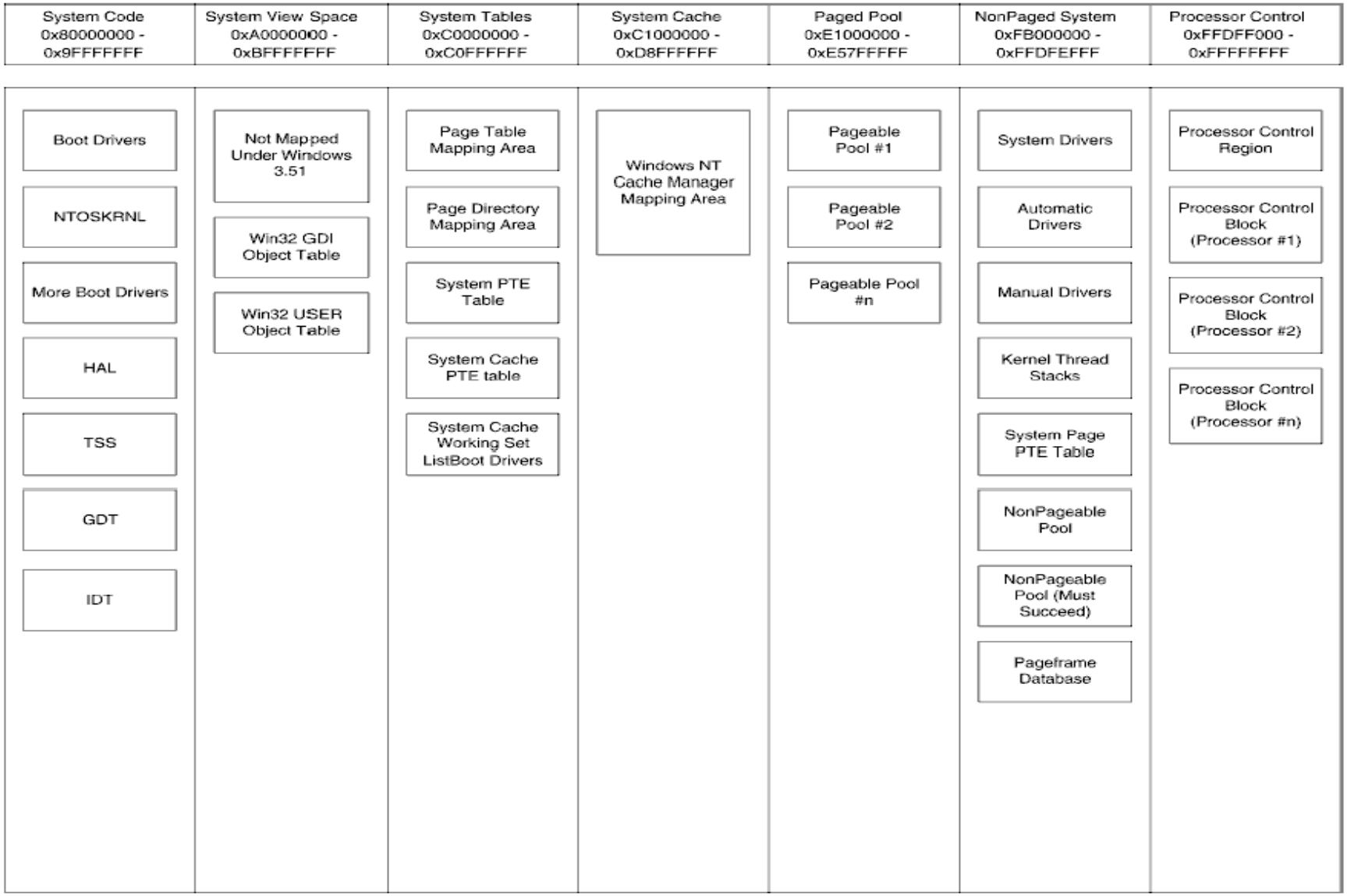
Взаимодействие между кодом пользовательского режима и ядром часто изображают так:



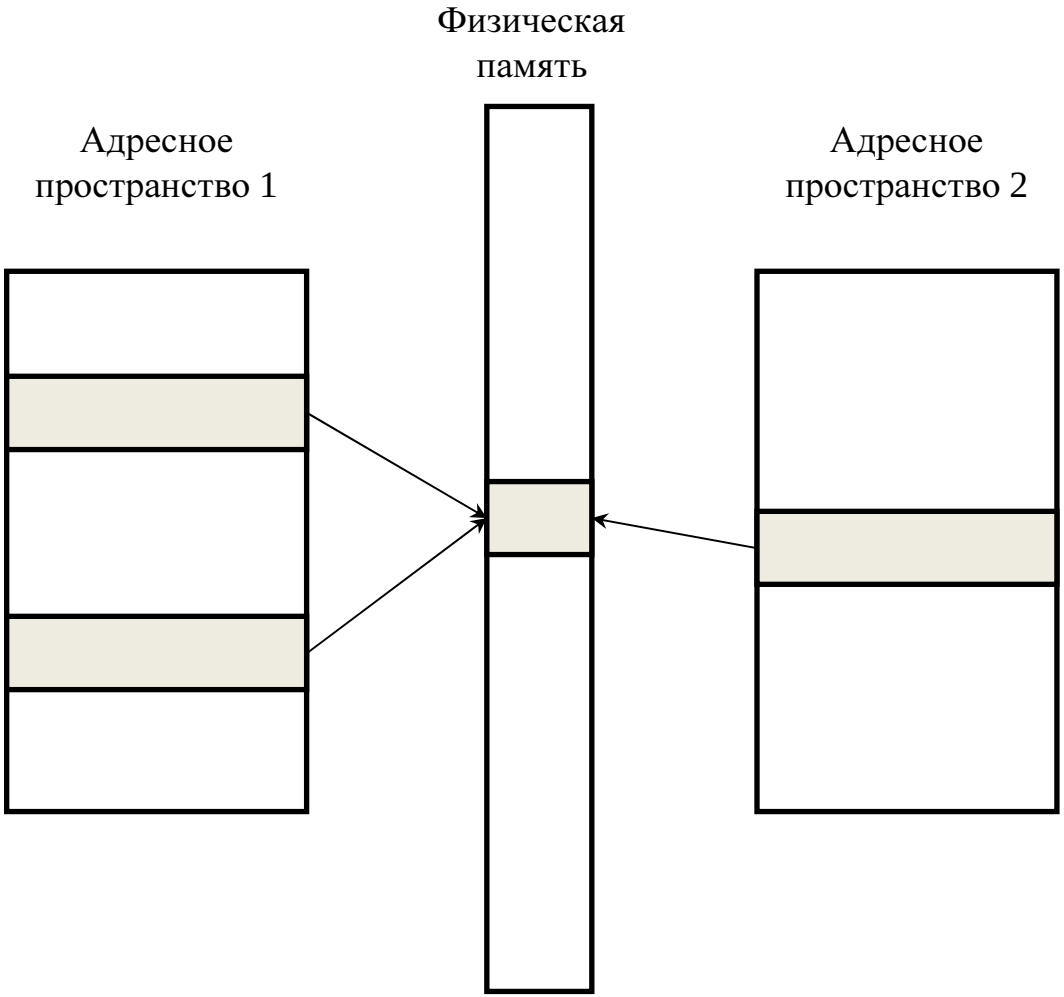
- Каталоги страниц всех процессов организованы так, что преобразование линейного в физический адрес для системного диапазона любого адресного пространства совпадают – диапазон ядра «общий» для всех адресных пространств



The following diagram shows the system memory map for Windows NT.

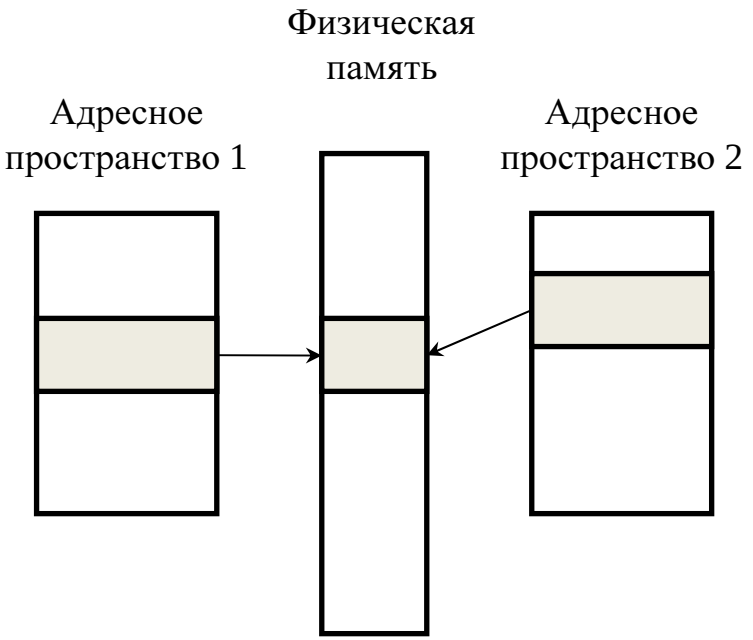


- Каталоги страниц позволяют создавать «совместно используемые» как внутри одного, так и между несколькими адресными пространствами области памяти путем трансляции разных виртуальных адресов в одинаковые физические.

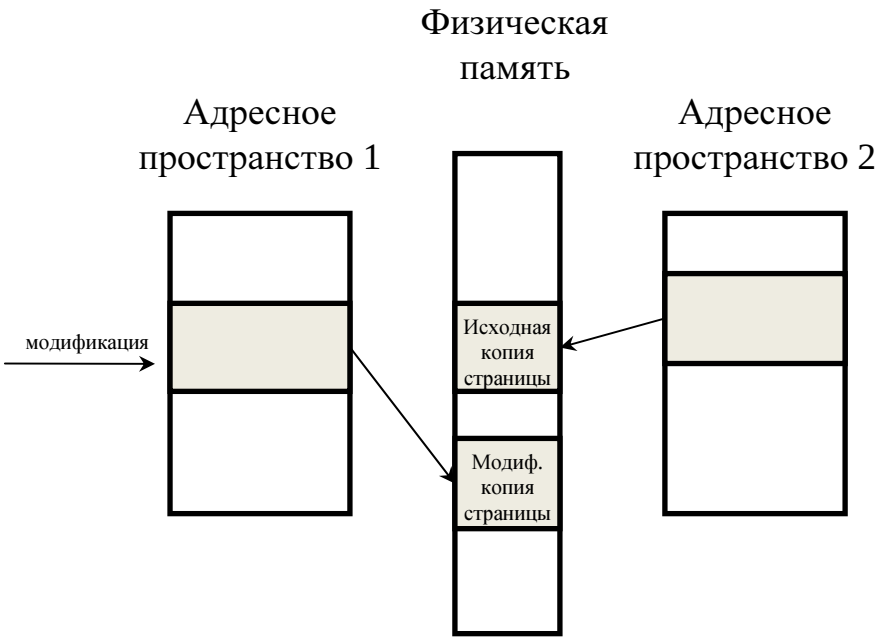


- Разновидность такого механизма – **copy-on-write** – используется для реализации механизма отображения файлов в память, через который в свою очередь подгружаются в адресное пространство все исполняемые модули - .exe, .dll, .sys и т.п.

Изначально все страницы адресных пространств, соответствующие исполняемому модулю, отображены в общий набор страниц физической памяти



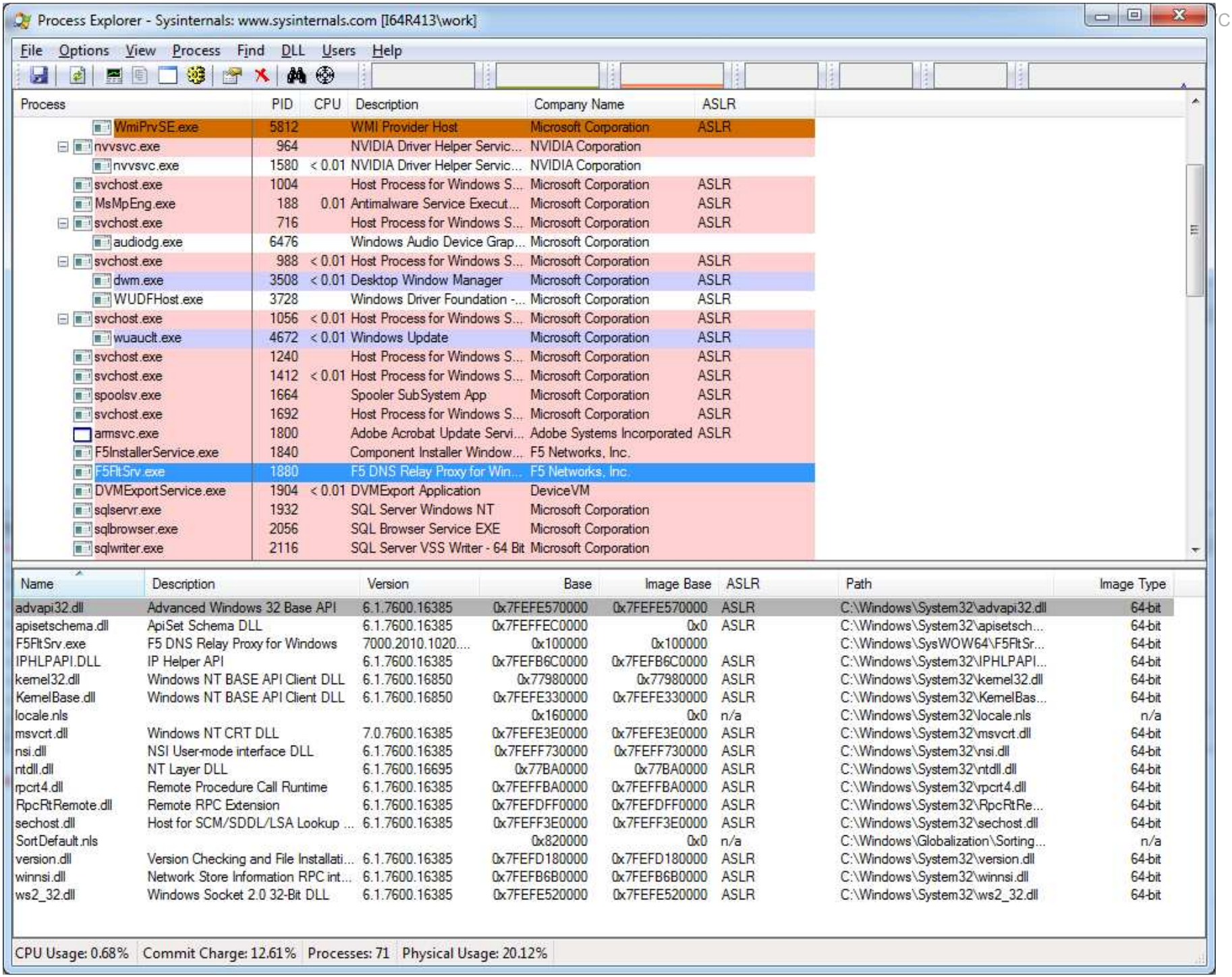
При попытке записи в какую-либо из таких страниц в одном из адресных пространств это адресное пространство вначале получает свою уникальную копию этой страницы в физической памяти, и только затем в эту копию осуществляется запись



Загрузка модулей в адресное пространство

- Исполняемые файлы в ОС Windows имеют формат PE (Portable Executable).
- При запуске программы создается адресное пространство процесса, причем системный диапазон адр. простр. запускаемого процесса создается простым копированием части каталога страниц запускающего процесса.
- Затем в пользовательский диапазон памяти отображается исполняемый модуль (обычно .exe) в соответствии с предпочтительным адресом загрузки, прописанном в PE-заголовке (задается компилятором, для MS VisualStudio обычно 0x400000).
- Далее в соответствии с таблицей импорта модуля начинается подгрузка (также через механизм отображения в память) всех динамических библиотек, с которыми связан текущий загружаемый модуль. Каждая такая библиотека (обычно .dll) также имеет формат PE, и для нее весь процесс повторяется заново.
- Если предпочтительный адрес загрузки уже занят, загрузчик перебазирует модуль на свободный участок виртуальной памяти. Кроме того, компилятор может создавать неперебазируемые модули, которые обязаны грузиться строго по указанному в заголовке адресу (обычно используется для некоторых модулей ядра). Сложность перебазирования определяется необходимостью загрузчику модифицировать инструкции, осуществляющие обращения по глобальным адресам (например при доступе к глобальным переменным) – компилятор формирует адреса таких переменных исходя из назначенного им же базового адреса загрузки, при перебазировании все такие адреса должны быть модифицированы. Список адресов кода, в которые требуется внести поправки при перебазировании, хранится в секции поправок (relocation table).

- При завершении загрузки модуля со всеми его зависимостями управление передается на функцию – точку входа в модуль, прописанную в его PE-заголовке (это касается всех загружаемых модулей).
- После отображения в память содержимое файла все еще не прочитано с диска. Такой файл начинает выступать в роли файла подкачки, а реальное чтение страниц произойдет при первом обращении к соответствующему адресу виртуальной памяти.
- Даже после выгрузки модуля ОС по возможности сохраняет информацию о выполненном ранее отображении файла и его физические страницы, для ускорения возможной повторной загрузки модуля.
- Перебазирование модуля – почти катастрофическая ситуация для ОС, поскольку вносимые поправки приводят к увеличению страниц физической памяти за счет срабатывания механизма copy-on-write. Кроме того, для перебазированного модуля становится затруднительным использовать оставшиеся от предыдущей загрузки страницы памяти.
- В ОС начиная с Vista появился механизм Address Space Layout Randomization (ASLR), предназначенный для усложнения атак со стороны вредоносного кода, направленных на прямой доступ к коду и данным (в т.ч. стеку) системных библиотек. В соответствии с этим механизмом, модули размещаются по «случайно» выбираемым адресам.
- В силу указанных выше проблем с перебазированием ОС не может допустить случайное размещение модуля в разных адресных пространствах. Вместо этого «случайные» адреса загрузки всех модулей вычисляются разово для всех модулей всех адресных пространств, и в ходе текущего сеанса работы ОС не пересчитываются.


Process Explorer - Sysinternals: www.sysinternals.com [I64R413\work]
File Options View Process Find DLL Users Help
Process PID CPU Description Company Name ASLR
WmiPrvSE.exe 5812 WMI Provider Host Microsoft Corporation ASLR
nvvsvc.exe 964 NVIDIA Driver Helper Servic... NVIDIA Corporation
nvvsvc.exe 1580 < 0.01 NVIDIA Driver Helper Servic... NVIDIA Corporation
svchost.exe 1004 Host Process for Windows S... Microsoft Corporation ASLR
MsMpEng.exe 188 0.01 Antimalware Service Execut... Microsoft Corporation ASLR
svchost.exe 716 Host Process for Windows S... Microsoft Corporation ASLR
audiodg.exe 6476 Windows Audio Device Grap... Microsoft Corporation
svchost.exe 988 < 0.01 Host Process for Windows S... Microsoft Corporation ASLR
dwm.exe 3508 < 0.01 Desktop Window Manager Microsoft Corporation ASLR
WUDFHost.exe 3728 Windows Driver Foundation -... Microsoft Corporation ASLR
svchost.exe 1056 < 0.01 Host Process for Windows S... Microsoft Corporation ASLR
wuauclt.exe 4672 < 0.01 Windows Update Microsoft Corporation ASLR
svchost.exe 1240 Host Process for Windows S... Microsoft Corporation ASLR
svchost.exe 1412 < 0.01 Host Process for Windows S... Microsoft Corporation ASLR
spoolsv.exe 1664 Spooler SubSystem App Microsoft Corporation ASLR
svchost.exe 1692 Host Process for Windows S... Microsoft Corporation ASLR
armsvc.exe 1800 Adobe Acrobat Update Servi... Adobe Systems Incorporated ASLR
F5InstallerService.exe 1840 Component Installer Window... F5 Networks, Inc.
F5FltSrv.exe 1880 F5 DNS Relay Proxy for Win... F5 Networks, Inc.
DVMExportService.exe 1904 < 0.01 DVMExport Application DeviceVM
sqlservr.exe 1932 SQL Server Windows NT Microsoft Corporation
sqlbrowser.exe 2056 SQL Browser Service EXE Microsoft Corporation
sqlwriter.exe 2116 SQL Server VSS Writer - 64 Bit Microsoft Corporation
Name Description Version Base Image Base ASLR Path Image Type
advapi32.dll Advanced Windows 32 Base API 6.1.7600.16385 0x7FEFE570000 0x7FEFE570000 ASLR C:\Windows\System32\advapi32.dll 64-bit
apisetschema.dll ApiSet Schema DLL 6.1.7600.16385 0x7FEFFEC0000 0x0 ASLR C:\Windows\System32\apisetsch... 64-bit
F5FltSrv.exe F5 DNS Relay Proxy for Windows 7000.2010.1020.... 0x100000 0x100000 C:\Windows\SysWOW64\F5FltSr... 64-bit
IPHLPAPI.DLL IP Helper API 6.1.7600.16385 0x7FEFB6C0000 0x7FEFB6C0000 ASLR C:\Windows\System32\IPHLPAPI... 64-bit
kernel32.dll Windows NT BASE API Client DLL 6.1.7600.16850 0x77980000 0x77980000 ASLR C:\Windows\System32\kernel32.dll 64-bit
KernelBase.dll Windows NT BASE API Client DLL 6.1.7600.16850 0x7FEFE330000 0x7FEFE330000 ASLR C:\Windows\System32\KernelBas... 64-bit
locale.nls 0x160000 0x0 n/a C:\Windows\System32\locale.nls n/a
msvcrt.dll Windows NT CRT DLL 7.0.7600.16385 0x7FEFE3E0000 0x7FEFE3E0000 ASLR C:\Windows\System32\msvcrt.dll 64-bit
nsi.dll NSI User-mode interface DLL 6.1.7600.16385 0x7FEFF730000 0x7FEFF730000 ASLR C:\Windows\System32\nsi.dll 64-bit
ntdll.dll NT Layer DLL 6.1.7600.16695 0x77BA0000 0x77BA0000 ASLR C:\Windows\System32\ntdll.dll 64-bit
rpcrt4.dll Remote Procedure Call Runtime 6.1.7600.16385 0x7FEFFBA0000 0x7FEFFBA0000 ASLR C:\Windows\System32\rpcrt4.dll 64-bit
RpcRtRemote.dll Remote RPC Extension 6.1.7600.16385 0x7FEFDFF0000 0x7FEFDFF0000 ASLR C:\Windows\System32\RpcRtRe... 64-bit
sechost.dll Host for SCM/SDDL/LSA Lookup ... 6.1.7600.16385 0x7FEFF3E0000 0x7FEFF3E0000 ASLR C:\Windows\System32\sechost.dll 64-bit
SortDefault.nls 0x820000 0x0 n/a C:\Windows\Globalization\Sorting... n/a
version.dll Version Checking and File Installati... 6.1.7600.16385 0x7FEFD180000 0x7FEFD180000 ASLR C:\Windows\System32\version.dll 64-bit
winnsi.dll Network Store Information RPC int... 6.1.7600.16385 0x7FEFB6B0000 0x7FEFB6B0000 ASLR C:\Windows\System32\winnsi.dll 64-bit
ws2_32.dll Windows Socket 2.0 32-Bit DLL 6.1.7600.16385 0x7FEFE520000 0x7FEFE520000 ASLR C:\Windows\System32\ws2_32.dll 64-bit
CPU Usage: 0.68% Commit Charge: 12.61% Processes: 71 Physical Usage: 20.12%

Поддержка многозадачности

- Механизм аппаратной поддержки многозадачности, предоставляемый архитектурой x86 (task switching) – в Windows не используется при штатном функционировании ОС. Единственный случай его применения – при обработке BSOD (“синий экран”), т.е. при фатальном сбое в работе ядра ОС, для обработки которого на время вывода диагностических сообщений и формирования crashdump требуется работоспособная программно-аппаратная среда. В момент штатной работы ОС все происходит в рамках одного task’а, для обработки BSOD происходит переключение в другую задачу, выхода из нее уже нет.

- Собственно многозадачность в терминах ОС Windows реализуется программно и будет рассмотрена позже.

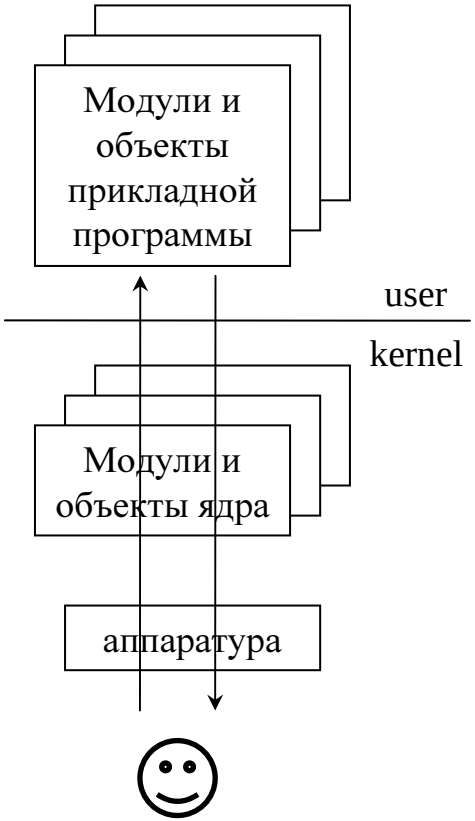
Основные понятия: подведение итогов

- **Процесс** – соответствует некоторому запущенному в ОС приложению.
- Процесс можно рассматривать как контейнер ресурсов, выделяемых приложению для работы.
- Одним из важнейших ресурсов является адресное пространство. Каждому процессу выделяется свое собственное, единственное, адресное пространство.
- Код выполняется в рамках потоков

- **Адресное пространство** – диапазон виртуальных адресов, доступный процессу.
- Для некоторой части этих адресов средствами ОС при поддержке аппаратуры осуществляется трансляция в физические адреса, т.е. обращение к таким адресам будет либо немедленно перенаправлено в ОЗУ или память устройств, или приведет к задействованию механизма подкачки страниц с диска в ОЗУ с последующим доступом к ним в ОЗУ.
- Если аппаратные механизмы контроля доступа к памяти не позволяют осуществить доступ, либо трансляция памяти для адресуемой виртуальной ячейке отсутствует, процессор генерирует исключение.
- Адресное пространство разбито на 2 диапазона: по младшим адресам – диапазон памяти, доступный и коду пользовательского режима (user mode = 3 кольцо защиты), и коду ядра (kernel mode = 4 к.з.; по старшим адресам – диапазон памяти ядра (доступен только коду ядре).
- Диапазон памяти ядра с точки трансляции адресов – «общий» для всех адресных пространств
- Вся память устройств, отображаемая на адресуемую физическую память, в виртуальном адресном пространстве отображается в диапазон памяти ядра. Доступ к портам в/в коду user mode также закрыт, в результате прикладной уровень не имеет возможности прямой работы с оборудованием.

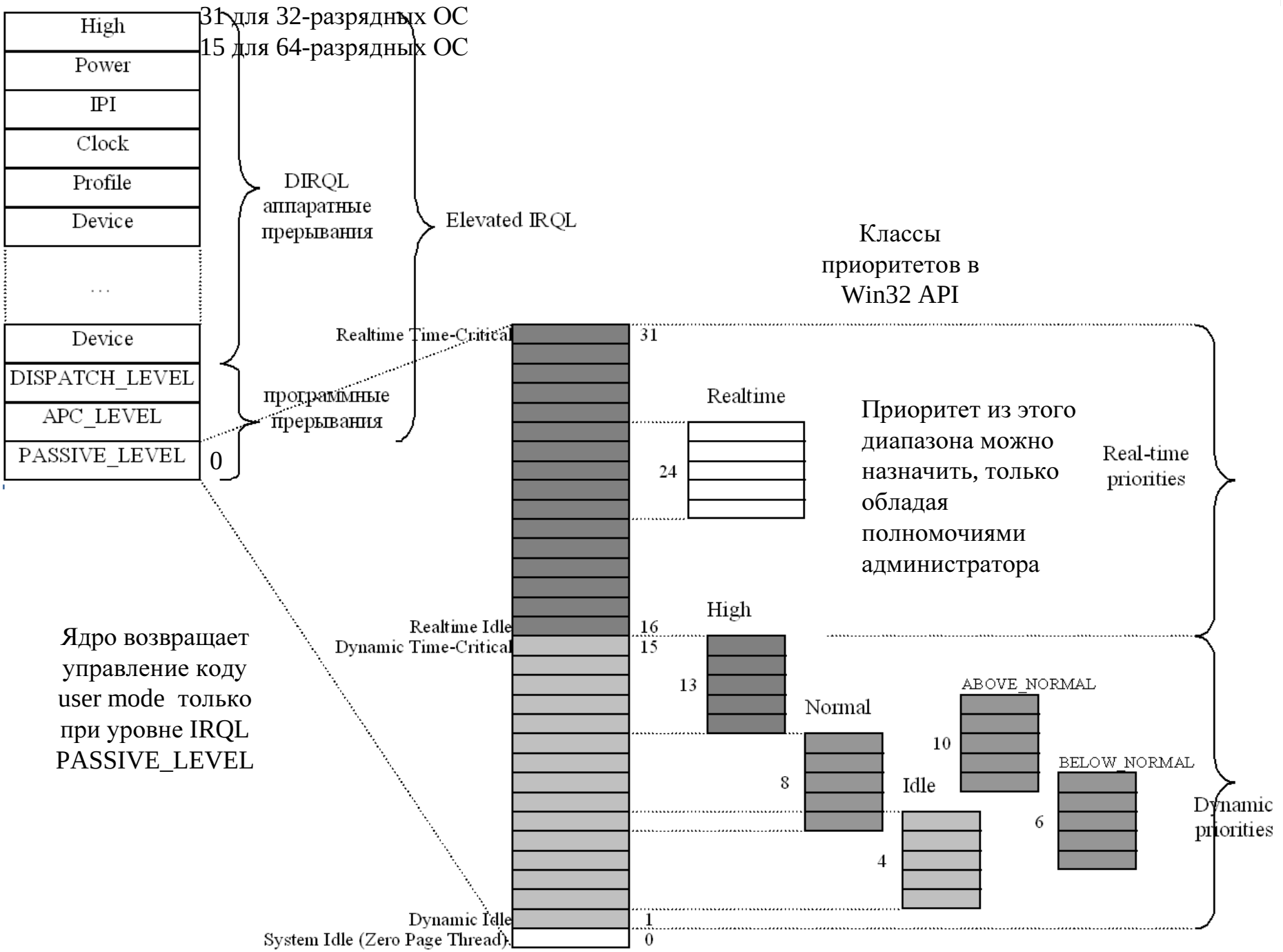
- **Поток** – единица исполнения в ОС Windows. Механизм вытесняющей многозадачности, реализуемый ОС, применяется именно к потокам.
- Программный код исполняется в потоке, изначально для процесса создается один поток, который затем может создать дополнительные.
- Все потоки одного процесса работают в рамках общего адресного пространства, но различаются контекстом – копией регистров, которая сохраняется перед приостановкой потока при исчерпании потоком кванта времени и подгружается при возобновлении потока при получении потоком очередного кванта времени.

Контекст потока (thread context) – термин, часто используемый в ядерной документации. Состояние работы потока определяется принадлежащей ему копией процессорных регистров, в том числе отдельной копией регистра вершины стека для каждого кольца защиты. Для ядерных функций различают два вида контекстов:
Arbitrary thread context – контекст случайного потока – как правило для функций, вызов которых произошел из-за возникновения асинхронного аппаратного события, прервав работу произвольного работающего потока (на рис. стрелка вверх)
Non-arbitrary thread context – контекст определенного потока – как правило – контекст потока, в котором произошла передача управления в код ядра (на рис. стрелка вниз)



Система приоритетов

- Windows NT имеет двухуровневую модель приоритетов:
- Приоритеты высшего уровня (**уровни запросов прерываний** - Interrupt ReQuest Level - IRQL) управляют аппаратными и программными прерываниями
- приоритеты низшего уровня (**приоритеты планирования**) управляются планировщиком и управляют исполнением потоков


High
Power
IPI
Clock
Profile
Device
...
Device
DISPATCH_LEVEL
APC_LEVEL
PASSIVE_LEVEL
0
31 для 32-разрядных ОС
15 для 64-разрядных ОС
DIRQL
аппаратные
прерывания
Elevated IRQL
программные
прерывания
Ядро возвращает управление коду user mode только при уровне IRQL PASSIVE_LEVEL
Классы приоритетов в Win32 API
Realtime Time-Critical
31
Realtime
24
Приоритет из этого диапазона можно назначить, только обладая полномочиями администратора
Real-time priorities
Realtime Idle
16
Dynamic Time-Critical
15
High
13
Normal
8
ABOVE_NORMAL
10
Idle
4
BELOW_NORMAL
6
Dynamic priorities
Dynamic Idle
1
System Idle (Zero Page Thread)
0

Приоритеты планирования

Схема приоритетов, реализуемая ядром ОС:

- При запуске **потоку** назначается **базовый приоритет (б.п.)**.
- В процессе работы – у потока текущий приоритет (т.п.)
- Если базовый приоритет в динамическом диапазоне, исходно **т.п. = б.п.,** затем в процессе работы ОС может менять т.п. по правилу: **б.п. ≤ т.п. ≤ 15**
- Если базовый приоритет диапазоне realtime, то всегда **т.п. = б.п.**

- Win32 API скрывает эту схему: **процессу** назначается **класс приоритета**, а его потокам – относительный приоритет – значение ±2 относительно класса приоритета потока.
- Вся эта схема все равно сводится к системе приоритетов, реализуемой ядром ОС.

- Планировщик Windows реализует «карусельную» (round-robbin) схему переключения потоков по правилу:
- Если есть потоки из диапазона realtime – переключение по карусельной схеме будет только для потоков с максимальным значением realtime-приоритета. Любые потоки с более низким приоритетом никогда не исполнятся.
- Если потоков с realtime-приоритетом нет – карусельная схема для всех потоков с динамическим диапазоном приоритетов. Для исполнения выбираются потоки с максимальным т.п., но у потоков с более низким т.п. он иногда инкрементируется, и в конце концов поток даже с самым низким базовым приоритетом в конце концов получит квант времени, после чего его т.п. сбрасывается.

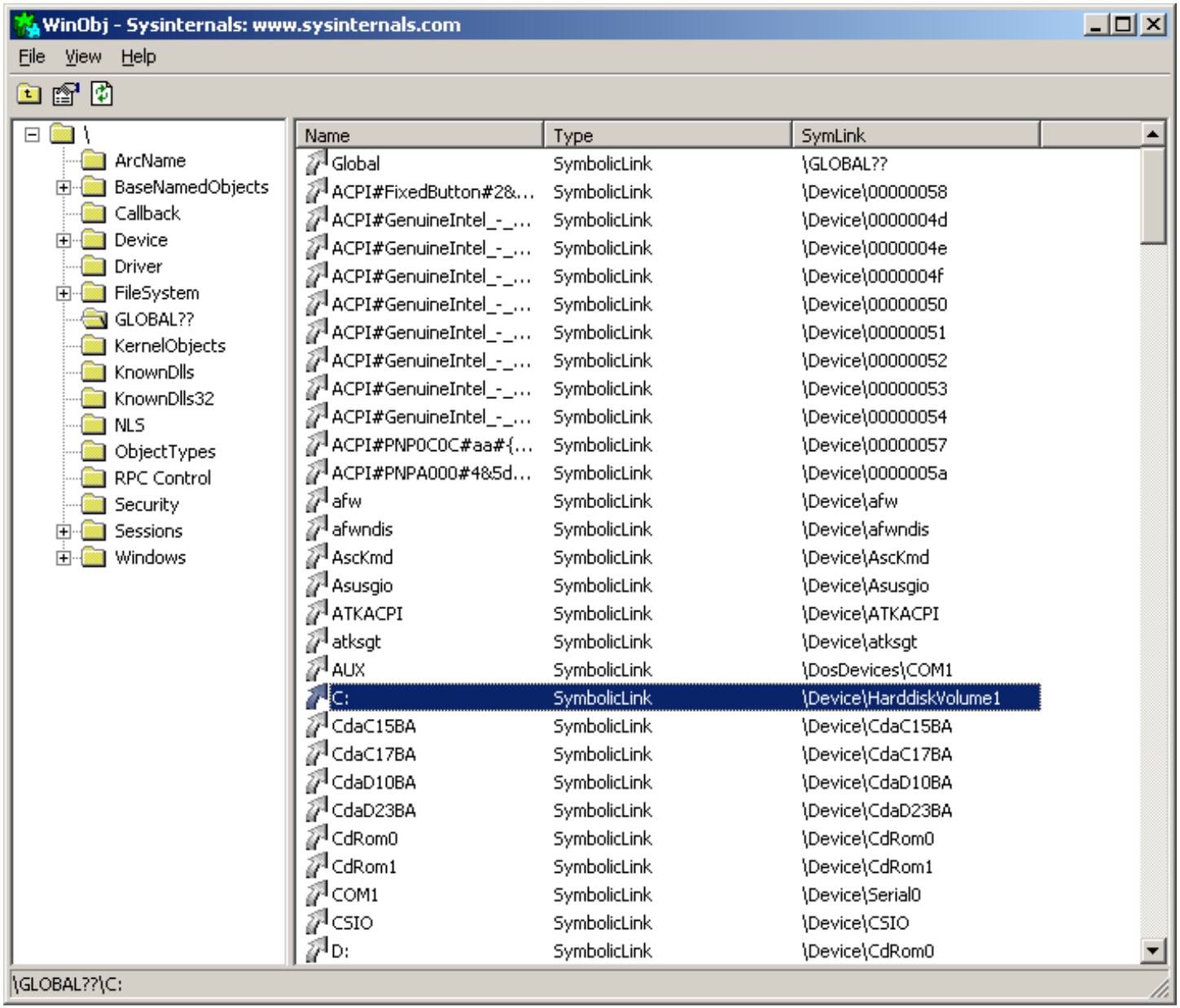
Уровни запросов прерываний (IRQL)

- IRQL – способ управления маскированием прерываний на конкретном процессоре. По сути – это некоторое число, назначаемое конкретному процессору и обозначающее факт маскирования на нем некоторых прерываний. Конкретное значение IRQL из диапазона DIRQL (Device IRQL) также связывается с каждым обработчиком аппаратных прерываний. Если такое прерывание в данный момент не маскировано и оно доставлено процессору, ОС производит смену текущего IRQL процессора на IRQL прерывания, маскируя при этом все прерывания с IRQL ≤ нового IRQL процессора. При завершении обработчика прерывания значение IRQL процессора восстанавливается на старое.
- На уровне IRQL_PASSIVE_LEVEL разрешены все прерывания и работает механизм многозадачности. Код пользовательского режима работает только на PASSIVE_LEVEL (возможно за исключением функций, вызываемых механизмом APC на уровне APC_LEVEL)
- На уровне APC_LEVEL реализуется механизм асинхронного вызова процедур Asynchronous Procedure Call (APC), через который в частности ядро ОС реализует механизм обратного вызова функций (callback functions) прикладного уровня для уведомления прикладной программы ядром о возникновении некоторой ожидаемой ситуации. На этом уровне маскированы прерывания с IRQL= APC_LEVEL. Доступ к выгружаемой (paged) памяти разрешен.

- На уровне IRQL DISPATCH_LEVEL работают так называемые «отложенные процедуры» (Deferred Procedure Calls - DPC), поэтому этот уровень иногда называют DPC_LEVEL. DPC создаются обработчиками прерываний, и туда выносится весь «долгоиграющий» код обработки прерывания. Задачей обработчика прерывания является как можно быстрее зафиксировать факт возникновения прерывания и данные, необходимые для дальнейшей обработки в DPC. Вызовы DPC помещаются в очередь (своя для каждого процессора), обработка которых ведется в соответствии с собственной системой приоритетов (low-medium-high). Возврата к IRQL<DISPATCH_LEVEL не будет до тех пор, пока в очереди DPC содержится хоть один запрос.
- На этом же уровне работают прерывания, отвечающие за подгрузку страниц в ОЗУ из файлов подкачки, а также за диспетчеризацию потоков. Отсюда самое главное ограничение на код ядра:

- при работе процессора на уровне IRQL≥DISPATCH_LEVEL запрещено обращение к выгружаемой памяти – оно немедленно приведет к BSOD с ошибкой IRQL_NOT_LESS_OR_EQUAL.
- На уровне IRQL>PASSIVE_LEVEL нельзя предпринимать никакие действия, которые могут привести к необходимости переключения потока (например использовать механизмы синхронизации на основе диспетчерских объектов (Dispatcher Objects) с помощью функции WaitFor(Single/Multiple)Objects с ненулевым временем ожидания) – замаскированы обслуживающие такое переключение прерывания уровня APC и DISPATCH_LEVEL
- Уровни выше DISPATCH_LEVEL в основном соответствуют обработчикам аппаратных прерываний (DIRQLs). Маскирована часть прерываний, к возникновению которых могут приводить в том числе вызовы многих служебных ядерных функций, поэтому множество разрешенных к вызову служебных функций сильно ограничено. Серьезная обработка должна быть вынесена в DPC, а иногда – и в рабочие потоки с IRQL PASSIVE_LEVEL.

Пространство имен диспетчера объектов



- Имена ядерных объектов размещаются в едином пространстве имен. Подсистема Win32 скрывает его наличие, транслируя обращения к именам объектов различных типов в обращения к конкретным директориям единого пространства имен, иногда полностью видоизменяя имя объекта. Например, обращение к файлу с именем *c:\foo.txt* будет последовательно трансформировано следующим образом:
- *\??\c:\foo.txt*
- В директории *\??* будет найден элемент с именем *C:*, имеющий тип *SymbolicLink* и значение *\Device\HarddiskVolume1*, в результате подстановки имя примет вид
- *\Device\HarddiskVolume1\foo.txt*
- В директории *\Device* будет найден элемент с типом *Device*, на этом разбор имени будет закончен, а соответствующему устройству будет отправлен запрос, частью которого будет оставшаяся неразобранной часть имени *\foo.txt* – дальше с ней будет разбираться драйвер устройства.