Лекция 8:

**Управление памятью**

**Виртуальная память**

Всем процессам в операционной системе *Windows* предоставляется важнейший *ресурс* – *виртуальная память* (*virtual* *memory*). Все данные, с которыми процессы непосредственно работают, хранятся именно в виртуальной памяти.

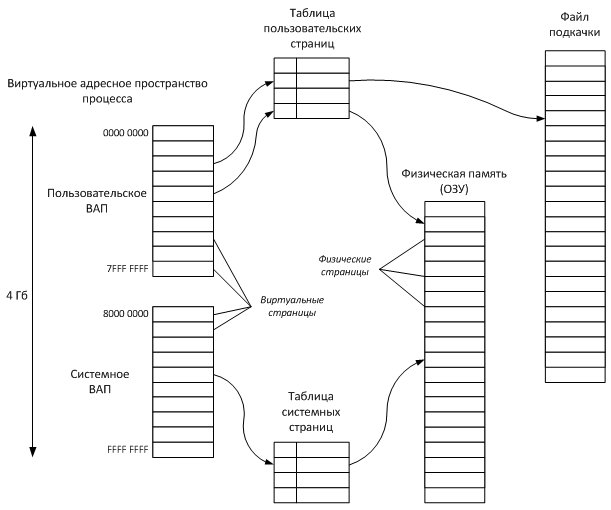
Название "виртуальная" произошло из-за того что процессу неизвестно реальное (физическое) расположение памяти – она может находиться как в оперативной памяти (*ОЗУ*), так и на диске. *Операционная система* предоставляет процессу *виртуальное* *адресное пространство* (ВАП, *virtual address* *space*) определенного размера и процесс может работать с ячейками памяти *по* любым виртуальным адресам этого пространства, не "задумываясь" о том, где реально хранятся данные.

Размер виртуальной памяти теоретически ограничивается разрядностью операционной системы. На практике в конкретной реализации операционной системы устанавливаются ограничения ниже теоретического предела. Например, для 32-разрядных систем (*x86*), которые используют для адресации 32 разрядные регистры и переменные, теоретический *максимум* составляет 4 ГБ (232 *байт* = 4 294 967 296 *байт* = 4 ГБ). Однако для процессов доступна только половина этой памяти – 2 ГБ, другая половина отдается системным компонентам. В 64 разрядных системах (x64) теоретический *предел* равен 16 экзабайт (264 *байт* = 16 777 216 ТБ = 16 ЭБ). При этом процессам выделяется 8 ТБ, ещё столько же отдается системе, остальное *адресное пространство* в нынешних версиях *Windows* не используется.

Введение виртуальной памяти, во-первых, позволяет прикладным программистам не заниматься сложными вопросами реального размещения данных в памяти, во-вторых, дает возможность операционной системе запускать несколько процессов одновременно, поскольку вместо дорогого ограниченного ресурса – оперативной памяти, используется дешевая и большая *по* емкости внешняя *память*.

**Реализация виртуальной памяти в Windows**

Схема реализации виртуальной памяти в 32-разрядной операционной системе *Windows* представлена на [рис.11.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=1#image.11.1). Как уже отмечалось, процессу предоставляется *виртуальное* *адресное пространство* размером 4 ГБ, из которых 2 ГБ, расположенных *по* младшим адресам (0000 0000 – 7FFF FFFF), процесс может использовать *по* своему усмотрению (пользовательское ВАП), а оставшиеся два гигабайта (8000 0000 – FFFF FFFF) выделяются под системные структуры данных и компоненты (системное ВАП)1. Отметим, что каждый процесс имеет свое собственное пользовательское ВАП, а системное ВАП для всех процессов одно и то же.



**Рис. 11.1.**Реализация виртуальной памяти в 32-разрядных Windows

*Виртуальная память* делится на блоки одинакового размера – виртуальные страницы. В *Windows* страницы бывают большие (*x86* – 4 МБ, x64 – 2 МБ) и малые (4 КБ). *Физическая память* (*ОЗУ*) также делится на страницы точно такого же размера, как и *виртуальная память*. Общее количество малых виртуальных страниц процесса в 32 разрядных системах равно 1 048 576 (4 ГБ / 4 КБ = 1 048 576).

Обычно процессы задействуют не весь объем виртуальной памяти, а только небольшую его часть. Соответственно, не имеет смысла (и, часто, возможности) выделять страницу в физической памяти для каждой виртуальной страницы всех процессов. Вместо этого в *ОЗУ* (говорят, "резидентно") находится ограниченное количество страниц, которые непосредственно необходимы процессу. Такое *подмножество* виртуальных страниц процесса, расположенных в физической памяти, называется рабочим набором процесса (working *set*).

Те виртуальные страницы, которые пока не требуются процессу, *операционная система* может выгрузить на *диск*, в специальный *файл*, называемый файлом подкачки (page file).

Каким образом процесс узнает, где в данный момент находится требуемая страница? Для этого служат специальные структуры данных – таблицы страниц (*page table*).

**Структура виртуального адресного пространства**

Рассмотрим, из каких элементов состоит *виртуальное* *адресное пространство* процесса в 32 разрядных *Windows* ([рис.11.2](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=1#image.11.2)).

В пользовательском ВАП располагаются исполняемый образ процесса, динамически подключаемые библиотеки (*DLL*, *dynamic-link library*), *куча* процесса и стеки потоков.

При запуске программы создается процесс (см. лекцию 6 "Процессы и потоки"), при этом в *память* загружаются код и данные программы (исполняемый образ, *executable image*), а также необходимые программе динамически подключаемые библиотеки (*DLL*). Формируется *куча* (*heap*) – область, в которой процесс может выделять *память* динамическим структурам данных (т. е. структурам, размер которых заранее неизвестен, а определяется в ходе выполнения программы). *По* умолчанию размер кучи составляет 1 МБ, но при компиляции приложения или в ходе выполнения процесса может быть изменен. Кроме того, каждому потоку предоставляется *стек* (stack) для хранения локальных переменных и параметров функций, также *по* умолчанию размером 1 МБ.



**Рис. 11.2.**Структура виртуального адресного пространства

В системном ВАП расположены:

* образы ядра (ntoskrnl.exe), исполнительной системы, HAL (hal.dll), драйверов устройств, требуемых при загрузке системы;
* таблицы страниц процесса;
* системный кэш;
* пул подкачиваемой памяти (paged pool) – системная куча подкачиваемой памяти;
* пул подкачиваемой памяти (nonpaged pool) – системная куча неподкачиваемой памяти;
* другие элементы (см. [[5](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.5)]).

Переменные, в которых хранятся границы разделов в системном ВАП, приведены в [[5](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.5), стр. 442]. Вычисляются эти переменные в функции MmInitSystem (*файл* base\ntos\mm\mminit.c, строка 373), отвечающей за инициализацию подсистемы памяти. В файле base\ntos\mm\i386\mi386.h приведена структура ВАП и определены *константы*, связанные с управлением памятью (например, стартовый *адрес* системного кэша MM\_SYSTEM\_CACHE\_START, строка 199).

**Выделение памяти процессам**

Существует несколько способов выделения виртуальной памяти процессам при помощи Windows API2. Рассмотрим два основных способа – с помощью функции VirtualAlloc и с использованием кучи.

1. WinAPI *функция* VirtualAlloc позволяет резервировать и передавать виртуальную *память* процессу. При резервировании запрошенный *диапазон* виртуального адресного пространства закрепляется за процессом (при условии наличия достаточного количества свободных страниц в пользовательском ВАП), соответствующие виртуальные страницы становятся зарезервированными (*reserved*), но доступа к этой памяти у процесса нет – при попытке чтения или записи возникнет *исключение*. Чтобы получить *доступ*, процесс должен передать *память* зарезервированным страницам, которые в этом случае становятся переданными (*commit*).

Отметим, что резервируются участки виртуальной памяти *по* адресам, кратным значению *константы* гранулярности выделения памяти MM\_ALLOCATION\_GRANULARITY (*файл* base\ntos\inc\mm.h, строка 54). Это *значение* равно 64 КБ. Кроме того, размер резервируемой области должен быть кратен размеру страницы (4 КБ).

WinAPI *функция* VirtualAlloc для выделения памяти использует функцию ядра NtAllocateVirtualMemory (*файл* base\ntos\mm\allocvm.c, строка 173).

2. Для более гибкого распределения памяти существует *куча* процесса, которая управляется диспетчером кучи (*heap* *manager*). Кучу используют WinAPI *функция* HeapAlloc, а также оператор языка C malloc и оператор C++ *new*. *Диспетчер* кучи предоставляет возможность процессу выделять *память* с гранулярностью 8 байтов (в 32-разрядных системах), а для обслуживания этих запросов использует те же функции ядра, что и VirtualAlloc.

Дескрипторы виртуальных адресов

Для хранения информации о зарезервированных страницах памяти используются дескрипторы виртуальных адресов (*Virtual Address* Descriptors, *VAD*). Каждый *дескриптор* содержит данные об одной зарезервированной области памяти и описывается структурой MMVAD (*файл* base\ntos\mm\mi.h, строка 3976).

Границы области определяются двумя полями – StartingVpn (начальный *VPN*) и EndingVpn (конечный *VPN*). *VPN* (*Virtual Page* Number) – это номер виртуальной страницы; страницы просто нумеруются, начиная с нулевой. Если размер страницы 4 КБ (212 *байт*), то *VPN* получается из виртуального адреса начала страницы отбрасыванием младших 12 *бит* (или 3 шестнадцатеричных цифр). Например, если *виртуальная страница* начинается с адреса 0x340000, то *VPN* такой страницы равен 0x340.

Дескрипторы виртуальных адресов для каждого процесса организованы в сбалансированное двоичное АВЛ дерево3 (*AVL* *tree*). Для этого в структуре MMVAD имеются поля указатели на левого и правого потомков: LeftChild и RightChild.

Для хранения информации о состоянии области памяти, за которую отвечает *дескриптор*, в структуре MMVAD содержится *поле* флагов VadFlags.

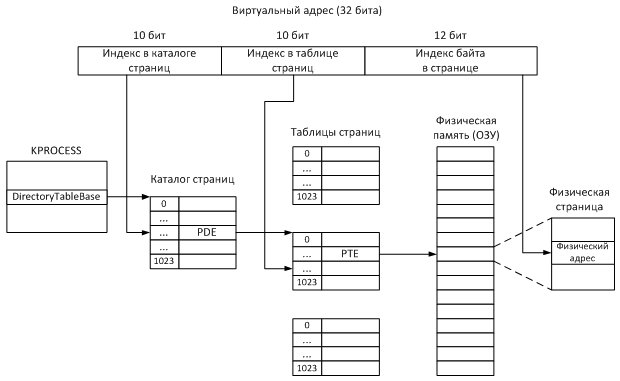
**Трансляция адресов**

*Трансляция* виртуального адреса – это *определение* реального (физического) расположение ячейки памяти с данным виртуальным адресом, т. е. преобразование виртуального адреса в физический. Принцип трансляции показан на [рис.11.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=1#image.11.1), здесь мы рассмотрим подробности трансляции и детали реализации в WRK.

Из [рис.11.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=1#image.11.1) видно, что *информация* о соответствии виртуальных адресов физическим хранится в таблицах страниц. В системе для каждого процесса поддерживается множество записей о страницах: если размер страницы 4 КБ, то чтобы хранить информацию обо всех виртуальных страницах в 32 разрядной системе требуется более миллиона записей (4 ГБ / 4 КБ = 1 048 576). Эти записи о страницах сгруппированы в таблицы страниц (*Page Table*), *запись* называется PTE (*Page Table* *Entry*). В каждой таблице содержится 1024 записи, таким образом, максимальное количество таблиц страниц для процесса – 1024 (1 048 576 / 1024 = 1024). Половина от общего количества – 512 таблиц – отвечают за пользовательское ВАП, другая половина – за системное ВАП.

Таблицы страниц хранятся в виртуальной памяти (см. [рис.11.2](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=1#image.11.2)). *Информация* о расположении каждой из таблиц страниц находится в каталоге страниц (Page *Directory*), единственном для процесса. Записи этого каталога называются PDE (Page *Directory* *Entry*). Таким образом, процесс трансляции является двухступенчатым: сначала *по* виртуальному адресу определяется *запись* PDE в каталоге страниц, затем *по* этой записи находится соответствующая *таблица страниц*, *запись* PTE которой указывает на требуемую страницу в физической памяти.

Откуда процесс знает, где в памяти хранится каталог страниц? За это отвечает *поле* DirectoryTableBase структуры KPROCESS (*файл* base\ntos\inc\ke.h, строка 958, первый элемент массива). Схема трансляции адресов показана на [рис.11.3](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=2#image.11.3).



**Рис. 11.3.**Трансляция адресов

Записи PDE и PTE представлены структурой MMPTE\_HARDWARE (base\ntos\mm\i386\mi386.h, строка 2508), содержащей следующие основные поля:

* флаг (однобитовое поле) Valid: если виртуальная страница расположена в физической памяти, Valid = 1;
* флаг Accessed: если к странице были обращения для чтения, Accessed = 1;
* флаг Dirty: если содержимое страницы было изменено (была произведена операция записи), Dirty = 1;
* флаг LargePage: если страница является большой (4 МБ), LargePage = 1;
* флаг Owner: если страница доступна из пользовательского режима, Owner = 1;
* 20 битовое поле PageFrameNumber: указывает номер страничного фрейма (PFN, Page Frame Number).

В *поле* PageFrameNumber хранится номер записи в базе данных *PFN* – системной структуре, отвечающей за информацию о страницах физической памяти. *Запись* *PFN* представлена структурой MMPFN (*файл* base\ntos\mm\mi.h, строка 1710) и подробно описана в [[5](https://www.intuit.ru/studies/courses/10471/1078/literature#literature.5), стр. 502].

**Ошибки страниц**

Страница может находиться либо в физической памяти (*ОЗУ*), либо на диске в файле подкачки.

Если в записи PTE флаг *Valid* = 1, то страница находится в физической памяти и к ней можно обращаться. Иначе (*Valid* = 0) – страница недоступна процессу. При попытке доступа к такой странице возникает страничная ошибка (*page fault*) и вызывается *функция* MmAccessFault (*файл* base\ntos\mm\mmfault.c, строка 101).

Причин страничных ошибок существует множество (см. [Руссинович и др., 2008, стр. 463]), мы рассмотрим только одну – страница выгружена в страничный *файл* (*файл* подкачки). В этом случае *запись* PTE имеет тип MMPTE\_SOFTWARE (*файл* base\ntos\mm\i386\mi386.h, строка 2446) и вместо поля PageFrameNumber имеет 20 разрядное *поле* PageFileHigh, отвечающее за расположение страницы в страничном файле.

Страничные файлы описываются структурой MMPAGING\_FILE (base\ntos\mm\mi.h, строка 4239), имеющей следующие поля:

* Size – текущий размер файла (в страницах);
* MaximumSize, MinimumSize – максимальный и минимальный размеры файла (в страницах);
* FreeSpace, CurrentUsage – число свободных и занятых страниц;
* PageFileName – имя файла;
* PageFileNumber – номер файла;
* FileHandle – дескриптор файла.

В 32 разрядных *Windows* поддерживается до 16 файлов подкачки размером до 4095 МБ каждый. *Список* файлов подкачки находится в ключе реестра HKLM\SYSTEM\CurrentControlSet\Control\Session *Manager*\*Memory* Management\PagingFiles. Соответствующий системный *массив* MmPagingFile[MAX\_PAGE\_FILES] типа PMMPAGING\_FILE описывается в файле base\ntos\mm\mi.h (строка 8045).

**Пределы памяти**

В таблицах [8.1](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=2#table.11.1), [8.2](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=2#table.11.2) и [8.3](https://www.intuit.ru/studies/courses/10471/1078/lecture/16578?page=2#table.11.3) приведены ограничения на виртуальную и физическую *память* в 32 разрядных и 64 разрядных операционных системах *Windows*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 11.1. Ограничения на виртуальную память | | | | |
| **Тип памяти** | | **32-разрядные Windows** | | **64-разрядные Windows** |
| Виртуальное адресное пространство | | 4 ГБ | | 16 ТБ (16 000 ГБ) |
| Пользовательское ВАП | | 2 ГБ; до 3 ГБ в случае использования специальных ключей при загрузке | | 8 ТБ |
| Системное ВАП | | 2 ГБ; от 1 до 2 ГБ в случае использования специальных ключей при загрузке | | 8 ТБ |
| Таблица 11.2. Ограничения на физическую память в клиентских версиях | | | | |
| **Версия Windows** | **32-разрядные** | | **64-разрядные** | |
| Windows XP | От 512 МБ (Starter) до 4 ГБ (Professional) | | 128 ГБ (Professional) | |
| Windows Vista | от 1 ГБ (Starter) до 4 ГБ (Ultimate) | | от 8 ГБ (Home Basic) до 128 ГБ (Ultimate) | |
| Windows 7 | от 2 ГБ (Starter) до 4 ГБ (Ultimate) | | от 8 ГБ (Home Basic) до 192 ГБ (Ultimate) | |

|  |  |  |
| --- | --- | --- |
| Таблица 11.3. Ограничения на физическую память в серверных версиях | | |
| **Версия Windows** | **32-разрядные** | **64-разрядные** |
| Windows Server 2003 R2 | От 4 ГБ (Standard) до 64 ГБ (Datacenter) | От 32 ГБ (Standard) до 1 ТБ (Datacenter) |
| Windows Server 2008 | От 4 ГБ (Web Server) до 64 ГБ (Datacenter) | От 32 ГБ (Web Server) до 1 ТБ (Datacenter) |
| Windows Server 2008 R2 | нет 32 разрядных версий | от 8 ГБ (Foundation) до 2 ТБ (Datacenter) |

**Резюме**

В лекции изучаются такие важные понятия как виртуальная и *физическая память*, *виртуальное* *адресное пространство*, рабочий набор процесса, *файл* подкачки. Рассматриваются структура виртуального адресного пространства процесса, способы выделения памяти процессам, дескрипторы виртуальных адресов, ошибки страниц. Описывается процесс трансляции виртуальных адресов в физические. Приводятся ограничения на размер виртуальной и физической памяти в различных версиях *Windows*.