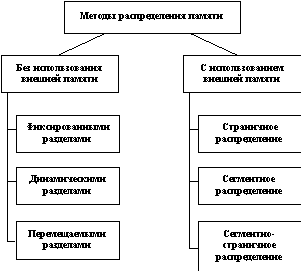
**Операционные системы**

Основы управления памятью

Методы управления памятью

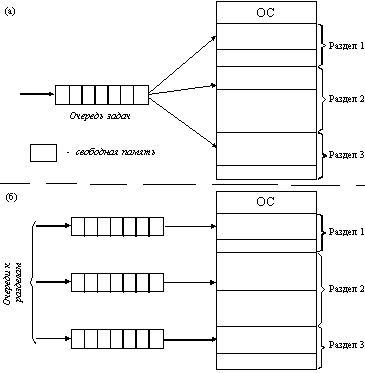


Все методы управления памятью могут быть разделены на два класса: методы, которые используют перемещение процессов между оперативной памятью и диском, и методы, которые не делают этого.

**Основы управления памятью**

Методы распределения памяти без использования дискового пространства

Распределение памяти фиксированными разделами



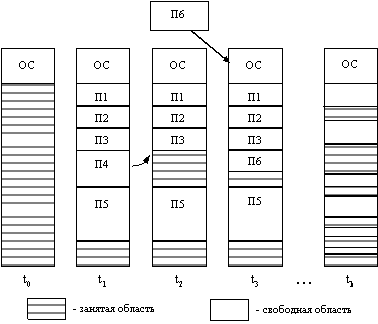
Эта схема была реализована в IBM OS/360, DEC RSX-11 и ряде других систем.

*а - с общей очередью;*

*б - с отдельными очередями*

* Самым простым способом управления оперативной памятью является разделение ее на несколько разделов фиксированной величины. Это может быть выполнено вручную оператором во время старта системы или во время ее генерации. Очередная задача, поступившая на выполнение, помещается либо в общую очередь (рисунок а), либо в очередь к некоторому разделу (рисунок б).
* Подсистема управления памятью в этом случае выполняет следующие задачи:
  + сравнивая размер программы, поступившей на выполнение, и свободных разделов, выбирает подходящий раздел,
  + осуществляет загрузку программы и настройку адресов.
  + При очевидном преимуществе - простоте реализации - данный метод имеет существенный недостаток - жесткость. Так как в каждом разделе может выполняться только одна программа, то уровень мультипрограммирования заранее ограничен числом разделов не зависимо от того, какой размер имеют программы.
* Другим существенным недостатком является то, что предлагаемая схема сильно страдает от ***внутренней фрагментации*** – потери части памяти, выделенной процессу, но не используемой им. *Фрагментация* возникает потому, что процесс не полностью занимает выделенный ему раздел или потому, что некоторые разделы слишком малы для выполняемых пользовательских программ

Распределение памяти разделами переменной величины



* В этом случае память машины не делится заранее на разделы. Сначала вся память свободна. Каждой вновь поступающей задаче выделяется необходимая ей память. Если достаточный объем памяти отсутствует, то задача не принимается на выполнение и стоит в очереди. После завершения задачи память освобождается, и на это место может быть загружена другая задача. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера.
* На рисунке показано состояние памяти в различные моменты времени при использовании динамического распределения. Так в момент t0 в памяти находится только ОС, а к моменту t1 память разделена между 5 задачами, причем задача П4, завершаясь, покидает память. На освободившееся после задачи П4 место загружается задача П6, поступившая в момент t3.

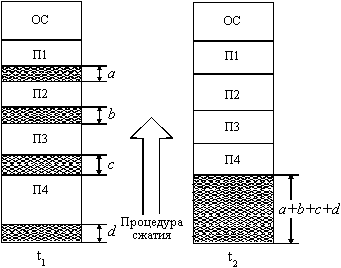
Задачами операционной системы при реализации данного метода управления памятью является:

* ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти,
* при поступлении новой задачи - анализ запроса, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей задачи,
* загрузка задачи в выделенный ей раздел (настройка адресов) и корректировка таблиц свободных и занятых областей,
* после завершения задачи корректировка таблиц свободных и занятых областей.

Программный код не перемещается во время выполнения, то есть может быть проведена единовременная настройка адресов посредством использования перемещающего загрузчика.

* В какой раздел помещать процесс? Наиболее распространены три стратегии.
  + Стратегия первого подходящего (First fit). Процесс помещается в первый подходящий по размеру раздел.
  + Стратегия наиболее подходящего (Best fit). Процесс помещается в тот раздел, где после его загрузки останется меньше всего свободного места.
  + Стратегия наименее подходящего (Worst fit). При помещении в самый большой раздел в нем остается достаточно места для возможного размещения еще одного процесса.
* Моделирование показало, что доля полезно используемой памяти в первых двух случаях больше, при этом первый способ несколько быстрее. Попутно заметим, что перечисленные стратегии широко применяются и другими компонентами ОС, например для размещения файлов на диске.
* По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток - *фрагментация памяти*.
* Фрагментация - это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

Перемещаемые разделы



* Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших либо в сторону младших адресов, так, чтобы вся свободная память образовывала единую свободную область (слайд). В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется "сжатием".
* Сжатие может выполняться либо при каждом завершении задачи, либо только тогда, когда для вновь поступившей задачи нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц, а во втором - реже выполняется процедура сжатия. Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов из виртуальной формы в физическую должно выполняться динамическим способом.
* Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества данного метода.

**Основы управления памятью**

Методы распределения памяти с использованием дискового пространства

Понятие виртуальной памяти

Виртуальная память (ВП) – это совокупность программно-аппаратных средств, позволяющих использовать программы, размер которых превосходит имеющуюся оперативную память.

Для этого менеджер ВП решает следующие задачи:

* размещает данные в запоминающих устройствах разного типа, например, часть программы в оперативной памяти, а часть на диске;
* перемещает по мере необходимости данные между запоминающими устройствами разного типа, например, подгружает нужную часть программы с диска в оперативную память;
* преобразует виртуальные адреса в физические.

Уже достаточно давно пользователи столкнулись с проблемой размещения в памяти программ, размер которых превышал имеющуюся в наличии свободную память. Решением было разбиение программы на части, называемые *оверлеями*. 0-ой оверлей начинал выполняться первым. Когда он заканчивал свое выполнение, он вызывал другой оверлей. Все оверлеи хранились на диске и перемещались между памятью и диском средствами операционной системы. Однако разбиение программы на части и планирование их загрузки в оперативную память должен был осуществлять программист.

Развитие методов организации вычислительного процесса в этом направлении привело к появлению метода, известного под названием *виртуальная память*.

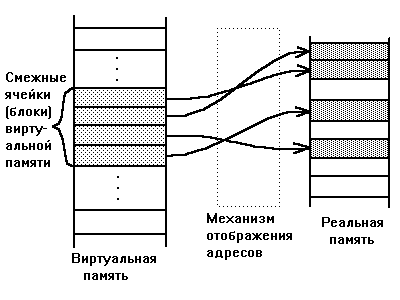
Термин *виртуальная память* обычно ассоциируется с возможностью адресовать пространство памяти, гораздо большее, чем емкость первичной (реальной) памяти конкретной вычислительной машины. Концепция виртуальной памяти является далеко не новой. впервые она была реализована в вычислительной машине Atlas, созданной в Манчестерском университете в Англии в 1960г. Однако широкое распространение системы виртуальной памяти получили относительно недавно. Существуют два наиболее общепринятых способа реализации виртуальной памяти - *страничная* и *сегментная* организации памяти. в некоторых системах применяется либо тот, либо другой из этих способов, а в некоторых - их комбинация.

Физические и виртуальные адреса

Суть концепции виртуальной памяти заключается в том, что адреса, к которым обращается выполняющийся процесс, отделяются от адресов, реально существующих в первичной памяти:

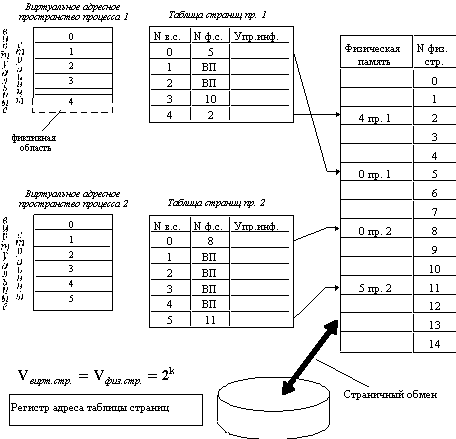
* адреса, на которые делает ссылки выполняющийся процесс, называются *виртуальными адресами (ВА).* Диапазон ВА, к которым может обращаться выполняющийся процесс, называется *пространством виртуальных адресов* V этого процесса.
* адреса, которые существуют в первичной памяти, называются *реальными* (или физическими) *адресами (ФА)*. Диапазон ФА, существующих в конкретной ЭВМ, называется *пространством реальных адресов* R этой ВС.

Отображение адресов

Для установления соответствия между ВА и ФА разработаны различные способы. Так, механизмы *динамического преобразования адресов* (DAT - Dynamic Address Translation) обеспечивают преобразование ВА в ФА время выполнения процесса.

Все подобные системы обладают общим свойством: смежные адреса виртуального адресного пространства процесса не обязательно будут смежными в реальной памяти, это свойство называют “*искусственной смежностью*”.

Страничное распределение



Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами. В общем случае размер виртуального адресного пространства не является кратным размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками).

Размер страницы обычно выбирается равным степени двойки: 512, 1024 и т.д., это позволяет упростить механизм преобразования адресов.

При загрузке процесса часть его виртуальных страниц помещается в оперативную память, а остальные - на диск. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. При загрузке операционная система создает для каждого процесса информационную структуру - таблицу страниц, в которой устанавливается соответствие между номерами виртуальных и физических страниц для страниц, загруженных в оперативную память, или делается отметка о том, что виртуальная страница выгружена на диск. Кроме того, в таблице страниц содержится управляющая информация, такая как признак модификации страницы, признак невыгружаемости (выгрузка некоторых страниц может быть запрещена), признак обращения к странице (используется для подсчета числа обращений за определенный период времени) и другие данные, формируемые и используемые механизмом виртуальной памяти.

При активизации очередного процесса в специальный регистр процессора загружается адрес таблицы страниц данного процесса.

При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение. Если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический. Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди готовых. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить ее в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то решается вопрос, какую страницу следует выгрузить из оперативной памяти.

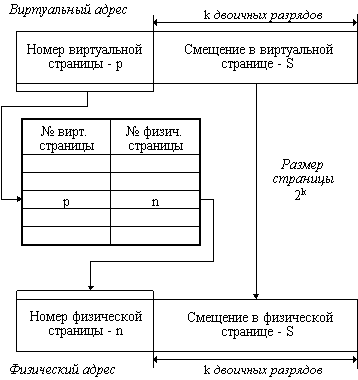
В данной ситуации может быть использовано много разных критериев выбора, наиболее популярные из них следующие:

* дольше всего не использовавшаяся страница,
* первая попавшаяся страница,
* страница, к которой в последнее время было меньше всего обращений.

В некоторых системах используется понятие рабочего множества страниц. Рабочее множество определяется для каждого процесса и представляет собой перечень наиболее часто используемых страниц, которые должны постоянно находиться в оперативной памяти и поэтому не подлежат выгрузке.

После того, как выбрана страница, которая должна покинуть оперативную память, анализируется ее признак модификации (из таблицы страниц). Если выталкиваемая страница с момента загрузки была модифицирована, то ее новая версия должна быть переписана на диск. Если нет, то она может быть просто уничтожена, то есть соответствующая физическая страница объявляется свободной.

Страничное распределение: преобразования ВА в ФА

1. на основании начального адреса таблицы страниц, номера виртуальной страницы и длины записи в таблице страниц определяется адрес нужной записи в таблице,
2. из этой записи извлекается номер физической страницы,
3. к номеру физической страницы присоединяется смещение.

Рассмотрим механизм преобразования виртуального адреса в физический при страничной организации памяти.

Виртуальный адрес при страничном распределении может быть представлен в виде пары (p, s), где p - номер виртуальной страницы процесса (нумерация страниц начинается с 0), а s - смещение в пределах виртуальной страницы. Учитывая, что размер страницы равен 2 в степени к, смещение s может быть получено простым отделением k младших разрядов в двоичной записи виртуального адреса. Оставшиеся старшие разряды представляют собой двоичную запись номера страницы p.

При каждом обращении к оперативной памяти аппаратными средствами выполняются следующие действия:

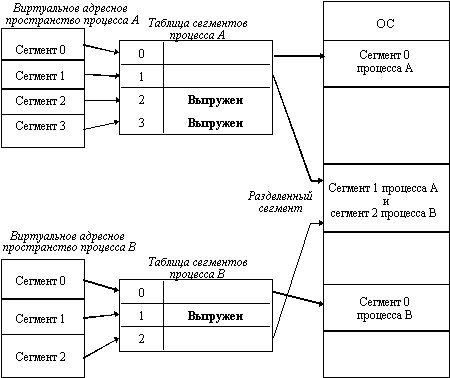
1. на основании начального адреса таблицы страниц (содержимое регистра адреса таблицы страниц), номера виртуальной страницы (старшие разряды виртуального адреса) и длины записи в таблице страниц (системная константа) определяется адрес нужной записи в таблице,
2. из этой записи извлекается номер физической страницы,
3. к номеру физической страницы присоединяется смещение (младшие разряды виртуального адреса).

Использование в пункте (3) того факта, что размер страницы равен степени 2, позволяет применить операцию конкатенации (присоединения) вместо более длительной операции сложения, что уменьшает время получения физического адреса, а значит повышает производительность компьютера.

На производительность системы со страничной организацией памяти влияют временные затраты, связанные с обработкой страничных прерываний и преобразованием виртуального адреса в физический. При часто возникающих страничных прерываниях система может тратить большую часть времени впустую, на свопинг страниц. Чтобы уменьшить частоту страничных прерываний, следовало бы увеличивать размер страницы. Кроме того, увеличение размера страницы уменьшает размер таблицы страниц, а значит уменьшает затраты памяти. С другой стороны, если страница велика, значит велика и фиктивная область в последней виртуальной странице каждой программы. В среднем на каждой программе теряется половина объема страницы, что в сумме при большой странице может составить существенную величину. Время преобразования виртуального адреса в физический в значительной степени определяется временем доступа к таблице страниц. В связи с этим таблицу страниц стремятся размещать в "быстрых" запоминающих устройствах. Это может быть, например, набор специальных регистров или память, использующая для уменьшения времени доступа ассоциативный поиск и кэширование данных.

Страничное распределение памяти может быть реализовано в упрощенном варианте, без выгрузки страниц на диск. В этом случае все виртуальные страницы всех процессов постоянно находятся в оперативной памяти. Такой вариант страничной организации хотя и не предоставляет пользователю виртуальной памяти, но почти исключает фрагментацию за счет того, что программа может загружаться в несмежные области, а также того, что при загрузке виртуальных страниц никогда не образуется остатков.

Сегментное распределение

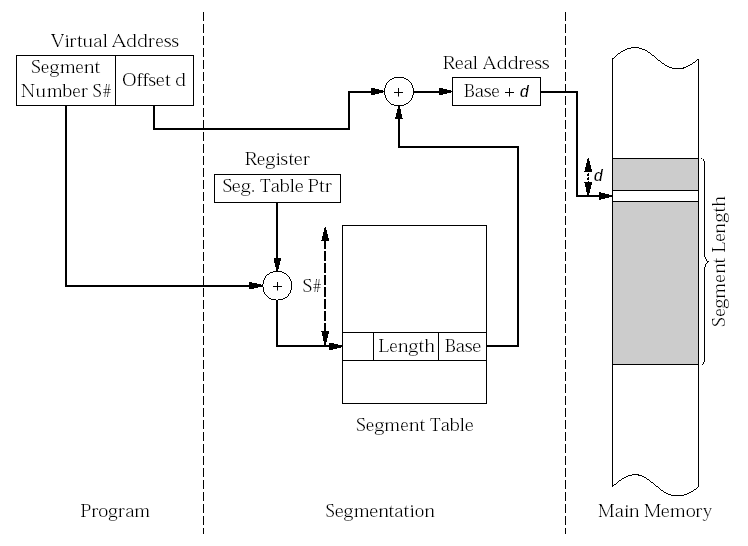


При страничной организации виртуальное адресное пространство процесса делится механически на равные части. Это не позволяет дифференцировать способы доступа к разным частям программы (сегментам), а это свойство часто бывает очень полезным. Например, можно запретить обращаться с операциями записи и чтения в кодовый сегмент программы, а для сегмента данных разрешить только чтение. Кроме того, разбиение программы на "осмысленные" части делает принципиально возможным разделение одного сегмента несколькими процессами. Например, если два процесса используют одну и ту же математическую подпрограмму, то в оперативную память может быть загружена только одна копия этой подпрограммы.

Рассмотрим, каким образом сегментное распределение памяти реализует эти возможности.

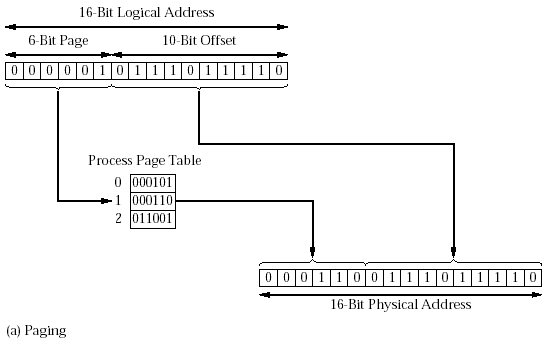
Виртуальное адресное пространство процесса делится на сегменты, размер которых определяется программистом с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т.п. Иногда сегментация программы выполняется по умолчанию компилятором.

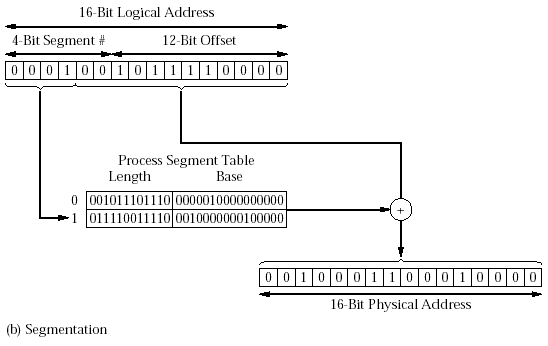
При загрузке процесса часть сегментов помещается в оперативную память (при этом для каждого из этих сегментов операционная система подыскивает подходящий участок свободной памяти), а часть сегментов размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки. Во время загрузки система создает таблицу сегментов процесса (аналогичную таблице страниц), в которой для каждого сегмента указывается начальный физический адрес сегмента в оперативной памяти, размер сегмента, правила доступа, признак модификации, признак обращения к данному сегменту за последний интервал времени и некоторая другая информация. Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок оперативной памяти, в который данный сегмент загружается в единственном экземпляре.

Система с сегментной организацией функционирует аналогично системе со страничной организацией: время от времени происходят прерывания, связанные с отсутствием нужных сегментов в памяти, при необходимости освобождения памяти некоторые сегменты выгружаются, при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Кроме того, при обращении к памяти проверяется, разрешен ли доступ требуемого типа к данному сегменту.

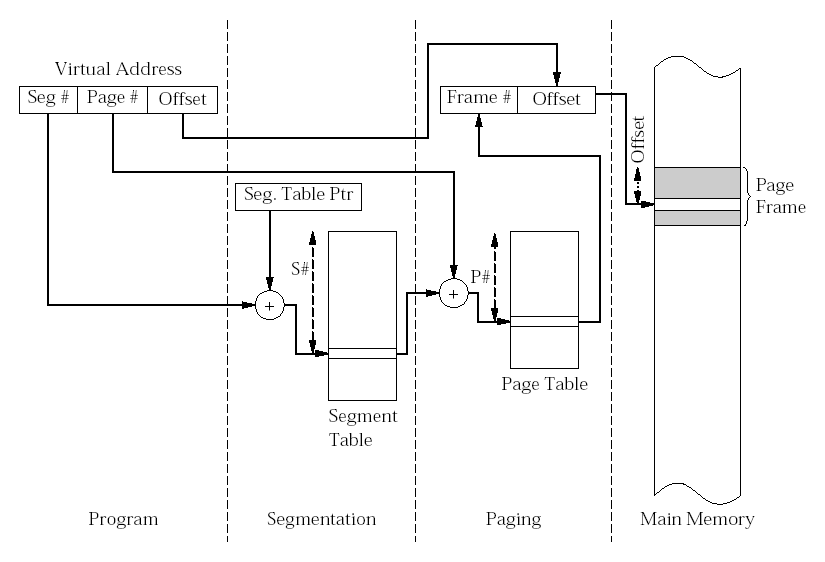
Виртуальный адрес при сегментной организации памяти может быть представлен парой (g, s), где g - номер сегмента, а s - смещение в сегменте. Физический адрес получается путем сложения начального физического адреса сегмента, найденного в таблице сегментов по номеру g, и смещения s.

Недостатком данного метода распределения памяти является фрагментация на уровне сегментов и более медленное по сравнению со страничной организацией преобразование адреса.



****

Странично-сегментное распределение

Как видно из названия, данный метод представляет собой комбинацию страничного и сегментного распределения памяти и, вследствие этого, сочетает в себе достоинства обоих подходов. Виртуальное пространство процесса делится на сегменты, а каждый сегмент в свою очередь делится на виртуальные страницы, которые нумеруются в пределах сегмента. Оперативная память делится на физические страницы. Загрузка процесса выполняется операционной системой постранично, при этом часть страниц размещается в оперативной памяти, а часть на диске. Для каждого сегмента создается своя таблица страниц, структура которой полностью совпадает со структурой таблицы страниц, используемой при страничном распределении. Для каждого процесса создается таблица сегментов, в которой указываются адреса таблиц страниц для всех сегментов данного процесса. Адрес таблицы сегментов загружается в специальный регистр процессора, когда активизируется соответствующий процесс.

На слайде показана схема преобразования виртуального адреса в физический для данного метода.

Алгоритмы замещения страниц (свопинга)

* Замещение случайной страницы
* **FIFO** (First In First Out) – замещение первой использованной страницы
* **LRU** (Least Recently Used) – замещение дольше всех неиспользовавшихся страниц
* **NRU** (Not Recently Used) или clock – замещение не использовавшихся в последнее время страницы
* **LFU** (Least Frequently Used) – замещение наименее часто используемых страниц

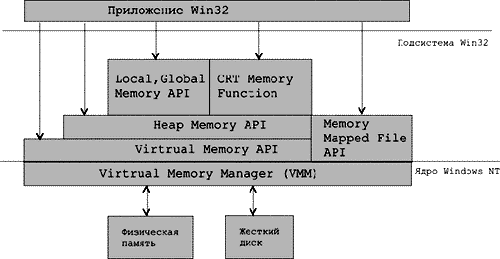
Идеальный алгоритм заключается в том, что бы выгружать ту страницу, которая будет запрошена позже всех.

Но этот алгоритм не осуществим, т.к. нельзя знать какую страницу, когда запросят.

**Операционные системы**

Архитектура памяти в Win32 API. Общие принципы

Архитектура API управления памятью

Диспетчер управления памятью (VMM) является составной частью ядра операционной системы. Приложения не могут получить к нему прямой доступ. Для управления памятью прикладным программам предоставляются различные интерфейсы (API).

Virtual Memory API - набор функций, позволяющих приложению работать с виртуальным адресным пространством: назначать физические страницы блоку адресов и освобождать их, устанавливать атрибуты защиты.

Memory Mapped File API - набор функций, позволяющий работать с файлами, отображаемыми в память. Новый механизм, предоставляемый Win32 API для работы с файлами и взаимодействия процессов.

Heap Memory API - набор функций, позволяющих работать с динамически распределяемыми областями памяти (кучами).

Local, Global Memory API - набор функций работы с памятью, совместимых с 16-битной Windows. Следует избегать их использования.

CRT Memory API - функции стандартной библиотеки языка “С” периода исполнения (runtime).

Два последних набора функций в настоящем курсе не рассматриваются.

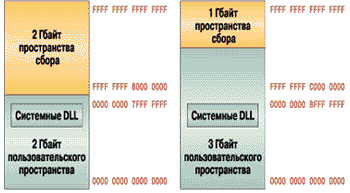
Менеджер виртуальной памяти

**VMM (Virtual Memory Manager)**

* управление виртуальным адресным пространством процесса;
* разделение памяти между процессами;
* защита виртуальной памяти одного процесса от других процессов.

Менеджер ВП обеспечивает для процессов следующие наборы функций:

* управление виртуальным адресным пространством процесса;
* разделение памяти между процессами;
* защита виртуальной памяти одного процесса от других процессов.

Адресное пространство процесса

В Windows NT используется плоская (flat) модель памяти. Каждому процессу выделяется "личное" изолированное адресное пространство. На 32-разрядных компьютерах размер этого пространства составляет 4 Гбайт и может быть расширен до 32 Гбайт при работе Windows NT 5.0 на процессоре Alpha.

В Windows NT 4.0 младшие 2 Гбайт памяти выделяются процессу для произвольного использования, а старшие 2 Гбайт резервируются и используются операционной системой. В младшую часть адресного пространства помещаются и некоторые системные динамически связываемые библиотеки (DLL). Желание расширить доступное процессу адресное пространство привело к тому, что Windows NT 4.0 Enterprise процессу выделяется дополнительный 1 Гбайт за счет сокращения системной области.

Средства защиты памяти

* **Отдельное адресное пространство для каждого процесса.** Аппаратура запрещает процессу доступ к физическим адресам другого процесса.
* **Два режима работы**: режим ядра, в котором процессам разрешен доступ к системным данным, и пользовательский режим, в котором это запрещен.
* **Страничный механизм защиты.** Каждая виртуальная страница имеет набор признаков, который определяет разрешенные типы доступа в пользовательском режиме и в режиме ядра.
* **Объектно-ориентированная защита памяти.** Каждый раз, когда процесс открывает указатель на секцию, монитор ссылок безопасности проверяет, разрешен ли доступ процесса к данному объекту.

Каждый раз, когда нить использует адрес, менеджер ВП вместе с аппаратными средствами транслирует виртуальный адрес в физический. Подсистема виртуальной памяти, управляя процессом трансляции виртуальных адресов, гарантирует, что нить одного процесса не сможет получить доступ к физической странице памяти, относящейся к другому процессу.

В дополнение к прямой защите, обеспечиваемой механизмом трансляции, каждый процессор, который поддерживает виртуальную память, реализует некоторую форму аппаратно-управляемой защиты памяти. Часто аппаратная защита бывает минимальной. Из-за этого менеджер виртуальной памяти Windows NT в гораздо большей степени зависит от аппаратуры, чем другие части операционной системы.

Каждый процесс NT executive имеет большое виртуальное адресное пространство размером в 4Гб, из которых 2 Гб резервируются для системных нужд. (Процессор MIPS R4000 требует, чтобы 2 Гб адресного пространства были зарезервированы для системы. Хотя другие процессоры требуют меньше, для переносимости системы Windows NT всегда резервирует 2 Гб.)

**Операционные системы**

Управление виртуальной памятью в Win32

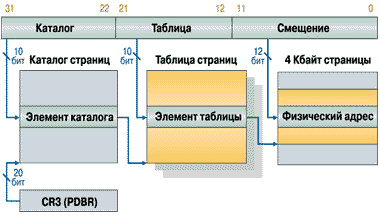
Каталог страниц и свопинг

Каждому процессу назначается свой каталог страниц. Именно поэтому адресное пространство каждого процесса изолировано, что очень хорошо с точки зрения защиты процессов друг от друга.

Для того, чтобы обеспечить все линейное адресное пространство процесса физическими ячейками памяти. Windows NT-2003 применяет свопинг (swapping).

Организацией свопинга занимается VMM. При генерации системы на диске образуется специальный файл свопинга, куда записываются те страницы, которым не находится места в физической памяти. Процессы могут захватывать память в своем 32-битном адресном пространстве и, затем, использовать ее. Страница может иметь различные состояния.

VMM использует алгоритм LRU (Least Recently Used) – замещение дольше всех неиспользовавшихся страниц.

Конечно, компьютер может и не иметь 4 Гбайт оперативной памяти, адресуемых процессорами Intel для того, чтобы обеспечить все линейное адресное пространство процесса физическими ячейками памяти. Windows NT, как и все другие операционные системы, применяет свопинг (swapping). Не используемые в конкретный момент страницы памяти могут быть вытеснены на диск в так называемый файл подкачки. В соответствующем элементе таблицы страниц эта страница помечается как отсутствующая, и при попытке обращения к ней возникает исключительная ситуация - "сбой" страницы. Обрабатывая ее, операционная система находит страницу на диске и переписывает ее в память, соответствующим образом подстраивая элемент таблицы страниц. После этого попытка выполнить команду, вызвавшую исключение, повторяется.

Страничное преобразование

180-2Виртуальная память в Windows NT имеет страничную организацию, принятую во многих современных операционных системах. В общем виде схема страничной организации описывается следующим образом: линейный адрес разбивается на несколько частей. Старшая часть адреса содержит в себе номер элемента в корневой таблице. Этот элемент содержит адрес таблицы следующего уровня. Следующая часть линейного адреса содержит номер элемента уже в этой таблице и так далее, до последней таблицы, которая содержит номер физической страницы. А самая младшая часть адреса уже является номером байта в этой физической странице.

Процессоры Intel начиная с Pentium Pro позволяют операционным системам применять одно-, двух- и трехступенчатые схемы. И даже разрешается одновременное использование страниц различного размера. Эта возможность, конечно, повысила бы эффективность страничного преобразования, будь она внедрена в Windows NT. Увы, эта ОС возникла раньше и поддерживает только двухступенчатую схему преобразования с фиксированным размером страниц. Размер страниц для платформы Intel составляет 4 Кбайт, а для DEC Alpha - 8 Кбайт. Дополнительные схемы, которые поддерживает процессор Pentium Pro, не используются.

32-разрядный линейный адрес разбивается на три части. Старшие 10 разрядов адреса определяют номер одного из 1024 элементов в каталоге страниц, адрес которого находится в регистре процессора CR3. Этот элемент содержит физический адрес таблицы страниц. Следующие 10 разрядов линейного адреса определяют номер элемента таблицы. Элемент, в свою очередь, содержит физический адрес страницы виртуальной памяти. Размер страницы - 4 Кбайт, и младших 12 разрядов линейного адреса как раз хватает (212 = 4096), чтобы определить точный физический номер адресуемой ячейки памяти внутри этой страницы.

Элемент таблицы страниц

* Защита – Win32 API поддерживает ряд значений, в том числе: *PAGE\_NOACCESS, PAGE\_READONLY, PAGE\_READWRITE, PAGE\_EXECUTE*.
* Базовый физический адрес страницы в памяти.
* Pagefile – индекс используемого файла подкачки (один из 16 возможных в системе файлов).
* State – состояние страницы в системе:
  + T (Transition) – отмечает страницу как переходную;
  + D (Dirty) – страница, в которую была произведена запись;
  + P (Present) – страница присутствует в ОП или находится в файле подкачки.

Отдельные состояния страниц

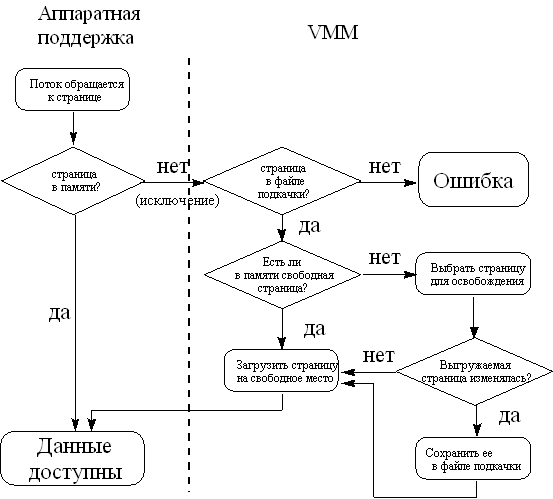
* Valid – страница используется процессом. Она реально существует в ОП и помечена в PTE как присутствующая в рабочем множестве процесса (P=1, D=0,1).
* Modified – содержимое страницы было изменено (D=1). В PTE страница помечена как отсутствующая (P=0) и переходная (T=1).
* Standby – содержимое страницы не изменялось (D=0). В PTE страница помечена как отсутствующая (P=0) и переходная (T=1).
* Free – страница, на которую не ссылается ни один PTE. Страница свободна, но подлежит обнулению, прежде чем будет использована.
* Zeroed – свободная и обнуленная страница, пригодная к непосредственному использованию любым процессом.
* Bad – страница, которая вызывает аппаратные ошибки и не может быть использована ни одним процессом.

Отдельные состояния страниц

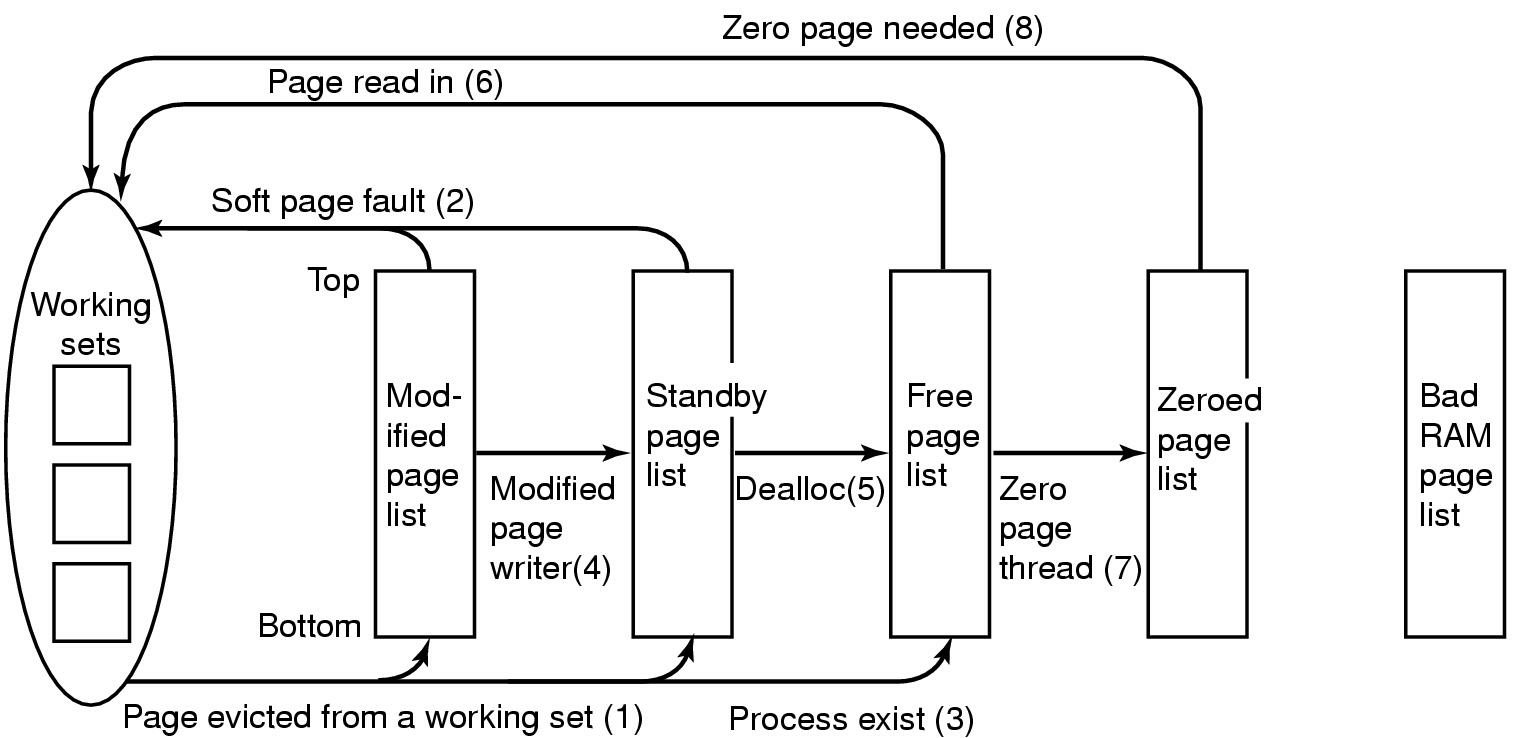
|  |  |  |  |
| --- | --- | --- | --- |
| **T** | **D** | **P** | **Page state** |
| 0 | - | 0 | Invalid page (Free) |
| - | 0 | 1 | Valid page |
| - | 1 | 1 | Valid dirty page |
| 1 | 0 | 0 | Invalid page in transition (Standby) |
| 1 | 1 | 0 | Invalid dirty page in transition (Modified) |

Реализация свопинга

С понятием свопинга связаны три стратегии:

* *выборка* (fetch);
* *размещение* (placement);
* *замещение* (replacement).
* Выборка определяет, в какой момент необходимо переписать страницу с диска в память. В Windows NT используется классическая схема выборки с упреждением: система переписывает в память не только выбранную страницу, но и несколько следующих по принципу пространственной локальности, гласящему: наиболее вероятным является обращение к тем ячейкам памяти, которые находятся в непосредственной близости от ячейки, к которой производится обращение в настоящий момент. Поэтому вероятность того, что будут востребованы последовательные страницы, достаточна высока. Их упреждающая подкачка позволяет снизить накладные расходы, связанные с обработкой прерываний.
* Размещение определяет, в какое место оперативной памяти необходимо поместить подгружаемую страницу. Для систем со страничной организацией данная стратегия практически не имеет никакого значения, и поэтому Windows NT выбирает первую попавшуюся свободную страницу.
* Замещение начинает действовать с того момента, когда в оперативной памяти компьютера не остается свободного места для размещения подгружаемой страницы. В этом случае необходимо решить, какую страницу вытеснить из физической памяти в файл подкачки. Механизм замещения заслуживает более пристального внимания.
* Периодически VMM просматривает список страниц с атрибутом Valid и пытается похитить их у процесса. Он помечает их как отсутствующие (P=0), но на самом деле оставляет их на месте, только переводит в разряд Modified или Standby в зависимости от значения бита D из PTE. Если похищенная страница принадлежит рабочему множеству, то к ней в ближайшее время произойдет обращение. Это, конечно, вызовет исключение - ведь страница-то помечена как отсутствующая. Но VMM очень быстро сделает эту страницу вновь доступной процессу, поскольку она реально находится в памяти. Если страница находится вне рабочего множества, то обращений к ней не будет и она со временем перейдет в разряд Free, а затем Zeroed и станет доступна другим процессам системы.

Переходы между состояниями страниц



**Архитектура памяти в Win32 API**

Организация «статической» виртуальной памяти

Работа приложений с виртуальной памятью

Блок адресов в адресном пространстве процесса может находиться в одном из трех состояний

* Выделен (committed) – блоку адресов назначена физическая память либо часть файла подкачки.
* Зарезервирован (reserved) – блок адресов помечен как занятый, но физическая память не распределена.
* Свободен (free) – блок адресов не выделен и не зарезервирован.

Функции API для работы виртуальной памятью

* ***VirtualAlloc***
* ***VirtualFree***
* ***VirtualQuery***
* ***VirtualLock***
* ***VirtualUnlock***
* ***VirtualProtect***
* ***VirtualProtectEx***

Для резервирования региона памяти в адресном пространстве процесса или выделения ее используется функция VirtualAlloc, а для освобождения – функция VirtualFree.

Выделенные страницы можно заблокировать в памяти, т.е. запретить их вытеснение в файл подкачки. Для этих целей служит пара функций VirtualLock и VirtualUnlock. Процессу не разрешается блокировать более 30 страниц.

Для изменения атрибутов защиты регионов используются функции VirtualProtect и VirtualProtectEx. Причем, первая позволяет изменять атрибуты защиты в адресном пространстве текущего процесса, а вторая – произвольного.

*LPVOID* ***VirtualAlloc*** *(*

*//* адрес, по которому надо зарезервировать или выделить память

*LPVOID lpAddress,*

*//* размер выделяемого региона

*DWORD dwSize,*

*//* тип распределения памяти

*DWORD flAllocationType,*

*//* тип защиты доступа

*DWORD flProtect*

*);*

Функция VirtualAlloc возвращает адрес выделенного региона или NULL в случае неудачи.

Параметры:

**lpAddress** - адрес, по которому надо зарезервировать или выделить память. Если этот параметр равен NULL, то система самостоятельно выбирает место в адресном пространстве процесса;

**dwSize** - размер выделяемого региона;

Параметр **flAllocationType** может принимать следующие значения:

**MEM\_RESERVE** - резервирует блок адресов без выделения памяти;

**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. В Windows 95 игнорируется.

**MEM\_DECOMMIT** - освободить выделенную память;

**MEM\_RELEASE** - освободить зарезервированный регион. При использовании этого флага параметр dwSize должен быть равен нулю.

**flProtect** - тип защиты доступа выделяемого региона:

**PAGE\_READONLY** - допускается только чтение;

**PAGE\_READWRITE** - допускается чтение и запись;

**PAGE\_EXECUTE** - допускается только выполнение;

**PAGE\_EXECUTE\_READ** - допускается исполнение и чтение;

**PAGE\_EXECUTE\_READWRITE** - допускается выполнение, чтение и запись;

**PAGE\_GUARD** - дополнительный флаг защиты, который комбинируется с другими флагами. При первом обращении к странице этот флаг сбрасывается и возникает исключение STATUS\_GUARD\_PAGE. Этот флаг используется для контроля размеров стека с возможностью его динамического расширения;

**PAGE\_NOCACHE** - запрещает кэширование страниц. Может быть полезен при разработке драйверов устройств (например, данные в видеобуфер должны переписываться сразу, без кэширования).

* Параметр **flAllocationType** может принимать следующие значения:
* **MEM\_RESERVE** - резервирует блок адресов без выделения памяти;
* **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. В Windows 95 игнорируется.
* **MEM\_DECOMMIT** - освободить выделенную память;
* **MEM\_RELEASE** - освободить зарезервированный регион. При использовании этого флага параметр dwSize должен быть равен нулю.
* **flProtect** - тип защиты доступа выделяемого региона:
* **PAGE\_READONLY** - допускается только чтение;
* **PAGE\_READWRITE** - допускается чтение и запись;
* **PAGE\_EXECUTE** - допускается только выполнение;
* **PAGE\_EXECUTE\_READ** - допускается исполнение и чтение;
* **PAGE\_EXECUTE\_READWRITE** - допускается выполнение, чтение и запись;
* **PAGE\_GUARD** - дополнительный флаг защиты, который комбинируется с другими флагами. При первом обращении к странице этот флаг сбрасывается и возникает исключение STATUS\_GUARD\_PAGE. Этот флаг используется для контроля размеров стека с возможностью его динамического расширения;
* **PAGE\_NOCACHE** - запрещает кэширование страниц. Может быть полезен при разработке драйверов устройств (например, данные в видеобуфер должны переписываться сразу, без кэширования).

LPVOID VirtualAllocEx(

HANDLE hProcess,

LPVOID lpAddress,

SIZE\_T dwSize,

DWORD flAllocationType,

DWORD flProtect);

BOOL WriteProcessMemory(

HANDLE hProcess,

LPVOID lpBaseAddress,

LPCVOID lpBuffer,

SIZE\_T nSize,

SIZE\_T\* lpNumberOfBytesWritten);

*BOOL* ***VirtualFree*** *(*

*//* адрес региона, который надо освободить

*LPVOID lpAddress,*

*//* размер освобождаемого региона

*DWORD dwSize,*

*//* тип освобождения

*DWORD dwFreeType*

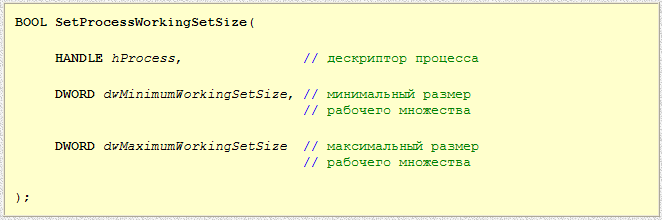
*);*

* ***dwSize*** *- размер, если мы будем использовать тип освобождения, как* ***MEM\_RELEASE****, то размер должен быть установлен в 0.*
* ***dwFreeType*** *- будет определять какая операция произойдет с памятью:* 
  + ***MEM\_RELEASE*** *- освобождена*
  + ***MEM\_DECOMMIT*** *- зарезервированные, но не используемые.*
* *При успешном выполнении функция вернет ненулевое значение.*

Функция VirtualFree Возвращает TRUE в случае успеха и FALSE - в случае неудачи.

* Функция **VirtualLock()** позволят предотвратить запись памяти на диск.
  + BOOL VirtualLock ( LPVOID lpAddress, *// адрес начала памяти* SIZE\_T dwSize *// количество байтов* );
* Если фиксация больше не нужна, то ее можно убрать функцией **VirtualUnlock()**.
  + BOOL VirtualUnlock ( LPVOID lpAddress, *// адрес начала памяти* SIZE\_T dwSize *// количество байтов* );
* При успешном выполнении обе функции возвращают ненулевое значение.
* Процессу не разрешается блокировать более 30 страниц.
* Для настройки рабочего множества процесса может использоваться и функция SetProcessWorkingSetSize(), которая может снять ограничение.

SetProcessWorkingSetSize



дескриптор должен иметь права доступа **PROCESS\_SET\_QUOTA**

Если значения или ***dwMinimumWorkingSetSize*** или ***dwMaximumWorkingSetSize*** больше, чем текущий размер рабочего множества памяти процессов, заданный процесс должен иметь привилегию **SE\_INC\_BASE\_PRIORITY\_NAME**.

**Архитектура памяти в Win32 API**

Организация «динамической» виртуальной памяти

Кучи (heaps)

Кучи (heaps) – это динамически распределяемые области данных.

* HANDLE GetProcessHeap( VOID ) – для получения дескриптора кучи по умолчанию;
* LPVOID HeapAlloc( HANDLE hHeap, DWORD dwFlags, DWORD dwSize ) – для выделения из кучи блока памяти заданного размера и возвращения указателя;
* LPVOID HeapReAlloc( HANDLE hHeap, DWORD dwFlags, LPVOID lpOldBlock, DWORD dwSize) – для изменения размера выделенного блока памяти с возможностью перемещения блока при необходимости;
* BOOL HeapFree(HANDLE hHeap, DWORD dwFlags, LPVOID lpMem ) – для освобождения выделенного блока памяти кучи.

При порождении процесса ему предоставляется куча размером 1 Мбайт по умолчанию. Ее размер может изменяться параметром /HEAP при построении исполняемого модуля. Функции библиотеки времени исполнения компилятора (malloc(), free() и т. д.) используют возможности куч.

Создание новых куч

* **для “защиты” друг от друга различных структур данных**;
* **для повышения эффективности управления памятью**. В системах со страничной организацией отсутствует проблема фрагментации физической памяти. Однако существует проблема фрагментации адресного пространства. В 4Gb адресном пространстве эта проблема не актуальна, но она имеет значение в 1Mb куче. Если элементы какой-либо структуры имеют один размер, а элементы другой структуры - другой размер, то полезно размещать эти структуры в разных кучах.
* **для уменьшения рабочего множества процесса**. В соответствии с принципом локальности, работа с разными структурами, чаще всего, происходит не одновременно. Границы элементов разных структур не выровнены на границу страницы. Обращение к элементам одной структуры вызывает подкачку всей страницы, а, значит и элементов другой структуры. Это увеличивает рабочее множество процесса.
* **для повышения эффективности работы многонитевых приложений;**
* **для быстрого освобождение всей памяти в куче**.

HANDLE HeapCreate (DWORD dwFlags, DWORD dwInitialSize, DWORD dwMaximumSize).

Параметр *fdwOptions* модифицирует способ выполнения операций над кучей. В нем можно указать 0, HEAP\_NO\_SERIALIZE, HEAP\_GENERATE\_EXCEPTIONS или комбинацию последних двух флагов.

BOOL HeapDestroy ( HANDLE hHeap);

|  |  |
| --- | --- |
| Флаг | Назначение |
| HEAP\_GENERATE\_EXCEPTIONS[1)](http://www.podgoretsky.com/ftp/Docs/Delphi/Memory/heap/90-footnote.html) | Указывает системе на то, что в случае возникновения ошибки необходимо генерировать исключительную ситуацию. Это будет происходить во всех случаях, когда функция должна была бы возвратить значение nil. |
| HEAP\_NO\_SERIALIZE[1)](http://www.podgoretsky.com/ftp/Docs/Delphi/Memory/heap/90-footnote.html) | Указывает, что пока выполняется текущий вызов HeapAlloc, к куче не будут происходить обращения из других потоков.  Важно! Это значение не может быть указано при обращении к куче процесса. Система может создавать свои собственные потоки без участия приложения, например такие, как обработчик нажатия Ctrl+C, которые могут выполнять доступ к куче процесса. |
| HEAP\_ZERO\_MEMORY[1)](http://www.podgoretsky.com/ftp/Docs/Delphi/Memory/heap/90-footnote.html) | Указывает, что выделяемая память должна инициализироваться нулями. В противном случае память не обязатльно инициализируется нулями. |

Дополнительные возможности по управлению кучами

* UINT HeapCompact( HANDLE hHeap, DWORD fdwFlags);
* BOOL HeapLock(HANDLE hHeap);   
  BOOL HeapUnlock(HANDLE hHeap);
* BOOL HeapWalk( HANDLE hHeap, PPROCESS\_HEAP\_ENTRY pHeapEntry);

**Архитектура памяти в Win32 API**

Файлы, проецируемые в память

Проецируемые файлы

“Как и виртуальная память, проецируемые файлы позволяют резервировать регион адресного пространства и передавать ему физическую память. Различие между этими механизмами состоит в том, что в последнем случае физическая память не выделяется из системного страничного файла, а берется из файла, уже находящегося на диске. Как только файл спроецирован в память, к нему можно обращаться так, как будто он в нее целиком загружен.”

(Джеффри Рихтер. Windows для профессионалов.)

Файлы, проецируемые (отображаемые) в память, - это один из самых замечательных сервисов, которые Win32 предоставляет программисту. Его существование стирает для программиста грань между оперативной и дисковой памятью. Действительно, с точки зрения классической теории кэш, оперативная память и дисковое пространство - это три вида памяти, отличающиеся скоростью доступа и размером. Но если заботу о перемещении данных между кэшем и оперативной памятью берут на себя процессор и операционная система, то перемещение данных между оперативной памятью и диском обычно выполняет прикладной процесс с использованием функций read() и write(). Win32 действует иначе: операционная система берет на себя заботу о перемещении страниц адресного пространства процесса, находящихся в файле подкачки, причем в качестве файла подкачки может быть использован любой файл. Иначе говоря, страницы виртуальной памяти любого процесса могут быть помечены как выгруженные, а в качестве места, куда они выгружены, может быть указан файл. Теперь при обращении к такой странице VMM произведет ее загрузку, используя стандартный механизм свопинга. Это позволяет работать с произвольным файлом как с регионом памяти.

Применение проецируемых файлов

Этот механизм имеет три применения в Win32:

* Для запуска исполняемых файлов (EXE) и динамически связываемых библиотек (DLL).
* Для работы с файлами.
* Для одновременного использования одной области данных двумя процессами.

Запуск исполняемых файлов и DLL

При исполнении функции ***CreateProcess*** система обращается к VMM для выполнения следующих действий

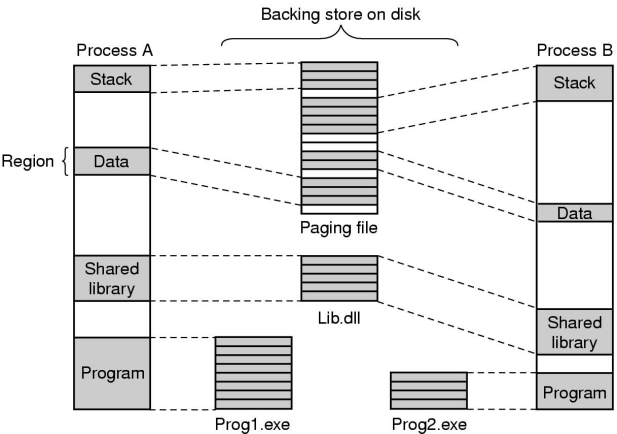
1. Создать адресное пространство процесса (размером 4Gb).
2. Резервировать в адресном пространстве процесса регион размером, достаточным для размещения исполняемого файла. Начальный адрес региона определяется в заголовке EXE-модуля. Обычно он равен 0x00400000.
3. Отобразить исполняемый файл на зарезервированное адресное пространство.
4. Таким же образом отобразить на адресное пространство процесса необходимые ему динамически связываемые библиотеки. Информация о необходимых библиотеках находится в заголовке EXE-модуля. Желательное расположение региона адресов описано внутри библиотеки.

При исполнении функции CreateProcess система обращается к VMM для выполнения следующих действий

1. Создать адресное пространство процесса (размером 4Gb)
2. Резервировать в адресном пространстве процесса регион размером, достаточным для размещения исполняемого файла. Начальный адрес региона определяется в заголовке EXE-модуля. Обычно он равен 0x00400000, но может быть изменен при построении файла параметром /BASE компоновщика.
3. Отобразить исполняемый файл на зарезервированное адресное пространство. Тем самым VMM распределяет физические страницы не из файла подкачки, а непосредственно из EXE-модуля.
4. Таким же образом отобразить на адресное пространство процесса необходимые ему динамически связываемые библиотеки. Информация о необходимых библиотеках находится в заголовке EXE-модуля. Желательное расположение региона адресов описано внутри библиотеки. Visual C++, например, устанавливает по умолчанию адрес 0x10000000. Этот адрес может так же изменяться параметром /BASE компоновщика. Если при загрузке выясняется, что данный регион занят, то система попытается переместить библиотеку в другой регион адресов, на основе настроечной информации, содержащейся в DLL-модуле. Однако эта операция снижает эффективность системы и, кроме того, если настроечная информация удалена при компоновке библиотеки параметром /FIXED, то загрузка становится вообще невозможной.

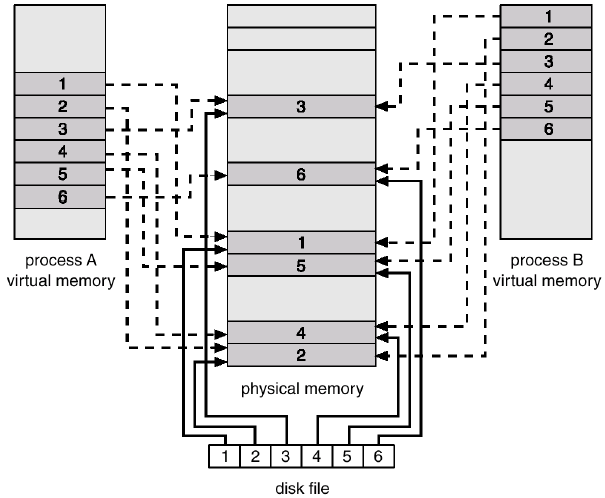
При одновременном запуске нескольких приложений Win32 отображает один и тот же исполняемый файл и библиотеки на адресные пространства различных процессов. При этом возникает проблема независимого использования процессами статических переменных и областей данных. Кроме того, изменение данных исполняющейся программой не должно приводить к изменению EXE-файла. Win32 откладывает решение этой проблемы на максимально возможный срок (Lazy Evaluation). При этом используется классический механизм отложенного копирования (copy-on-write - копирование при попытке записи). Все страницы адресного пространства процесса получают атрибут защиты PAGE\_WRITECOPY. При попытке записи в такую страницу возникает исключение нарушения защиты и VMM копирует страницу для обратившегося процесса. В дальнейшем эта страница будет выгружаться в файл подкачки. После копирования происходит рестарт команды, вызвавшей исключение.

Запуск EXE-файлов и DLL-библиотек



The *lib.dll* file is mapped into two address spaces at same time.

Одновременное использование одной области данных двумя процессами



Файлы данных, проецируемые в память

**Проецирование файла данных в память:**

* Создается объект ядра “файл”. Для создания объекта “файл” используется функция ***CreateFile***.
* С помощью функции ***CreateFileMapping*** создается объект ядра “проецируемый файл”. При этом используется дескриптор файла, возвращенный функцией CreateFile.
* Производится отображение объекта “проецируемый файл” или его части на адресное пространство процесса. Для этого применяется функция **MapViewOfFile**.

**Завершение проецирования файла данных:**

* Выполняется открепление файла от адресного пространства процесса с помощью функции ***UnmapViewOfFile***.
* Выполняется уничтожение объектов “файл” и “проецируемый файл” с помощью функции ***CloseHandle***.

Проецирование файла данных в адресное пространство процесса предоставляет мощный механизм работы с файлами. Спроецировав файл на адресное пространство процесса, программа получает возможность работать ним, как с массивом. Проецирование файла в память выполняется в три этапа.

Для открепления файла от адресного пространства процесса используется функция UnmapViewOfFile, а для уничтожения объектов “файл” и “проецируемый файл” - функция CloseHandle.

**HANDLE CreateFileMapping(**

**HANDLE** *hFile***,** // дескриптор файла

**LPSECURITY\_ATTRIBUTES** *lpAttributes***,** // защита

**DWORD** *flProtect***,** // атрибуты защиты

**DWORD** *dwMaximumSizeHigh***,** // старшее слово размера

**DWORD** *dwMaximumSizeLow***,** // младшее слово размера

**LPCTSTR** *lpName* // имя объекта **);**

**HANDLE OpenFileMapping(**

**DWORD dwDesiredAccess, // режим доступа**

**BOOL bInheritHandle, // флажок наследования**

**LPCTSTR lpName // имя объекта );**

typedef struct \_SECURITY\_ATTRIBUTES {  
DWORD nLength,  
LPVOID lpSecurityDescriptor;  
BOOL bInheritHandle;  
} SECURITY\_ATTRIBUTES;

**LPVOID MapViewOfFile(**

**HANDLE hFileMappingObject, // дескр. объекта проецируемый файл**

**DWORD dwDesiredAccess, // режим доступа**

**DWORD dwFileOffsetHigh, // старшее DWORD смещения**

**DWORD dwFileOffsetLow, // младшее DWORD смещения**

**SIZE\_T dwNumberOfBytesToMap // число отображаемых байтов );**

**LPVOID MapViewOfFileEx(**

**HANDLE** *hFileMappingObject***,** // дескриптор отображаемого объекта

**DWORD** *dwDesiredAccess***,** // режим доступа

**DWORD** *dwFileOffsetHigh***,** // старшее DWORD смещения

**DWORD** *dwFileOffsetLow***,** // младшее DWORD смещения

**SIZE\_T** *dwNumberOfBytesToMap***,** // число отображаемых байтов

**LPVOID** *lpBaseAddress* // начальный адрес **);**

**BOOL UnmapViewOfFile(**

**LPCVOID** *lpBaseAddress* // начальный адрес **);**

Пример работы с проецированными файлами

HANDLE hFile, hFileMapping;

PVOID pMassive;

hFile = CreateFile( “File Name”, ... );

hFileMapping = CreateFileMapping( hFile, ... );

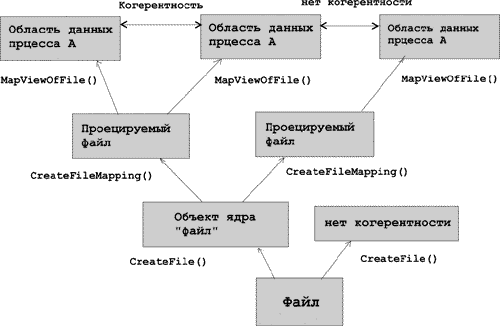
CloseHandle( hFile ) ;

pMassive = MapViewOfFile( hFileMapping, ... );

//Здесь производится работа с массивом pMassive

UnmapViewOfFile( pMassive );

Взаимодействие процессов через общую область данных



Для обеспечения *когерентности* процессы должны работать с одним объектом “проецируемый файл”, а не с одним файлом.

Два процесса могут совместно использовать объект “проецируемый файл”. При этом, при помощи функции *MapViewOfFile* каждый процесс проецирует этот объект на свое адресное пространство и используют эту часть адресного пространства как разделяемую область данных. Общий механизм таков: один процесс создает объект “проецируемый файл” с помощью функции *CreateFileMapping* и порождает другой процесс, передавая ему в наследство описатель этого объекта. Дочерний процесс может пользоваться этим описателем наравне с родительским. Проблема состоит только в том, как сообщить дочернему процессу, какой из переданных ему в наследство описателей является описателем “проецируемого файла”. Это можно сделать любым способом. Например передачей параметров при запуске процесса, через переменные среды, передачей сообщения в главное окно процесса и так далее.

Общая область данных может быть создана не только путем проецирования файла, но и путем проецирования части файла подкачки. Для этого в функцию CreateFileMapping необходимо передать в качестве параметра не описатель ранее открытого файла, а -1. В этом случае необходимо задать размеры выделяемой области. Кроме того, в параметре lpName можно задать имя объекта, которое является глобальным в системе. Если это имя задается в системе впервые, то процессу выделяется новая область данных, а если имя было уже задано, то именованная область данных предоставляется для совместного использования.

Если один процесс меняет разделяемую область данных, то она меняется и для другого процесса. Операционная система обеспечивает ***когерентность*** разделяемой области данных для всех процессов. Но для обеспечения когерентности процессы должны работать с одним объектом “проецируемый файл”, а не с одним файлом.