

## **ТЕМА 3.1. УПРАВЛЕНИЕ ПАМЯТЬЮ**

В данной теме рассматриваются следующие вопросы:

- Организация памяти, основные механизмы управления памятью.
- Адресное пространство процесса.
- Концепция рабочего множества.
- Защита памяти.
- Классификация запоминающих устройств.
- Иерархия памяти.
- Оперативные и постоянные запоминающие устройства.

Лекции – 2 часа, лабораторные занятия – 2 часа, самостоятельная работа – 2 часа.

Экзаменационные вопросы по теме:

- Управление памятью: адресное пространство процесса, организация памяти, основные механизмы управления памятью, концепция рабочего множества.
- Классификация запоминающих устройств. Иерархия памяти. Оперативные и постоянные запоминающие устройства.

### 3.1.1. Организация памяти, основные механизмы управления памятью

Стоимость памяти обычно увеличивается с ростом ее объема и быстродействия. В то же время при выполнении не вся эта память необходима одновременно. Совершенно логично редко используемые данные хранить в более медленной и дешевой памяти, а данные, которые потребуются процессору в самом ближайшем будущем. Концепция иерархии памяти, согласно которой компьютеры обладают несколькими мегабайтами очень быстродействующей, дорогой и энергозависимой кэш-памяти, несколькими гигабайтами памяти, средней как по скорости, так и по цене, а также несколькими терабайтами памяти на довольно медленных, сравнительно дешевых дисковых накопителях, не говоря уже о сменных накопителях, таких как DVD и флеш-устройства USB. Превратить эту иерархию в абстракцию, то есть в удобную модель, а затем управлять этой абстракцией — и есть задача операционной системы [1].

Часть операционной системы, которая управляет иерархией памяти (или ее частью), называется менеджером, или диспетчером, памяти. Он предназначен для действенного управления памятью и должен следить за тем, какие части памяти используются, выделять память процессам, которые в ней нуждаются, и освобождать память, когда процессы завершат свою работу.

Физические адреса — это реальные адреса, используемые для выбора микросхем физической памяти, содержащих данные. Физическая память организована в виде последовательности 8-разрядных байтов. Каждому байту присвоен уникальный адрес, который может изменяться от 0 до  $2^{32}-1$  (или  $2^{36}-1$  у P6+). Физическая память составляет единое целое с компьютером и управляется процессором (с некоторой помощью средств прямого доступа к памяти). Организационные методы распределения памяти предоставляют программистам возможность эффективного использования всей компьютерной системы [2].

При использовании средств процессора для управления памятью, программа может использовать одну из трех моделей доступа к памяти: сплошная ("плоская") модель памяти, сегментированная модель памяти, модель режима реального адреса.

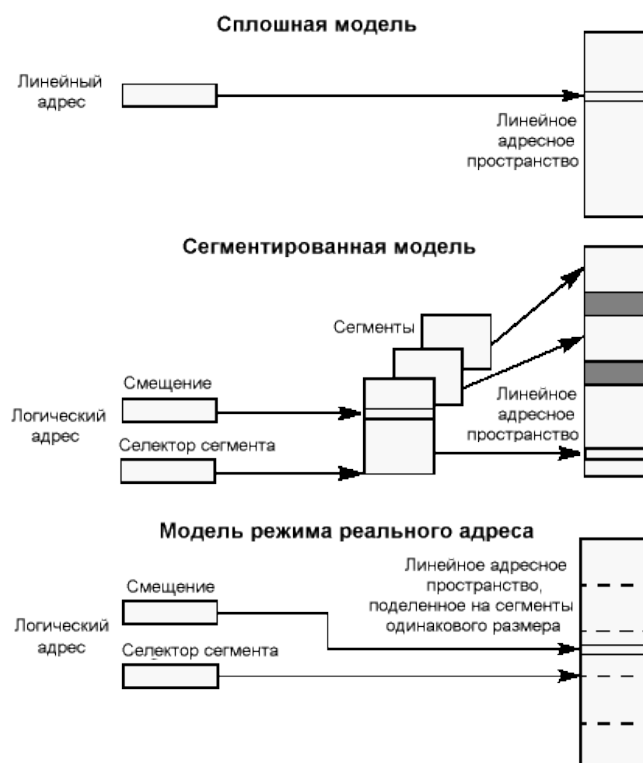


Рис. 3.1.1. Три модели доступа к памяти

Ранние универсальные машины (до 1960 года), ранние мини-компьютеры (до 1970 года) и ранние персональные компьютеры (до 1980 года) не использовали абстракции памяти. Каждая программа просто видела физическую память. Когда программа выполняла команду

```
MOV REGISTER1,1000
```

компьютер просто перемещал содержимое физической ячейки памяти 1000 в REGISTER1. Таким образом, модель памяти, предоставляемая программисту, была простой физической памятью, набором адресов от 0 до некоторого максимального значения, где каждый адрес соответствовал ячейке, содержащей какое-то количество бит (обычно 8).

Даже в условиях, когда в качестве модели памяти выступает сама физическая память, возможны несколько вариантов использования памяти. Три из них показаны на рис. 3.1.2. Операционная система может (рис. 3.1.2, а) размещаться в нижней части адресов, в оперативном запоминающем устройстве (ОЗУ), или, по-другому, в памяти с произвольным доступом — RAM (Random Access Memory). Она может размещаться также в постоянном запоминающем устройстве (ПЗУ), или, иначе, в ROM (Read-Only Memory), в верхних адресах памяти (рис. 3.1.2, б). Или же драйверы устройств могут быть в верхних адресах памяти, в ПЗУ, а остальная часть системы — в ОЗУ, в самом низу (рис. 3.1.2, в). Первая модель прежде использовалась на универсальных машинах и мини-компьютерах, а на других машинах — довольно редко. Вторая модель использовалась на некоторых КПК и встроенных системах. Третья модель использовалась на ранних персональных компьютерах (например, на тех, которые работали под управлением MS-DOS), где часть системы, размещавшаяся в ПЗУ, называлась базовой системой ввода-вывода — BIOS (Basic Input Output System). Недостаток моделей, изображенных на рис. 3.1.2, а и в, заключается в том, что ошибка в программе пользователя может затереть операционную систему, и, возможно, с весьма пагубными последствиями.



Рис. 3.1.2. Три простых способа организации памяти при наличии операционной системы и одного пользовательского процесса (существуют и другие варианты)

Чтобы в этих моделях запустить другую программу, операционная система должна сохранить все содержимое памяти в файле на диске, а затем загрузить и запустить следующую программу. Эта концепция называется заменой данных (или свопингом).

Даже в наши дни можно найти такую реализацию модели памяти в простых встроенных устройствах, как микроволновые печи и стиральные машины.

Чтобы допустить одновременное размещение в памяти нескольких приложений без создания взаимных помех, нужно решить две проблемы, относящиеся к защите и перемещению. Для этого придумали новую абстракцию для памяти: адресное пространство. Так же как понятие процесса создает своеобразный абстрактный

центральный процессор для запуска программ, понятие адресного пространства создает своеобразную абстрактную память, в которой существуют программы.

**Адресное пространство** — это набор адресов, который может быть использован процессом для обращения к памяти.

У каждого процесса имеется собственное адресное пространство, независимое от того адресного пространства, которое принадлежит другим процессам (за исключением тех особых обстоятельств, при которых процессам требуется совместное использование их адресных пространств).

В одном из простых решений использовалась весьма примитивная версия динамического перераспределения памяти. При этом адресное пространство каждого процесса просто проецируется на различные части физической памяти. Каждый центральный процессор оснащался двумя специальными аппаратными регистрами, которые обычно называются базовым и ограничительным регистрами (рис. 3.1.3). При использовании этих регистров программы загружаются в последовательно расположенные свободные области памяти без модификации адресов в процессе загрузки. При запуске процесса в базовый регистр загружается физический адрес, с которого начинается размещение программы в памяти, а в ограничительный регистр загружается длина программы.

При каждой ссылке процесса на память с целью извлечения команды или записи слова данных аппаратура центрального процессора перед выставлением адреса на шине памяти добавляет к адресу, сгенерированному процессом, значение базового регистра. Одновременно аппаратура проверяет, не равен ли предлагаемый адрес значению ограничительного регистра или не превышает ли он это значение (в этом случае генерируется отказ и доступ прерывается). Недостатком перемещений с использованием базовых и ограничительных регистров является необходимость применения операций сложения и сравнения к каждой ссылке на ячейку памяти.

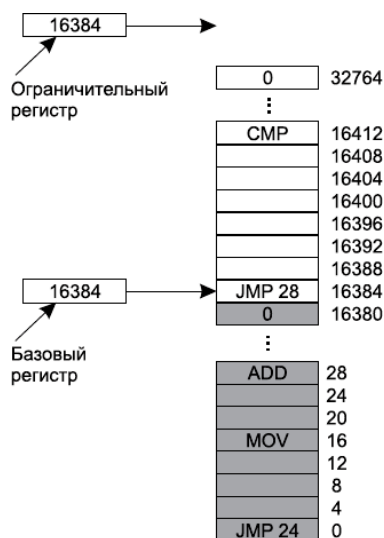


Рис. 3.1.3. Для предоставления каждому процессу отдельного адресного пространства могут использоваться базовый и ограничительный регистры

Если у компьютера недостаточно памяти для загрузки всех процессов, то требуются другие подходы к организации памяти. Самый простой из них, называемый свопингом, заключается в размещении в памяти всего процесса целиком, его запуске на некоторое время, а затем сбросе на диск. Бездействующие процессы большую часть времени хранятся на диске и в нерабочем состоянии не занимают пространство оперативной памяти (хотя некоторые из них периодически активизируются, чтобы проделать свою работу, после чего опять приостанавливаются). Второй подход называется виртуальной памятью, он позволяет программам запускаться даже в том случае, если они находятся в оперативной памяти лишь частично.

При свопинге после многократного запуска и завершения процессов, использующих различный объем памяти, рано или поздно наступит фрагментация памяти. Эти свободные области можно объединить в одну большую за счет перемещения при первой же возможности всех процессов в нижние адреса. Эта технология известна как уплотнение памяти. Но зачастую она не выполняется, поскольку отнимает довольно много процессорного времени. Вторая проблема — динамическое выделение дополнительной памяти для процесса. Если в памяти к этому процессу примыкает другой процесс, то нужны дополнительные действия: или перемещение процессов в памяти, или сброс на диск.

Если память распределяется в динамическом режиме, то управлять этим должна операционная система. В общих чертах, существуют два способа отслеживания использования памяти: битовые матрицы и списки свободного пространства.

При использовании битовых матриц память делится на единичные блоки размером от нескольких слов до нескольких килобайт. С каждым единичным блоком соотносится один бит в битовой матрице, который содержит 0, если единичный блок свободен, и 1, если он занят (или наоборот).

Другим способом отслеживания памяти является ведение связанных списков распределенных и свободных сегментов памяти, где сегмент либо содержит процесс, либо является пустым пространством между двумя процессами.

Когда процессы и пустые пространства содержатся в списке отсортированными по адресам, то для выделения памяти создаваемому процессу (или существующему процессу, загружаемому в результате свопинга с диска) могут быть использованы несколько алгоритмов. Предположим, что диспетчер памяти знает, сколько памяти нужно выделить. Простейший алгоритм называется **«первое подходящее»**. Диспетчер памяти сканирует список сегментов до тех пор, пока не найдет пустое пространство подходящего размера. Затем пустое пространство разбивается на две части: одна для процесса и одна для неиспользуемой памяти, за исключением того статистически маловероятного случая, когда процесс в точности помещается в пустое пространство. «Первое подходящее» — это быстрый алгоритм, поскольку поиск ведется с наименьшими затратами времени.

Незначительной вариацией алгоритма «первое подходящее» является алгоритм **«следующее подходящее»**. Он работает так же, как и «первое подходящее», за исключением того, что отслеживает свое местоположение, как только находит подходящее пустое пространство. При следующем вызове для поиска пустого пространства он начинает поиск в списке с того места, на котором остановился в прошлый раз, а не приступает к поиску с самого начала, как при работе алгоритма «первое подходящее». Моделирование работы алгоритма «следующее подходящее», выполненное Бэйсом (Bays, 1977), показало, что его производительность несколько хуже, чем алгоритма «первое подходящее».

Другой хорошо известный и широко используемый алгоритм — **«наиболее подходящее»**. При нем поиск ведется по всему списку, от начала до конца, и выбирается наименьшее соответствующее пустое пространство. Вместо того чтобы разбивать большое пустое пространство, которое может пригодиться чуть позже, алгоритм «наиболее подходящее» пытается подыскать пустое пространство, близкое по размеру к необходимому, чтобы наилучшим образом соответствовать запросу и имеющимся пустым пространствам.

Алгоритм «наиболее подходящее» работает медленнее, чем «первое подходящее», поскольку при каждом вызове он должен вести поиск по всему списку. Как ни странно, но его применение приводит к более расточительному использованию памяти, чем использование алгоритмов «первое подходящее» и «следующее подходящее», поскольку он стремится заполнить память, оставляя небольшие бесполезные пустые

пространства. В среднем при использовании алгоритма «первое подходящее» образуются более протяженные пустые пространства.

При попытке обойти проблему разбиения практически точно подходящих пространств памяти на память, отводимую под процесс, и небольшие пустые пространства можно прийти к идее алгоритма **«наименее подходящее»**, то есть к неизменному выбору самого большого подходящего пустого пространства, чтобы вновь образующееся пустое пространство было достаточно большим для дальнейшего использования. Моделирование показало, что применение алгоритма «наименее подходящее» также далеко не самая лучшая идея.

Еще один алгоритм распределения памяти называется **«быстро искомое подходящее»**. Его использование предусматривает ведение отдельных списков для некоторых наиболее востребованных искомых размеров.

При использовании алгоритма «быстро искомое подходящее» поиск пустого пространства требуемого размера выполняется исключительно быстро, но в нем имеется недостаток, присущий всем системам, сортирующим пустые пространства по размеру, а именно когда процесс завершается или выгружается процедурой свопинга, слишком много времени тратится на то, чтобы определить, можно ли высвобождаемое пространство объединить с соседними. Если не проводить объединение, то память очень быстро окажется разбитой на большое количество небольших по размеру пустых фрагментов, в которых не смогут поместиться процессы.

### **Виртуальная память**

Свопинг — не слишком привлекательный выбор для поддержки одновременно запущенных программ, каждая из которых помещается в память, но все вместе они превышают имеющийся объем памяти, поскольку обычный диск с интерфейсом SATA обладает пиковой скоростью передачи данных в несколько сотен мегабайт в секунду, а это означает, что свопинг программы объемом 1 Гбайт займет секунды, и еще столько же времени будет потрачено на загрузку другой программы в 1 Гбайт. Вторая проблема — некоторые программы превышали размер доступной оперативной памяти. Проблемы требовали решения.

В 1960-е годы было принято решение разбивать программы на небольшие части, называемые оверлеями. Оверлеи хранились на диске, и их свопинг с диска в память и обратно осуществлялся администратором загрузки оверлеев.

Хотя сама работа по свопингу оверлеев с диска в память и обратно выполнялась операционной системой, разбиение программ на части выполнялось программистом в ручном режиме. Разбиение больших программ на небольшие модульные части было очень трудоемкой, скучной и не застрахованной от ошибок работой.

Изобретенный метод (Fotheringham, 1961) стал известен как виртуальная память. В основе виртуальной памяти лежит идея, что у каждой программы имеется собственное адресное пространство, которое разбивается на участки, называемые страницами. Каждая страница представляет собой непрерывный диапазон адресов. Эти страницы отображаются на физическую память, но для запуска программы одновременное присутствие в памяти всех страниц необязательно. Когда программа ссылается на часть своего адресного пространства, находящегося в физической памяти, аппаратное обеспечение осуществляет необходимое отображение на лету. Когда программа ссылается на часть своего адресного пространства, которое не находится в физической памяти, операционная система предупреждается о том, что необходимо получить недостающую часть и повторно выполнить потерпевшую неудачу команду. В некотором смысле виртуальная память является обобщением идеи базового и ограничительного регистров.

При использовании виртуальной памяти вместо отдельного перемещения только сегмента текста или только сегмента данных программы на физическую память в сравнительно небольших блоках может быть отображено все адресное пространство. Реализация виртуальной памяти будет показана в теме 3.2.

Виртуальная память неплохо работает и в многозадачных системах, когда в памяти одновременно содержатся составные части многих программ. Пока программа ждет считывания какой-либо собственной части, центральный процессор может быть отдан другому процессу.

Адреса могут генерироваться с использованием индексной адресации, базовых регистров, сегментных регистров и другими способами.

Такие сгенерированные программным способом адреса называются виртуальными адресами, именно они и формируют виртуальное адресное пространство.

### Диспетчер памяти

Виртуальные адреса не выставляются напрямую на шине памяти. Вместо этого они поступают в диспетчер памяти (Memory Management Unit (MMU)), который отображает виртуальные адреса на адреса физической памяти (рис. 3.1.4).

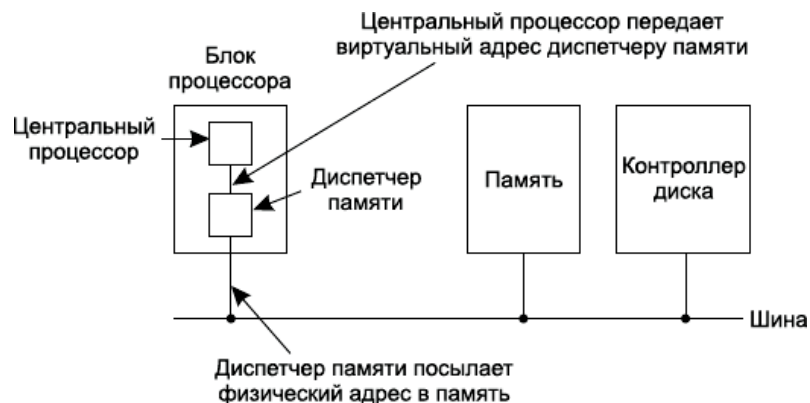


Рис. 3.1.4. Расположение и предназначение диспетчера памяти. Здесь он показан в составе микросхемы центрального процессора, как это чаще всего и бывает в наши дни. Но логически он может размещаться и в отдельной микросхеме, как было в прошлом

**Блок управления памятью (MMU)** — это аппаратный компонент компьютера, который обрабатывает все операции с памятью и кэшированием, связанные с процессором. Другими словами, MMU отвечает за все аспекты управления памятью. Обычно он интегрирован в процессор, хотя в некоторых системах он занимает отдельную интегральную схему (ИС).

Эффективное управление памятью требует от системных администраторов постоянного мониторинга использования и производительности памяти. Инструменты, помогающие в этом процессе, включены в операционную систему компьютера и доступны сторонним поставщикам.

Работа MMU делится на три основные категории:

- **Аппаратное управление памятью** контролирует и регулирует использование процессором оперативной памяти и кэш-памяти.
- **Управление памятью ОС** обеспечивает наличие достаточных ресурсов памяти для объектов и структур данных каждой запущенной программы.
- **Управление памятью приложений** выделяет необходимую память каждой отдельной программе, а затем повторно использует освобожденное пространство памяти после завершения операции.



Память является наиболее важной частью компьютера после центрального процессора (ЦП). Здесь выполняется работа и выполняются инструкции и другие вычислительные функции. Память обеспечивает правильную работу утилит компьютера, а также выполнение приложений и функций ввода/вывода.

MMU является важной частью подсистем памяти, обеспечивая эффективную работу всех функций, связанных с памятью. **Управление памятью помогает выполнять следующие функции:**

- **Распределение памяти.** MMU перемещает ресурсы памяти туда, где они могут наилучшим образом удовлетворить системные требования, обычно до и после выполнения системных процессов.
- **Мониторинг.** Он отслеживает все ресурсы памяти.
- **Эффективность.** Это сохраняет основную память и другие ресурсы памяти.
- **Целостность системы.** Это снижает вероятность фрагментации памяти и плохого распределения памяти.
- **Целостность данных.** Это гарантирует сохранение целостности данных.
- **Уменьшение рисков с данными.** Это сводит к минимуму риск повреждения данных.
- **Контроль за уровнем издержек.** Это снижает затраты, связанные с памятью.
- **Управление физической и логической памятью**

Основные различия между логической и физической памятью заключаются в следующем:

**Логическая память.** Адреса логической памяти, также называемые виртуальным адресным пространством, создаются ЦП и выполняют множество различных функций. Логическую память также называют виртуальной памятью, поскольку ее размер может динамически изменяться в зависимости от требований обработки данных пользователем.

**Физическая память.** Напротив, MMU вычисляет физические адреса. Затем они загружаются в регистр адреса памяти подсистемы памяти, который также называется физической памятью. Когда физические адреса содержат логические адреса, они называются физическим адресным пространством. В отличие от логических адресов, физические адреса не меняются.

### **Страничная организация памяти**

Виртуальное адресное пространство состоит из блоков фиксированного размера, называемых страницами. Соответствующие блоки в физической памяти называются страничными блоками. Страницы и страничные блоки имеют, как правило, одинаковые размеры. В нашем примере их размер составляет 4 Кбайт, но в реальных системах используются размеры страниц от 512 байт до 1 Гбайт. **Перенос информации между оперативной памятью и диском всегда осуществляется целыми страницами.** Многие процессоры поддерживают несколько размеров страниц, которые могут быть смешаны и подобраны по усмотрению операционной системы. Например, архитектура x86-64 поддерживает страницы размером 4 Кбайт, 2 Мбайт и 1 Гбайт, поэтому для пользовательских приложений можно использовать страницы размером 4 Кбайт, а для ядра — одну страницу размером 1 Гбайт.

Если программа ссылается на неотображаемые адреса, диспетчер памяти замечает, что страница не отображена, и заставляет центральный процессор передать управление операционной системе. Это системное прерывание называется ошибкой отсутствия страницы (page fault). Операционная система выбирает редко используемый страничный блок и сбрасывает его содержимое на диск (если оно еще не там). Затем она извлекает (также с диска) страницу, на которую была ссылка, и помещает ее в только что освободившийся страничный блок, вносит изменения в таблицы и заново запускает прерванную команду.



На рис. 3.1.5 показан пример виртуального адреса 8196 (**001000000000100** в двоичной записи), отображенного с использованием карты диспетчера памяти. Входящий 16-разрядный виртуальный адрес делится на 4-битный номер страницы и 12-битное смещение. Выделяя 4 бита под номер страницы, мы можем иметь 16 страниц, а с 12 битами под смещение можем адресовать все 4096 байт внутри страницы. **Номер страницы используется в качестве индекса внутри таблицы страниц для получения номера страничного блока, соответствующего виртуальной странице.** Если бит присутствия-отсутствия установлен в 0, то вызывается системное прерывание. Если бит установлен в 1, из таблицы страниц берется номер страничного блока, который копируется в старшие 3 бита выходного регистра вместе с 12-битным смещением, которое копируется в неизменном виде из входящего виртуального адреса. Вместе они формируют 15-разрядный физический адрес. Затем значение выходного регистра выставляется на шине памяти в качестве физического адреса.

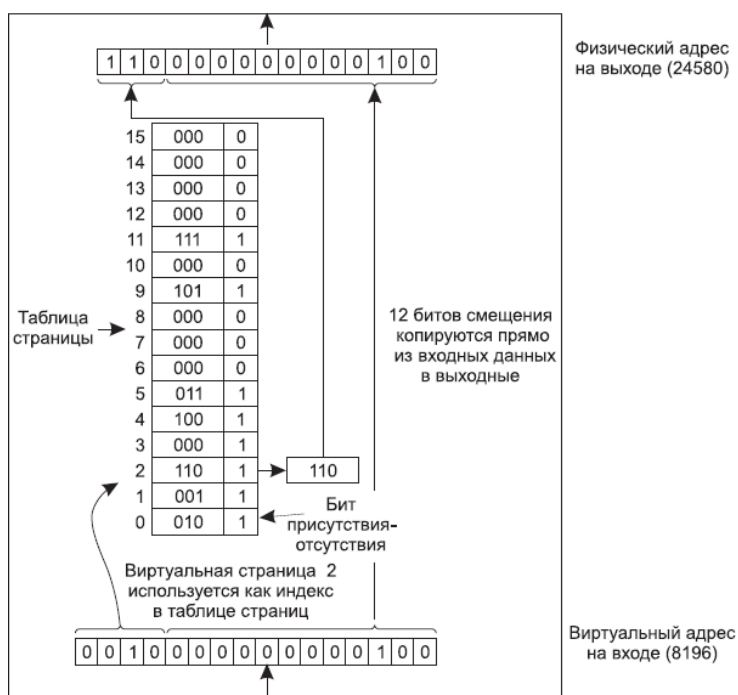


Рис. 3.1.5. Преобразование диспетчером памяти виртуального адреса в физический для 16 страниц по 4 Кбайт

### Таблицы страниц

**Номер виртуальной страницы используется в качестве индекса внутри таблицы страниц, который нужен для поиска записи для этой виртуальной страницы. Из записи в таблице страниц берется номер страничного блока (если таковой имеется). Номер страничного блока присоединяется к старшим битам смещения, заменяя собой номер виртуальной страницы, чтобы сформировать физический адрес, который может быть послан к памяти.**

**Таким образом, предназначение таблицы страниц заключается в отображении виртуальных страниц на страничные блоки.** С математической точки зрения таблица страниц — это функция, в которой в качестве аргумента выступает номер виртуальной страницы, а результатом является номер физического блока. При использовании результата этой функции поле виртуальной страницы в виртуальном адресе можно заменить полем страничного блока, формируя таким образом адрес физической памяти.

Точный формат записи сильно зависит от конструкции машины, но вид присутствующей в ней информации примерно одинаков для всех машин. На рис. 3.1.6 показан пример записи в таблице страниц. Размер варьируется от компьютера к компьютеру, но обычно он составляет 32 бита. **Наиболее важным является поле номера страничного блока (Page frame number).** В конечном счете цель страничного отображения и состоит в выдаче этого

значения. Следующим по значимости является бит присутствия-отсутствия. Если он установлен в 1, запись имеет смысл и может быть использована. А если он установлен в 0, то виртуальная страница, которой принадлежит эта запись, в данный момент в памяти отсутствует. Обращение к записи таблицы страниц, у которой этот бит установлен в 0, вызывает ошибку отсутствия страницы.

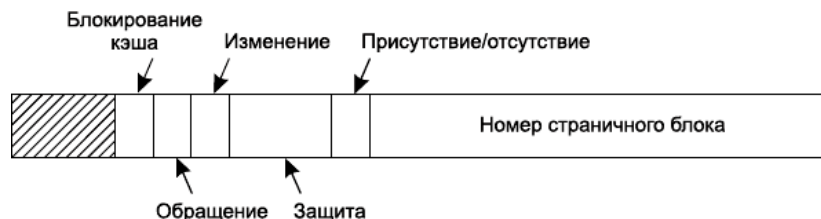


Рис. 3.1.6. Типичная запись таблицы страниц

**Биты защиты** сообщают о том, какого рода доступ разрешен. В простейшей форме это поле состоит из 1 бита со значением 0 для чтения-записи и значением 1 только для чтения. При более сложном устройстве имеется 3 бита, по одному для разрешения чтения, записи и исполнения страницы.

**Биты модификации и ссылки** отслеживают режим использования страницы. Когда в страницу осуществляется запись, аппаратура автоматически устанавливает бит модификации. Этот бит имеет значение, когда операционная система решает регенерировать страничный блок. Если содержащаяся в нем страница подвергалась модификации (то есть является измененной), ее нужно сбросить обратно на диск. Если же она не подвергалась модификации (то есть является неизменной), от нее можно отказаться, поскольку ее дисковая копия не утратила актуальности. Этот бит иногда называется битом изменения, поскольку он отражает состояние страницы.

**Бит ссылки** устанавливается при обращении к странице как для чтения, так и для записи. Он призван помочь операционной системе выбрать выселяемую страницу при возникновении ошибки отсутствия страницы. Страницы, к которым не было обращений, являются более предпочтительными кандидатами, чем часто используемые, и этот бит играет важную роль в ряде алгоритмов замещения страниц.

И наконец, оставшийся бит позволяет блокировать кэширование страницы. Эта возможность актуальна для тех страниц, которые отображаются на регистры устройств, а не на память. Если операционная система вошла в цикл ожидания отклика какого-нибудь устройства ввода-вывода на только что выданную ею команду, очень важно, чтобы аппаратура продолжала извлечение слова из устройства, а не использовала старую копию, попавшую в кэш. Благодаря этому биту кэширование может быть отключено. Те машины, у которых есть отдельное пространство ввода-вывода и которые не используют ввод-вывод с отображением данного пространства в память, в этом бите не нуждаются.

Потребность в обширном и быстром отображении страниц является весьма существенным ограничением на пути создания компьютеров. Простейшая конструкция (по крайней мере, концептуально) состоит в использовании одной таблицы страниц, состоящей из массива быстродействующих аппаратных регистров, имеющей по одной записи для каждой виртуальной страницы, проиндексированной по номеру виртуальной страницы (см. рис. 3.1.5). При запуске процесса операционная система загружает регистры таблицей страниц этого процесса, которая берется из копии, хранящейся в оперативной памяти. Во время выполнения процесса таблице страниц не нужны никакие дополнительные ссылки на память. Преимуществами этого метода являются простота и отсутствие каких-либо обращений к памяти во время отображения. Его недостаток — в чрезмерных затратах при большом размере таблицы страниц, что зачастую просто

непрактично. Еще один недостаток заключается в необходимости загрузки всей таблицы страниц при каждом переключении контекста, что полностью убьет производительность. Другой крайностью является конструкция, при которой вся таблица страниц может целиком находиться в оперативной памяти. При этом аппаратуре нужно иметь лишь один регистр, указывающий на начало таблицы страниц. Такая конструкция позволяет отображению виртуальной памяти на физическую меняться при переключении контекста путем перезагрузки всего лишь одного регистра. Здесь, разумеется, есть и недостаток, поскольку для считывания записей таблицы страниц во время выполнения каждой команды требуется одно или несколько обращений к памяти, что существенно замедляет работу.

### Буферы быстрого преобразования адреса

Большинство программ склонны большинство своих обращений направлять к небольшому количеству страниц, а не наоборот. Поэтому интенсивному чтению подвергается лишь небольшая часть записей таблицы страниц, а остальная часть практически не используется.

Поэтому компьютеры начали оснащать небольшим устройством для отображения виртуальных адресов на физические без просмотра таблицы страниц — **буфером быстрого преобразования адреса** (Translation Lookaside Buffer (**TLB**)), которое иногда еще называют ассоциативной памятью. Зачастую это устройство находится внутри диспетчера памяти и состоит из небольшого количества записей. Каждая запись содержит информацию об одной странице, включающую номер виртуальной страницы, бит, устанавливающийся при модификации страницы, код защиты (разрешение на чтение, запись и выполнение) и физический страничный блок, в котором расположена страница. Эти поля имеют точное соответствие полям в таблице страниц, за исключением номера виртуальной страницы, который в таблице страниц не нужен. Еще один бит показывает задействованность страницы (то есть используется она или нет).

Когда диспетчеру памяти предоставляется для преобразования виртуальный адрес, аппаратура сначала проверяет, не содержится ли номер его виртуальной страницы в TLB, одновременно (то есть параллельно) сравнивая его значение со всеми записями. Для этого потребуется специальное оборудование, имеющееся у всех диспетчеров памяти с TLB-буферами. Если будет найдено соответствие и биты защиты не будут препятствовать доступу, номер страничного блока будет взят непосредственно из TLB, без обращения к таблице страниц. Если номер виртуальной страницы присутствует в TLB, но команда пытается осуществить запись в страницу, предназначенную только для чтения, генерируется ошибка защиты.

Если номер виртуальной страницы не будет найден в TLB, диспетчер памяти обнаруживает его отсутствие и осуществляет обычный поиск в таблице страниц. Затем он выселяет одну из записей в TLB, заменяя ее только что найденной записью из таблицы страниц. Поэтому, если вскоре эта страница будет востребована снова, то во второй раз она уже будет найдена в TLB. Когда запись удаляется из TLB, бит модификации копируется обратно в таблицу страниц, находящуюся в памяти. Другие значения, за исключением бита ссылки, там уже присутствуют. Когда TLB загружается из таблицы страниц, все поля берутся из памяти.

При использовании программного управления TLB важно понять разницу между различными видами ошибок отсутствия записей. Программная ошибка отсутствия происходит, когда страница, к которой идет обращение, отсутствует в TLB, но присутствует в памяти. Для ее устранения требуется лишь обновление TLB и не требуется выполнение операций ввода-вывода с обращением к диску. Обычно устранение программной ошибки отсутствия требует 10–20 машинных команд и может быть завершено за несколько наносекунд. В отличие от нее аппаратная ошибка отсутствия

происходит, когда сама страница отсутствует в памяти (и, разумеется, запись о ней отсутствует в TLB). Для получения страницы требуется обращение к диску, занимающее в зависимости от используемого диска несколько миллисекунд. Аппаратная ошибка отсутствия обрабатывается почти в миллион раз медленнее, чем программная. Просмотр отображения в иерархии таблиц страниц называется просмотром таблиц страниц (page table walk).

### Таблицы страниц для больших объемов памяти

Возникают проблемы обслуживания очень больших виртуальных адресных пространств. Далее будут рассмотрены два способа, позволяющие справиться с этими проблемами.

В качестве первого подхода рассмотрим использование многоуровневой таблицы страниц, простой пример которой показан на рис. 3.1.7. В его левой части показан 32-разрядный виртуальный адрес, разбитый на 10-битное поле PT1, 10-битное поле PT2 и 12-битное поле смещения. Поскольку под смещение отведено 12 бит, страницы имеют размер 4 Кбайт и их общее количество составляет 220. Секрет метода использования многоуровневой таблицы страниц заключается в отказе от постоянного хранения всех таблиц страниц в памяти. В частности, вообще не должны храниться те таблицы, в которых нет необходимости.

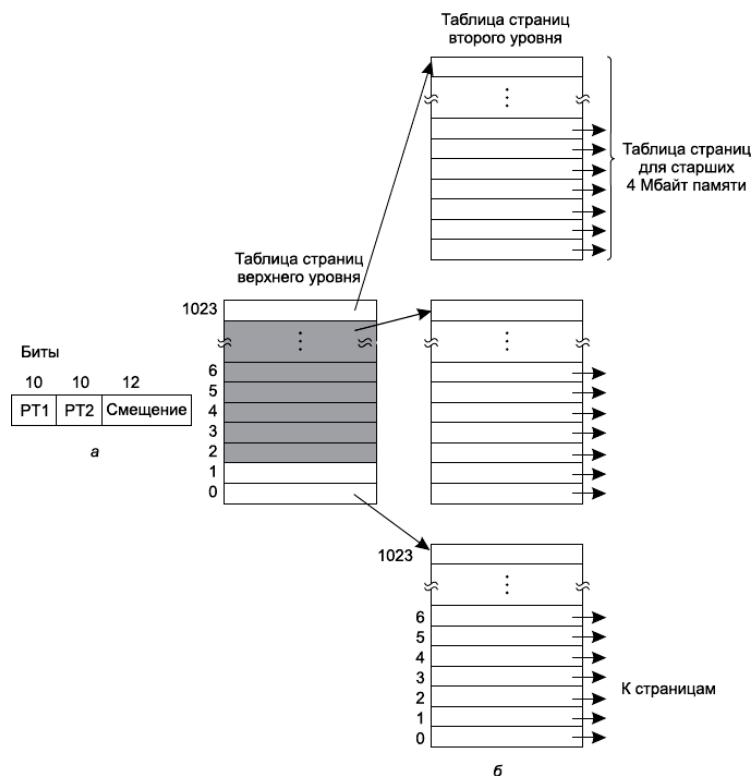


Рис. 3.1.7. Многоуровневая таблица страниц: а — 32-разрядный адрес с двумя полями таблиц страниц; б — двухуровневая таблица страниц

Из записи, место которой определяется путем индексирования таблицы страниц верхнего уровня, извлекается адрес или номер страничного блока таблицы страниц второго уровня. Запись 0 таблицы страниц верхнего уровня указывает на таблицу страниц для текста программы, запись 1 — на таблицу страниц для данных, а запись 1023 — на таблицу страниц для стека. Другие (закрашенные) записи не используются. Поле PT2 теперь используется в качестве индекса на выбранную таблицу страниц второго уровня, предназначенного для поиска номера страничного блока для самой страницы.

Альтернатива постоянно растущим уровням иерархии страничной адресации называется инвертированными таблицами страниц. Впервые они использовались такими процессорами, как PowerPC, UltraSPARC и Itanium. В данной конструкции имеется одна запись для каждого страничного блока в реальной памяти, а не одна запись на каждую

страницу в виртуальном адресном пространстве. Например, при использовании 64-разрядных виртуальных адресов, страниц размером 4 Кбайт и оперативной памяти размером 4 Гбайт инвертированные таблицы требовали только 1 048 576 записей. В каждой записи отслеживается, что именно находится в страничном блоке (процесс, виртуальная страница).

Хотя инвертированные таблицы страниц экономят значительное количество пространства, по крайней мере в том случае, когда виртуальное адресное пространство намного объемнее физической памяти, у них есть один серьезный недостаток: преобразование виртуальных адресов в физические становится намного сложнее. Когда процесс  $n$  обращается к виртуальной странице  $p$ , аппаратура уже не может найти физическую страницу, используя  $p$  в качестве индекса внутри таблицы страниц. Вместо этого она должна провести поиск записи  $(n, p)$  по всей инвертированной таблице страниц. Более того, этот поиск должен быть проведен при каждом обращении к памяти, а не только при ошибках отсутствия страницы. Но просмотр таблицы размером 256 К записей при каждом обращении к памяти может серьезно ухудшить быстродействие.

Решение этой дилеммы состоит в использовании TLB. Если в этом буфере можно будет хранить информацию обо всех интенсивно используемых страницах, преобразование может происходить так же быстро, как и при использовании обычных таблиц страниц. Но при отсутствии нужной записи в TLB программа должна просмотреть инвертированную таблицу страниц. Одним из приемлемых способов осуществления этого поиска является ведение хэш-таблицы, созданной на основе виртуальных адресов.

На рис. 3.1.8 показано, что все находящиеся на данный момент в памяти виртуальные страницы, имеющие одинаковые хэш-значения, связываются в одну цепочку. Если у хэш-таблицы столько же строк, сколько физических страниц у машины, средняя цепочка будет длиной всего лишь в одну запись, позволяя существенно ускорить отображение. Как только будет найден номер страничного блока, в TLB будет введена новая пара значений (виртуального, физического).

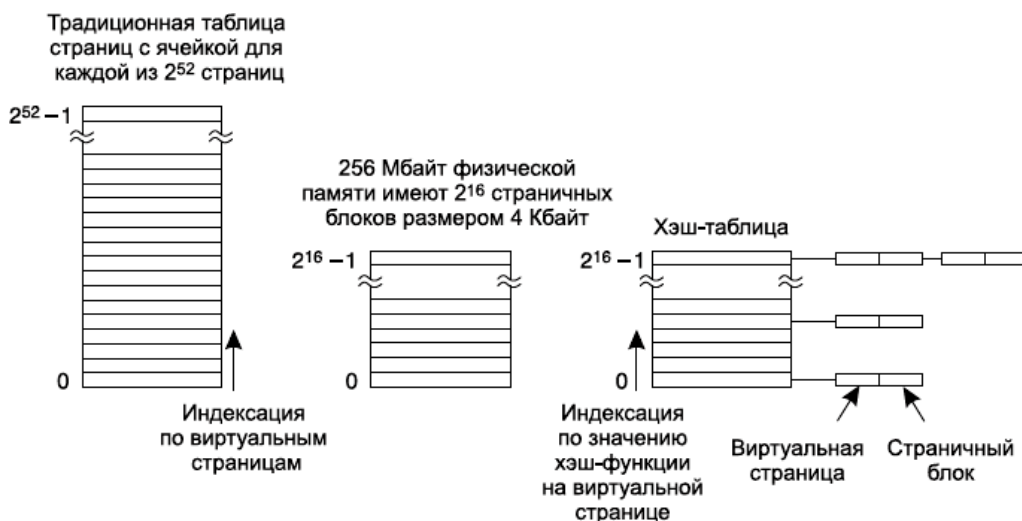


Рис. 3.1.8. Сопоставление традиционной таблицы страниц с инвертированной

Инвертированные таблицы страниц нашли широкое применение на 64-разрядных машинах, поскольку даже при очень больших размерах страниц количество записей в обычных таблицах страниц будет для них просто гигантским. К примеру, при размере страниц 4 Мбайт и 64-разрядных виртуальных адресах понадобится  $2^{42}$  записей в таблице страниц.

Практическая реализация и способы управления виртуальной памятью в Windows и Linux будет рассматриваться в теме 3.2 настоящей учебной дисциплины.

### 3.1.2. Адресное пространство процесса

В дополнение к тому, что ядро должно управлять своей памятью, оно также должно, управлять и адресным пространством процессов — тем, как память видится для каждого процесса в системе. Как было показано в предыдущем разделе, для всех процессов выполняется виртуализация ресурсов памяти. Для каждого процесса создается иллюзия того, что он один использует всю физическую память в системе. Еще более важно, что адресное пространство процессов может быть даже значительно больше объема физической памяти [3].

**Адресное пространство процесса** состоит из диапазона адресов, которые выделены процессу, и, что более важно, в этом диапазоне выделяются адреса, которые процесс может так или иначе использовать. Каждому процессу выделяется "плоское" 32- или 64-битовое адресное пространство. Термин "плоское" обозначает, что адресное пространство состоит из одного диапазона адресов (например, 32-разрядное адресное пространство занимает диапазон адресов от 0 до 429496729). Некоторые операционные системы предоставляют сегментированное адресное пространство — адресное пространство состоит больше чем из одного диапазона адресов, т.е. состоит из сегментов. Современные операционные системы обычно предоставляют плоское адресное пространство. Размер адресного пространства зависит от аппаратной платформы. Обычно для каждого процесса существует свое адресное пространство. Адрес памяти в адресном пространстве одного процесса не имеет никакого отношения к такому же адресу памяти в адресном пространстве другого процесса. Тем не менее несколько процессов могут совместно использовать одно общее адресное пространство. Такие процессы называются потоками.

Значение адреса памяти — это заданное значение из диапазона адресов адресного пространства, как, например, 41021f000. Это значение идентифицирует определенный байт в 32-битовом адресном пространстве. Важной частью адресного пространства являются интервалы адресов памяти, к которым процесс имеет право доступа, как, например, 08048000–0804c000. Такие интервалы разрешенных адресов называются областями памяти (memory area). С помощью ядра процесс может динамически добавлять и удалять области памяти своего адресного пространства.

Процесс имеет право доступа только к действительным областям памяти. Более того, на область памяти могут быть установлены права только для чтения или запрет на выполнение. Если процесс обращается к адресу памяти, который не находится в действительной области памяти, или доступ к действительной области выполняется запрещенным образом, то ядро уничтожает процесс с сообщением "Segmentation Fault" (ошибка сегментации).

Области памяти могут содержать следующую нужную информацию.

- Отображение выполняемого кода из выполняемого файла в область памяти процесса, которая называется сегментом кода (text section).
- Отображение инициализированных переменных из выполняемого файла в область памяти процесса, которая называется сегментом данных (data section).
- Отображение страницы памяти, заполненной нулями, в область памяти процесса, которая содержит неинициализированные глобальные переменные.
- Отображение страницы памяти, заполненной нулями, в память процесса, которая используется в качестве стека процесса пространства пользователя (не нужно путать со стеком процесса в пространстве ядра, который является отдельной структурой данных и управляется и используется ядром).
- Дополнительные сегменты кода, данных и BSS каждой совместно используемой библиотеки, загружаемых в адресное пространство процесса.
- Все файлы, содержимое которых отображено в память.

- Все области совместно используемой памяти.
- Все анонимные отображения в память.

Каждое действительное значение адреса памяти в адресном пространстве процесса принадлежит только и только одной области памяти (области памяти не перекрываются). Как будет показано, для каждого отдельного участка памяти в выполняющемся процессе существует своя область: стек, объектный код, глобальные переменные, отображенный в память файл и т.д.

### **Адресное пространство в 32-хразрядных версиях Windows**

Виртуальное адресное пространство для 32-разрядной версии Windows составляет 4 гигабайта (ГБ) и разделено на две секции: одну для использования процессом, а другую — для использования системой.

#### **Виртуальное адресное пространство по умолчанию для 32-разрядной версии Windows**

<b>Диапазон памяти</b>	<b>Использование</b>
Низкий объем 2 ГБ (0x00000000 по 0x7FFFFFFF)	Используется процессом.
Высокая емкость 2 ГБ (0x80000000 до 0xFFFFFFFF)	Используется системой.

#### **Виртуальное адресное пространство для 32-разрядной версии Windows с 4GT**

<b>Диапазон памяти</b>	<b>Использование</b>
Низкий объем 3 ГБ (0x00000000 до 0xBFFFFFFF)	Используется процессом.
Высокая емкость 1 ГБ (0xC0000000–0xFFFFFFFF)	Используется системой.

После включения 4GT процесс с флагом IMAGE\_FILE\_LARGE\_ADDRESS\_AWARE, установленным в заголовке образа, получит доступ к дополнительным 1 ГБ памяти, превышающим 2 ГБ.

### **Адресное пространство в 64-хразрядных версиях Windows**

#### **Виртуальное адресное пространство в пользовательском режиме для каждого 32-разрядного процесса:**

- 2 ГБ** без флага IMAGE\_FILE\_LARGE\_ADDRESS\_AWARE (по умолчанию)
- 4 ГБ** с флагом IMAGE\_FILE\_LARGE\_ADDRESS\_AWARE

#### **Виртуальное адресное пространство в пользовательском режиме для каждого 64-разрядного процесса:**

- 2 ГБ** без флага IMAGE\_FILE\_LARGE\_ADDRESS\_AWARE
- с флагом IMAGE\_FILE\_LARGE\_ADDRESS\_AWARE** (по умолчанию):

**128 ТБ** (Windows 8.1 и Windows Server 2012 R2 или более поздней версии)

**8 ТБ** (Windows 8 и Windows Server 2012 или более ранние)

#### **Виртуальное адресное пространство в режиме ядра:**

**128 ТБ** (Windows 8.1 и Windows Server 2012 R2 или более поздней версии)

**8 ТБ** (Windows 8 и Windows Server 2012 или более ранние)

### **Ограничения физической памяти: Windows 11**

<b>Версия</b>	<b>X64</b>	<b>ARM64</b>
Windows 11 Корпоративная	6 ТБ	6 ТБ
Windows 11 для образовательных учреждений	2 ТБ	2 ТБ
Windows 11 Pro для рабочих станций	6 ТБ	6 ТБ
Windows 11 Pro	2 ТБ	2 ТБ
Windows 11 Домашняя	128 ГБ	128 ГБ

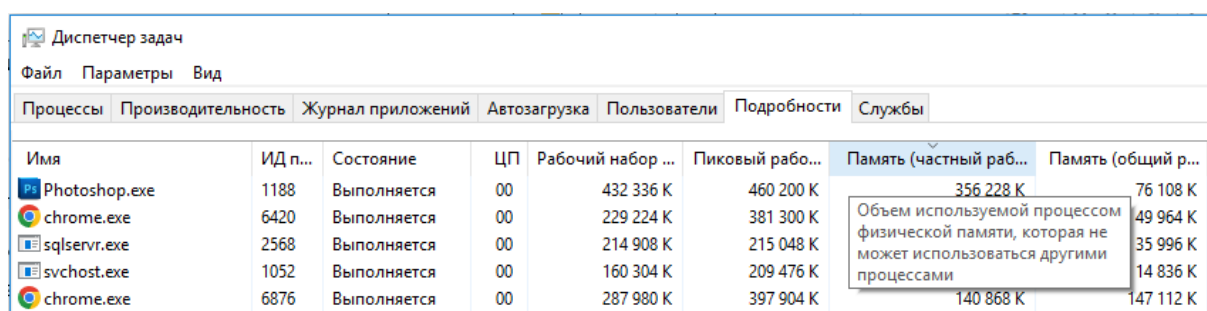


## Ограничения физической памяти: Windows 10

Версия	X86	X64
Windows 10 Корпоративная	4 ГБ	6 ТБ
Windows 10 для образовательных учреждений	4 ГБ	2 ТБ
Windows 10 Pro для рабочих станций	4 ГБ	6 ТБ
Windows 10 Pro	4 ГБ	2 ТБ
Windows 10 Домашняя	4 ГБ	128 ГБ

### 3.1.3. Концепция рабочего множества (набора)

Рабочий набор процесса — это набор страниц в виртуальном адресном пространстве процесса, которые в настоящее время находятся в физической памяти. Рабочий набор содержит только страничные выделения памяти. В рабочий набор не включаются ресурсы памяти, не допускающие подкачку, такие как расширения адресных окон (AWE) или большие страницы [4].



Имя	ИД п...	Состояние	ЦП	Рабочий набор ...	Пиковый рабо...	Память (частный раб...	Память (общий р...
Photoshop.exe	1188	Выполняется	00	432 336 К	460 200 К	356 228 К	76 108 К
chrome.exe	6420	Выполняется	00	229 224 К	381 300 К	49 964 К	35 996 К
sqlservr.exe	2568	Выполняется	00	214 908 К	215 048 К	14 836 К	14 836 К
svchost.exe	1052	Выполняется	00	160 304 К	209 476 К	140 868 К	147 112 К
chrome.exe	6876	Выполняется	00	287 980 К	397 904 К		

Рис. 3.1.9. Отображение информации о рабочих наборах в диспетчере задач Windows

Если процесс ссылается на страницу памяти, которая в настоящее время не входит в рабочий набор, происходит ошибка страницы. Обработчик ошибки системной страницы пытается устранить ошибку страницы, и в случае успеха страница добавляется в рабочий набор. (Доступ к AWE или выделению больших страниц никогда не приводит к ошибке страницы, так как эти выделения не являются страничными.)

Ошибка жесткой страницы должна быть устранена путем чтения содержимого страницы из резервного хранилища страницы, которое является системным файлом подкачки или файлом, сопоставленным в памяти, созданным процессом. Ошибка мягкой страницы может быть устранена без доступа к резервному хранилищу. Ошибка мягкой страницы возникает в следующих случаях:

- Страница находится в рабочем наборе какого-то другого процесса, поэтому она уже находится в памяти.
- Страница находится в состоянии перехода, так как она либо была удалена из рабочих наборов всех процессов, которые использовали страницу и еще не была перепрофилирована, либо она уже находится в результате операции предварительной выборки диспетчера памяти.
- Процесс впервые ссылается на выделенную виртуальную страницу (иногда это называется ошибкой с нулевым спросом).

Страницы можно удалить из рабочего набора процесса в результате следующих действий:

- Процесс уменьшает или очищает рабочий набор, вызывая функцию `SetProcessWorkingSetSize`, `SetProcessWorkingSetSizeEx` или `EmptyWorkingSet`.
- Процесс вызывает функцию `VirtualUnlock` в не заблокированном диапазоне памяти.

- Процесс отменяет сопоставление сопоставленного представления файла с помощью функции `UnmapViewOfFile`.
- Диспетчер памяти обрезает страницы из рабочего набора, чтобы создать больше доступной памяти.
- Диспетчер памяти должен удалить страницу из рабочего набора, чтобы освободить место для новой страницы (например, потому, что рабочий набор имеет максимальный размер).

Если несколько процессов совместно используют страницу, удаление страницы из рабочего набора одного процесса не влияет на другие процессы. После удаления страницы из рабочих наборов всех процессов, которые ее использовали, страница становится страницей перехода. Страницы перехода остаются кэшируемыми в ОЗУ, пока на страницу не будет повторно ссылаться какой-либо процесс или не будет перепрофилирована (например, заполнены нулями и передана другому процессу). Если страница перехода была изменена с момента последней записи на диск (т. е. если страница имеет значение "грязное"), то перед перепрофилированием страницу необходимо записать в ее резервное хранилище. Система может начать писать грязные переходные страниц в их резервное хранилище, как только такие страницы станут доступными.

Каждый процесс имеет минимальный и максимальный размер рабочего набора, который влияет на поведение процесса подкачки виртуальной памяти. Чтобы получить текущий размер рабочего набора указанного процесса, используйте функцию `GetProcessMemoryInfo`. Чтобы получить или изменить минимальный и максимальный размер рабочего набора, используйте функции `GetProcessWorkingSetSizeEx` и `SetProcessWorkingSetSizeEx`.

### 3.1.4. Защита памяти

Память, принадлежащая процессу, неявно защищена его частным виртуальным адресным пространством. Кроме того, Windows обеспечивает защиту памяти с помощью оборудования виртуальной памяти. Реализация этой защиты зависит от процессора, например, кодовые страницы в адресном пространстве процесса могут быть помечены как доступные только для чтения и защищены от изменения потоками пользовательского режима [4].

**Защита памяти** (Memory protection) — это способ управления правами доступа к отдельным регионам памяти. Используется большинством многозадачных операционных систем. Основной целью защиты памяти является запрет доступа процессу к той памяти, которая не выделена для этого процесса. Такие запреты повышают надёжность работы как программ, так и операционных систем, так как ошибка в одной программе не может повлиять непосредственно на память других приложений.

Альтернативное определение: защита памяти — избирательная способность предохранять выполняемую задачу от записи или чтения памяти предназначенной другой программе.

Чаще всего реализуется в рамках виртуальной адресации памяти.

Многие архитектуры, использующие страничную организацию памяти, в том числе x86 и x64, реализуют защиту памяти на уровне страниц. (При страничной организации памяти приложение не имеет возможности обратиться к странице, отсутствующей в её таблице страниц. Если при обращении по произвольному адресу не было найдено подходящее отображение, происходит исключительная ситуация `page fault`.)

Дескриптор страницы может задавать дополнительные ограничения доступа.

## NX bit

Атрибут (бит) NX-Bit (no execute bit в терминологии фирмы AMD) или XD-Bit (execute disable bit в терминологии фирмы Intel) — бит запрета исполнения, добавленный в страницы для реализации возможности предотвращения выполнения данных как кода.

Используется для предотвращения уязвимости типа «переполнение буфера», позволяющей выполнять произвольный код на атакуемой системе локально или удалённо. Технология требует программной поддержки со стороны ядра операционной системы.

Один из сценариев атак состоит в том, что воспользовавшись переполнением буфера в программе (зачастую это демон, предоставляющий некоторый сетевой сервис), специально написанная вредоносная программа (эксплойт) может записать некоторый код в область данных уязвимой программы таким образом, что в результате ошибки этот код получит управление и выполнит действия, запрограммированные злоумышленником (зачастую это запрос выполнить программу-оболочку ОС, с помощью которой злоумышленник получит контроль над уязвимой системой с правами владельца уязвимой программы; очень часто это root).

## Рандомизация размещения адресного пространства

ASLR (address space layout randomization) — технология, применяемая в операционных системах, при использовании которой случайным образом изменяется расположение в адресном пространстве процесса важных структур данных, а именно образов исполняемого файла, подгружаемых библиотек, кучи и стека.

Технология ASLR создана для усложнения эксплуатации нескольких типов уязвимостей. Например, если при помощи переполнения буфера или другим методом атакующий получит возможность передать управление по произвольному адресу, ему нужно будет угадать, по какому именно адресу расположен стек, куча или другие структуры данных, в которые можно поместить шелл-код. Сходные проблемы возникнут и при атаке типа «возврат в библиотеку» (return-to-libc): атакующему не будет известен адрес, по которому загружена библиотека. В приведённых примерах от атакующего скрыты конкретные адреса, и, если не удастся угадать правильный адрес, приложение скорее всего аварийно завершится, тем самым лишив атакующего возможности повторной атаки и привлекая внимание системного администратора.

Для использования ASLR исполняемые файлы требуется собирать со специальными флагами. В результате в коде не будут использоваться постоянные адреса, но при этом увеличится размер кода исполняемых файлов и увеличится время загрузки в память каждого исполняемого файла.

## Защита копирование-при-записи (COW)

Защита **копирование-при-записи** — это оптимизация, которая позволяет нескольким процессам сопоставлять виртуальные адресные пространства таким образом, чтобы они совместно используют физическую страницу до тех пор, пока один из процессов не изменит страницу. Это часть метода, называемого отложенной оценкой, которая позволяет системе экономить физическую память и время, не выполняя операцию до крайней необходимости [5].

Например, предположим, что два процесса загружают страницы из одной библиотеки DLL в свои виртуальные пространства памяти. Эти страницы виртуальной памяти сопоставляются с теми же страницами физической памяти для обоих процессов. Пока ни один процесс не выполняет запись на эти страницы, они могут сопоставлять и совместно использовать те же физические страницы, как показано на рис. 3.1.10.

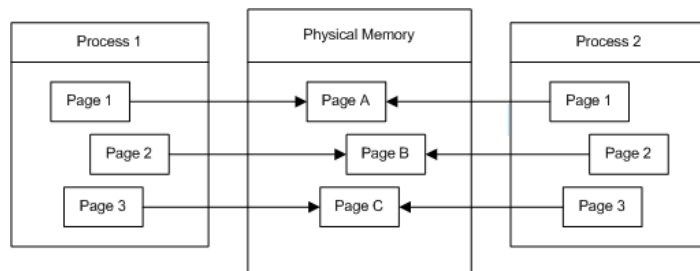


Рис. 3.1.10. Страница В разделяется между двумя процессами

Если процесс 1 выполняет запись на одну из этих страниц, содержимое физической страницы копируется на другую физическую страницу, а карта виртуальной памяти обновляется для процесса 1. Теперь оба процесса имеют собственный экземпляр страницы в физической памяти (рис. 3.1.11). Таким образом, один процесс не может выполнять запись на общую физическую страницу, а другой процесс не может видеть изменения.

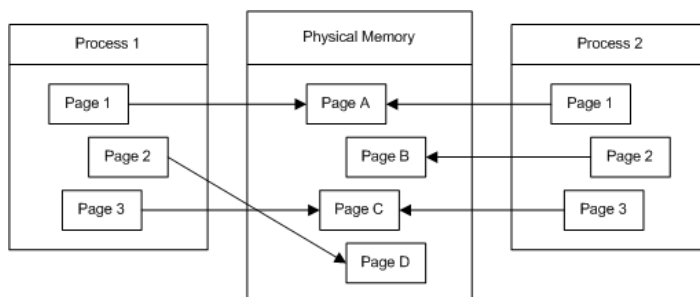


Рис. 3.1.11. Процесс 1 получил собственную копию страницы В — страницу D

### Загрузка приложений и библиотек DLL

При загрузке нескольких экземпляров одного приложения Windows каждый экземпляр запускается в собственном защищенном виртуальном адресном пространстве. Однако дескрипторы экземпляров (hInstance) обычно имеют одинаковое значение. Это значение представляет базовый адрес приложения в его виртуальном адресном пространстве. Если каждый экземпляр можно загрузить в базовый адрес по умолчанию, он может сопоставляться с теми же физическими страницами и совместно использовать их с другими экземплярами с помощью защиты от копирования при записи. Система позволяет этим экземплярам совместно использовать одни и те же физические страницы, пока одна из них не изменит страницу. Если по какой-либо причине один из этих экземпляров не может быть загружен в нужный базовый адрес, он получает собственные физические страницы.

Библиотеки DLL создаются с базовым адресом по умолчанию. Каждый процесс, использующий библиотеку DLL, будет пытаться загрузить библиотеку DLL в пределах собственного адресного пространства по виртуальному адресу библиотеки DLL по умолчанию. Если несколько приложений могут загрузить библиотеку DLL по виртуальному адресу по умолчанию, они могут совместно использовать одни и те же физические страницы для библиотеки DLL. Если по какой-либо причине процесс не может загрузить библиотеку DLL по адресу по умолчанию, он загружает библиотеку DLL в другое место. Защита копирования-при-записи заставляет некоторые страницы библиотеки DLL копироваться в разные физические страницы для этого процесса, так как исправления для инструкций перехода написаны на страницах библиотеки DLL, и они будут отличаться для этого процесса. Если раздел кода содержит много ссылок на раздел данных, это может привести к копированию всего раздела кода на новые физические страницы.

## Обзор Credential Guard

**Credential Guard** предотвращает атаки на кражу учетных данных, защищая хэши паролей NTLM, билеты на предоставление билетов Kerberos (TGT) и учетные данные, хранящиеся приложениями в качестве учетных данных домена [6].

Credential Guard использует безопасность на основе виртуализации (VBS) для изоляции секретов, чтобы доступ к ним могли получить только привилегированные системные программы. Несанкционированный доступ к этим секретам может привести к атакам кражи учетных данных, таким как передача хэша и передача билета.

Если этот параметр включен, Credential Guard предоставляет следующие преимущества:

- **Аппаратная безопасность:** NTLM, Kerberos и Credential Manager используют преимущества функций безопасности платформы, включая безопасную загрузку и виртуализацию, для защиты учетных данных.
- **Безопасность на основе виртуализации:** ntlm, производные учетные данные Kerberos и другие секреты выполняются в защищенной среде, изолированной от работающей операционной системы.
- **Защита от сложных постоянных угроз:** при защите учетных данных с помощью VBS методы и средства атаки на кражу учетных данных, используемые во многих целевых атаках, блокируются. Вредоносная программа, запущенная в операционной системе с правами администратора, не может извлекать секреты, защищенные с помощью VBS

### 3.1.5. Классификация запоминающих устройств

**Запоминающее устройство** — носитель информации, предназначенный для записи и хранения данных. В основе работы запоминающего устройства может лежать любой физический эффект, обеспечивающий приведение системы к двум или более устойчивым состояниям.

Классификация запоминающих устройств по устойчивости записи и возможности перезаписи:

- **Постоянные (ПЗУ)**, содержание которых не может быть изменено конечным пользователем (например, BIOS). ПЗУ в рабочем режиме допускает только считывание информации.
- **Записываемые (ППЗУ)**, в которые конечный пользователь может записать информацию только один раз (например, CD-R).
- **Многokrратно перезаписываемые (ПППЗУ)** (например, CD-RW).
- **Оперативные (ОЗУ)** — обеспечивают режим записи, хранения и считывания информации в процессе её обработки. Быстрые, но дорогие ОЗУ (SRAM) строят на триггерах, более медленные, но более дешёвые разновидности ОЗУ — динамические ЗУ (DRAM) строят на элементах, состоящих из ёмкости (конденсатора) и полевого транзистора, используемого в качестве ключа разрешения записи-чтения. В обоих видах ЗУ информация исчезает после отключения от источника питания (например, тока).

По типу доступа ЗУ делятся на:

- **устройства с последовательным доступом** (например, магнитные ленты).
- **устройства с произвольным доступом (RAM)** (например, оперативная память).
- **устройства с прямым доступом** (например, жесткие магнитные диски).
- **устройства с ассоциативным доступом** (специальные устройства, для повышения производительности БД)

Классификация запоминающих устройств по геометрическому исполнению:

- **дисковые** (магнитные диски, оптические, магнитооптические);
- **ленточные** (магнитные ленты, перфоленты);
- **барабанные** (магнитные барабаны);
- **карточные** (магнитные карты, перфокарты, флэш-карты, и др.)
- **печатные** платы (карты DRAM).

Классификация запоминающих устройств по физическому принципу:

- **перфорационные** (перфокарта; перфолента);
- **с магнитной записью** (ферритовые сердечники, магнитные диски, магнитные ленты, магнитные карты);
- **оптические** (CD, DVD, HD-DVD, Blu-ray Disc);
- **использующие эффекты в полупроводниках** (флэш-память) и другие.

### 3.1.6. Иерархия памяти

Современная система памяти образует иерархию от быстрых типов памяти маленького размера до медленных типов памяти большого размера. Мы говорим, что конкретный уровень иерархии кэширует или является кэшем для данных, расположенных на более низком уровне. Это значит, что он содержит копии данных с более низкого уровня. Когда процессор хочет получить какие-то данные, он их сперва ищет на самых быстрых высоких уровнях. И спускается на более низкие, если не может найти.

На вершине иерархии находятся **регистры** процессора. Доступ к ним занимает 0 тактов, но их всего несколько штук.

Далее идёт несколько килобайт **кэш-памяти первого уровня**, доступ к которой занимает примерно 4 такта.

Потом идёт пара сотен килобайт более медленной **кэш-памяти второго уровня**.

Потом несколько мегабайт **кэш-памяти третьего уровня**.

Она гораздо медленней, но всё равно быстрее оперативной памяти.

Далее расположена относительно медленная **оперативная память**.

Оперативную память можно рассматривать как кэш для **локального диска** (локальные диски тоже можно разделить на отдельные категории).

Локальный диск сам может рассматриваться как кэш для данных, расположенных на **удалённых серверах**.

Резервные копии данных можно хранить на **магнитных лентах**.

Современная система имеет примерно такие характеристики:

Тип кэша	Время доступа (тактов)	Размер кэша
Регистры	0	десятки штук
L1 кэш	4	32 KB
L2 кэш	10	256 KB
L3 кэш	50	8 MB
Оперативная память	200	8 GB
Буфер диска	100'000	64 MB
Локальный диск	10'000'000	1000 GB
Удалённые сервера	1'000'000'000	∞

## Сравнение дисковых систем

Существует два типа дисков: жесткий диск HDD (hard disk drive) и твердотельные накопители SSD (solid-state drive) и NVMe (Non-Volatile Memory Express) [7].

### HDD-диск

HDD-диски стоят в большинстве персональных компьютеров и ноутбуках. Внутри диска находятся несколько алюминиевых пластин. Операции чтения и записи происходят за счет вращения пластин и расположенной в нескольких нанометрах считывающей головки. Скорость пластин достигает 15 000 оборотов в минуту, отсюда и привычный шум, и высокая температура при работе дисков. Такие диски стали популярными за счет большого объема дискового пространства (до 18 ТБ на одном HDD-диске), высокой степени надежности, устойчивости к операциям чтения и записи.

Недостатки HDD-дисков относительно SSD-дисков:

- низкая скорость операций чтения/записи
- высокое энергопотребление
- высокий уровень шума

HDD-диски подходят для операций, в которых не требуется частое чтение и запись информации: организации хранилища данных, системы резервного копирования, почтового сервера, организации потокового видео, для организации сервера под виртуальные машины.

### SSD-диск

В SSD-дисках используются микросхемы памяти, а за счет отсутствия вращающихся элементов, такие диски полностью бесшумны, потребляют меньше электроэнергии и меньше HDD-дисков в размерах.

Операции чтения и записи в SSD-дисках проходят быстрее (файлы быстрее открываются, сохраняются и удаляются с диска).

Отношение скорости передачи данных к размеру передаваемого блока определяется показателем IOPS (Input/Output Operations Per Second). IOPS показывает какое количество блоков успевает записаться/считаться за секунду. Для сравнения, в HDD-дисках этот показатель около 80-100 IOPS, а в SSD-дисках — больше 8000 IOPS.

Однако, каждый цикл перезаписи постепенно "сжигает" диск, что уменьшает срок его службы.

SSD-диски подходят для высоконагруженных проектов, которые чувствительны к скорости процессов записи и чтения. SSD-диски увеличивают скорость работы сайта, разработанного на любой современной CMS.

Для подключения дисков к серверу используют дисковый интерфейс.

### NVMe-диск

NVMe это тот же накопитель семейства SSD, но использующий специально разработанный протокол доступа и подключаемый по шине PCI Express.

Интерфейс NVMe разработан специально для увеличения производительности твердотельных накопителей, он позволяет снять ограничения стандарта SATA, разработанного когда-то для HDD. Такой диск развивает скорость до 3,5 ГБ/с.

NVMe обладает аналогичными преимуществами и недостатками SSD-диска, рекомендуем использовать его для требовательных CMS, например, 1С-Битрикс, передачи «тяжёлых» файлов и когда производительности SSD для ваших проектов уже недостаточно.



### 3.1.7. Оперативные и постоянные запоминающие устройства

В ПЗУ находятся:

- программа управления работой процессора;
- программа запуска и останова компьютера;
- программы тестирования устройств, проверяющие при каждом включении компьютера правильность работы его блоков;
- программы управления дисплеем, клавиатурой, принтером, внешней памятью;
- информация о том, где на диске находится операционная система.

Оперативная память (также оперативное запоминающее устройство, ОЗУ) - предназначена для временного хранения данных и команд, необходимых процессору для выполнения им операций. Оперативная память передаёт процессору данные непосредственно, либо через кэш-память. Каждая ячейка оперативной памяти имеет свой индивидуальный адрес.

Устанавливается в виде модулей объемом до 64 Гб. Наиболее распространенный тип модулей — DIMM, тип памяти — DDR.

Различные поколения DDR-модулей отличаются электрически и физически (разное количество контактов). Чтобы избежать их повреждения, положения ключа в них отличается (рис. 3.1.12).

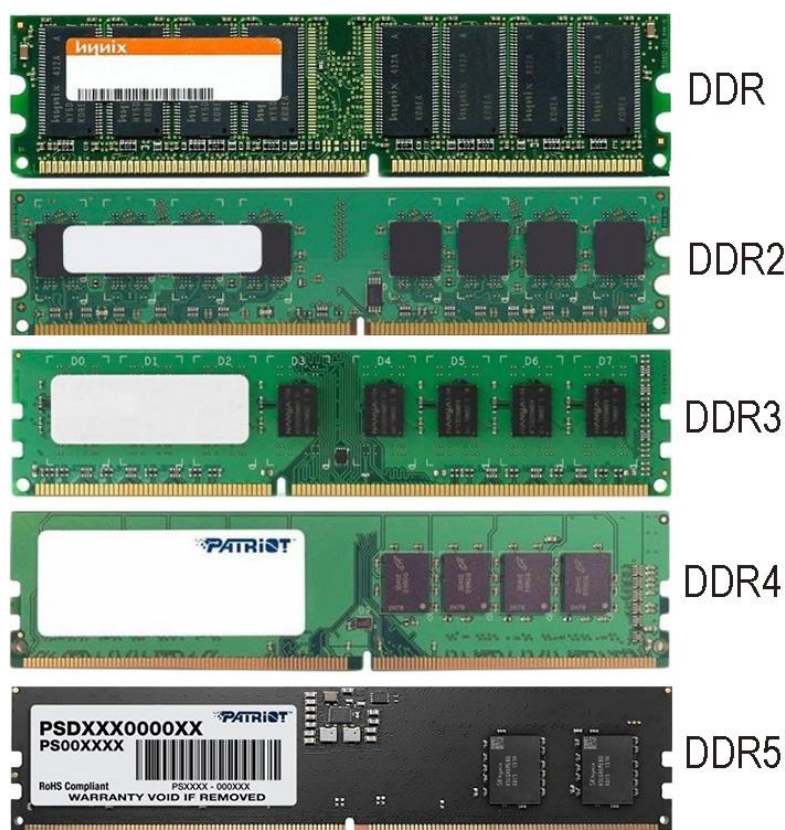


Рис. 3.1.12. Сравнение форм-фактора различных поколений DDR-памяти

С каждым поколением DDR-памяти увеличивается быстродействие и другие характеристики. Сравнительная таблица приведена на рис. 3.1.13.

	DDR5	DDR4	DDR3
Макс. скорость передачи	6.4 Гбит/с	3.2 Гбит/с	1.6 Гбит/с
Макс. размер модуля памяти	128 ГБ	32 ГБ	8 ГБ
Макс. плотность одного ядра	64 Гбит	16 Гбит	4 Гбит
Каналов	2	1	1
Напряжение	1.1 В	1.2 В	1.5 В
Ширина (Non-ECC)	64-бит (2×32)	64-бит	64-бит
Банки (Per Group)	4	4	8
Группы банков	8/4	4/2	1

Рис. 3.1.13. Сравнительная таблица поколений DDR-памяти

## **Список использованных источников**

1. Таненбаум, Э. Современные операционные системы. / Э. Таненбаум, Х. Бос. — 4-е изд. — СПб.: Питер, 2015. — 1120 с.
2. 3.2 Организация памяти  
[https://dfe.petrso.ru/x86/env\\_2.shtml](https://dfe.petrso.ru/x86/env_2.shtml)
3. Глава 14 Адресное пространство процесса  
<https://it.wikireading.ru/1927>
4. Рабочий набор  
<https://learn.microsoft.com/ru-ru/windows/win32/memory/working-set>
5. Защита памяти  
<https://learn.microsoft.com/ru-ru/windows/win32/memory/memory-protection>
6. Обзор Credential Guard  
<https://learn.microsoft.com/ru-ru/windows/security/identity-protection/credential-guard/>
7. HDD, SSD и NVMe: сравнение дисковых систем и интерфейсов  
<https://ispserver.ru/help/differences-between-disks>