**Текст практического занятия №2. Тема: «Организация памяти».**

**Вопрос 1. Функции операционной системы по управлению памятью.**

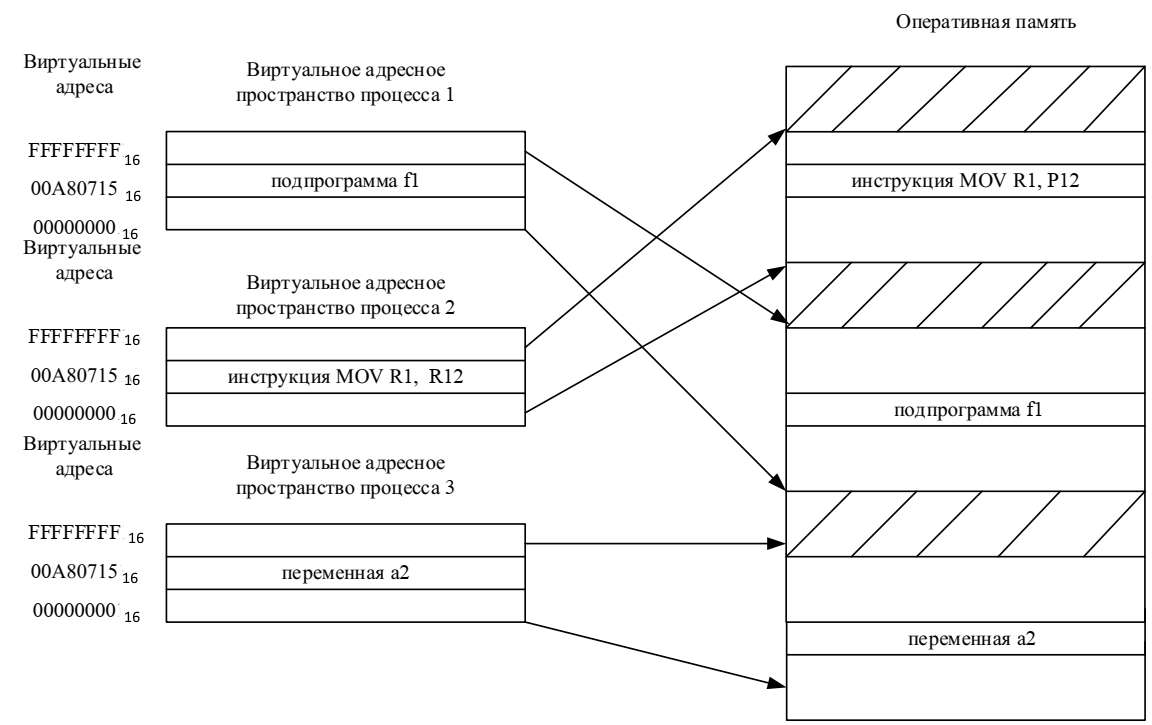
В ранних ОС управление памятью сводилось просто к загрузке программы и ее данных с некоторого внешнего накопителя в память. С появлением мультипрограммирования перед ОС были поставлены новые задачи, связанные с распределением имеющейся памяти между несколькими одновременно выполняющимися программами.

*Функциями ОС по управлению памятью в мультипрограммной системе являются:*

* отслеживание свободной и занятой памяти;
* выделение памяти процессам и освобождение памяти при завершении процессов;
* вытеснение кодов и данных процессов из оперативной памяти на диск (полное или частичное), когда размеры основной памяти недостаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место;
* настройка, адресов программы на конкретную область физической памяти;
* защита памяти – запрет выполняемому процессу записывать данные в память, назначенной другому процессу, или читать их оттуда. Эта функция, как правило, реализуется программными модулями ОС в тесном взаимодействии с аппаратными средствами.

Помимо первоначального выделения памяти процессам при их создании, ОС должна также заниматься динамическим распределением памяти, то есть выполнять запросы приложений во время выполнения на выделение им дополнительной памяти. После того как приложение перестает нуждаться в дополнительной памяти, оно может возвратить ее системе. Выделение памяти случайной длины в случайные моменты времени из общего пула памяти приводит к фрагментации и, вследствие этого, к неэффективному ее использованию.

Дефрагментация памяти тоже является функцией операционной системы. Во время работы операционной системы ей часто приходится создавать новые служебные информационные структуры, такие как описатели процессов и потоков, различные таблицы распределения ресурсов, буферы, используемые процессами для обмена данными, синхронизирующие объекты и т. П. Все эти системные объекты требуют памяти. В некоторых ОС заранее (во время установки) резервируется некоторый фиксированный объем памяти для системных нужд. В других же ОС используется более гибкий подход, при котором память для системных целей выделяется динамически. В таком случае разные подсистемы ОС при создании своих таблиц, объектов, структур и т. П. обращаются к подсистеме управления памятью с запросами.

**Типы адресов**. Для идентификации переменных и команд на разных этапах жизненного цикла программы используются символьные имена (метки), виртуальные адреса и физические адреса (рис. 1).

*Рис. 1. Виртуальные адресные пространства нескольких программ.*

Символьные имена (метки) присваивает пользователь при написании программы на алгоритмическом языке или ассемблере.

Виртуальные, математические, или логические, адреса вырабатывает транслятор, переводящий программу на машинный язык. Поскольку во время трансляции в общем случае неизвестно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам условные виртуальные адреса, обычно считая по умолчанию, что начальным адресом программы будет нулевой адрес.

Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности расположены или будут расположены переменные и команды.

Совокупность виртуальных адресов процесса называется *виртуальным адресным* пространством.

Диапазон возможных адресов виртуального пространства у всех процессов является одним и тем же. Например, при использовании 32-разрядных виртуальных адресов этот диапазон задается границами 0000000016 и FFFFFFFF16.

В то же время каждый процесс имеет собственное виртуальное адресное пространство – транслятор присваивает виртуальные адреса переменным и кодам каждой программе независимо.

Совпадение виртуальных адресов переменных и команд различных процессов не приводит к конфликтам, так как в том случае, когда эти переменные одновременно присутствуют в памяти, операционная система отображает их на разные физические адреса.

В разных операционных системах используются разные способы структуризации виртуального адресного пространства. В одних ОС виртуальное адресное пространство процесса подобно физической памяти представлено в виде непрерывной последовательности виртуальных адресов. Такую структуру адресного пространства называют линейной, или плоской (*flat*). При этом виртуальным адресом является единственное число, представляющее собой смещение относительно начала (обычно это значение 000…000) виртуального адресного пространства (рис. 2 а). Адрес такого типа называют линейным виртуальным адресом.

В других ОС виртуальное адресное пространство делится на части, называемые сегментами (или секциями, или областями, или другими терминами). В этом случае говорят, что виртуальное адресное пространство имеет сегментированную структуру. Для позиционирования данных здесь помимо линейного виртуального адреса может быть использован виртуальный адрес, учитывающий разбиение на сегменты и представляющий собой пару чисел (*n*, *m*), где *n* определяет сегмент, а *m* – смещение внутри сегмента (рис. 2 б).

*Рис. 2. Типы виртуальных адресных пространств: плоское (а), сегментированное (б).*

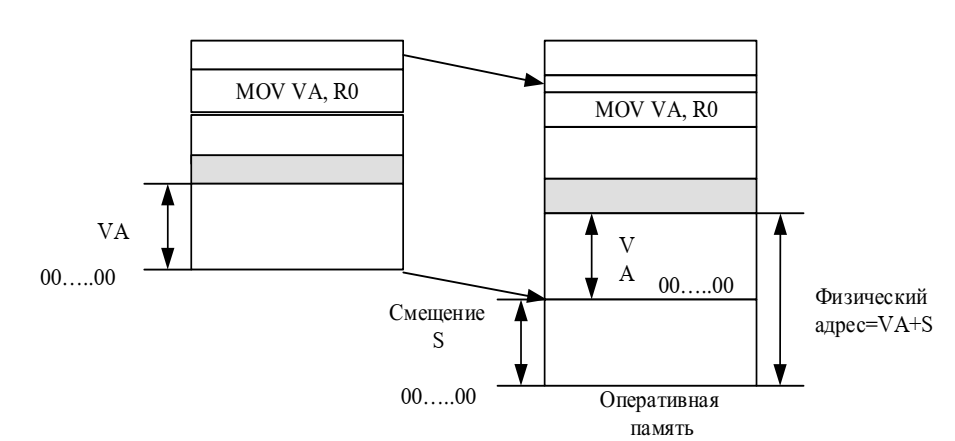
Существуют и более сложные способы структуризации виртуального адресного пространства, когда виртуальный адрес образуется тремя или даже более числами.

Задачей операционной системы является отображение индивидуальных виртуальных адресных пространств, всех, одновременно выполняющихся процессов, на общую физическую память. При этом ОС отображает либо все виртуальное адресное пространство, либо только определенную его часть. Процедура преобразования виртуальных адресов в физические должна быть максимально прозрачна для пользователя и программиста.

Существует два принципиально отличающихся подхода к преобразованию виртуальных адресов в физические. В *первом* случае замена виртуальных адресов физическими выполняется один раз для каждого процесса во время начальной загрузки программы в память. Специальная системная программа – перемещающий загрузчик – на основании имеющихся у нее исходных данных о начальном адресе физической памяти, в которую предстоит загружать программу, а также информации, предоставленной транслятором об адресно-зависимых элементах программы, выполняет загрузку программы, совмещая ее с заменой виртуальных адресов физическими.

*Второй* способ заключается в том, что программа загружается в память в неизмененном виде в виртуальных адресах, то есть операнды инструкций, и адреса переходов имеют те значения, которые выработал транслятор. В наиболее простом случае, когда виртуальная и физическая память процесса представляют собой единые непрерывные области адресов, операционная система выполняет преобразование виртуальных адресов в физические по следующей схеме. При загрузке операционная система фиксирует смещение действительного расположения программного кода относительно виртуального адресного пространства.

Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Схема такого преобразования показана на рис. 3.

*Рис. 3. Схема динамического преобразования адресов.*

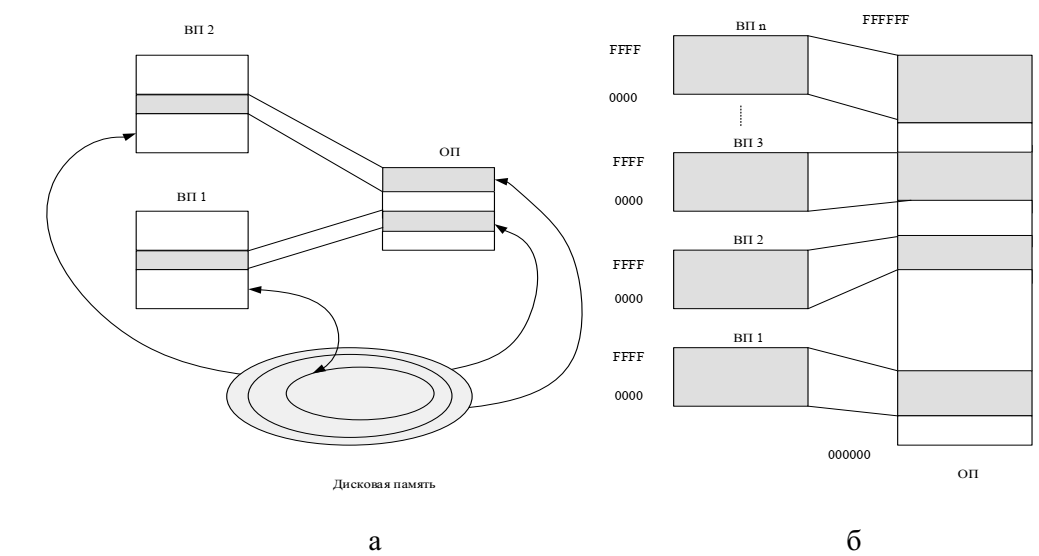
В некоторых случаях, когда заранее точно известно, в какой области оперативной памяти будет выполняться программа, транслятор выдает исполняемый код сразу в физических адресах.

Необходимо различать максимально возможное виртуальное адресное пространство процесса и назначенное (выделенное) процессу виртуальное адресное пространство. В первом случае речь идет о максимальном размере виртуального адресного пространства, определяемом архитектурой компьютера, на котором работает ОС, и, в частности, разрядностью его схем адресации (32-разрядная, 64-разрядная и т. п.). Например, при работе на компьютерах с 32-разрядными процессорами *Intel Pentium* операционная система может предоставить каждому процессу виртуальное адресное пространство размером до 4 Гбайт.

Однако это значение представляет собой только потенциально возможный размер виртуального адресного пространства, который редко на практике необходим процессу. Процесс использует только часть доступного ему виртуального адресного пространства.

Назначенное виртуальное адресное пространство представляет собой набор виртуальных адресов, действительно нужных процессу для работы. Эти адреса первоначально назначает программе транслятор на основании текста программы, когда создает кодовый (текстовый) сегмент, а также сегмент или сегменты данных, с которыми работает программа. Затем при создании процесса ОС фиксирует назначенное виртуальное адресное пространство в своих системных таблицах. В ходе своего выполнения процесс может увеличить размер первоначального назначенного виртуального ему адресного пространства, запросив у ОС создания дополнительных сегментов или увеличения размера существующих. В любом случае операционная система обычно следит за корректностью использования процессом виртуальных адресов – процессу не разрешается оперировать виртуальным адресом, выходящим за пределы назначенных ему сегментов.

Максимальный размер виртуального адресного пространства ограничивается только разрядностью адреса, присущей данной архитектуре компьютера, и, как правило, не совпадает с объемом физической памяти, имеющимся в компьютере.

Сегодня для машин универсального назначения типична ситуация, когда объем виртуального адресного пространства превышает доступный объем оперативной памяти. В таком случае операционная система для хранения данных виртуального адресного пространства процесса, не помещающихся в оперативную память, использует внешнюю память, которая в современных компьютерах представлена жесткими дисками (рис. 4а).

*Рис. 4. Соотношение объемов виртуального адресного пространства и физической памяти: виртуальное адресное пространство превосходит объем физической памяти (а), виртуальное адресное пространство меньше объема физической памяти (б)*

Именно на этом принципе основана виртуальная память – наиболее совершенный механизм, используемый в операционных системах для управления памятью.

Необходимо подчеркнуть, что виртуальное адресное пространство и виртуальная память – это различные механизмы, и они не обязательно реализуются в операционной системе одновременно. Можно представить себе ОС, в которой поддерживаются виртуальные адресные пространства для процессов, но отсутствует механизм виртуальной памяти. Это возможно только в том случае, если размер виртуального адресного пространства каждого процесса меньше объема физической памяти.

Однако соотношение объемов виртуальной и физической памяти может быть и обратным. Так, в миникомпьютерах 80-х годов разрядности поля адреса нередко не хватало для того, чтобы охватить всю имеющуюся оперативную память. Несколько процессов могло быть загружено в память одновременно и целиком (рис. 4 б).

Обычно виртуальное адресное пространство процесса делится на две непрерывные части: системную и пользовательскую.

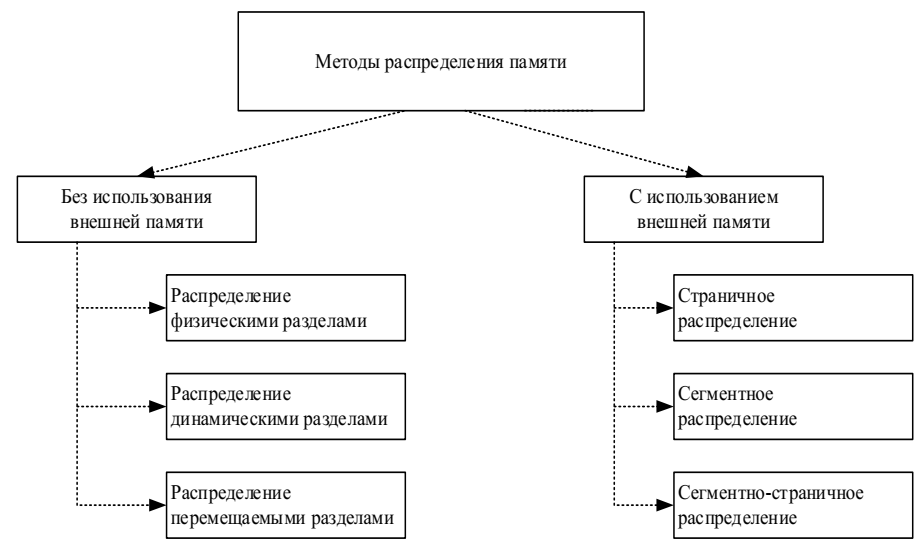
Деление виртуального адресного пространства на две непрерывные области не является обязательным – можно представить себе ОС, которая обходится без него, чередуя свои сегменты и сегменты прикладных программ и размокая их в виртуальном адресном пространстве в произвольном порядке.

Описанное назначение двух частей виртуального адресного пространства – для сегментов ОС и для сегментов прикладной программы – является типичным, но не абсолютным. Имеются и исключения из общего правила. В некоторых ОС существуют системные процессы, порожденные для решения внутренних задач ОС. В этих процессах отсутствуют сегменты прикладной программы, но они могут расположить некоторые свои сегменты (сегменты ОС) в общей части виртуального адресного пространства, а некоторые – в индивидуальной части, обычно предназначенной для прикладных сегментов. И, наоборот, в общей, системной части виртуального адресного пространства размещаются сегменты прикладного кода, предназначенные для совместного использования несколькими прикладными процессами.

Обычно аппаратура накладывает свои ограничения на порядок использования виртуального адресного пространства. Некоторые процессоры (например, MIPS) предусматривают для определенной области системной части адресного пространства особые правила отображения на физическую память. При этом виртуальный адрес прямо отображается на физический (последний либо полностью соответствует виртуальному адресу, либо равен его части). Такая особая область памяти не подвергается страничному вытеснению, и поскольку достаточно трудоемкая процедура преобразования адресов исключается, то доступ к располагаемым здесь кодам и данным осуществляется очень быстро.

*MIPS (англ. Microprocessor without Interlocked Pipeline Stages) — микропроцессор, разработанный компанией MIPS Computer Systems (в настоящее время MIPS Technologies) в соответствии с концепцией проектирования процессоров RISC (то есть для процессоров с упрощенным набором команд).*

**Алгоритмы распределения памяти**. Следует ли назначать каждому процессу одну непрерывную область физической памяти или можно выделять память «кусками»? Должны ли сегменты программы, загруженные в память, находиться на одном месте в течение всего периода выполнения процесса или можно их время от времени сдвигать? Что делать, если сегменты программы не помещается в имеющуюся память? Разные ОС по-разному отвечают на эти и другие базовые вопросы управления памятью. Далее будут рассмотрены наиболее общие подходы к распределению памяти, которые были характерны для разных периодов развития операционных систем. Некоторые из них сохранили актуальность и широко используются в современных ОС, другие же представляют в основном только познавательный интерес, хотя их и сегодня можно встретить в специализированных системах.

На рис. 5 все алгоритмы распределения памяти разделены на два класса: алгоритмы, в которых используется перемещение сегментов процессов между оперативной памятью и диском, и алгоритмы, в которых внешняя память не привлекается.

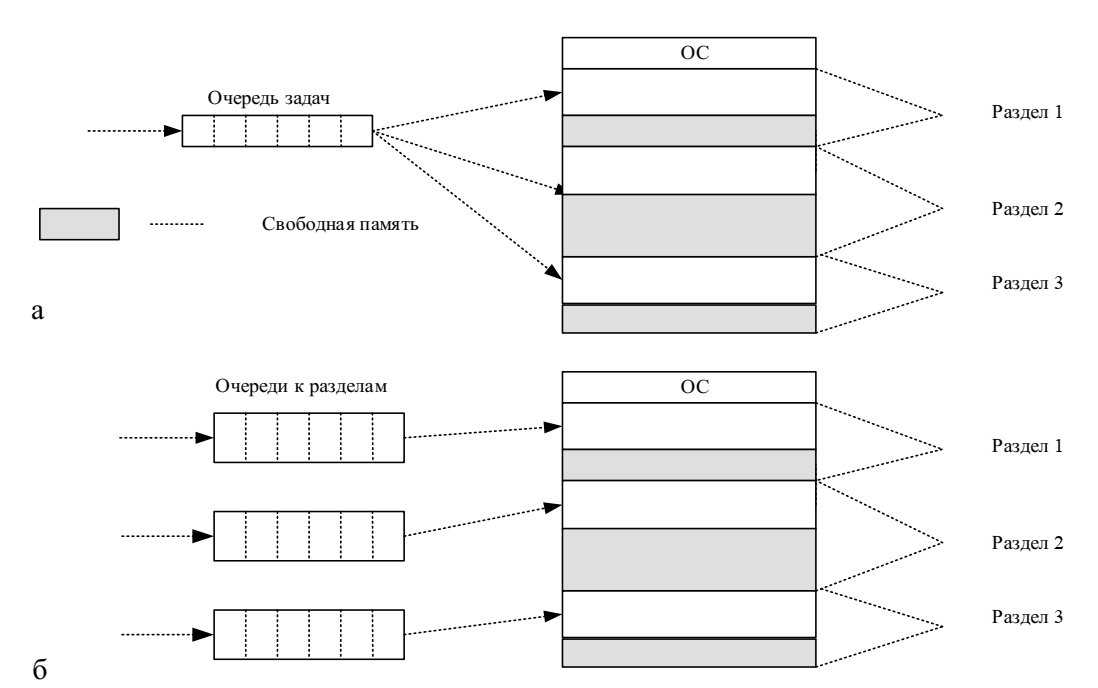
*Рис. 5. Методы распределения памяти.*

**Фиксированные разделы**. Простейший способ управления оперативной памятью состоит в том, что память разбивается на несколько областей фиксированной величины, называемых разделами. Такое разбиение может быть выполнено вручную оператором во время старта системы или вовремя ее установки. После этого границы разделов не изменяются.

Очередной новый процесс, поступивший на выполнение, помещается либо в общую очередь (рис. 6а), либо в очередь к некоторому разделу (рис. 6 б). При распределении памяти фиксированными разделами подсистема управления памятью решает следующие задачи:

* сравнивает объем памяти, требуемый для вновь поступившего процесса, с размерами свободных разделов и выбирает подходящий раздел;
* осуществляет загрузку программы в один из разделов и настройку адресов.

При очевидном преимуществе – простоте реализации, данный метод имеет существенный недостаток – жесткость. Так как в каждом разделе может выполняться только один процесс, то уровень мультипрограммирования заранее ограничен числом разделов. Независимо от размера программы она будет занимать весь раздел. Так, например, в системе с тремя разделами невозможно выполнять одновременно более трех процессов, даже если им требуется совсем мало памяти.

**Такой способ управления памятью применялся в ранних мультипрограммных ОС. Однако и сейчас метод распределения памяти фиксированными разделами находит применение в системах реального времени.

*Рис. 6. Распределение памяти фиксированными разделами: с общей очередью (а), с отдельными очередями (б)*

Детерминированность вычислительного процесса систем реального времени (заранее известен набор выполняемых задач, их требования к памяти, а иногда и моменты запуска) компенсирует недостаточную гибкость данного способа управления памятью.

**Динамические разделы**. В следующем способе управления памятью память машины не делится заранее на разделы. Сначала вся память, отводимая для приложений, свободна. Каждому вновь поступающему на выполнение приложению на этапе создания процесса выделяется вся необходимая ему память (если достаточный объем памяти отсутствует, то приложение не принимается на выполнение и процесс для нее не создается). После завершения процесса память освобождается, и на это место может быть загружен другой процесс.

Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера. На рис. 7 показано состояние памяти в различные моменты времени при динамическом распределении. Так, в момент t0 в памяти находится только ОС, а к моменту tx память оказывается разделенной между 5 процессами, причем процесс П4, завершаясь, покидает память. На освободившееся от процесса П4 место загружается процесс П6, поступивший в момент t3.

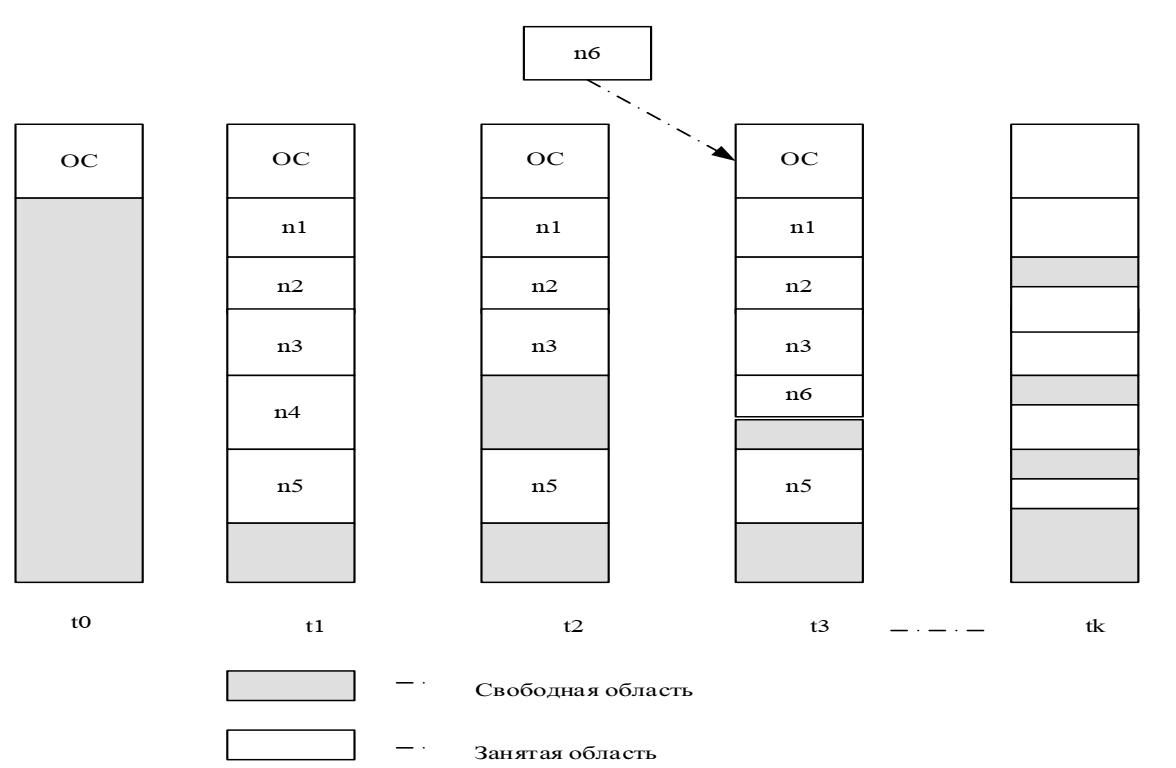
Распределение памяти динамическими разделами сводится к выполнению операционной системой следующих функций:

* ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти;
* при создании нового процесса – анализ требований к памяти, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения кодов и данных нового процесса;
* загрузка программы в выделенный ей раздел и корректировка таблиц свободных и занятых областей;
* после завершения процесса – корректировка таблиц свободных и занятых областей.
* выбор раздела для загрузки кодов процесса при его создании может осуществляться по разным правилам, например: «первый попавшийся раздел достаточного размера», «раздел, имеющий наименьший достаточный размер» или «раздел, имеющий наибольший достаточный размер».

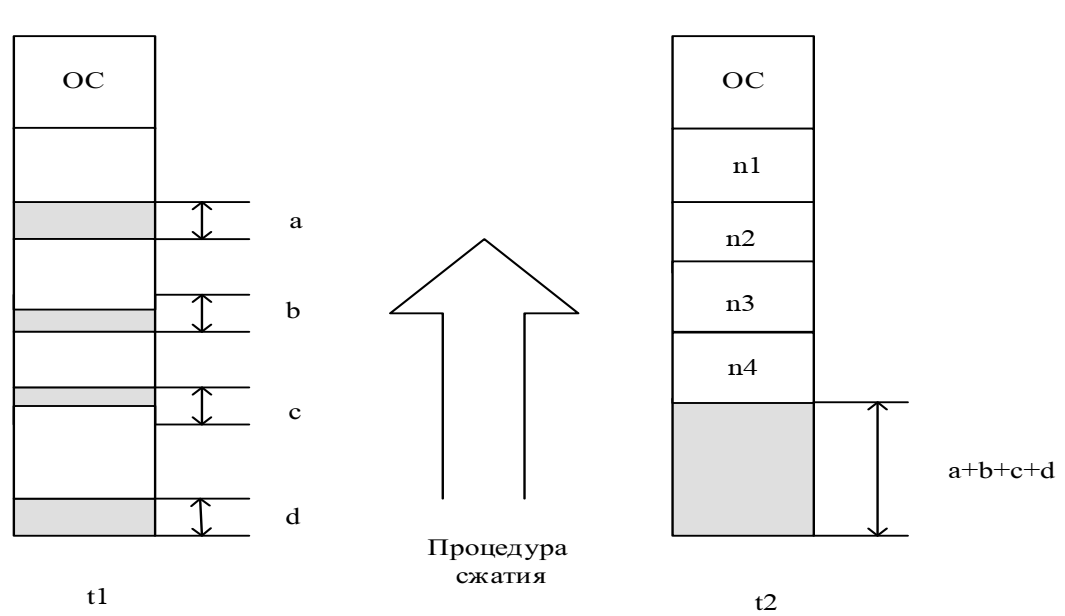
По сравнению с методом распределения памяти фиксированными разделами данный метод обладает существенно большей гибкостью, но ему присущ очень серьезный недостаток – фрагментация памяти.

Фрагментация памяти – это процесс появления большого числа несмежных участков свободной памяти маленького размера (фрагментов), таких, 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.

Распределение памяти динамическими разделами лежит в основе подсистем управления памятью многих мультипрограммных операционных системах 60-70-х годов.



*Рис. 7. Распределение памяти динамическими разделами.*

**Перемещаемые разделы**. Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших или младших адресов, так чтобы вся свободная память образовала единую область (рис. 8).

*Рис. 8. Распределение памяти перемещаемыми разделами.*

В дополнение к функциям, которые выполняет ОС при распределении памяти динамическими разделами, в случае перемещаемых разделов 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 процесса, либо только тогда, когда для вновь создаваемого процесса нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц свободных и занятых областей, во втором реже выполняется процедура сжатия.

Так как программы в ходе своего выполнения перемещаются по оперативной памяти, в данном случае невозможно выполнить настройку адресов с помощью перемещающего загрузчика. Здесь более подходящим оказывается динамическое преобразование адресов. Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может требовать значительного времени, что часто перевешивает преимущества данного метода.

Концепция сжатия применяется и при использовании других методов распределения памяти, когда отдельному процессу выделяется не одна сплошная область памяти, а несколько несмежных участков памяти произвольного размера. Такой подход был характерен для ранних версий систем ОС, в которых память распределялась сегментами, а возникавшая при этом фрагментация периодически устранялась путем перемещения сегментов.

При страничном распределении памяти виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницам.

В общем случае, размер виртуального адресного пространства процесса не кратен 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, или кадрами). Размер страницы выбирается кратным степени двойки: 512, 1024, 4096 байт и т. д., что позволяет упростить механизм преобразования адресов.

При создании процесса ОС загружает в оперативную память несколько его виртуальных страниц (начальные страницы кодового сегмента и сегмента данных). Копия всего виртуального адресного пространства процесса находится на диске. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах.

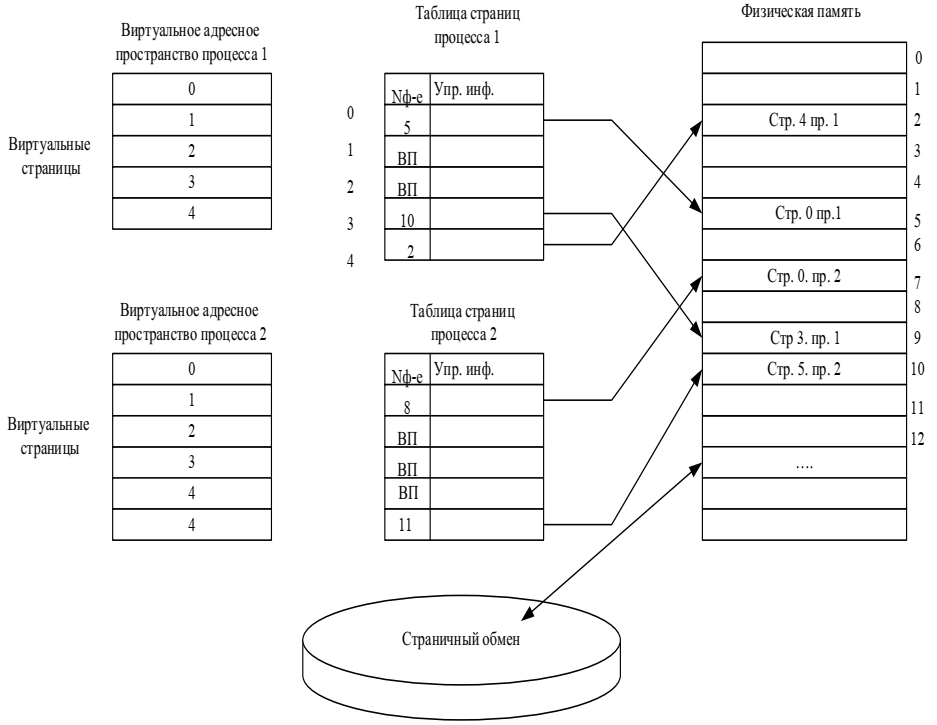
Вопрос 2. Часть 1. Страничная организация виртуальной памяти.

На рис. 9 показана схема страничного распределения памяти.

Для каждого процесса операционная система создает таблицу страниц – информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

Дескриптор страницы – это отдельная запись таблицы. Она включает следующую информацию:

* номер физической страницы, в которую загружена данная виртуальная страница;
* признак присутствия, устанавливаемый в единицу, если виртуальная страница находится в оперативной памяти;
* признак модификации страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;
* признак обращения к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице.

Признаки присутствия, модификации и обращения в большинстве моделей современных процессоров устанавливаются схемами процессора аппаратно при выполнении операции с памятью. Информация из таблиц страниц используется для решения вопроса о необходимости перемещения той или иной страницы между памятью и диском, а также для преобразования виртуального адреса в физический адрес.

*Рис. 9. Страничное распределение памяти.*

Сами таблицы страниц, так же, как и описываемые ими страницы, размещаются в оперативной памяти. Адрес таблицы страниц включается в контекст соответствующего процесса. При активизации очередного процесса операционная система загружает адрес его таблицы страниц в специальный регистр процессора. При каждом обращении к памяти выполняется поиск номера виртуальной страницы, содержащей требуемый адрес, затем по этому номеру определяется нужный элемент таблицы страниц и из него извлекается описывающая страницу информация.

Далее анализируется признак присутствия, и если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический, то есть виртуальный адрес заменяется, указанным в записи таблицы, физическим адресом. 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 процессов, находящихся в состоянии готовности. 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 на основании принятой в данной системе стратегии замещения страниц решается вопрос о том, какую страницу следует выгрузить из оперативной памяти.

После того как выбрана страница, которая должна покинуть оперативную память, обнуляется ее бит присутствия и анализируется ее признак модификации.

Если выталкиваемая страница за время последнего пребывания в оперативной памяти была модифицирована, то ее новая версия должна быть переписана на диск. Если нет, то принимается во внимание, что на диске уже имеется предыдущая копия этой виртуальной страницы, и никакой записи на диск не производится. Физическая страница объявляется свободной. Из соображений безопасности в некоторых системах освобождаемая страница обнуляется, с тем, чтобы невозможно было использовать содержимое выгруженной страницы.

Для хранения информации о положении вытесненной страницы в страничном файле ОС может использовать поля таблицы страниц или же другую системную структуру данных (например, дескриптор сегмента при сегментностраничной организации виртуальной памяти).

Виртуальный адрес при страничном распределении может быть представлен в виде пары (р, su), где р – порядковый номер виртуальной страницы процесса (нумерация страниц начинается с 0), a – смещение в пределах виртуальной страницы.

Физический адрес также может быть представлен в виде пары (n, sf), где n – номер физической страницы, a sf – смещение в пределах физической страницы.

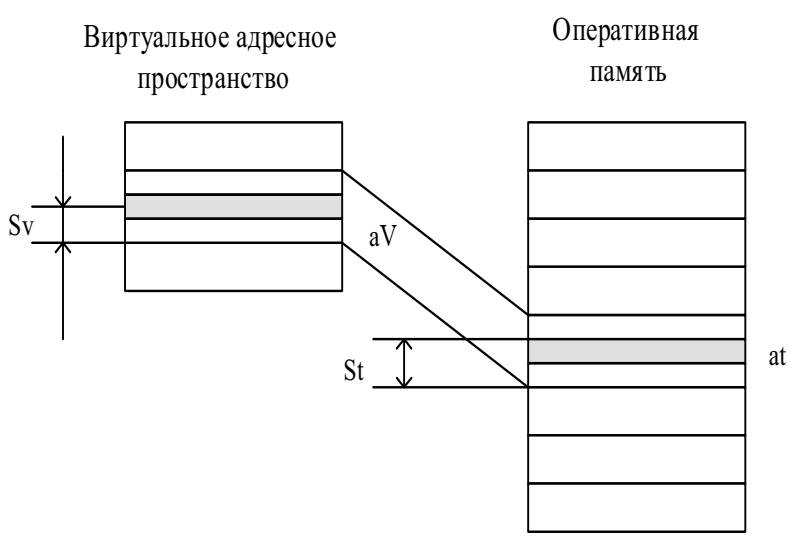
Задача подсистемы виртуальной памяти состоит в отображении пары (р, su) на (n, sf).

Прежде чем приступить к рассмотрению схемы преобразования виртуального адреса в физический адрес остановимся на двух базисных свойствах страничной организации.

Первое из них состоит в том, что объем страницы выбирается кратным степени двойки – 2k. Из этого следует, что смещение *s* может быть получено простым отделением к младших разрядов в двоичной записи адреса, а оставшиеся старшие разряды адреса представляют собой двоичную запись номера страницы (при этом неважно, является страница виртуальной или физической). Например, если размер страницы составляет 1 Кбайт (210), то из двоичной записи адреса 50718 − 1010001110012 можно определить, что он принадлежит странице, номер которой в двоичном выражении равен 102 и смещен относительно ее начала на 10001110012 байт (рис. 10).

*Рис. 10. Двоичное представление адресов.*

Из рисунка хорошо видно, что номер страницы и ее начальный адрес легко могут быть получены один из другого дополнением или отбрасыванием *k* нулей, соответствующих смещению. Именно по этой причине часто говорят, что таблица страниц содержит начальный физический адрес страницы в памяти (а не номер физической страницы), хотя на самом деле в таблице указаны только старшие разряды адреса. Начальный адрес страницы называется базовым адресом.



*Рис. 11. Схема отображения виртуального адреса на физический адрес.*

Второе свойство заключается в том, что в пределах страницы непрерывная последовательность виртуальных адресов однозначно отображается на непрерывную последовательность физических адресов, а значит смещения в виртуальном и физическом адресах *Sv* и *Sy* равны между собой (см. рис. 11).

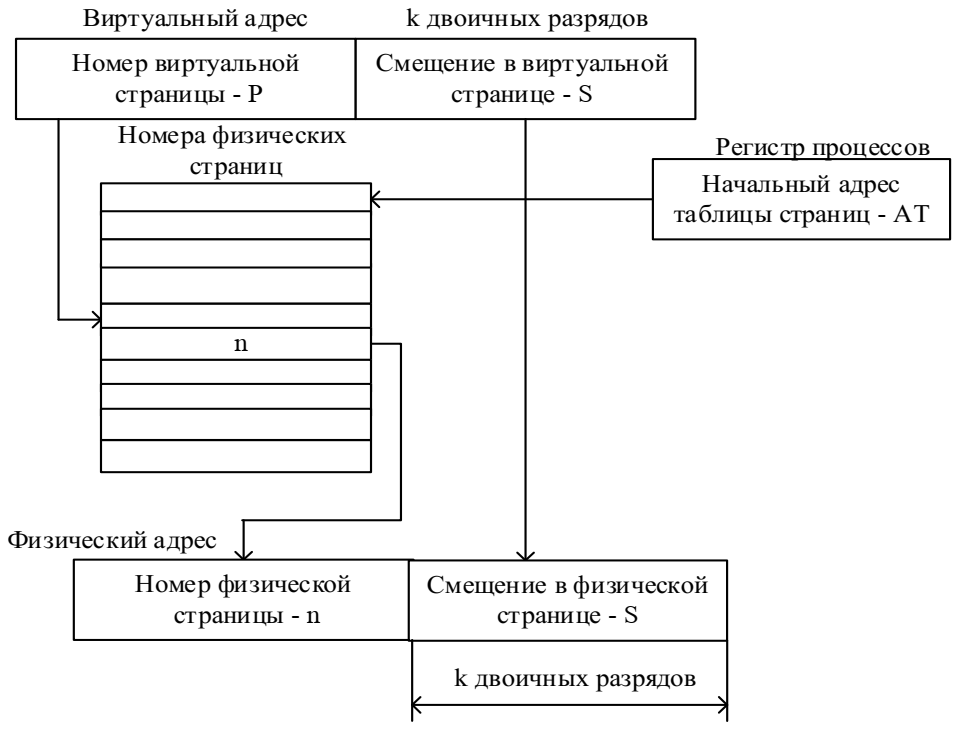
Отсюда следует простая схема преобразования виртуального адреса в физический (рис. 12).

Младшие разряды физического адреса, соответствующие смещению, получаются переносом такого же количества младших разрядов из виртуального адреса. Старшие разряды физического адреса, соответствующие номеру физической страницы, определяются из таблицы страниц, в которой указывается соответствие виртуальных и физических страниц.

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 преимуществ работы с виртуальной памятью большого объема, сохраняет другое достоинство страничной организации – позволяет успешно бороться с фрагментацией физической памяти.

Действительно, во-первых, программу можно разбить на части и загрузить в разрозненные участки свободной памяти, во-вторых, при загрузке виртуальных страниц никогда не образуется неиспользуемых остатков, так как размеры виртуальных и физических страниц совпадают.



*Рис. 12. Схема преобразования виртуального адреса в физический адрес при страничной организации памяти.*

Такой режим работы системы управления памятью используется в некоторых специализированных ОС, когда требуется высокая реактивность системы и способность выполнять переменный набор приложений.

Вопрос 2. Часть 2. Сегментная организация виртуальной памяти.

При страничной организации виртуальное адресное пространство процесса делится на равные части механически, без учета смыслового значения данных. В одной странице могут оказаться и коды команд, и инициализируемые переменные, и массив исходных данных программы. Такой подход не позволяет обеспечить дифференцированный доступ к разным частям программы, а это свойство могло бы быть очень полезным во многих случаях. Например, можно было бы запретить обращаться с операциями записи в сегмент программы, содержащий коды команд, разрешив эту операцию для сегментов данных.

Кроме того, разбиение виртуального адресного пространства на «осмысленные» части делает принципиально возможным совместное использование фрагментов программ разными процессами.

Пусть, например, двум процессам требуется одна и та же подпрограмма, которая к тому же обладает свойством реентерабельности. Тогда коды этой подпрограммы могут быть оформлены в виде отдельного сегмента и включены в виртуальные адресные пространства обоих процессов. При отображении на физическую память сегменты, содержащие коды подпрограммы из обоих виртуальных пространств, проецируются на одну и ту же область физической памяти. Таким образом, оба процесса получают доступ к одной и той же копии подпрограммы (рис. 13).

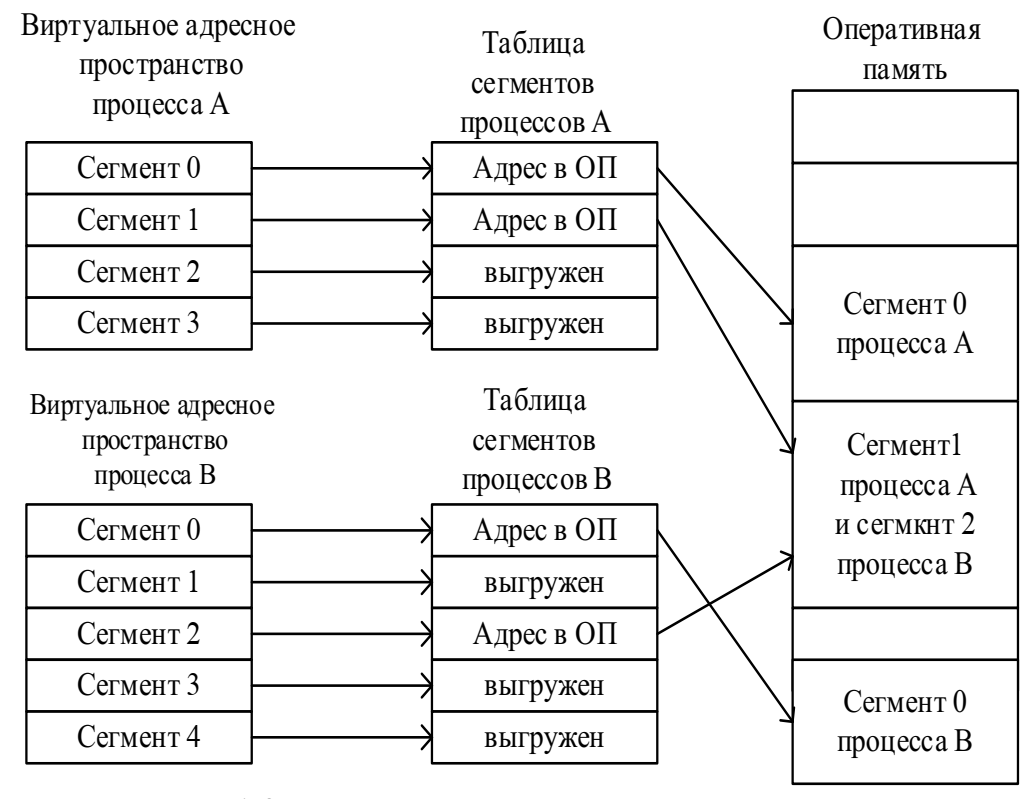
При сегментной организации памяти виртуальное адресное пространство процесса делится на части – сегменты, размер которых определяется с учетом смыслового значения содержащейся в них информации.

На этапе создания процесса во время загрузки его образа в оперативную память система создает таблицу сегментов процесса.

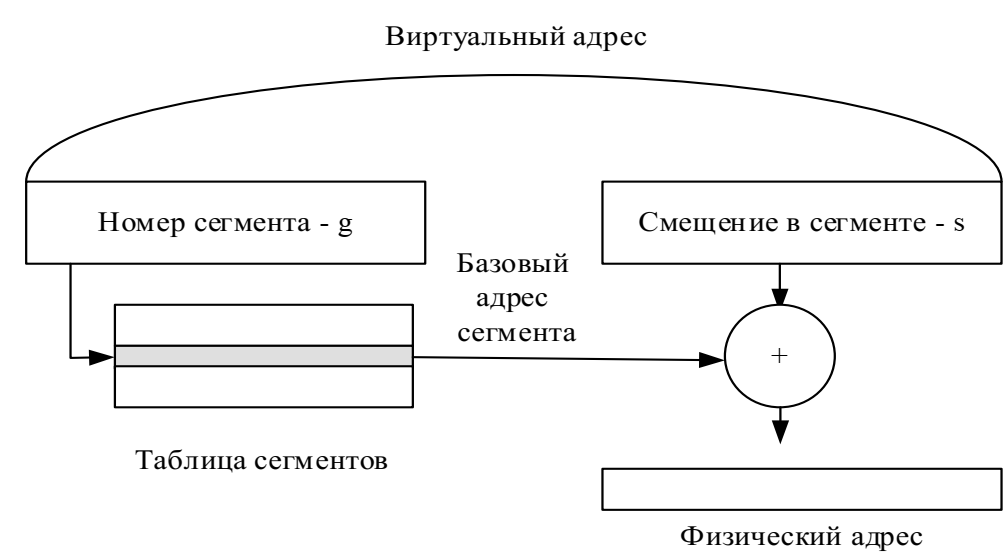
Каждая запись в таблице сегментов содержит следующие характеристики соответствующего сегмента:

* базовый физический адрес сегмента в оперативной памяти;
* размер сегмента;
* правила доступа к сегменту;
* признаки модификации, присутствия и обращения к данному сегменту, а также 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.

Как видно, сегментное распределение памяти имеет много общего со страничным распределением.

*Рис. 13. Распределение памяти сегментами.*

Механизмы преобразования адресов этих двух способов управления памятью тоже весьма схожи, однако в них имеются и существенные отличия, которые являются следствием того, что сегменты, в отличие от страниц, имеют произвольный размер.

*Рис. 14. Преобразование виртуального адреса при сегментной организации памяти.*

Виртуальный адрес при сегментной организации памяти может быть представлен парой (*g*, *s*), где *g* – номер сегмента, a *s* – смещение в сегменте.

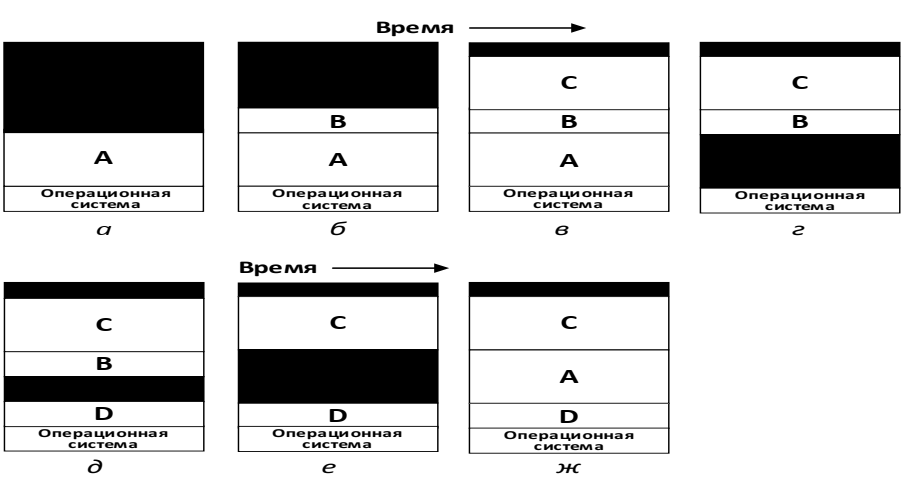
Физический адрес получается путем сложения базового адреса сегмента, который определяется по номеру сегмента *g* из таблицы сегментов и смещения *s* (см. рис. 14).

Одним из существенных отличий сегментной организации памяти от страничной является возможность задания дифференцированных прав доступа процесса к его сегментам.

Например, один сегмент данных, содержащий исходную информацию для приложения, может иметь права доступа «только чтение», а сегмент данных, представляющий результаты, – «чтение и запись». Это свойство означает принципиальное преимущество сегментной модели памяти над страничной.

Вопрос 2. Часть 3. Свопинг.

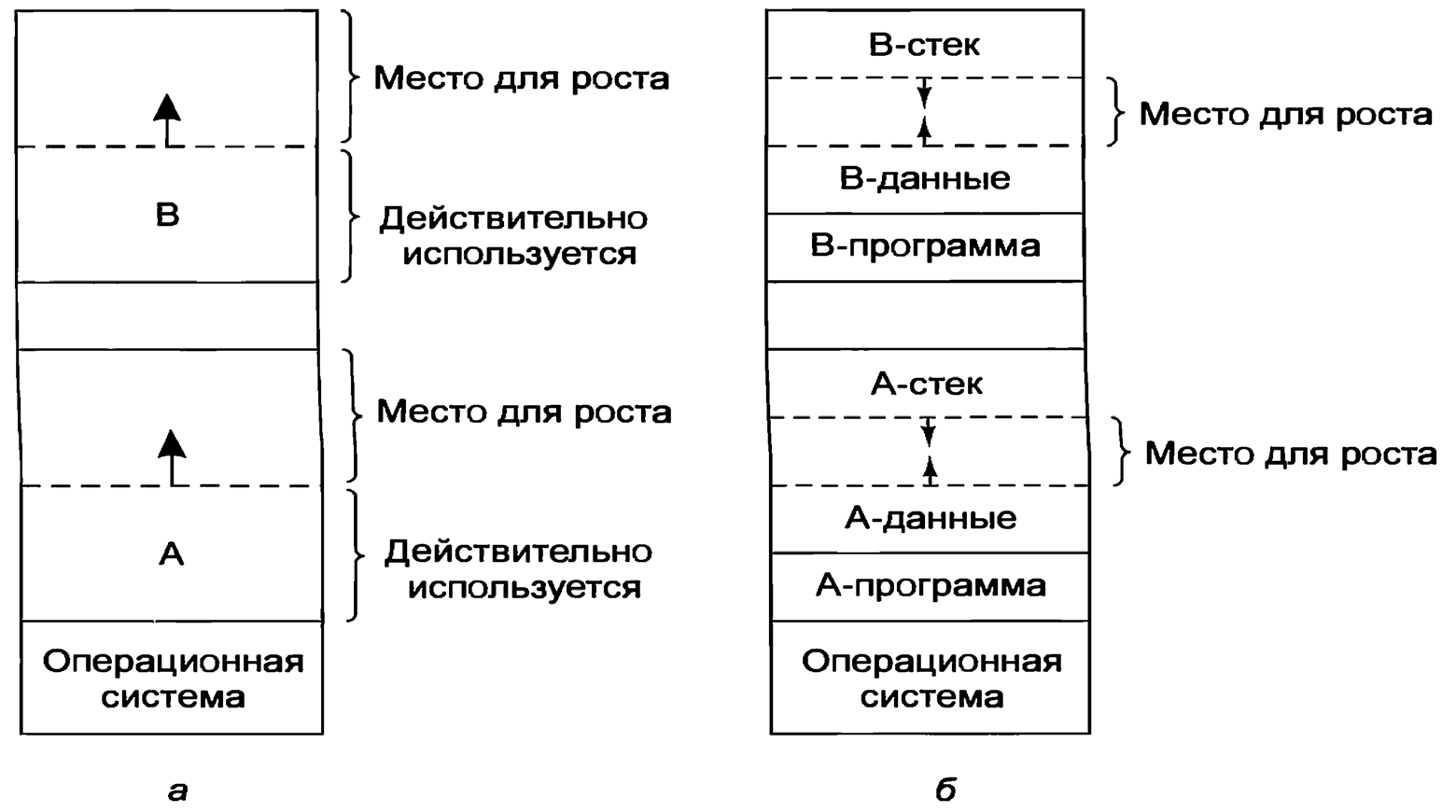
Как правило, суммарный объем оперативной памяти, необходимый для размещения всех процессов, зачастую значительно превышает имеющийся объем ОЗУ. Следовательно, постоянное содержание всех процессов в памяти требует огромных объемов и не может быть осуществлено при дефиците памяти.

Для преодоления перегрузки памяти были выработаны два основных подхода. Самый простой из них, называемый свопингом, заключается в размещении в памяти всего процесса целиком, в запуске его на некоторое время, а затем в сбросе его 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).

*Рис. 15. Изменения в выделении памяти по мере появления процессов в памяти и выгрузки их из нее (неиспользованные области памяти заштрихованы).*

Второй подход называется виртуальной памятью, он позволяет программам запускаться даже в том случае, если они находятся в оперативной памяти лишь частично. Работа системы с использованием свопинга показана на рис. 15.

Свопингу на диск должна подвергаться только реально задействованная память, копировать при этом еще и дополнительно выделенную память будет слишком расточительно. На рис. 16а показана конфигурация памяти, в которой пространства для разрастания были выделены двум процессам.



*Рис. 16. Выделение памяти под разрастающийся сегмент данных (а); выделение памяти под разрастающийся стек и сегмент данных (б)*

Если процессы могут иметь два разрастающихся сегмента, к примеру сегмент данных, используемый в качестве динамически выделяемой и освобождаемой памяти для переменных, и сегмент стека для обычных локальных переменных и адресов возврата, то предлагается альтернативная структура распределения памяти, показанная на рис. 1.40б.

На этом рисунке видно, что у каждого изображенного процесса в верхних адресах выделенной ему памяти имеется стек, растущий вниз, и сразу над текстом программы – сегмент данных, растущий вверх. Разделяющая их память может использоваться обоими сегментами. Если она иссякнет, процесс должен быть перемещен в свободное пространство достаточного размера (или же перемещен из памяти в область свопинга) до тех пор, пока не будет создано достаточное по размеру свободное пространство, либо он должен быть уничтожен.

Вопрос 3. Разделяемая память.

**Вопрос 3. Часть 1. Совместный доступ к одному сегменту памяти.**

Подсистема виртуальной памяти представляет собой удобный механизм для решения задачи совместного доступа нескольких процессов к одному и тому же сегменту памяти, который в этом случае называется разделяемой памятью.

Хотя основной задачей операционной системы при управлении памятью является защита областей оперативной памяти, принадлежащей одному из процессов, от доступа к ней остальных процессов, в некоторых случаях оказывается полезным организовать контролируемый совместный доступ нескольких процессов к определенной области памяти.

Для организации разделяемого сегмента при наличии подсистемы виртуальной памяти достаточно поместить его в виртуальное адресное пространство каждого процесса, которому нужен доступ к данному сегменту, а затем настроить параметры отображения этих виртуальных сегментов так, чтобы они соответствовали одной и той же области оперативной памяти. Детали такой настройки зависят от типа используемой в ОС модели виртуальной памяти: сегментной или сегментно-страничной (чисто страничная организация не поддерживает понятие «сегмент», что делает невозможным решение рассматриваемой задачи).

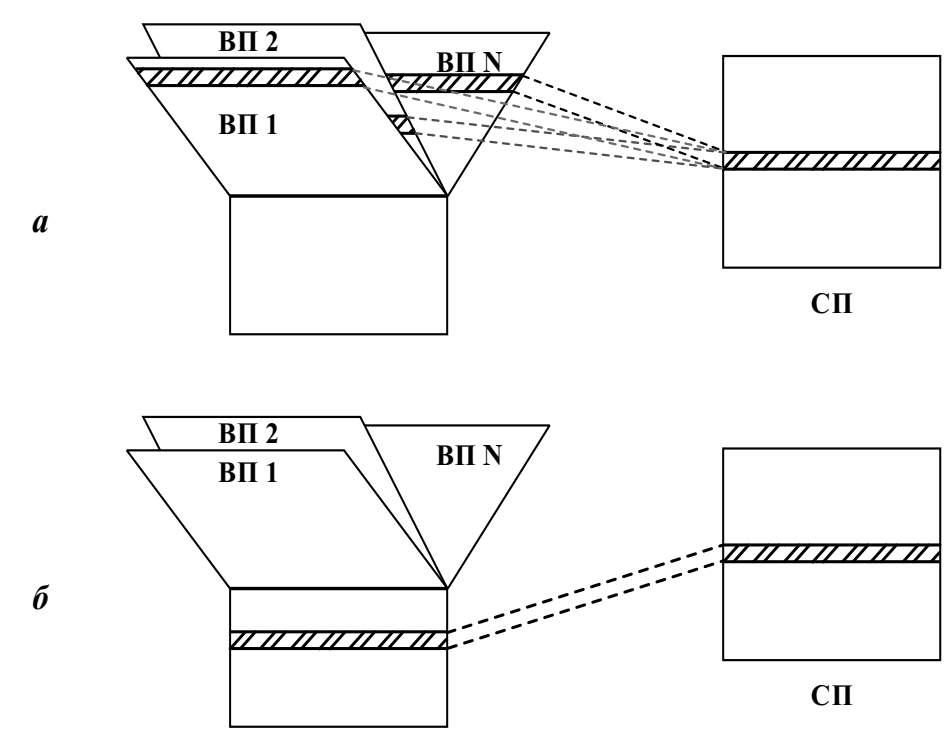
В приведенном выше описании подразумевалось, что разделяемый сегмент помещается в индивидуальную часть виртуального адресного пространства каждого процесса (рис. 17а) и описывается в каждом процессе индивидуальным дескриптором сегмента (и индивидуальными дескрипторами страниц, если используется сегментно-страничный механизм). «Попадание» же этих виртуальных сегментов на общую часть оперативной памяти достигается за счет согласованной настройки операционной системой многочисленных дескрипторов для множества процессов.

Возможно и более экономичное для ОС решение этой задачи – помещение единственного разделяемого виртуального сегмента в общую часть виртуального адресного пространства процессов, то есть в ту часть, которая обычно используется для модулей ОС (17б).

В этом случае настройка дескриптора сегмента (и дескрипторов страниц) выполняется только один раз, а все процессы пользуются такой настройкой и совместно используют часть оперативной памяти.

Операционная система может создавать разделяемые сегменты, как по явному запросу, так и по умолчанию. В первом случае прикладной процесс должен выполнить соответствующий системный вызов, по которому операционная система создает новый сегмент в соответствии с указанными в вызове параметрами: размером сегмента, разрешенными над ним операциями (чтение/запись) и идентификатором.

Во втором случае операционная система сама в определенных ситуациях принимает решение о том, что нужно создать разделяемый сегмент.

**

*Рис. 17. Два способа создания разделяемого сегмента памяти Разделяемые сегменты выгружаются на диск системой виртуальной памяти по тем же алгоритмам и с помощью тех же механизмов, что и индивидуальные*.

**Вопрос 3. Часть 2. Функции ядра ОС МСВС для работы с разделяемой памятью.**

Процессы должны предварительно «договориться» о правилах использования разделяемой памяти. Например, пока один из процессов производит запись данных в разделяемую память, другие процессы должны воздержаться от работы с ней. К счастью, задача кооперативного использования разделяемой памяти, заключающаяся в синхронизации выполнения процессов, легко решается с помощью семафоров.

Примерный сценарий работы с разделяемой памятью выглядит следующим образом:

* + - 1. Сервер получает доступ к разделяемой памяти, используя семафор.
      2. Сервер производит запись данных в разделяемую память.
      3. После завершения записи сервер освобождает разделяемую память с помощью семафора.
      4. Клиент получает доступ к разделяемой памяти, запирая ресурс с помощью семафора.
      5. Клиент производит чтение данных из разделяемой памяти и освобождает ее, используя семафор.

Для каждой области разделяемой памяти, ядро поддерживает структуру данных *shmid\_ds*, основными полями которой являются:

|  |  |
| --- | --- |
| Struct ipc\_perm shm\_perm | – права доступа, владельца и создателя области; |
| int shm\_segsz | – размер выделяемой памяти; |
| ushort shm\_nattch | – число процессов, использующих разделяемую память; |
| time\_t shm\_atime | – время последнего присоединения к разделяемой памяти; |
| time\_t shm\_dtime | – время последнего отключения от разделяемой памяти; |
| time\_t shm\_ctime | – время последнего изменения. |

Системные функции для работы с разделяемой памятью:

* функция *shmget* создает новую область разделяемой памяти или возвращает адрес уже существующей области;
* функция *shmat* логически присоединяет область к виртуальному адресному пространству процесса;
* функция *shmdt* отсоединяет ее;
* функция *shmctl* имеет дело с различными параметрами, связанными с разделяемой памятью.

Процессы ведут чтение и запись данных в области разделяемой памяти, используя для этого те же самые машинные команды, что и при работе с обычной памятью. После присоединения к виртуальному адресному пространству процесса область разделяемой памяти становится доступна так же, как любой

участок виртуальной памяти; для доступа к находящимся в ней данным не нужны обращения к каким-то дополнительным системным функциям.

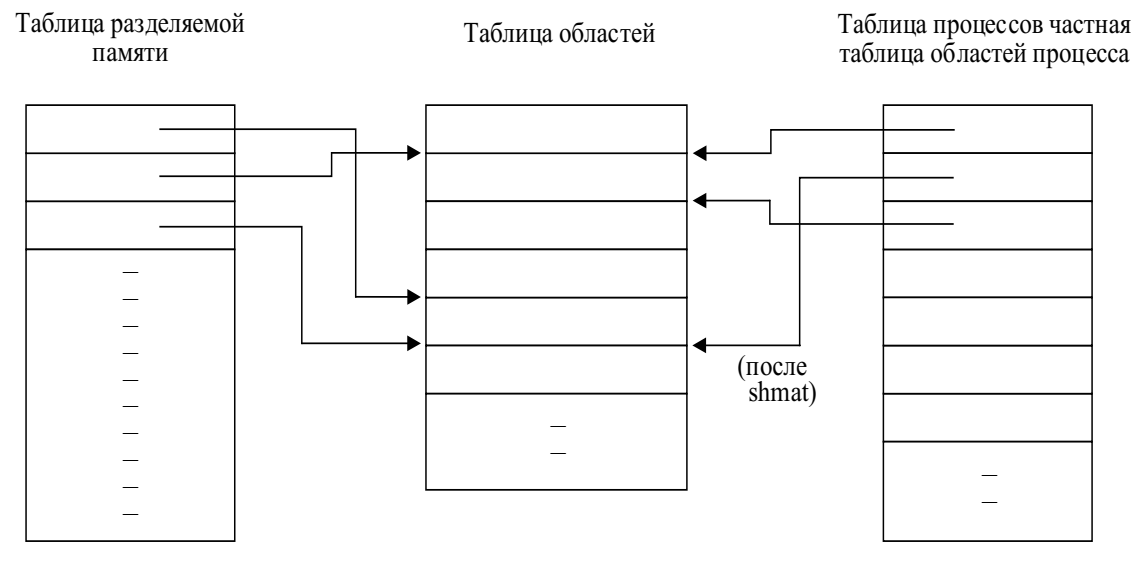
**Вопрос 3. Часть 3. Функции *API* OC МСВС для работы с разделяемой памятью.**

Для создания или для доступа к уже существующей разделяемой памяти используется системный вызов *shmget*(2).

Функция возвращает дескриптор разделяемой памяти в случае успеха, и -1 в случае неудачи. Аргумент *size* определяет размер создаваемой области памяти в байтах.

Значения аргумента *shmflag* задают права доступа к объекту и специальные флаги *IPC\_CREAT и IPC\_EXCL*.

Заметим, что вызов *shmget*(2) лишь создает или обеспечивает доступ к разделяемой памяти, но не позволяет работать с ней. Если запись не найдена и если пользователь установил флаг *IPC\_CREAT*, указывающий на необходимость создания новой области, ядро проверяет нахождение размера области в установленных системой пределах и выделяет область по алгоритму *allocreg*.

Ядро записывает установки прав доступа, размер области и указатель на соответствующую запись таблицы областей в таблицу разделяемой памяти (рис. 18) и устанавливает флаг, свидетельствующий о том, что с областью не связана отдельная память.

*Рис. 18. Структуры данных, используемые при разделении памяти Области выделяется память (таблицы страниц и т. п.) только тогда, когда процесс присоединяет область к своему адресному пространству.*

Ядро устанавливает также флаг, говорящий о том, что по завершении последнего связанного с областью процесса область не должна освобождаться. Таким образом, данные в разделяемой памяти остаются в сохранности, даже если она не принадлежит ни одному из процессов (как часть виртуального адресного пространства последнего).

Для работы с разделяемой памятью (чтение и запись) необходимо сначала присоединить (*attach*) область вызовом shmat(2):

**#*incluude <sys/types.h*>**

**#*include <sys/ipc.h*>**

**#*include <sys/shm.h>***

***char \*shmat*(*int shmid, char \*shmaddr, int shmflag*);**

Вызов *shmat*(2) возвращает адрес начала области в адресном пространстве процесса размером *size*, заданным предшествующем вызовом *shmget*(2). В этом адресном пространстве взаимодействующие процессы могут размещать требуемые структуры данных для обмена информацией.

Правила получения этого адреса следующие:

* если аргумент *shmaddr* нулевой, то система самостоятельно выбирает адрес;
* если аргумент *shmaddr* отличен от нуля, значение возвращаемого адреса зависит от наличия флажка *SHM\_RND* в аргументе *shmflag*;
* если флажок *SHM\_RND* не установлен, система присоединяет разделяемую память к указанному *shmaddr* адресу;
* если флажок *SHM\_RND* установлен, система присоединяет разделяемую память к адресу, полученному округлением в меньшую сторону *shmaddr* до некоторой определенной величины *SHMLBA*.

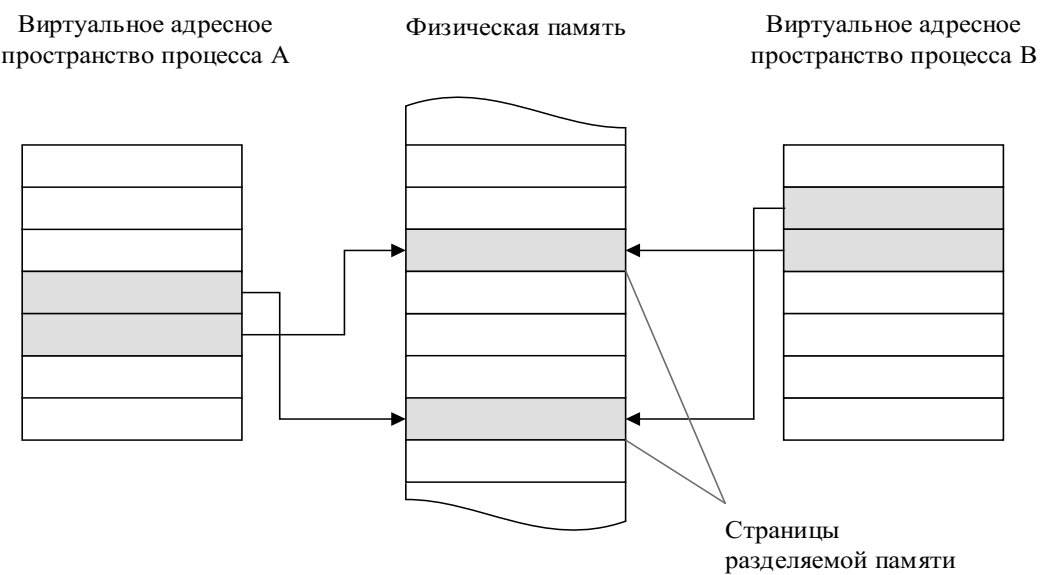
По умолчанию разделяемая память присоединяется с правами на чтение и запись. Эти права можно изменить, указав флажок *SHM\_RDONLY* в аргументе *shmflag*.

Таким образом, несколько процессов могут отображать область разделяемой памяти в различные участки собственного виртуального адресного пространства, как это показано на рис. 19.

В начале выполнения системной функции *shmat* ядро проверяет наличие у процесса необходимых прав доступа к области. Оно исследует указанный пользователем адрес; если он равен 0, ядро выбирает виртуальный адрес по своему усмотрению.

Область разделяемой памяти не должна пересекаться в виртуальном адресном пространстве процесса с другими областями; следовательно, ее выбор должен производиться разумно и осторожно.

Так, например, процесс может увеличить размер принадлежащей ему области данных с помощью системной функции *brk*, и новая область данных будет содержать адреса, смежные с прежней областью; поэтому, ядру не следует присоединять область разделяемой памяти слишком близко к области данных процесса.

**

*Рис. 19. Совместное использование разделяемой памяти.*

Так же не следует размещать область разделяемой памяти вблизи от вершины стека, чтобы стек при своем последующем увеличении не залезал за ее пределы. Если, например, стек растет в направлении увеличения адресов, необходимо разместить область разделяемой памяти непосредственно перед началом области стека.

Ядро проверяет возможность размещения области разделяемой памяти в адресном пространстве процесса и присоединяет ее с помощью алгоритма *attachreg*. Если вызывающий процесс является первым процессом, который присоединяет область, ядро выделяет для области все необходимые таблицы, используя алгоритм *growreg*, записывает время присоединения в соответствующее поле таблицы разделяемой памяти и возвращает процессу виртуальный адрес, по которому область была им подключена фактически.

Окончив работу с разделяемой памятью, процесс отключает (*detach*) область вызовом *shmdt* (2).

При работе с разделяемой памятью необходимо синхронизировать выполнение взаимодействующих процессов: когда один из процессов записывает данные в разделяемую память, остальные процессы ожидают завершения операции. Обычно синхронизация обеспечивается с помощью семафоров, назначение и число которых определяется конкретным использованием разделяемой памяти.