Все **алгоритмы для управления областями памяти** можно отнести к нескольким категориям, по типам объектов, с которыми они оперируют:

1. Объекты одного типа. Для выделения памяти под эти объекты нецелесообразно использовать malloc. Так как размер таких объектов постоянен, лучше использовать свой менеджер объектов, либо так называемый [slab аллокатор](http://www.ibm.com/developerworks/ru/library/l-linux-slab-allocator/).
2. Объекты определенного размера (например, страницы физической памяти). Для таких объектов наиболее подходят метод битовых масок или алгоритм "близнецов".
3. Объекты произвольного размера и типа (malloc). Для таких целей, как правило, применяется метод граничных маркеров.

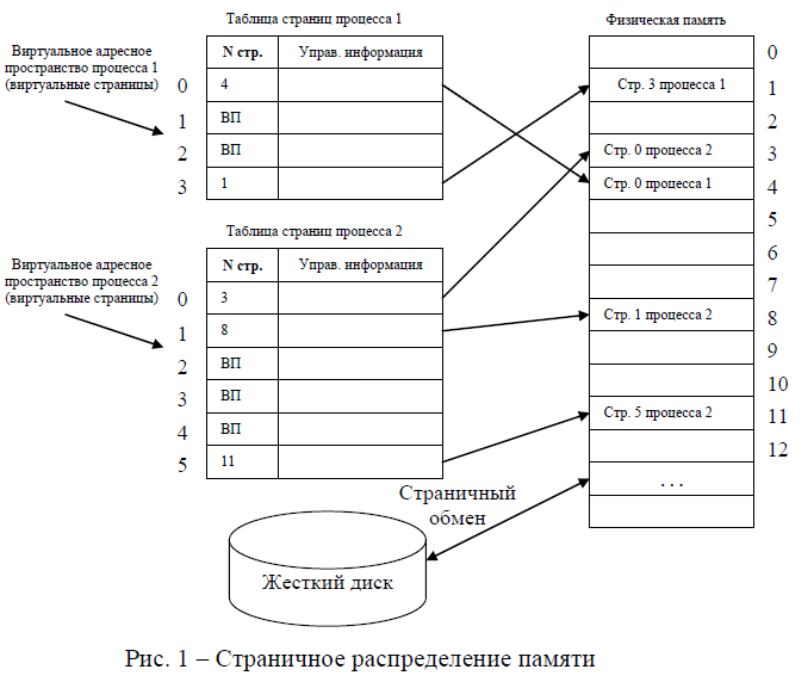
В современных системах используется двухуровневый подход. На нижнем уровне идет выделение страниц памяти, а на верхнем уровне происходит выделение кусков произвольной длины. Верхний уровень запрашивает память у системы с помощью диспетчера страниц. Существуют также системы, которые не нуждаются в нижнем уровне (диспетчере страниц), это, например, встроенные системы с ограниченными ресурсами, выполняющие статический набор процессов. В таких случаях диспетчер памяти может захватить при инициализации отведенное для него пространство и больше никогда не обращаться к диспетчеру страниц. Также это часто применяется в системах реального времени, где жертвуют предсказуемостью за счет удобства.

Для справки(необяз.):

**Страничная память** — способ организации [виртуальной памяти](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C), при котором единицей отображения виртуальных адресов на физические является регион постоянного размера (т. н. *страница*). Типичный размер страницы — 4096 байт, для некоторых архитектур — до 128 КБ.

На рисунке 1 показана схема страничного распределения памяти. Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами (virtual pages). В общем случае размер виртуального адресного пространства процесса не кратен размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

**Виртуа́льная па́мять** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *virtual memory*) — метод [управления памятью](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C%D1%8E) [компьютера](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), позволяющий выполнять программы, требующие больше [оперативной памяти](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C), чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем



Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками, или кадрами). Размер страницы выбирается равным степени двойки: 512, 1024, 4096 байт и т. д. Это позволяет упростить механизм преобразования адресов. Операционная система при создании процесса 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. Для каждого процесса операционная система создает таблицу страниц – информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

**Менеджер объектов**

Данный метод подходит для выделения и освобождения большого количества одинаковых объектов, ограничение на количество заданно на момент компиляции системы.

Под объекты выделяется массив памяти из максимального количества объектов данного типа, и содержащиеся в нем объекты связываются некоторой структурой данных, например списком свободных объектов. Тогда выделение объекта - это всего лишь возвращение ссылки на голову списка, а освобождение объекта - это помещение ссылки на него в конец очереди свободных объектов.

Данный метод может применяться также в системах реального времени, поскольку известно что поиск подходящего куска памяти произвольной длины более трудоемкая задача и требует больших временных затрат.

В качестве доказательства достаточно привести временные результаты следующего теста. В первом случае происходит выделение с помощью системных функций malloc или new множества однотипных объектов. Во втором случае однократно выделяется пул для всех объектов сразу, и с помощью например списка создается список этих объектов. После этого в программе объекты выделяются из этого списка.

Во втором случае затраченное время будет существенно ниже

**Slab менеджер**

Менеджер объектов обладает хорошими временными характеристиками, но достаточно накладно писать для каждого объекта свой менеджер. Чтобы избежать дублирования кода, но сохранить предсказуемость и скорость применяют [slab allocator](http://www.ibm.com/developerworks/ru/library/l-linux-slab-allocator/). Основная суть этого метода заключается в частом выделении однотипных объектов, то есть ситуация описанная в разделе **Менеджер объектов**. Но в отличие него в системе вводится единый интерфейс для выделения и освобождения различных однотипных объектов. Для каждого из этих объектов создается свой пул, а в системе создается кеш подобных пулов для различных зарегистрированных объектов.

**Алгоритм битовой маски.** При выделении процессу области памяти набор блоков заданного размера маркируется специальным массивом, определяющий занятость или возможность доступа к заданному блоку. При необходимости выделения диспетчер просматривает битовый массив, определяет блок доступ, к которому возможен, возвращает ссылку на этот блок в качестве результата выделения памяти и изменяет значение битовой маски (1 — занято, 0 - свободно). При освобождении достаточно изменить значение соответствующего бита маски, что приведет к возможности повторного использования данного участка, а, следовательно, к потере хранимых в нем данных

**Алгоритмом близнецов** используется когда различные размеры являются степенями числа 2, как 512 байт, 1Кбайт, 2Кбайта и т.д. Он состоит в том, что мы ищем блок требуемого размера в соответствующем списке. Если этот список пуст, мы берем список блоков вдвое большего размера. Получив блок большего размера, мы делим его пополам. Ненужную половину мы помещаем в соответствующий список свободных блоков. Одно из преимуществ этого метода состоит в простоте объединения блоков при их освобождении. Действительно, адрес блока-близнеца получается простым инвертированием соответствующего бита в адресе нашего блока. Нужно только проверить, свободен ли этот близнец. Если он свободен, то мы объединяем братьев в блок вдвое большего размера, и т.д.

Алгоритм близнецов значительно снижает фрагментацию памяти и резко ускоряет поиск блоков. Наиболее важным преимуществом этого подхода является то, что даже в наихудшем случае время поиска не превышает. Это делает алгоритм близнецов труднозаменимым для ситуаций, когда необходимо гарантированное время реакции - например, для задач реального времени. Часто этот алгоритм или его варианты используются для выделения памяти внутри ядра ОС. Например, функция kmalloc, используемая в ядре ОС *Linux*, основана именно на алгоритме близнецов.

**Bitmap**

Для представления памяти используется битовая карта, в которой каждому блоку соответствует один бит. Предназначен для выделения блоков памяти, выровненных по адресу и длине, например, страниц физической памяти

**Метод граничных маркеров (Boundary markers)**

В основе метода - двусвязный список всех свободных и занятых блоков памяти. Как правило, элементами списка являются сами блоки памяти, точнее их "шапки", называемые маркерами. Модификация списка происходит в несколько простых операций. Существует несколько алгоритмов поиска свободного блока памяти заданного размера. Среди них - алгоритм первого соответствия, следующего соответствия, наилучшего и наихудшего соответствия. Кроме того, можно поддерживать списки блоков для часто используемых запросов (т.н. алгоритм быстрого соответствия). Предназначен для выделения блоков памяти произвольной длины. Используется в аллокаторах общего назначения.

Алгоритм граничных маркеров хорошо подходит под выделение объектов произвольной длины, для сегмента же кода обычно используют другие алгоритмы. Алгоритм описан в книге Дэвида Кнута "Искусство программирование часть 1".

Для работы данного алгоритма в каждый свободный блок добавляется два дескриптора (расположенные в начале и в конце каждого блока) и два указателя (расположенные в начале блока) на предыдущий и последующий свободные блоки. Таким образом все свободные блоки соединяются в двунаправленный список, по которому и происходит поиск подходящего блока для выделения.

При выделении блока памяти помимо запрашиваемой памяти выделяется память под дескрипторы блоков, точно так же, один дескриптор в начале блока, другой - в конце. Таким образом выделяемый блок не может быть меньше, чем два указателя, которые могут понадобиться при освобождении блока под ссылки на предыдущий и последующий свободные блоки.

Дескриптор блока представляет из себя размер блока и флаг (свободен или занят) в терминах Кнута (Тэга блока). При использовании структуры, описанной выше, процессы выделения и освобождения блока приобретают следующий вид:

*Выделение*

Берем голову списка и, итерируясь по указателям, ищем подходящий блок. Могут быть различные стратегии:

* первый подходящий
* наилучший подходящий
* наихудший подходящий - в этом случае берется наибольший блок и от него вырезается требуемый размер.

Первый вариант быстрее, но второй и третий приводят к меньшей фрагментации памяти.

*Освобождение*

Основное преимущество данного алгоритма - в скорости освобождения памяти. Поскольку при освобождении мы знаем адрес освобождаемого блока, на предыдущем адресе лежит дескриптор блока. Кроме того, если освобождаемый блок не является крайним, непосредственно слева и справа от него расположены дескрипторы соседних блоков, содержащую информацию о занятости и размере. Это значительно ускоряет процесс объединения соседних свободных от блоков, образовавшихся при освобождении данного. Таким образом, при освобождении блока проверяется соседний младший блок (с младшими адресами), и, если он свободен, эти блоки соединяются, а затем проверяется блок, следующий за освобождаемым, и, если он свободен, блоки тоже объединяются.

Последней операцией является корректировка указателей в списке. Все склеенные блоки (если есть) удаляются из списка свободных. Так как применяется двунаправленный список, операция удаления тривиальна. После этого освобождаемый блок добавляется в список свободных.