В компьютерном хранилище **фрагментация**-это явление, при котором пространство хранения, основное хранилище или вторичное хранилище, используется неэффективно, снижая емкость или производительность, а часто и то, и другое. Точные последствия **фрагментации** зависят от конкретной используемой системы распределения памяти и конкретной формы **фрагментации**.

Существует 2 типа фрагментации:

Внутренняя : когда выделяется больше памяти, чем запрашивалось, избыток памяти не используется

Внешняя : свободная память в процессе выделения или освобождения разделяется на мелкие блоки и в результате не обслуживаются некоторые запросы на выделение памяти

Что такое кэш и что такое кэширование? Согласно определению кэш – это промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Соответственно кэширование – это процесс размещения и хранения какой либо информации в кэше с целью увеличения скорости доступа к ней. С

Действительно, активное использование функций malloc/free может привести к тому, что вся доступная память будет разбита на блоки маленького размера, и попытка выделения большого блока завершится неудачей, даже если сумма длин маленьких блоков намного больше требуемой

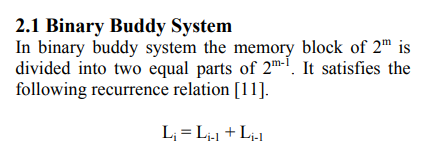
В стандартных библиотеках языков высокого уровня, таких как malloc/free/realloc в C, new/dispose в Pascal и т.д., как правило, используются алгоритмы, рассчитанные на худший случай: программа требует блоки случайного размера в случайном порядке и освобождает их также случайным образом.

Кроме того, большое количество блоков требует длительного поиска.

Это увеличивает появления фрагментации.  если мы нашли блок с размером больше заданного, мы должны отделить «хвост» и пометить его как новый свободный блок.

Поэтому у меня стоит задача модифицировать алгоритм управления памятью. В задаче стоит изучить алгоритм близнецы.

Основой этого алгоритма является сохранение выделенной памяти, содержащей объект определённого типа, и повторное использование этой памяти при следующем выделении для объекта того же типа.



Система buddy выделяет память из сегмента фиксированного размера, состоящего из

физически смежных страниц. Память выделяется из этого сегмента с использованием

распределителя степени 2, который удовлетворяет запросы в единицах, размер которых равен степени 2

(4 КБ, 8 КБ, 16 КБ и так далее). Запрос в единицах, не соответствующих размеру

, округляется до следующей по величине степени 2. Например, запрос на 11 КБ

удовлетворяется сегментом в 16 КБ.

Очевидным недостатком системы buddy является то, что округление до

следующей по величине степени, равной 2, с большой вероятностью приведет к фрагментации внутри выделенных

сегментов. Например, запрос размером 33 КБ может быть удовлетворен только с помощью 64-

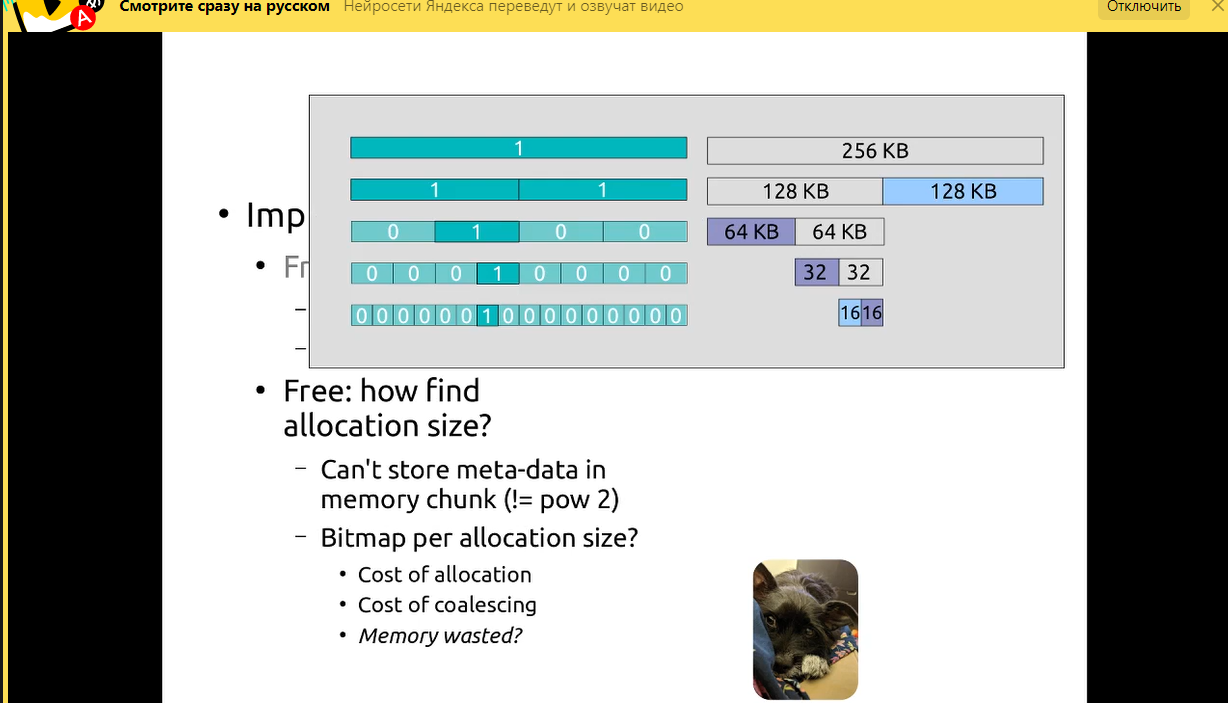
Сегмент КБ. На самом деле, мы не можем гарантировать, что менее 50 процентов

выделенного блока будет потрачено впустую из-за внутренней фрагментации. В следующем

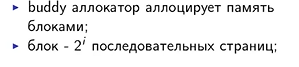
разделе мы рассмотрим схему распределения памяти, при которой пространство не теряется из-за

фрагментации.

coalescing.- объединение двух блоков памяти в один.

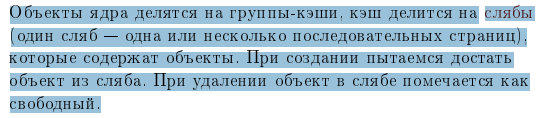


Buddy allocator не претендует на универсальность алгоритма, он предназначен для работы с большими участками памяти



**Пространство памяти, используемое распределителем Slab, выделяется через алгоритм партнера, но Slab реализует свой собственный алгоритм для этих областей памяти, а затем управляет небольшими блоками памяти.**

В случае же с распределением slab, при использовании программистом определённых системных вызовов, участки памяти, подходящие для размещения объектов данных определённого типа и размера, заранее предопределены. Распределитель slab хранит информацию о размещении этих участков, известных также как кэши. Таким образом, если поступает запрос на выделение памяти для объекта данных определённого размера, он может мгновенно удовлетворить запрос уже выделенным слотом.

.

1. **Кэш**: кэш представляет собой небольшой объём очень быстрой памяти. Здесь мы используем кэш как память для хранения таких [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), как [семафоры](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), дескрипторы [процессов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), объекты [файлов](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB) и т. д. Каждый кэш способен хранить только один тип объектов.
2. **Slab**: slab представляет собой непрерывный участок памяти, обычно составленный из нескольких физических смежных страниц.

**Адаптер Slab фактически основан на алгоритме системы партнера. Пространство памяти, используемое распределителем Slab, выделяется через алгоритм партнера, но Slab реализует свой собственный алгоритм для этих областей памяти, а затем управляет небольшими блоками памяти.**

Вторая стратегия выделения памяти ядра известна как выделение сляба.

Сляб состоит из одной или нескольких физически смежных страниц. Кэш состоит из

одного или нескольких блоков. Существует единый кэш для каждой уникальной структуры данных ядра

— например, отдельный кэш для структуры данных, представляющей

дескрипторы процессов, отдельный кэш для файловых объектов, отдельный кэш для семафоров

и так далее. Каждый кэш заполняется объектами, которые являются экземплярами

структуры данных ядра, которую представляет кэш. Например, кэш, представляющий

семафоры хранят экземпляры объектов семафора, кэш, представляющий

дескрипторы процесса, хранит экземпляры объектов дескриптора процесса и так далее.

Взаимосвязь между слябами, кэшами и объектами показана на рисунке 9.27. На

рисунке показаны два объекта ядра размером 3 КБ и три объекта размером 7 КБ, каждый

из которых хранится в отдельном кэше.

Алгоритм распределения слябов использует кэши для хранения объектов ядра. Когда создается

кэш, в кэш выделяется ряд объектов, которые изначально помечены как свободные

. Количество объектов в кэше зависит от размера

связанного сляба. Например, плита размером 12 КБ (состоящая из трех непрерывных

Страницы размером 4 КБ) может хранить шесть объектов размером 2 КБ. Изначально все объекты в кэше

помечаются как свободные. Когда требуется новый объект для структуры данных ядра,

распределитель может назначить любой свободный объект из кэша для удовлетворения запроса. То

объект, назначенный из кэша, помечается как используемый.

Запросы на память могут быть быстро удовлетворены. Таким образом, схема распределения slab

особенно эффективна для управления памятью, когда объекты

часто выделяются и освобождаются, как это часто бывает с запросами

от ядра. Процесс выделения и освобождения памяти может занять

много времени. Однако объекты создаются заранее и

, следовательно, могут быть быстро выделены из кэша. Более того, когда

ядро завершает работу с объектом и освобождает его, он помечается как свободный и

возвращается в свой кэш, что делает его немедленно доступным для последующих

запросов от ядра.

Список литературы:

1. <https://poisk-ru.ru/s61380t20.html>
2. <https://www.geeksforgeeks.org/operating-system-allocating-kernel-memory-buddy-system-slab-system/>
3. <https://en.wikipedia.org/wiki/Buddy_memory_allocation> (статья с Вики)
4. https://sci-hub.cat/downloads/2019-11-23/dc/feng2019.pdf?rand=633ab33a3dc40?download=true (https://ieeexplore.ieee.org/abstract/document/8759177/references#references)
5. [Кнут, Дональд](https://en.wikipedia.org/wiki/Donald_Knuth) (1997). *Фундаментальные алгоритмы*. [Искусство компьютерного программирования](https://en.wikipedia.org/wiki/The_Art_of_Computer_Programming). Том 1 (Второе изд.). Рединг, Массачусетс: Эддисон-Уэсли. стр. 435-455. [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [0-201-89683-4](https://en.wikipedia.org/wiki/Special:BookSources/0-201-89683-4).
6. <http://www.csl.ece.upatras.gr/os/Silberschatz.pdf>
7. <https://students.mimuw.edu.pl/ZSO/Wyklady/06_memory2/BuddySlabAllocator.pdf>
8. <https://russianblogs.com/article/1034274803/>
9. <https://cs.stackexchange.com/questions/152260/buddy-system-allocator-and-slab-allocator-in-linux-kernel>
10. <https://www.youtube.com/watch?v=pFi-JKgoX-I>
11. <https://ru.wikipedia.org/wiki/Slab>
12. <https://web.archive.org/web/20160831101859/http://k806.ru/osprogram/os_2011-11-12.pdf#page=26>
13. <https://www.programmersought.com/article/35715057658/>
14. <https://www.youtube.com/watch?v=DRAHRJEAEso>
15. <https://russianblogs.com/article/895968699/>
16. <https://russianblogs.com/article/261428559/>
17. <http://www.helenos.org/doc/design/html.chunked/mm.html>
18. <https://forum.osdev.org/viewtopic.php?f=15&t=30717>
19. <http://www.brokenthorn.com/Resources/OSDev26.html>
20. <https://ru.wikipedia.org/wiki/Динамическое_распределение_памяти>
21. <https://studwood.net/1601074/informatika/sistema_dvoynikov>
22. <https://revolution.allbest.ru/programming/00674849_0.html>
23. <https://tproger.ru/articles/memory-model/>
24. <https://www.memorymanagement.org/mmref/alloc.html>
25. <https://wikixw.ru/Управление_памятью>
26. <http://s2.bitdl.ir/Ebook/Computer%20Science/Blunden%20-%20Memory%20Management%20-%20Algor.%20and%20Impl.%20in%20C-C++%20(Wordware,%202002).pdf>
27. <https://sci-hub.ru/https://dl.acm.org/doi/abs/10.1145/359605.359626>
28. <https://sci-hub.ru/https://link.springer.com/chapter/10.1007/3-540-60368-9_19>
29. <https://dzen.ru/media/id/5eba7f439f339d116671be06/model-pamiati-v-iazykah-programmirovaniia-5f45f576cc336558bd7368cd>

Введение

Работа посвящена управлению памятью, являющимся ключевым вопросом в операционных системах. Память является очень важным ресурсом, поэтому требуется эффективное управление ею. Основной вопрос димнамического выделения памяти: “Какой именно блок памяти нужно выделить?” Наиболее распространены три стратегии:

**Стратегия первого подходящего (First fit).** Процесс помещается в первый подходящий по размеру раздел.

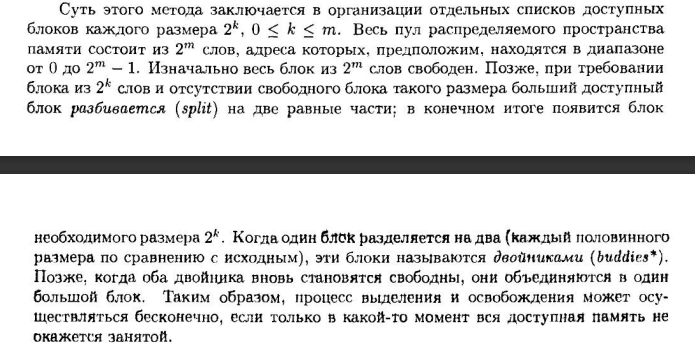
* **Стратегия наиболее подходящего (Best fit).** Процесс помещается в тот раздел, где после его загрузки останется меньше всего свободного места.
* **Стратегия наименее подходящего (Worst fit).** При помещении в самый большой раздел в нем остается достаточно места для возможного размещения еще одного процесса

**Наиболее подробно об этих алгоритмах можно почитать в учебнике Кнута(!**). Стоить упоминуть о ряде проблем, которые сопровождают данные алгоритмы.

1. **Внутренняя фрагментация**
2. **Время прохождения**

**Всё это позволяет решить одна** из альтернатив алгоритмов, с название Buddy system(алгоритм Двойников, алгоритм близнецы).

**Одной из альтернативой алгоритмов, стал алгоритм(“Buddy”, близнецы).** Одно из преимуществ этого метода состоит в простоте объединения блоков при их освобождении.



Более подробное описание алгоритма будет изложена в главе n.

Примером одной из основных частей ОС является менеджер распределения памяти. Главной операцией управления памятью является размещение программы в основной памяти для ее выполнения процессором. Практически во всех современных многозадачных системах эта задача предполагает использование сложной схемы, известной как виртуальная память. Виртуальная память, в свою очередь, основана на использовании одной или обеих базовых технологий -- сегментов и страниц.

Система двойников представляет собой разумный компромисс для преодоления недостатков схем фиксированного и динамического распределения, но в современных операционных системах ее превосходит виртуальная память, основанная на страничной организации и сегментации. Модифицированная версия системы двойников используется для распределения памяти ядром Linux.

Необходимо разработать программу для изучения этого алгоритма, которая позволит моделировать его работу при выделении и освобождении памяти. Данная программа может применяться как пособие-пример студенту, моделирующая работу менеджера памяти.

Результатом данной работы должна быть программа, которая сможет использоваться в процессе образования людьми, которым интересны основы системного программного обеспечения.

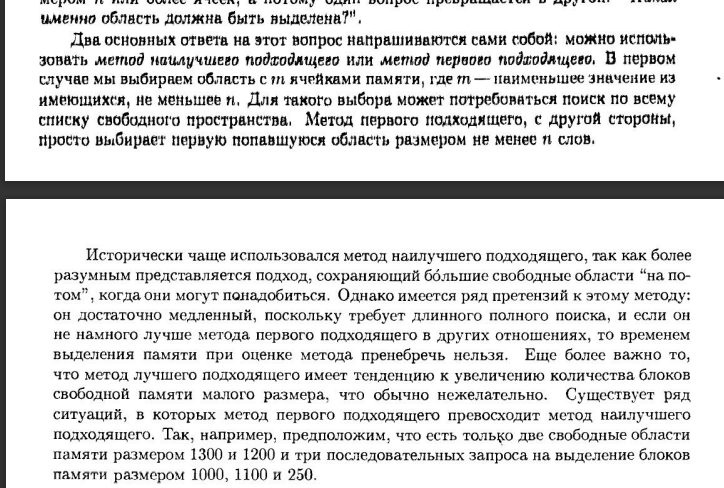
Одно из преимуществ этого метода состоит в простоте объединения блоков при их освобождении. Адрес блока-близнеца получается простым инвертированием соответствующего бита в адресе нашего блока. Нужно только проверить, свободен ли этот близнец. Если он свободен, то мы объединяем братьев в блок вдвое большего размера, и т. д. Даже в наихудшем случае время поиска не превышает О (log(Smax)-log(Smin)), где Smax и Smin обозначают, соответственно, максимальный и минимальный размеры используемых блоков. Это делает алгоритм близнецов трудно заменимым для ситуаций, в которых необходимо гарантированное время реакции – например, для задач реального времени. Часто этот алгоритм или его варианты используются для выделения памяти внутри ядра ОС.

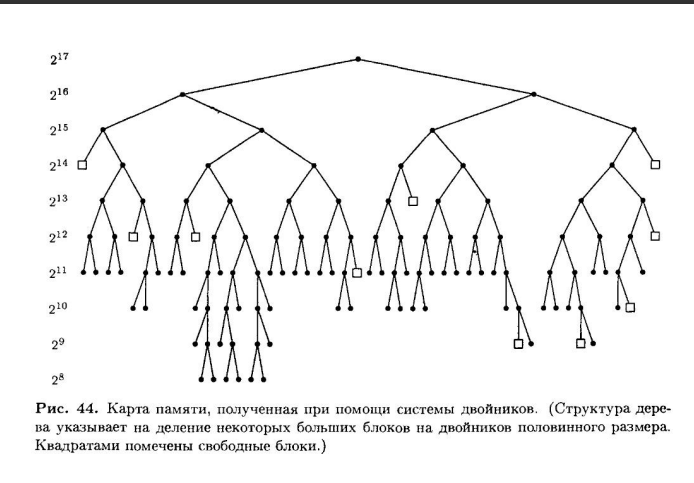
При **динамическом выделении памяти** запросы на выделение памяти формируются во время исполнения задачи. Динамическое выделение, таким образом, противопоставляется **статическому**, когда запросы формируются на этапе компиляции программы. В конечном итоге, и те, и другие запросы нередко обрабатываются одним и тем же алгоритмом выделения памяти в ядре ОС. Но во многих случаях статическое выделение можно реализовать намного более простыми способами, чем динамическое.

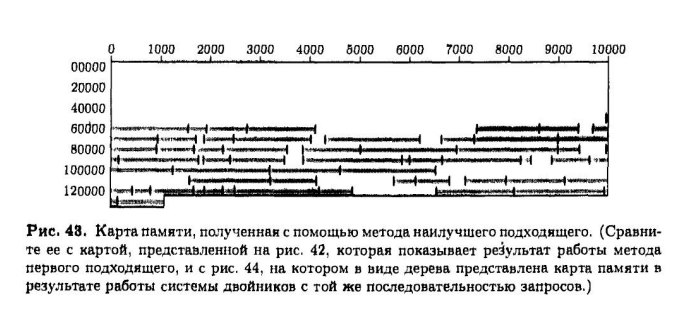
Главная сложность здесь в том, что при статическом выделении кажется неестественной – и поэтому редко требуется – возможность отказаться от ранее выделенной памяти. При динамическом же распределении часто требуется предоставить возможность отказываться от запрошенных блоков так, чтобы освобожденная память могла использоваться для удовлетворения последующих запросов. Таким образом, динамический распределитель вместо простой границы между занятой и свободной памятью (которой достаточно в простых случаях статического распределения) вынужден хранить список возможно несвязных областей свободной памяти, называемый пулом или кучей.

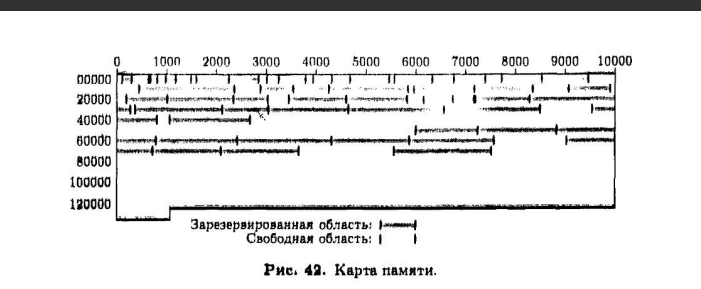
Обычно все свободные блоки памяти объединяются в двунаправленный связанный список. Список должен быть двунаправленным для того, чтобы из него в любой момент можно было извлечь любой блок. Впрочем, если все действия по извлечению блока производятся после поиска, то можно слегка усложнить процедуру поиска и всегда сохранять указатель на предыдущий блок. Это решает проблему извлечения и можно ограничиться однонаправленным списком. Беда только в том, что многие алгоритмы при объединении свободных блоков извлекают их из списка в соответствии с адресом, поэтому для таких алгоритмов двунаправленный список остро необходим.

Поиск в списке может вестись тремя способами: до нахождения **первого подходящего** (**first fit**) блока, до блока, размер которого ближе всего к заданному – **наиболее подходящего** (**best fit**), и, наконец, до нахождения самого большого блока, **наименее подходящего** (**worst fit**).









Система buddy используется во многих современных операционных

системах; в частности, в большинстве версий

UNIX / Linux. Различные

реализации имеют свои особые изюминки.

Например, большинство версий Linux пытаются избежать

потенциальных затрат O(log n) на выделение

и освобождение небольших блоков, сохраняя

освобожденные небольшие блоки в списках. (Slab allocator).

Система buddy поддерживает список свободных

блоков каждого размера (называемый свободным списком), так что

легко найти блок нужного размера, если таковой

имеется.

В схемах simple buddy

вся область кучи концептуально разделяется на две большие области, а

затем эти области разделяются на две меньшие области и так далее. Это иерархическое

разделение памяти используется для ограничения того, где размещаются объекты, каковы их

допустимые размеры и как они могут быть объединены в более крупные свободные области. Для каждого

допустимого размера поддерживается отдельный список свободных мест в массиве списков избранных.

Таким образом, системы на самом деле являются частным случаем сегрегированных подгонок, использующих классы размеров

с округлением и своеобразную ограниченную технику разделения и объединения.

Свободный блок может быть объединен только со своим приятелем, который является его уникальным соседом

на том же уровне в двоичном иерархическом разделении. Таким образом, результирующий свободный блок

всегда является одной из свободных областей на следующем более высоком уровне иерархии разделения памяти - на любом уровне первый блок может быть объединен только со

следующим блоком, который следует за ним в памяти; и наоборот, второй блок может

быть объединен только с первым, который предшествует это в памяти. Это ограничение на

слияние гарантирует, что результирующая объединенная свободная область всегда будет выровнена по

одна из границ иерархического расщепления.

Цель ограничения распределения бадди состоит в том, чтобы гарантировать, что при

освобождении блока его (уникальный) бадди всегда можно найти с помощью простого

вычисления адреса, и его бадди всегда будет либо целым, полностью свободным блоком,

либо недоступным блоком. Недоступный блок может быть выделен полностью или может

быть разделен, и некоторые из его подразделов выделены, но не другие. В любом

случае, вычисление адреса всегда будет в состоянии найти начало объекта

- оно никогда не найдет середину выделенного объекта. Приятель будет

либо целый (выделенный или свободный) блок определенного размера, либо начало

блока такого размера, который был разделен определенным образом. Если (и только

если) окажется, что это заголовок свободного блока, а блок - это весь

приятель, приятели могут быть объединены. Если приятель полностью или частично выделен,

приятели не могут быть объединены - даже если в пределах

(разделенного) приятеля есть смежная свободная область.

Объединение бадди происходит относительно быстро, но, возможно, самым большим преимуществом в

некоторых контекстах является то, что оно требует небольших затрат пространства на объект -

для каждого бадди требуется всего один бит, чтобы указать, является ли бадди смежной свободной областью.

Это может быть реализовано с помощью однобитового заголовка для каждого объекта или свободного блока.

Было разработано несколько существенных вариаций систем бадди:

Бинарные приятели. Бинарные друзья - это самый простой и наиболее известный вид системы приятелей

[Kno65]. В этой схеме все размеры приятелей равны степени двойки, и каждый размер

делится на две равные части. Это упрощает вычисления адресов, потому

что все партнеры выровнены по смещению границы в степени двойки от начала области

кучи, и каждый бит в смещении блока представляет один уровень в

иерархическом разделении памяти системы партнеров - если бит равен 0, он является первым из

пара приятелей, и если бит равен 1, то это второй. Эти операции могут быть

эффективно реализованы с помощью побитовых логических операций.

С другой стороны, системы, основанные на более близком расстоянии между классами размеров, могут быть столь же эффективными, если таблицы поиска используются для быстрого выполнения сопоставлений классов размеров.

Основная проблема с бинарными приятелями заключается в том, что внутренняя фрагментация обычно

относительно высока - ожидаемый случай составляет (очень грубо) около 28% [Knu73, PN77],74

потому что любой размер объекта должен быть округлен до ближайшей степени двойки (минус

слово для заголовка, если поле размера сохранено).

Выделение и освобождение структур - одна из наиболее распространенных операций внутри любого

ядра. Linux архивирует это с помощью своего механизма распределения слябов.

Основная идея распределителя слябов заключается в том, чтобы кэши часто используемых объектов хранились

в инициализированном состоянии, доступном для использования ядром. Без распределителя

на основе объектов ядро будет тратить большую часть своего времени на выделение, инициализацию и освобождение одного и того же объекта.

Распределитель слябов предназначен для кэширования освобожденного объекта, чтобы базовая структура сохранялась

между использованиями. (Бонвик, 1994)

Распределитель слябов состоит из переменного числа кэшей, которые связаны друг с другом в

двусвязном циклическом списке, называемом цепочкой кэша.

Распределитель плит имеет три основные цели:

 Выделение небольших блоков памяти для устранения внутренней фрагментации

в противном случае это было бы вызвано системой приятелей;

 Кэширование часто используемых объектов, чтобы система не тратила время

на выделение, инициализацию и уничтожение объектов. Тесты на Solaris показали

отличные улучшения скорости для распределений с использованием распределителя slab.

Основная работа программистов стоит на работе с данными, которые где-то нужно хранить