Основная работа программистов-это умение работать с памятью. Порядочный программист должен обладать знаниями о хранение данных в памяти, о том что память статическая, автоматическая и динамическая. Сегодня речь пойдёт о динамическом выделение памяти и управлению ею. Динамическое управление занимается тем, что выделяет память по запросу приложения. Если бы процесс был один, то думать было бы нечего. Просто можно было отдать все что есть, но в многозадачной операционной системе так поступать нельзя. Такой ценный ресурс как память может понадобиться любому процессу в абсолютно любой момент времени. Такой важный вопрос как распределение памяти между различными процессами находится в ведении операционной системы.

Наверняка вам уже приходилось работать с ручным управлением памятью, выделать и освобождать память при помощи функций malloc/free( в C) или new/delete(в C++). Определение того, когда выделяется память, как правило, тривиально и не вызывает проблем. Реальной проблемой является [уничтожение](https://translated.turbopages.org/proxy_u/en-ru.ru.3b292d50-63514abe-53b520c3-74722d776562/https/en.wikipedia.org/wiki/Object_destruction) алгоритма - определение того, когда объект больше не нужен (т.е. является мусором), обычно эта задача также стоит за программистом, а если уничтожения не происходит, то это приводит к утечкам памяти, но сегодня не об этом. Рядовой программист не всегда внедряется в глубокий анализ сущности функций управляющие памятью. Пропуская тот момент, что базовые аллокаторы(по другому распределители памяти) не так уж и эффективны и имеют ряд проблем. Существует ряд стратегий управлений памятью. К примеру, популярный алгоритм

-**first-fit**Распределитель памяти просматривает список областей до тех пор, пока не находит достаточ­но большой свободный участок Затем этот участок делится на две части: одна отдается процессу, а другая остается неиспользуемой. Так происходит всегда, кроме статистически нереального случая точного соответствия свободного участ­ка и процесса. Это быстрый алгоритм, потому что поиск уменьшен настолько, на­сколько возможно.

-**best-fit**(блок в heap, вмещающий запрошенный размер с наименьшим превышением). Когда блок памяти больше не нужен, он возвращаются в heap. Основная проблема этой стратегии распределения — фрагментация, и деградация системы с течением длительного времени непрерывной эксплуатации. Проблемой вторичного порядка малости является высокая затратность времени для управления свободным пространством heap. Алгоритм, который мы сегодня просмотрим более подробно - называется buddy system (алгоритм близницов/двойников/приятелей.

Суть этого метода заключается в организации отдельных списков доступных блоков каждого размера 2^k, 0<=k<=m. Весь пул распределяемого пространства памяти состоит из 2^m слов, адреса которых, предположим, находятся в диапазоне от 0 до 2^m - 1. Изначально весь блок из 2^m слов свободен. Позже, при требовании блока из 2^k слов и отсутствии свободного блока такого размера больший доступный блок разбивается на две равные части; в конечном итоге появится блок необходимого размера 2^k. Когда один блок разделяется на два(каждый половинного размера по сравнению с исходным), эти блоки называются двойниками(близнецами, buddies)Позже, когда оба двойника вновь становятся свободны, они объединяются в один большой блок. Таким образом, процесс выделения и освобождения может осуществляться бесконечно, если только в какой-то момент вся доступная память не окажется занятой. Но он может существенно увеличить непродуктивное расходование памяти, например если мы запросим блок размером в 518 бит, то удовлетворить запрос мы можем только блоком в 1024 бит, а этом алгоритме мы не можем разделить блок в 1024 бит на 518 и 506 бит, поэтому в блоке в 1024 бит будет находиться пустое неиспользованное пространство. Мы можем модифицировать алгоритм, использовав ещё одну стратегию, которая называется slab allocation. Идея такого распределителя состоит в том, что запросы на выделение памяти под объекты равного размера удовлетворяются из области одного кэша (слаба), а запросы на объекты другого размера (пусть отличающиеся от первого случая самым незначительным образом) - удовлетворяются из совершенно другого такого же кэша.

***Примечание:****Сам термин слаб переводится близко к «облицовочная плитка», и принцип очень похож: любую вынутую из плоскости плитку можно заменить другой такой же, потому, что их размеры в точности совпадают.*

Buddy allocator занимается пулом свободной памяти

