Operating Systems

Memory Allocation

Me

February 13, 2016

План

1. Классы памяти и кто за них отвечает.
2. Простые аллокаторы памяти (SLOB) и Border Tags.
3. Аллокация больших блоков памяти.
4. Кеширующие аллокаторы (SLAB).
5. TCMalloc и эффективная аллокация для многопточных программ.

Классы памяти

По привелегиям доступа:

1. привелигерованная (kernel space)
2. не привелигерованная (user space)

Классы памяти

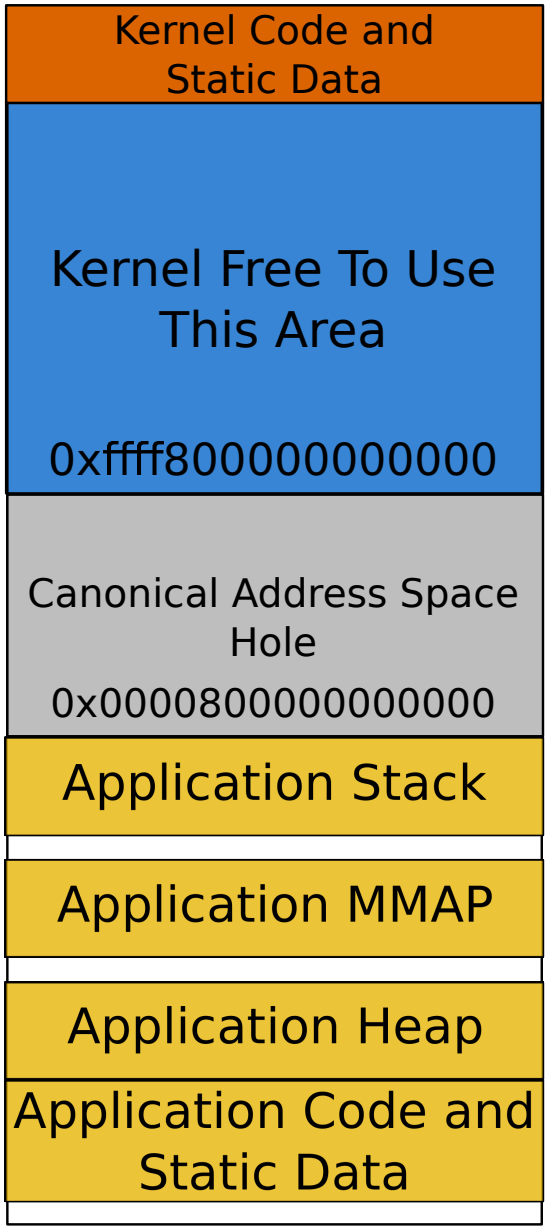
По привелегиям доступа:

1. привелигерованная (kernel space)
2. не привелигерованная (user space)

По способу аллокации:

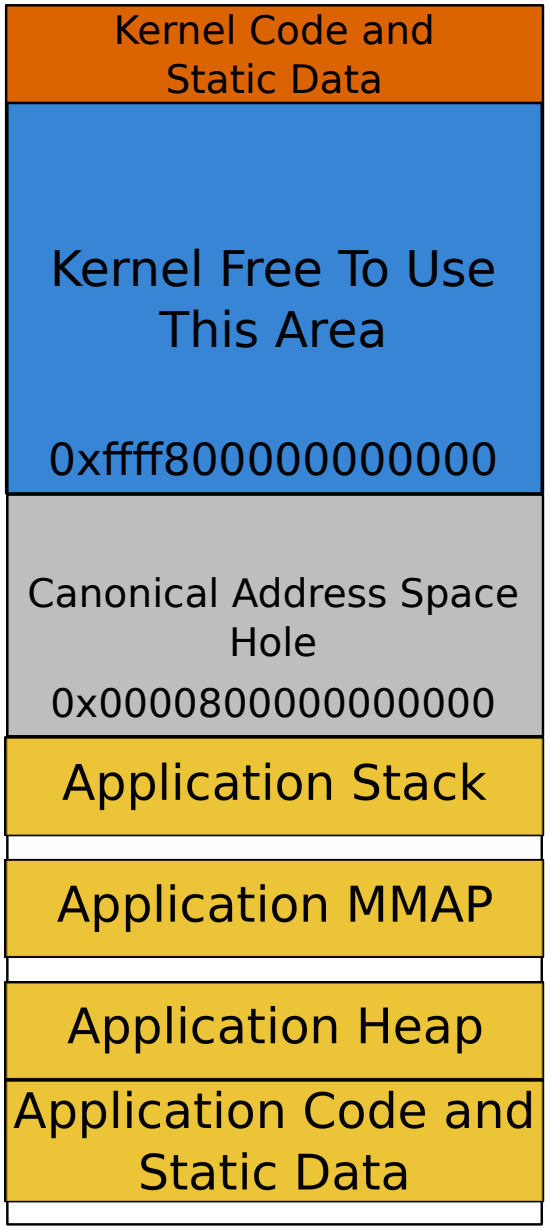
1. статическая память (код, глобальные перменные - размер известен заранее)
2. динамическая память (куча, free store и тд - размер не известен заранее)

Карта памяти



- Kernel Code and Static Data - привелигерованная статическая память (System V ABI amd64, 3.5.1 Architectural Constraints, Kernel code model)
- Canonical Address Space Hole - недопустимые адреса памяти (Intel® 64 and IA-32 Architectures Software Developer's Manual, 3.3.7.1 Canonical Addressing)

Карта памяти



- Application Stack - аллоцируется ОС при старте программы
- Application MMAP - разделяемые библиотеки, mmap/munmap
- Application Heap - malloc берет память отсюда, изменяется системным вызовом sbrk
- Application Code and Static Data - статическая не привелигерованная память

Простые алгоритмы аллокации памяти
Постановка задачи

Чего мы хотим:

Простые алгоритмы аллокации памяти
Постановка задачи

Чего мы хотим:

- реализовать malloc и free

Простые алгоритмы аллокации памяти
Постановка задачи

Чего мы хотим:

- реализовать malloc и free
- чем быстрее тем лучше

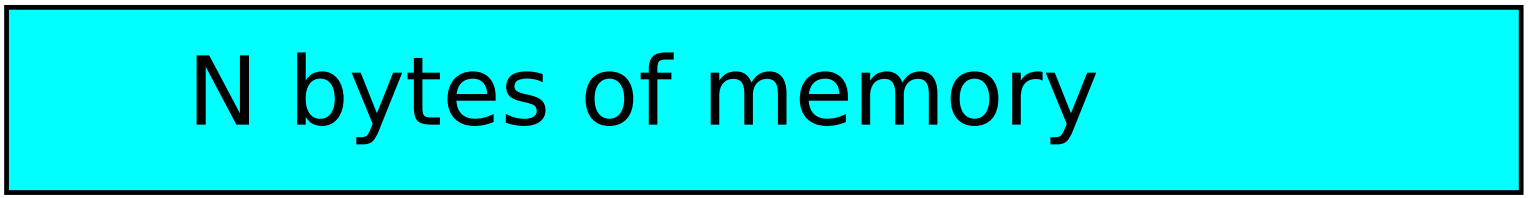
Простые алгоритмы аллокации памяти
Постановка задачи

Чего мы хотим:

- реализовать malloc и free
- чем быстрее тем лучше
- избежать фрагментации памяти, если получится

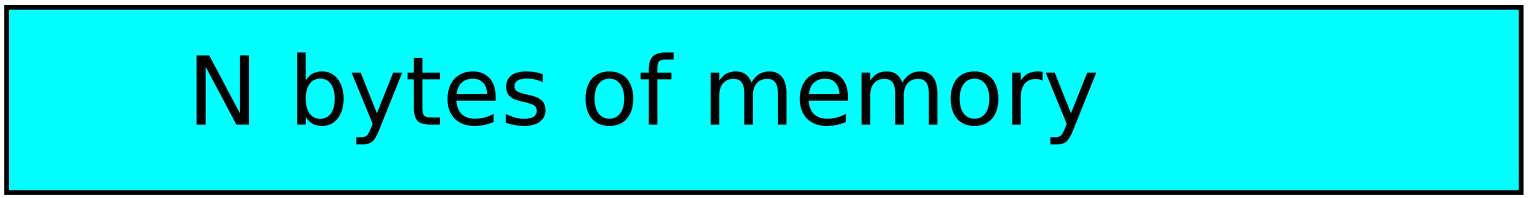
Простые алгоритмы аллокации памяти
Постановка задачи

Исходные данные - участок памяти:

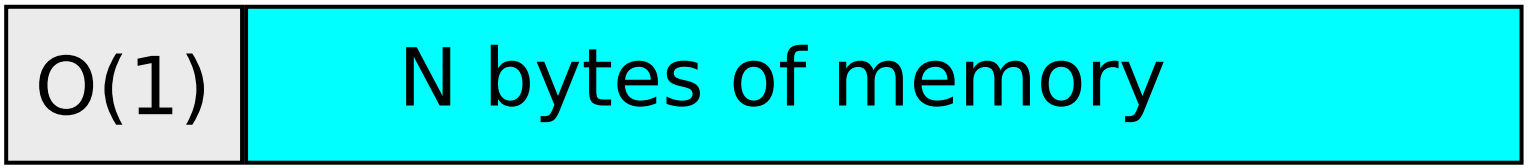


Простые алгоритмы аллокации памяти
Постановка задачи

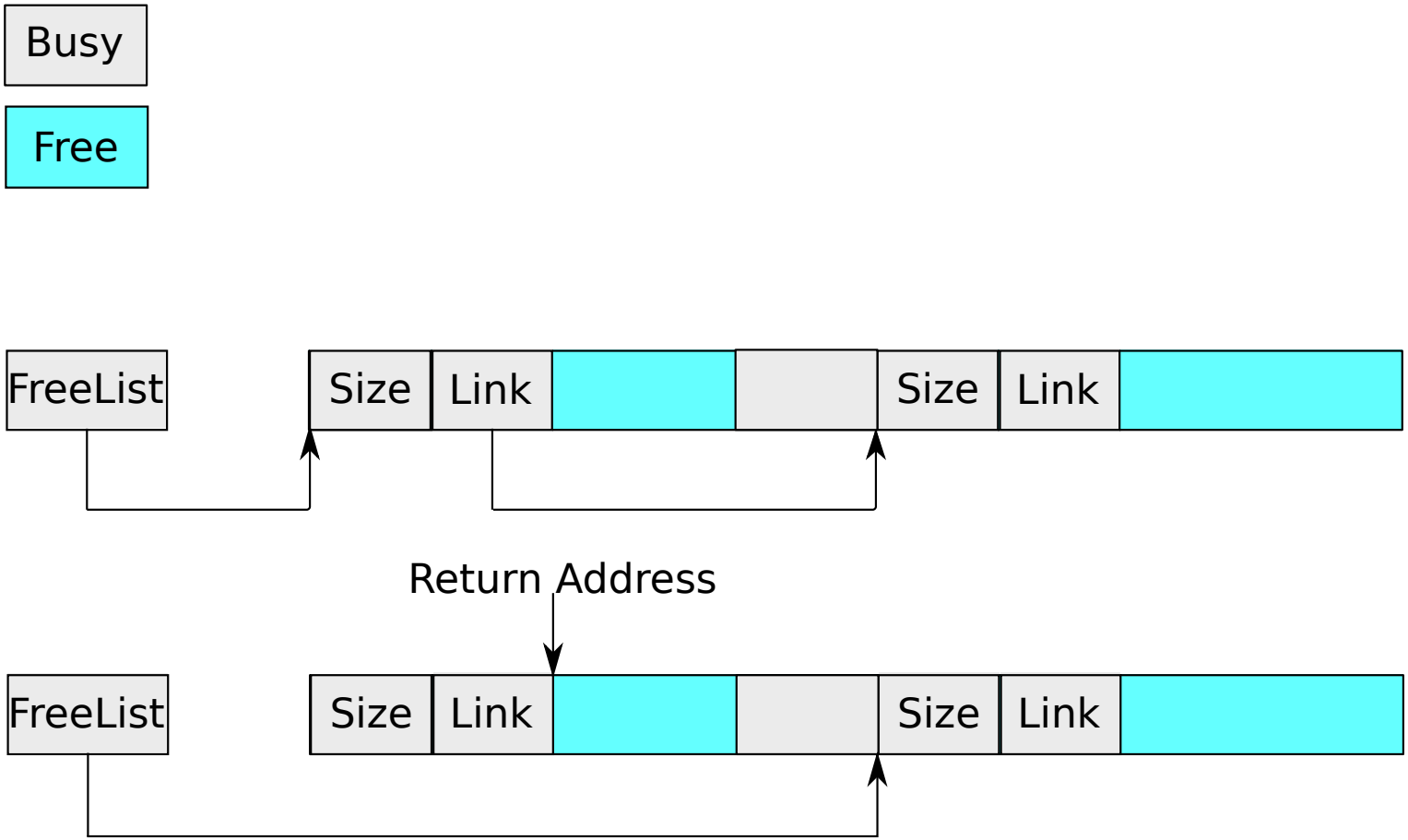
Исходные данные - участок памяти:



и, для удобства, какая-то память константного размера:



Построим в памяти связный список свободных участков:



Простые алгоритмы аллокации памяти

Аллокация

При аллокации проходим список свободных участков и выбираем подходящий.

Простые алгоритмы аллокации памяти

Аллокация

При аллокации проходим список свободных участков и выбираем подходящий.
Как выбрать подходящий?

Простые алгоритмы аллокации памяти
Аллокация

При аллокации проходим список свободных участков и выбираем подходящий.
Как выбрать подходящий?

- Best Fit - проходим весь список, выбираем наименьший из подходящих

Простые алгоритмы аллокации памяти
Аллокация

При аллокации проходим список свободных участков и выбираем подходящий.
Как выбрать подходящий?

- Best Fit - проходим весь список, выбираем наименьший из подходящих
- First Fit - выбираем из списка первый подходящий

Простые алгоритмы аллокации памяти
Аллокация

При аллокации проходим список свободных участков и выбираем подходящий.
Как выбрать подходящий?

- Best Fit - проходим весь список, выбираем наименьший из подходящих
- First Fit - выбираем из списка первый подходящий

Какая стратегия лучше?

Простые алгоритмы аллокации памяти
Аллокация

При аллокации проходим список свободных участков и выбираем подходящий.
Как выбрать подходящий?

- Best Fit - проходим весь список, выбираем наименьший из подходящих
- First Fit - выбираем из списка первый подходящий

Какая стратегия лучше?

1. науке это не известно - разные приложения используют память по разному

Простые алгоритмы аллокации памяти
Аллокация

При аллокации проходим список свободных участков и выбираем подходящий.
Как выбрать подходящий?

- Best Fit - проходим весь список, выбираем наименьший из подходящих
- First Fit - выбираем из списка первый подходящий

Какая стратегия лучше?

1. науке это не известно - разные приложения используют память по разному
2. зачастую First Fit лучше - не нужно проходить весь список при прочих равных (неизвестных)

Простые алгоритмы аллокации памяти
Аллокация

При аллокации проходим список свободных участков и выбираем подходящий.
Как выбрать подходящий?

- Best Fit - проходим весь список, выбираем наименьший из подходящих
- First Fit - выбираем из списка первый подходящий

Какая стратегия лучше?

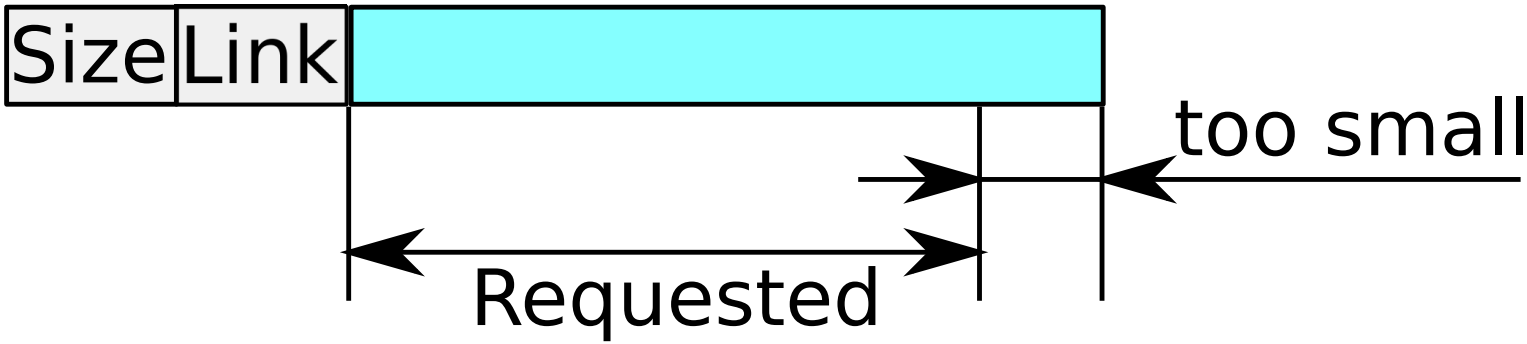
1. науке это не известно - разные приложения используют память по разному
2. зачастую First Fit лучше - не нужно проходить весь список при прочих равных (неизвестных)
3. простые алгоритмы редко используются на практике - есть лучшие подходы

Простые алгоритмы аллокации памяти
Аллокация

Почему бы пользователю не запоминать размер аллоцируемого блока? В простых случаях это будет работать хорошо:

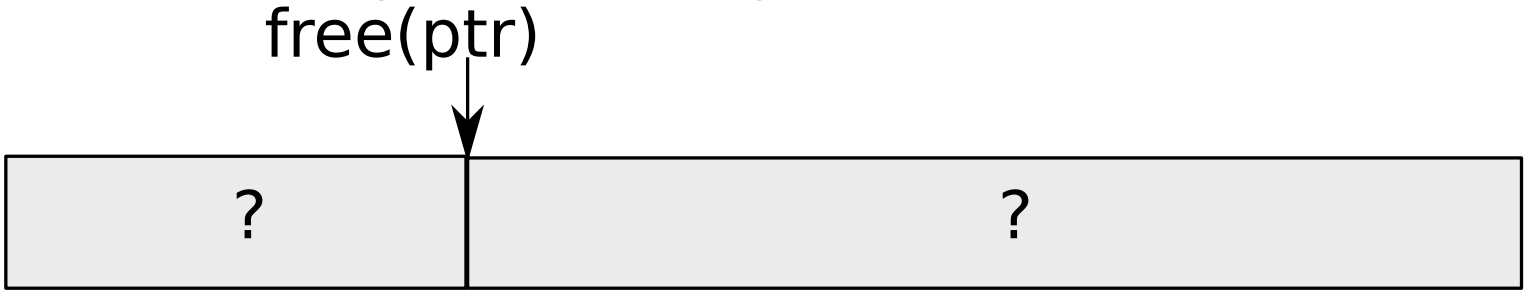
- часто размер известен в момент компиляции (sizeof(...))
- часто размер объекта вам нужен сам по себе - можно избежать дублирования
- пользователь сам определяет, где и как хранить размер

Однако в нашем случае это не будет работать - аллоцированный блок может быть больше запрошенного:



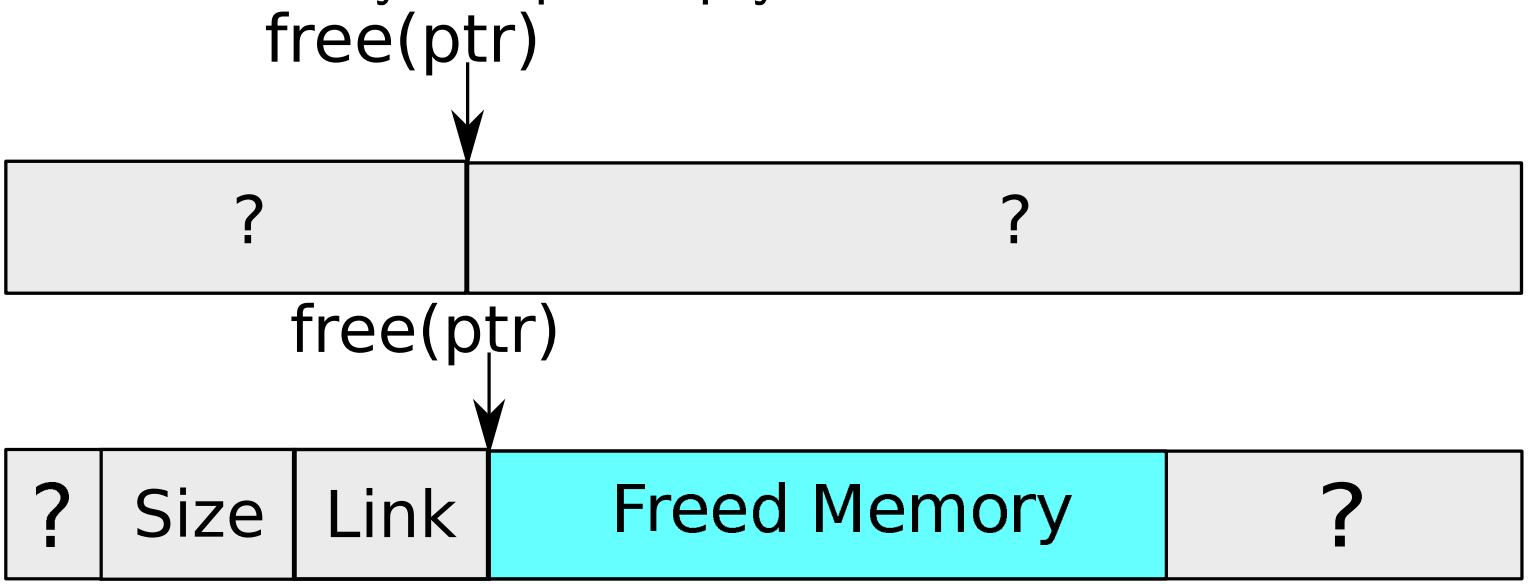
Простые алгоритмы аллокации памяти
Освобождение

Функция освобождения принимает указатель как аргумент, нам так же нужен размер участка памяти:



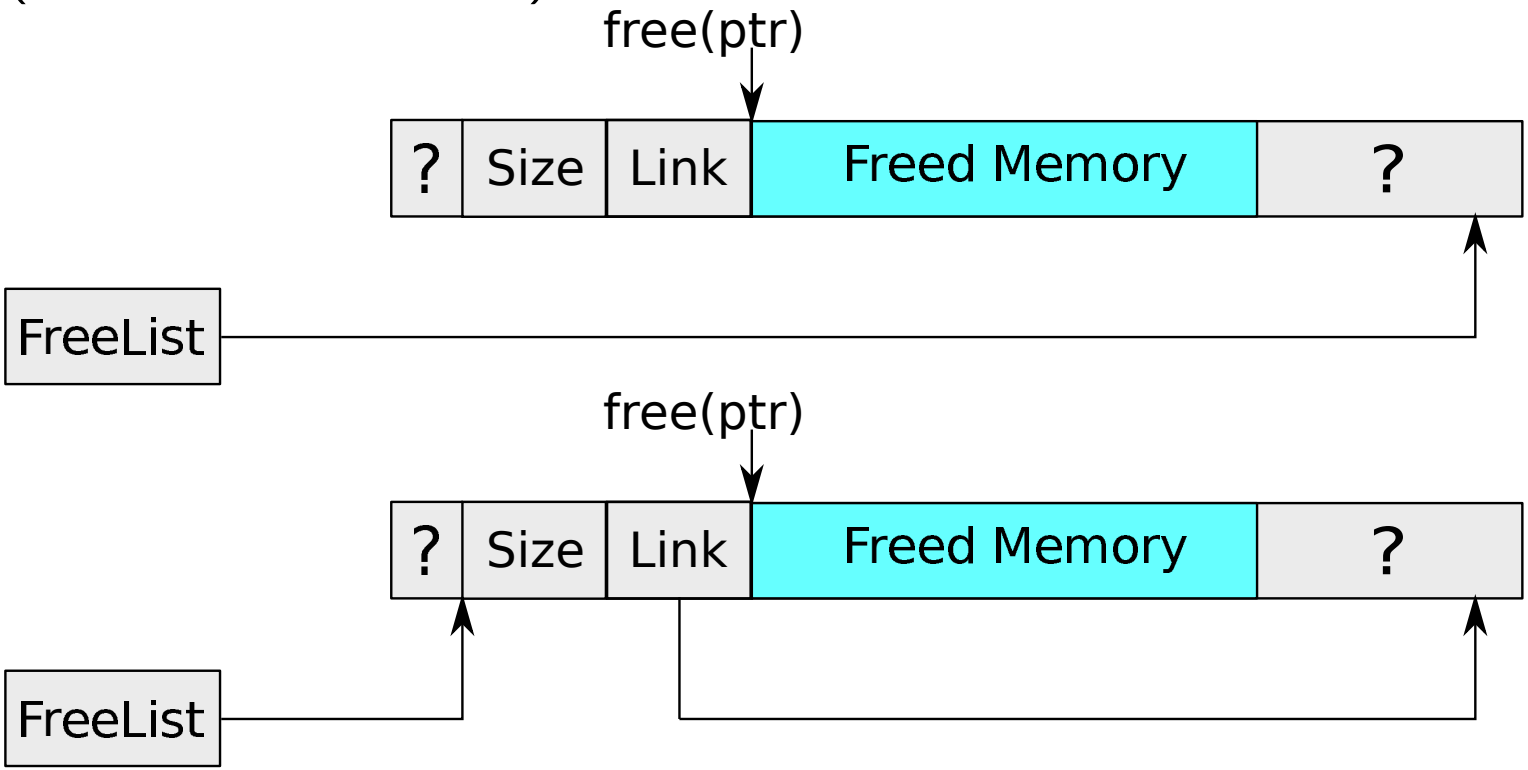
Простые алгоритмы аллокации памяти
Освобождение

Функция освобождения принимает указатель как аргумент, нам так же нужен размер участка памяти:



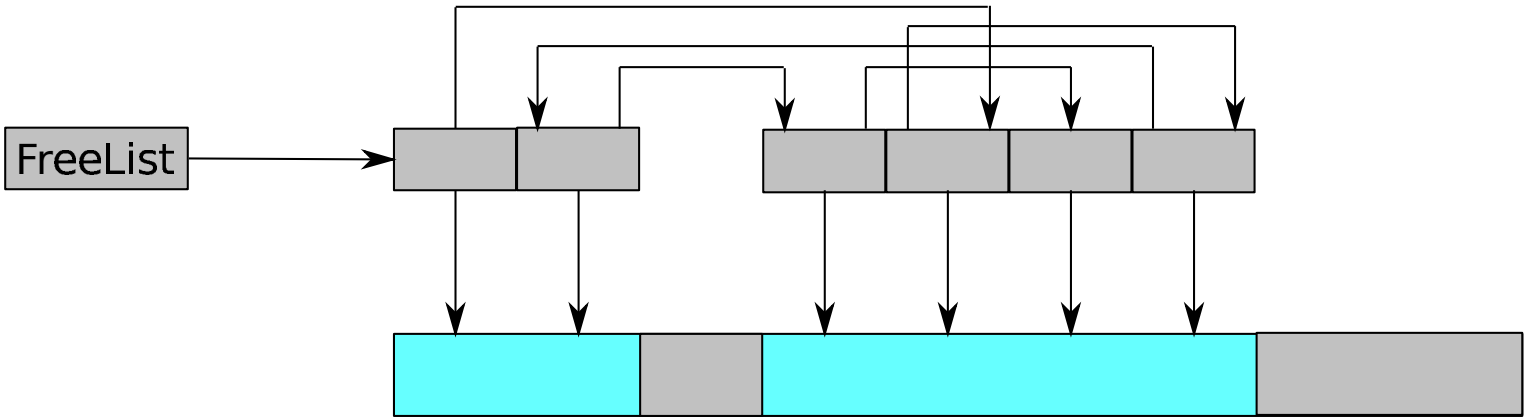
Простые алгоритмы аллокации памяти
Освобождение

Самый простой вариант, просто добавить элемент в список (начало или конец):



Простые алгоритмы аллокации памяти
Освобождение

Рано или поздно это приведет к проблеме:



Простые алгоритмы аллокации памяти
Освобождение

При освобождении необходимо объединять соседние свободные блоки:

Простые алгоритмы аллокации памяти
Освобождение

При освобождении необходимо объединять соседние свободные блоки:

- чтобы не фрагментировать память, иначе мы не сможем аллоцировать большие участки памяти

Простые алгоритмы аллокации памяти
Освобождение

При освобождении необходимо объединять соседние свободные блоки:

- чтобы не фрагментировать память, иначе мы не сможем аллоцировать большие участки памяти
- чтобы список свободных блоков не разрастался - плохо влияет на время аллокации памяти

Простые алгоритмы аллокации памяти
Освобождение

Как объединять соседние свободные блоки:

Простые алгоритмы аллокации памяти
Освобождение

Как объединять соседние свободные блоки:

- мы можем поддерживать список отсортированным по адресу (классический версия malloc в UNIX, aka SLOB, описана в The C Programming Language)

Простые алгоритмы аллокации памяти
Освобождение

Как объединять соседние свободные блоки:

- мы можем поддерживать список отсортированным по адресу (классический версия malloc в UNIX, aka SLOB, описана в The C Programming Language)
- вместо списка можно использовать дерево - жертвуем памятью в обмен на производительность

Простые алгоритмы аллокации памяти
Освобождение

Как объединять соседние свободные блоки:

- мы можем поддерживать список отсортированным по адресу (классический версия malloc в UNIX, aka SLOB, описана в The C Programming Language)
- вместо списка можно использовать дерево - жертвуем памятью в обмен на производительность
- можно использовать Border Tags (авторство приписывают Кнуту, но идея очень очевидная)

Простые алгоритмы аллокации памяти

Освобождение

Мы уже храним служебную информацию в начале блока, давайте добавим еще и в конец:

Простые алгоритмы аллокации памяти
Освобождение

Мы уже храним служебную информацию в начале блока, давайте добавим еще и в конец:

- TAG - индикатор свободности/занятости блока (это и есть Border Tag)

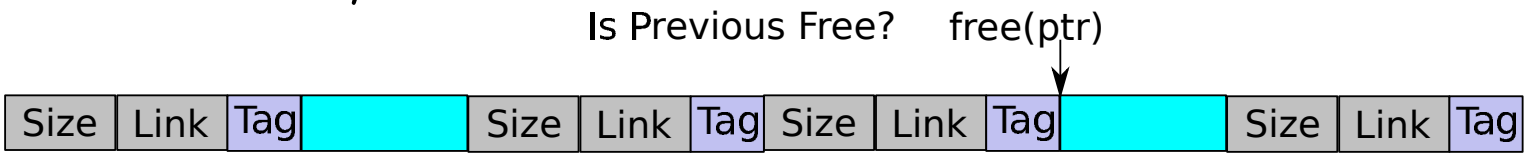
Простые алгоритмы аллокации памяти
Освобождение

Мы уже храним служебную информацию в начале блока, давайте добавим еще и в конец:

- TAG - индикатор свободности/занятости блока (это и есть Border Tag)
- Link-и в начале и в конце можно использовать как ссылки на следующий и ссылки на предыдущий блоки (просто для экономии памяти)

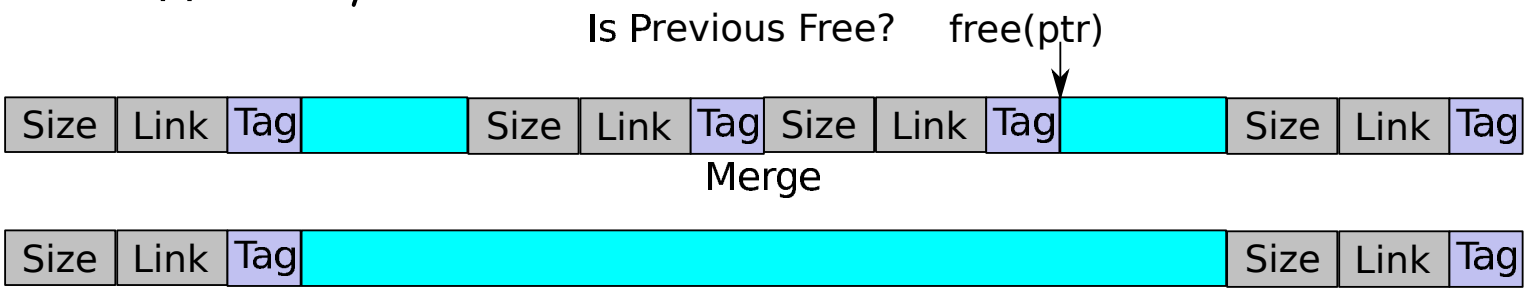
Простые алгоритмы аллокации памяти
Освобождение

Мы знаем где находится Border Tag предыдущего (следующего) блока и можем легко проверять свободность/занятость:



Простые алгоритмы аллокации памяти
Освобождение

Мы знаем где находится Border Tag предыдущего (следующего) блока и можем легко проверять свободность/занятость:



При объединении сразу трех блоков нужно удалить один из двусвязного спика - O(1).

Продвинутые алгоритмы аллокации памяти

Аллокация в несколько этапов

Современные аллокаторы памяти выделяют две стадии:

Продвинутые алгоритмы аллокации памяти

Аллокация в несколько этапов

Современные аллокаторы памяти выделяют две стадии:

- аллокация больших блоков (Buddy Allocator и Ко.):
  - аллокации просиходят нечасто, большие объекты живут долго
  - чем больше блок тем меньше накладные расходы на служебные структуры алокатора - можем хранить больше информации

Продвинутые алгоритмы аллокации памяти
Аллокация в несколько этапов

Современные аллокаторы памяти выделяют две стадии:

- аллокация больших блоков (Buddy Allocator и Ко.):
  - аллокации просиходят нечасто, большие объекты живут долго
  - чем больше блок тем меньше накладные расходы на служебные структуры алокатора - можем хранить больше информации
- аллокация маленьких блоков фиксированного размера (SLAB и Ко.):
  - блоки фиксированного размера проще аллоцировать
  - блоки фиксированного размера требуют меньше служебной информации
  - блоки имеют одинаковый размер не случайно - часто это объекты одного типа и это можно использовать