

Operating Systems

Memory Allocation

Me

February 14, 2016

- 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 многопоточных программ.

По привелегиям доступа:

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

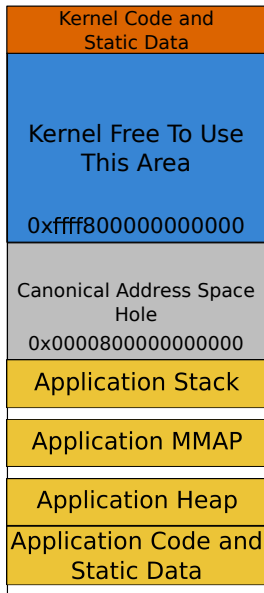
По привелегиям доступа:

- ① привелигированная (kernel space)
- ② не привелигированная (user space)

По способу аллокации:

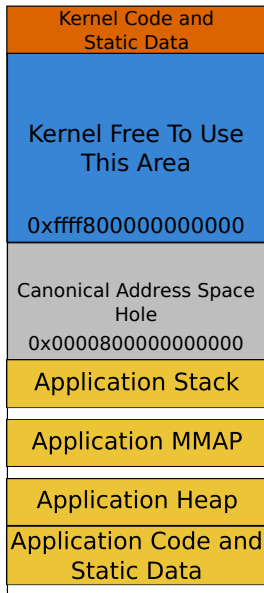
- ① статическая память (код, глобальные переменные - размер известен заранее)
- ② динамическая память (куча, free store и тд - размер не известен заранее)

Карта памяти



- Kernel Code and Static Data - привелигированная 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 привелигированная 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:

N bytes of memory

Простые алгоритмы аллокации памяти

Постановка задачи

Исходные данные - участок памяти:

N bytes of memory

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

$O(1)$

N bytes of memory

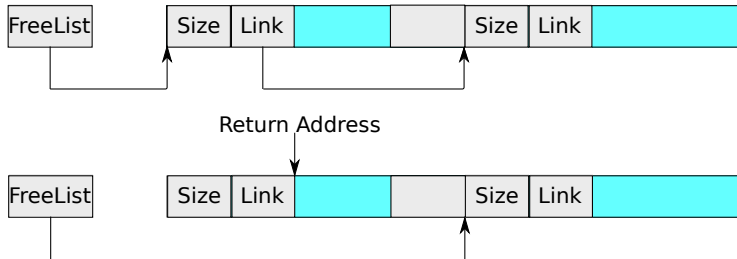
Простые алгоритмы аллокации памяти

Аллокация

Построим в памяти связный список свободных участков:

Busy

Free



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

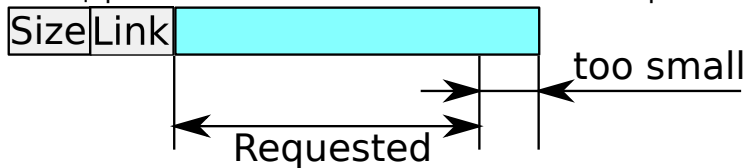
Почему бы пользователю не запоминать размер аллоцируемого блока? В простых случаях это будет работать хорошо:

- 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

Простые алгоритмы аллокации памяти

Аллокация

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:

`free(ptr)`



Простые алгоритмы аллокации памяти

Освобождение

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

`free(ptr)`



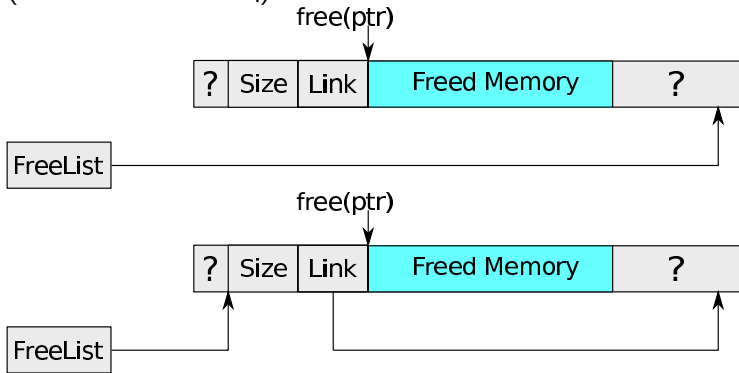
`free(ptr)`



Простые алгоритмы аллокации памяти

Освобождение

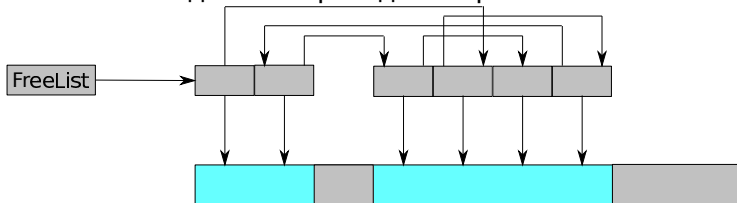
Самый простой вариант, просто добавить элемент в список (начало или конец):



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:

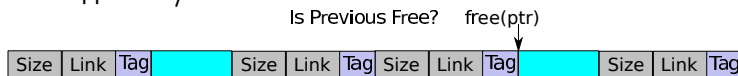


- TAG - индикатор свободности/занятости блока (это и есть Border Tag)
- Link-и в начале и в конце можно использовать как ссылки на следующий и ссылки на предыдущий блоки (просто для экономии памяти)

Простые алгоритмы аллокации памяти

Освобождение

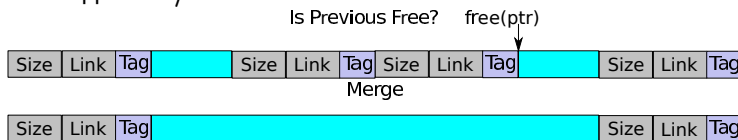
Мы знаем где находится Border Tag предыдущего (следующего) блока и можем легко проверять свободу/занятость:



Простые алгоритмы аллокации памяти

Освобождение

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



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

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 Co.):
 - 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 Co.):
 - 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

Buddy Allocator

Вводные положения

- вся аллоцируемая память разбита на большие блоки фиксированного размера (будем называть их PAGE)
- каждому PAGE поставлен в соответствие дескриптор (мы легко можем получить дескриптор по номеру PAGE и наоборот, считайте, что у нас есть массив таких дескрипторов), хранящий служебную информацию (свободен/занят, порядок свободного блока)
- память аллоцируется и освобождается блоками по $2^i \times PAGE$, i будем называть порядком блока
- порядок блока хранит пользователь и передает его в как функцию аллокации, так и в функцию освобождения

Ключевой концепцией для Buddy Allocator-а является понятие Buddy:

- Buddy Allocator хранит информацию о блоках в отдельных списках по порядкам этих блоков (т. е. для каждого возможного порядка блока есть свой список);
- смежные (в памяти, а не в списке) блоки одного порядка называются Buddies (plural for buddy);
- два смежных блока (Buddies) в объединении дают один блок большего порядка, и наоборот из одного блока можно получить два Buddies меньшего порядка.

Buddy Allocator

Buddies

Order 2

Buddies

Order 1

Order 1 Block Order 1 Block

Order 0

PAGE PAGE PAGE PAGE

Buddies

Buddies

Not Buddies