Куча. Алгоритмы работы malloc/free

Происхождение термина «куча»

Согласно Дональду Кнуту, «Several authors began about 1975 to call the pool of available memory a "heap."».

В стеке элементы расположены один над другим.

В куче нет определенного порядка в расположении элементов.

Особенности использования динамической памяти

Для хранения данных используется «куча».

Создать переменную в «куче» нельзя, но можно выделить память под нее.

“+”

Все «минусы» локальных переменных.

“-”

Ручное управление временем жизни.

Свойства области, выделенной malloc

- malloc выделяет по крайней мере указанное количество байт (меньше нельзя, больше можно).
- Указатель, возвращенный malloc, указывает на выделенную область (т.е. область, в которую программа может писать и из которой может читать данные).
- Ни один другой вызов malloc не может выделить эту область или ее часть, если только она не была освобождена с помощью free.

Реализация malloc/free

- Для моделирования области памяти, используемой под кучу, воспользуемся одномерным массивом.

Реализация malloc/free

- Пусть программист уже выделил 100 байт и хочет выделить еще 32 байта.

- Нельзя использовать 100 байт, которые уже были выделены, и еще не были освобождены.
- Начиная с какого места можно выделять память?
- Как найти нужный блок после выделения (например, чтобы освободить)?

Реализация malloc/free

Необходимо вести учет выделенных и свободных областей, но где хранить эти данные?

- Мы не можем воспользоваться malloc, потому что сами реализуем эту функцию :(
- Но мы можем выделить область чуть больше, чем нужно, и в ее начале расположить необходимые данные.

Реализация malloc/free

Какие сведения об области нам нужны?

- Размер.
- Состояния (выделена/свободна).
- Где находится следующая область?

```
struct block_t
{
    size_t size;
    int free;
    struct block_t *next;
};
```

Реализация malloc/free

```
#define MY_HEAP_SIZE 1000000

// пространство под "кучу"
static char my_heap[MY_HEAP_SIZE];

// список свободных/занятых областей
static struct block_t *free_list = (struct block_t*) my_heap;

// начальная инициализация списка свободных/занятых областей
static void initialize(void)
{
    free_list->size = sizeof(my_heap) - sizeof(struct block_t);
    free_list->free = 1;
    free_list->next = NULL;
}
```

Реализация malloc/free

Выделение области памяти (malloc)

- Просмотреть список занятых/свободных областей памяти в поисках свободной области подходящего размера.
- Если область имеет точно такой размер, как запрашивается, пометить найденную область как занятую и вернуть указатель на начало области памяти.
- Если область имеет больший размер, разделить ее на части, одна из которых будет занята (выделена), а другая останется в свободной.
- Если область не найдена, вернуть нулевой указатель.

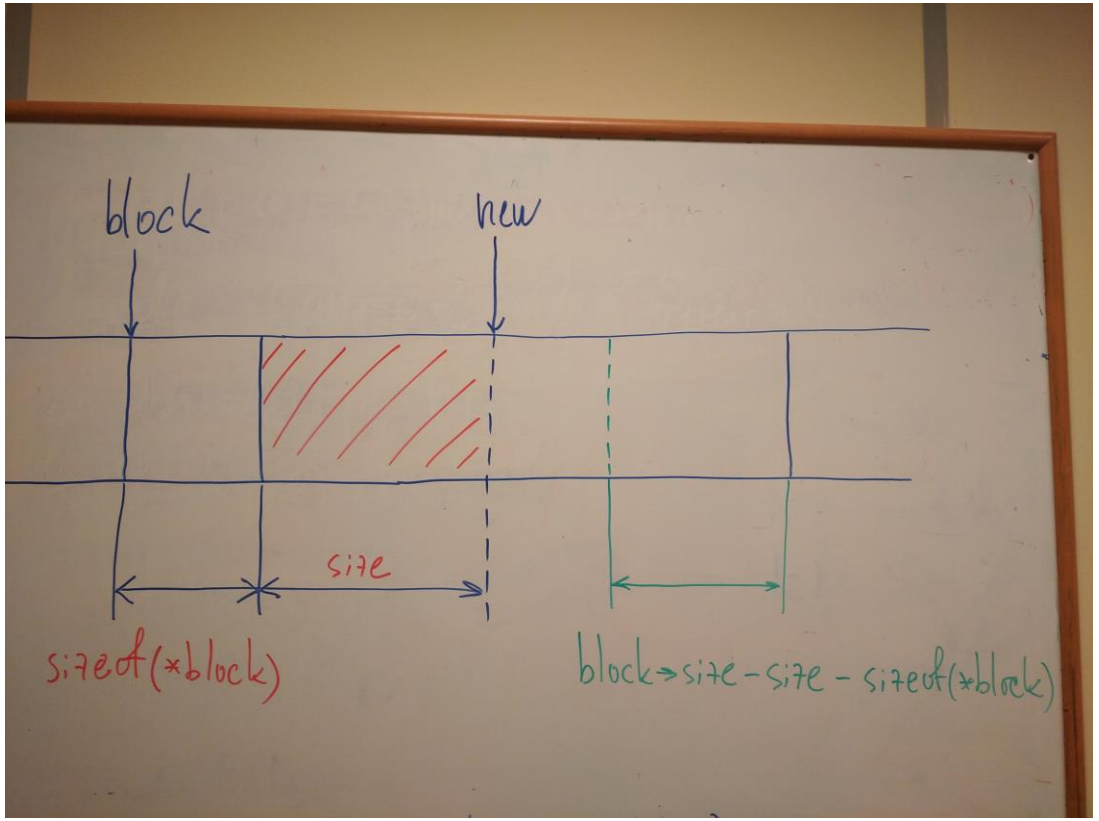
Реализация malloc/free

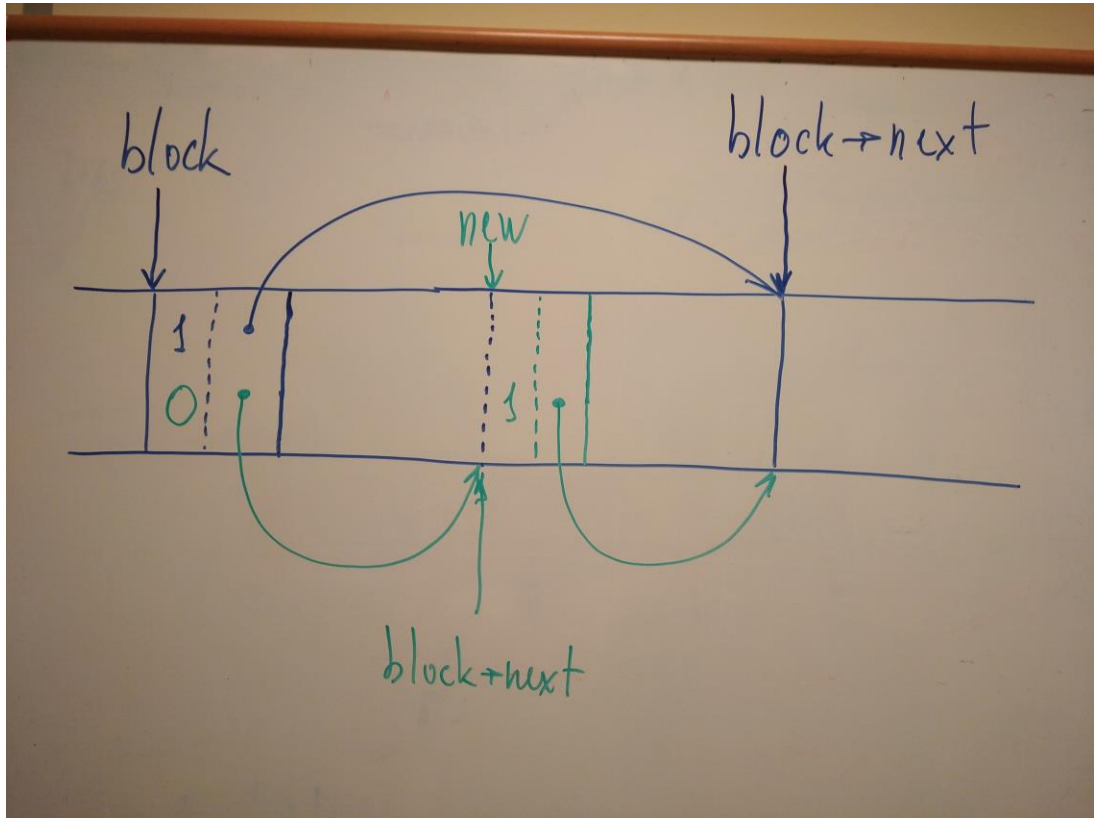
```
void* my_malloc(size_t size)
{
    struct block_t *cur;
    void *result;

    if (!free_list->size)
        initialize();

    cur = free_list;
    while (cur && (cur->free == 0 ||
                   cur->size < size))
        cur = cur->next;

    if (!cur)
    {
        result = NULL;
        printf("Out of memory\n");
    }
    else if (cur->size == size)
    {
        cur->free = 0;
        result = (void*) (++cur);
    }
    else
    {
        split_block(cur, size);
        result = (void*) (++cur);
    }

    return result;
}
```

Реализация malloc/free

```
static void split_block(struct block_t *block, size_t size)
{
    size_t rest = block->size - size;

    if (rest > sizeof(struct block_t))
    {
        struct block_t *new = (void*)((char*)block + size + sizeof(struct block_t));

        new->size = block->size - size - sizeof(struct block_t);
        new->free = 1;
        new->next = block->next;

        block->size = size;
        block->free = 0;
        block->next = new;
    }
    else
        block->free = 0;
}
```

Реализация malloc/free





Реализация malloc/free

Освобождение области памяти (free)

- Просмотреть список занятых/свободных областей памяти в поисках указанной области.
- Пометить найденную область как свободную.
- Если освобожденная область вплотную граничит со свободной областью с какой-либо из двух сторон, то объединить их в единую область большего размера.

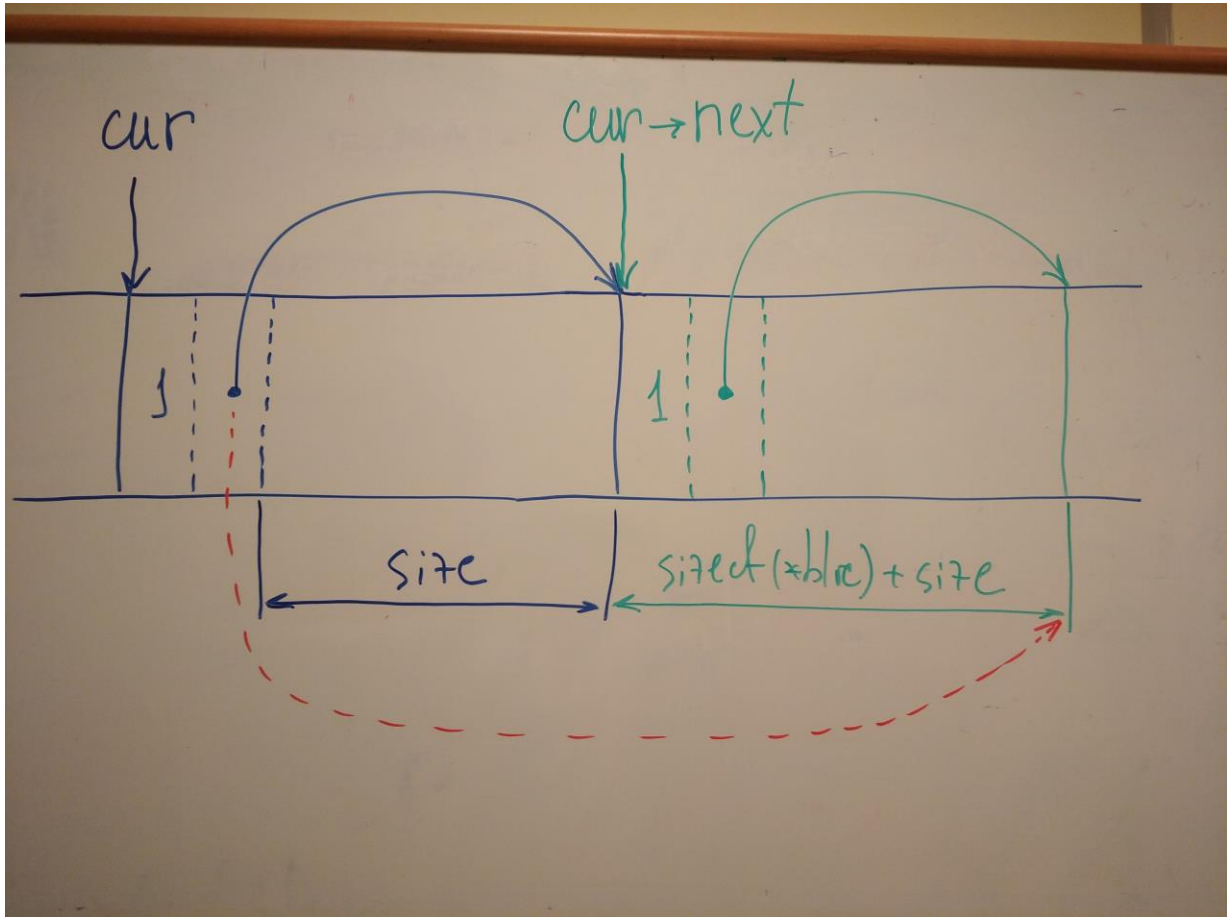
Реализация malloc/free

```
void my_free(void *ptr)
{
    if (my_heap <= (char*) ptr && (char*) ptr <= my_heap + sizeof(my_heap))
    {
        struct block_t *cur = ptr;

        --cur;
        cur->free = 1;

        merge_blocks();
    }
    else
        printf("Wrong pointer\n");
}
```

Реализация malloc/free

```
static void merge_blocks(void)
{
    struct block_t *cur = free_list;

    while (cur && cur->next != NULL)
    {
        if (cur->free && cur->next->free)
        {
            cur->size += cur->next->size + sizeof(struct block_t);
            cur->next = cur->next->next;
        }
        else
            cur = cur->next;
    }
}
```

Реализация malloc/free



Реализация malloc/free: недоделки и т.п.

- Выравнивание.
- Фрагментация.
- Возможность увеличения области, отведенной под кучу.

Реализация malloc/free: недоделки и т.п.

Выравнивание данных

По Кернигану, Ритчи

Для хранения произвольных объектов блок должен быть правильно выровнен. В каждой системе есть самый «требовательный» тип данных – если элемент этого типа можно поместить по некоторому адресу, то любые другие элементы тоже можно поместить туда.

Реализация malloc/free: недоделки и т.п.

```
// По Кернигану, Ритчи

typedef long align_t;

union block_t
{
    struct
    {
        size_t size;
        int free;
        union block_t *next;
    } block;

    align_t x;
};
```

Запрашиваемый размер области обычно округляется до размера кратного размеру заголовка.

```
n_blocks = (size + sizeof(union block_t) - 1) /
                                sizeof(union block_t) + 1;

alloc_size = n_blocks*sizeof(sizeof(union block_t));
```

Реализация malloc/free: недоделки и т.п.

Фрагментация

Размер «кучи» 1000 байт. 600 байт занято. Пользователю нужно выделить область в 400 байт :(

Использованные материалы

- Б. Керниган, Д. Ритчи «Язык программирования С»
- M. Burelle «A Malloc Tutorial»
- T. Madurapperuma «How to write your own Malloc and Free using C?»
- Лабораторная работа по курсу CS170 - Operating Systems