Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики.

Кафедра вычислительной техники

Языки системного программирования

Лабораторная работа №6

Кастомный аллокатор памяти

Выполнил:

Студент группы P3210

Глушков Дмитрий Сергеевич

Санкт-Петербург
2018 г.

Исходный текст:

```
#include "mem.h"
#include "alloc_list.h"

//инициализация кучи
void* heap_init(size_t size)
{
	void* pointer = mmap(HEAP_START, to_page_size(size), PROT_READ | PROT_WRITE,
MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
	mem_t *head = (mem_t*)pointer;
	head -> next = NULL;
	head -> capacity = to_page_size(size) - sizeof(mem_t);
	head -> is_free = 1; // debug: = true
	return (void*)( (char*)pointer + sizeof(mem_t) );
}

//аллокация блока
mem_t* allocate (mem_t* last, size_t size)
{
	char *allocated = mmap((char*)(last) + last -> capacity + sizeof(mem_t),
to_page_size(size), PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
	if (!allocated)
		allocated = mmap(NULL, to_page_size(size), PROT_READ | PROT_WRITE, MAP_PRIVATE
| MAP_ANONYMOUS, -1, 0);
	if(allocated == (char*)(last) + last -> capacity+sizeof(mem_t))
	{
		last -> capacity += to_page_size(size - last -> capacity);
		return last;
	}
	else
	{
		last -> next = (mem_t*)allocated;
		last -> next -> next = NULL;
		last -> next -> capacity = to_page_size(size) - sizeof(mem_t);
		last -> next -> is_free = 1;
		return (mem_t*)allocated;
	}
}

//обрезается излишек
mem_t*  cut_extra(mem_t *pointer, size_t size)
{
	mem_t *free = NULL;
	if(pointer->capacity >= size + sizeof(mem_t))
	{
		free = (mem_t*)((char*)(pointer) + size + sizeof(mem_t));
		free -> is_free = 1;
		free -> capacity = pointer->capacity - size - sizeof(mem_t);
		free -> next = NULL;
		pointer -> capacity = size;
		pointer -> is_free = 0;
	}
	return free;
}


//приводим размер блока к кратному BLOCK_SIZE
size_t to_page_size(size_t size)
{
	return size % BLOCK_SIZE ? size + (BLOCK_SIZE - size % BLOCK_SIZE) : size;
}

//выделение памяти
void* _malloc(size_t size)
{
	mem_t *pointer = list_find_space(HEAP_START, size);
	mem_t *new = NULL;
	mem_t *free = NULL;
	if(pointer)
	{
		new = (mem_t*)((char*)pointer + size + sizeof(mem_t));
		new -> capacity = pointer->capacity - size - sizeof(mem_t);
```

```
            new -> is_free = 1;
            list_set_front(new, pointer);
            pointer -> capacity = size;
            pointer -> is_free = 0;
      }
      else
      {
            pointer = list_get_elem_last (HEAP_START);
            new = allocate(pointer, size);
            free =  cut_extra(new, size);
            list_set_front(free, new);
            pointer = (mem_t*)(new);

      }
      return (char*)pointer + sizeof(mem_t);
}

//объединение пустых блоков
void merge()
{
      mem_t* current = HEAP_START;
      while(current -> next != NULL)
      {
        if(current -> is_free && current -> next -> is_free)
            {
            current -> capacity += current -> next -> capacity + sizeof(mem_t);
            current -> next = current -> next -> next;
        }
            else
            {
            current = current->next;
        }
    }
}

//free()
void _free(void *pointer)
{
      mem_t *mem_block = list_get_elem(HEAP_START, (mem_t*)((char*)pointer -
sizeof(mem_t)));
      if(mem_block)
      {
            mem_block -> is_free = 1;
            merge();
      }
}

void memalloc_debug_struct_info(FILE* f, mem_t const* const address)
{

    size_t i;
    fprintf(f,
            "start: %p\nsize: %lu\nis_free: %d\n",
            (void*)address,
            address -> capacity,
            address -> is_free);
    for (i = 0; i < DEBUG_FIRST_BYTES && i < address -> capacity; ++i)
            fprintf(f, "%hhX", ((char*)address)[sizeof(mem_t) + i ]);
    putc('\n', f);
}

void memalloc_debug_heap( FILE* f,  mem_t const* pointer )
{
            fprintf(f, "--- start ---\n");
    for( ; pointer; pointer = pointer->next)
        memalloc_debug_struct_info(f, pointer);
            fprintf(f, "--- finish ---\n");
}
```

Вывод

В результате проделанной работы была написана программа, реализующая кастомный аллокатор памяти.