Университет ИТМО
Факультет программной инженерии и компьютерной техники

Низкоуровневое программирование

Лабораторная работа №7

Выполнила: Машина Е.А.

Группа P33113

Преподаватель: Логинов И.П.

Санкт-Петербург
2020 г.

Задание

In this assignment, we are going to implement our own version of malloc and free based on the memory mapping system call mmap and a linked list of chunks of arbitrary sizes. It can be viewed as a simplified version of a memory manager typical for the standard C library and shares most of its weaknesses. For this assignment, the usage of malloc/calloc, free and realloc is forbidden.

Код

https://github.com/mashinakatherina/Low_Level_Programming/blob/master/Lab7

main.c

```c
#include <stdio.h>
#include "allocation.h"
#include "mem_debug.h"

#define MIN_BLOCK_SIZE 1

int main() {
    init(1000);

    char *a = (char *) _malloc(sizeof(char) * 10000);
    for (int i = 0; i < 10000; ++i) {
        a[i] = 64;
    }

    char *b = (char *) _malloc(sizeof(char) * 3);
    for (int i = 0; i < 3; ++i) {
        b[i] = 2;
    }

    char *c = (char *) _malloc(sizeof(char) * 1);
    c[0] = 'f';

    FILE *f = fopen("heap.txt", "w");
    memalloc_debug_heap(f, HEAP_START);

    _free(a);
    _free(c);
    _free(b);

    f = fopen("heap_after_free.txt", "w");
    memalloc_debug_heap(f, HEAP_START);
    puts("done");
    return 0;
}
```

allocatin.h

```c
#ifndef LAB7_ALLOCATION_H
#define LAB7_ALLOCATION_H

#include <stdbool.h>
#include <stddef.h>

#define MIN_BLOCK_SIZE 32
```

```c

void *HEAP_START;

typedef struct __attribute__((packed)) {
    struct header *next;
    size_t capacity;
    bool is_free;
} header;

void *init(size_t init_size);

void *_malloc(size_t query);

void _free(void *p);

#endif //LAB7_ALLOCATION_H
```

allocation.c

```c
#include <stdbool.h>
#include <stdio.h>
#include "allocation.h"
#include <sys/mman.h>

void *init(size_t init_size) {
    if (init_size < sizeof(header))
        return NULL;

    HEAP_START = mmap(NULL, init_size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

    if (HEAP_START == MAP_FAILED)
        return NULL;

    header header_p;
    header_p.next = NULL;
    header_p.capacity = init_size;
    header_p.is_free = true;

    *(header *) HEAP_START = header_p;
}

void *_malloc(size_t query) {
    if (MIN_BLOCK_SIZE <= 0) {
        return NULL;
    }

    if (query < MIN_BLOCK_SIZE) {
        query = MIN_BLOCK_SIZE;
    }

    header *block = (header *) HEAP_START;

    while (true) {
        if (block->is_free && query < block->capacity - sizeof(header) -
                                      MIN_BLOCK_SIZE) {
            size_t temp_size = block->capacity - query - sizeof(header);
            header *new = (header *) ((void *) block +
                                      block->capacity);
            *new = *block;
            new->capacity = temp_size;
            block->capacity = query + sizeof(header);
            block->is_free = false;
            block->next = (void *) new;
            void *place =
                    (void *) block + sizeof(header);
            return place;
        }

        if (!block->next) {
            void *p = mmap((void *) block + block->capacity, query, PROT_READ | PROT_WRITE,
                           MAP_PRIVATE | MAP_ANONYMOUS | MAP_FIXED, -1,
                           0);
            if (p == MAP_FAILED) {
                p = mmap(NULL, query, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
                if (p == MAP_FAILED)
                    return NULL;
            }
            header header_p;
            header_p.next = NULL;
            header_p.capacity = query;
            header_p.is_free = false;
            *(header *) p = header_p;
            block->next = p;
            return p + sizeof(header_p);
        }

        block = (header *) block->next;
    }

}

void _free(void *p) {
    p -= sizeof(header);
    header *head = (header *) p;
    head->is_free = true;
    while (head->next != NULL && ((header *) (head->next))->is_free) {
        head->capacity += ((header *) (head->next))->capacity + sizeof(header);
        head->next = ((header *) (head->next))->next;
    }
}
```

mem_debug.h

```
#ifndef LAB7_MEM_DEBUG_H
#define LAB7_MEM_DEBUG_H

#include <stdio.h>
#include "allocation.h"

#define DEBUG_FIRST_BYTES 4

void memalloc_debug_struct_info(FILE *f, header const *const address);

void memalloc_debug_heap(FILE *f, header const *ptr);

#endif //LAB7_MEM_DEBUG_H
```

mem_debug.c

```c
#include <bits/types/FILE.h>
#include <stdio.h>
#include "mem_debug.h"

#include "allocation.h"


void memalloc_debug_struct_info(FILE *f, header 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(header) + i]);
    putc('\n', f);
}

void memalloc_debug_heap(FILE *f, header const *ptr) {
    for (; ptr; ptr = ptr->next)
        memalloc_debug_struct_info(f, ptr);
}
```

Вывод

Выполнив эту лабораторную работу, я реализовала malloc и free.