Университет ИТМО
Кафедра ВТ

Лабораторная работа №7
Низкоуровневое программирование

Выполнил: Федоров Сергей
Группа: P33113

Санкт-Петербург
2020 г.

Задание лабораторной работы:

Реализовать на языке C аналог функции `malloc` и `free` из стандартного `<malloc.h>`, используя для выделения памяти функцию `mmap`.

Выполнение:

```
Lab7/
.
├── Makefile
├── main.c
├── mem.h
├── mem.c
├── mem_debug.h
└── mem_debug.c
```

Makefile

```
all: run

run: main
	./main

main: main.o
	gcc -o main *.o

main.o: main.c mem.c mem_debug.c
	gcc -Wpedantic -Wall -Werror -c *.c

clean:
	rm -f *.o main
```

main.c

```c
//
// Created by Sergey Fedorov on 11/19/20.
//

#include "mem.h"
#include "mem_debug.h"

#include <stdio.h>

struct movie{
    int id;
    double rating;
    char* name;
};

int main() {

    void* heap_start = NULL;

    void* big_mem_test = mem_malloc(MEMORY_CHUNK * 3);

    heap_start = (char*) big_mem_test - sizeof(struct mem);
    memalloc_debug_heap(stdout, heap_start);

    puts("^^ FIRST\n");
```

```
    struct movie * gump = mem_malloc(sizeof(struct movie));
    gump->id = 109830;
    gump->rating = 8.8;
    gump->name = "Forrest Gump";

    memalloc_debug_heap(stdout, heap_start);

    puts("^^ SECOND\n");

    struct movie * citizen_kane = mem_malloc(sizeof(struct movie));
    citizen_kane->id = 33467;
    citizen_kane->rating = 8.3;
    citizen_kane->name = "Citizen Kane";

    memalloc_debug_heap(stdout, heap_start);

    puts("^^ THIRD\n");

    struct movie * only_lovers_left_alive = mem_malloc(sizeof(struct movie));
    only_lovers_left_alive->id = 1714915;
    only_lovers_left_alive->rating = 7.3;
    only_lovers_left_alive->name = "Only Lovers Left Alive";

    memalloc_debug_heap(stdout, heap_start);

    puts("^^ FORTH (BEFORE FREE)\n");

    mem_free(only_lovers_left_alive);
    mem_free(citizen_kane);
    mem_free(gump);

    memalloc_debug_heap(stdout, heap_start);

    puts("^^ FORTH\n");
}
```

mem.h

```
//
// Created by Sergey Fedorov on 11/19/20.
//

#ifndef LOW_LEVEL_PROGRAMMING_ITMO_2020_MEM_H
#define LOW_LEVEL_PROGRAMMING_ITMO_2020_MEM_H

#include <stddef.h>
#include <stdint.h>
#include <stdio.h>

#define HEAP_START ((void*) 0x04321000)
#define MEMORY_CHUNK 4 * 1024 * 1024 // 4Kb

struct mem;

#pragma pack(push, 1)
struct mem {
    struct mem* next;
    size_t capacity;
    char is_free;
};
#pragma pack(pop)

void* mem_malloc(size_t query);
void  mem_free(void* mem);
```

```
void* heap_init(size_t initial_size);

#endif //LOW_LEVEL_PROGRAMMING_ITMO_2020_MEM_H
```

mem.c

```
//
// Created by Sergey Fedorov on 11/19/20.
//

#include "mem.h"

#include <sys/mman.h>

// STATIC POINTER TO THE START OF THE HEAP
static void* heap_start = NULL;

static struct mem* find_available(size_t query);
static struct mem* find_last(void);
static void  align_mem(void);

size_t threshold_size(size_t init);

void* mem_malloc(size_t query) {
    struct mem* ptr;
    struct mem* end;

    ptr = (struct mem*) find_available(query + sizeof(struct mem));

    if (ptr != NULL) {
        end = (struct mem*) (((char*) ptr) + sizeof(struct mem) + query);
        end->next = NULL;
        end->capacity = ptr->capacity - query - sizeof(struct mem);
        end->is_free = 1;

        ptr->next = end;
        ptr->capacity = query;
        ptr->is_free = 0;

        return ptr + 1;
    } else {
        size_t new_size = threshold_size(query + sizeof(struct mem));

        struct mem* last_ptr = NULL;
        if (heap_start != NULL) last_ptr = find_last();
        ptr = (struct mem*) heap_init(new_size);
        if (last_ptr != NULL) last_ptr->next = ptr;

        if ((long long) ptr->capacity - (long long) query - (long long) sizeof(struct
mem) > 0) {
            end = (struct mem*) (((char*) ptr) + sizeof(struct mem) + query);
            end->next = NULL;
            end->capacity = ptr->capacity - query - sizeof(struct mem);
            end->is_free = 1;

            ptr->next = end;
        }
        ptr->capacity = query;
        ptr->is_free = 0;

        return ptr + 1;
    }
}
void mem_free(void* mem) {
    mem = (char*) mem - sizeof(struct mem);
```

```
    ((struct mem*) mem)->is_free = 1;
    align_mem();
}

void* heap_init(size_t initial_size) {
    struct mem* new_ptr =
            (struct mem*) mmap(HEAP_START, initial_size, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);

    if (heap_start == NULL) heap_start = (void*) new_ptr;

    new_ptr->next = NULL;
    new_ptr->capacity = initial_size - sizeof(struct mem);
    new_ptr->is_free = 1;

    return new_ptr;
}

// HELPERS
static struct mem* find_available(size_t query) {
    struct mem* ptr = (struct mem*) heap_start;
    while (ptr) {
        if (ptr->is_free && ptr->capacity >= query)
            return ptr;
        ptr = ptr->next;
    }
    return NULL;
}

static struct mem* find_last(void) {
    struct mem* ptr = (struct mem*) heap_start;
    while (ptr->next != NULL) ptr = ptr->next;
    return ptr;
}

static void align_mem(void) {
    struct mem* ptr = (struct mem*) heap_start;
    while (ptr != NULL && ptr->next != NULL) {
        if (
            ptr->is_free &&
            ptr->next->is_free &&
            (struct mem*) ((char*) ptr + ptr->capacity + sizeof(struct mem)) == ptr->next
            ) {
            ptr->capacity += ptr->next->capacity + sizeof(struct mem);
            ptr->next = ptr->next->next;
        }
        ptr = ptr->next;
    }
}

size_t threshold_size(size_t init) {
    return init > MEMORY_CHUNK ? init : MEMORY_CHUNK;
}
```

mem_debug.h

```
//
// Created by Sergey Fedorov on 11/19/20.
//

#ifndef LOW_LEVEL_PROGRAMMING_ITMO_2020_MEM_DEBUG_H
#define LOW_LEVEL_PROGRAMMING_ITMO_2020_MEM_DEBUG_H

#include "mem.h"
```

```
#define DEBUG_FIRST_BYTES 50
#define LOG

void memalloc_debug_struct_info(FILE* f, struct mem const* const address);
void memalloc_debug_heap(FILE* f, struct mem const* ptr);

#endif //LOW_LEVEL_PROGRAMMING_ITMO_2020_MEM_DEBUG_H
```

mem_debug.c

```
//
// Created by Sergey Fedorov on 11/19/20.
//

#include "mem_debug.h"

void memalloc_debug_struct_info(FILE* f, struct mem const* const address) {

    fprintf(f, "start: %p\nsize: %lu\nis_free: %d\nnext: %p\n",
                (void*) address,
                address->capacity,
                address->is_free,
                (void*) address->next
            );

    size_t i;
    for (i = 0; i < DEBUG_FIRST_BYTES && i < address->capacity; i++)
        fprintf(f, "%hhX", ((char*) address)[sizeof(struct mem) + i]);

    putc('\n', f);
}

void memalloc_debug_heap(FILE* f, struct mem const* ptr) {
#ifdef LOG
    printf("\nStarted debugging heap with start in %p\n", (void*) ptr);
#endif
    for(; ptr; ptr = ptr->next) {
#ifdef LOG
        printf("Debugging %p\n", (void*) ptr);
#endif
        memalloc_debug_struct_info(f, ptr);
    }
}
```

Вывод:

На самом деле не особо много мыслей по поводу данной лабораторной работе, но вот следующая ремарка найдется:

Понравилось узнать как работает, а точнее узнать принцип, того как работает malloc внутри языка C. Было интересно как хранится meta-информация которая позволяет отчищать память после того как она была предоставлена программе. Также, до лабораторной работы предполагал что malloc еще использует mmap (так как функционал схож и логично переиспользовать функции).

Еще, наверное, есть такое замечание. В такой задаче как выделение памяти я сейчас использовал обычный linked list с информацией о голове. Выглядит так что это возможно не самый эффективный вариант. Возможно лучше использовать список с еще информацией и о хвосте, а возможно лучше использовать двусвязный или может что-то еще. Короче, здесь есть о чем подумать, и следовательно, скорее всего стандартная реализация malloc будет хотя бы чутка но все же сложнее чем то что здесь написал я.