## Университет ИТМО

## Кафедра ВТ

Лабораторная работа №7

Низкоуровневое программирование

Выполнил: Федоров Сергей

Группа: P33113

Санкт-Петербург

2020 г.

## Задание лабораторной работы:

Реализовать на языке С 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**

*//  
// 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 С. 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.