Университет ИТМО

Кафедра ВТ

**Языки системного программирования**

Лабораторная работа №6

Группа P3210

Нгу Фыонг Ань

ПРОВЕРИЛ:

Лукьянов Николай Михайлович

2018 год

**Assignment**

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 программы**

**#main.c**

#include "mem.h"

#include <stdio.h>

int main(void) {

int\* arr1 = \_malloc(sizeof (int)\*2);

printf("arr1: %p\n", (mem\*) ((size\_t) arr1 - sizeof (mem)));

printf("\n");

print\_all\_blocks();

int\* arr2 = \_malloc(sizeof (int)\*4);

printf("arr2: %p\n", (mem\*) ((size\_t) arr2 - sizeof (mem)));

printf("\n");

print\_all\_blocks();

//\_free(arr1);

//\_free(arr2);

print\_all\_blocks();

int\* arr3 = \_malloc(sizeof (int)\*8);

printf("arr3: %p\n", (mem\*) ((size\_t) arr3 - sizeof (mem)));

printf("\n");

print\_all\_blocks();

return 0;

}

**#mem.c**

#include "mem.h"

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#define BLOCK\_MIN\_SIZE 4

#define true 1

#define false 0

#define MAX\_INT 65536;

static mem\* heap\_start;

void mem\_init(mem \* const block, const size\_t capacity);

void \*page\_align(const void \*ptr);

void\* heap\_init(size\_t initial\_size);

void merge();

void\* find\_block(const size\_t query);

void \_free(void\* ptr) {

mem\* ptr\_struct = (mem\*) ((size\_t) ptr - sizeof (mem));

ptr\_struct->is\_free = true;

merge();

return;

}

void merge() {

mem\* current = heap\_start->next;

mem\* previous = heap\_start;

for (; current->next != NULL; current = current->next) {

if ((current->is\_free) && (current->next->is\_free)) {

current->capacity = current->capacity + current->next->capacity;

current->next = (current->next)->next;

current = previous;

}

}

}

void \*page\_align(const void \*ptr) {

const size\_t page\_size = getpagesize();

size\_t addr = (size\_t) ptr;

addr = addr - addr % page\_size + page\_size;

return (void \*) addr;

}

void\* find\_block(const size\_t query) {

mem\* current = heap\_start;

mem\* previous = heap\_start;

for (; current != NULL; current = current->next) {

if (current->is\_free == true) {

if ((current->capacity) > query) {

mem\* second\_block = (mem\*) ((size\_t) current + query + sizeof (mem));

size\_t capacity\_second\_block = current->capacity - sizeof (mem) - query;

second\_block->is\_free = true;

second\_block->capacity = capacity\_second\_block;

second\_block->next = current->next;

current->capacity = query;

current->is\_free = false;

current->next = second\_block;

return (void\*) ((size\_t) current + sizeof (mem));

}

}

previous = current;

}

size\_t needed\_mem = query - previous->capacity;

size\_t size = (size\_t) page\_align((void\*) needed\_mem);

void\* addr = (void\*) ((size\_t) previous + sizeof (mem) + previous->capacity);

void\* new\_page = mmap(addr, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS | MAP\_FIXED, -1, 0);

if (new\_page != MAP\_FAILED) {

mem\* second\_block = (mem\*) ((size\_t) previous + sizeof (mem) + query);

second\_block->is\_free = true;

second\_block->capacity = size - needed\_mem - sizeof (mem);

second\_block->next = NULL;

previous->capacity = query;

previous->is\_free = false;

previous->next = second\_block;

return (void\*) ((size\_t) previous + sizeof (mem));

}

return NULL;

}

void mem\_init(mem \* const block, const size\_t capacity) {

block->next = NULL;

block->capacity = capacity;

block->is\_free = false;

}

void\* heap\_init(size\_t initial\_size) {

size\_t size = (size\_t) page\_align((void \*) (initial\_size + sizeof (mem)));

heap\_start = (mem\*) mmap(HEAP\_START, size, PROT\_READ | PROT\_WRITE, MAP\_ANONYMOUS | MAP\_SHARED, -1, 0);

if (heap\_start == NULL) {

return NULL;

}

mem\_init(heap\_start, size - sizeof (mem));

mem\* second\_block = (mem\*) ((size\_t) heap\_start + sizeof (mem) + initial\_size);

second\_block->is\_free = true;

second\_block->capacity = heap\_start->capacity - sizeof (mem);

second\_block->next = NULL;

heap\_start->next = second\_block;

print\_all\_blocks();

return heap\_start;

}

void\* \_malloc(size\_t query) {

if (query > 65536) {

printf("Memory is not enough");

exit(EXIT\_SUCCESS);

}

int value = (int) query;

if (value < BLOCK\_MIN\_SIZE) query = BLOCK\_MIN\_SIZE;

if (heap\_start != HEAP\_START) {

heap\_init(0);

}

void\* block = find\_block(query);

if (block == NULL) return NULL;

return block;

}

void print\_all\_blocks() {

mem\* current = heap\_start;

for (; current != NULL; current = current->next) {

if (current->is\_free == false) printf("%p\n", current);

}

printf("\n");

}

**#mem.h**

#ifndef Lab\_6

#define Lab\_6

#include <stddef.h>

#include <stdint.h>

#include <sys/mman.h>

#define HEAP\_START ((void\*)0x04040000)

#pragma pack(push, 1)

typedef struct mem {

struct mem\* next;

size\_t capacity;

uint8\_t is\_free;

} mem;

#pragma pack(pop)

void\* \_malloc( size\_t query );

void\* \_calloc(size\_t num, size\_t size);

void\* \_realloc(void\* ptr, size\_t new\_size);

void \_free( void\* ptr );

void print\_all\_blocks();

#endif