УНИВЕРСИТЕТ ИТМО

ПИиКТ

Языки Системного Программирования

ЛАБОРАТОРНАЯ РАБОТА №6

Группа: P3200

Студент: Базарова Анна Александровна

2018 год

Санкт-Петербург

Цель: написать свой аллокатор памяти, не используя функции malloc/calloc, free, realloc.

Main.c

#define \_GNU\_SOURCE

#include <stdlib.h>

#include <stdio.h>

#include <stdint.h>

#include <sys/mman.h>//here lies mmap

#include <stdbool.h>

#define MIN\_QUERY\_SIZE 8

#define INIT\_BLOCK\_SIZE 4096

typedef struct block\_header block\_header;

char\* start;

struct \_\_attribute\_\_((packed)) block\_header {

block\_header\* next;//следующий блок где

size\_t capacity;//размер

bool is\_free;//свободен ли

};

block\_header\* init() {

start = mmap(start, INIT\_BLOCK\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS | MAP\_FIXED, 0, 0);//откуда, длина, доступ на чтение и запись, доступ только одному процессу, дескриптор, отступ

if (start == MAP\_FAILED)

start = mmap(NULL, INIT\_BLOCK\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, 0, 0);//если с точки старта не удалось, поставим с null

block\_header\* header = (block\_header\*)start;

header->next = NULL;

header->capacity = INIT\_BLOCK\_SIZE;

header->is\_free = true;

return header;

}

void merge(block\_header\* header) {

if (header->next != NULL && header->is\_free == true && header->next->is\_free == true) {//если оба соседних блока свободны то сделаем из них один

header->capacity += header->next->capacity + sizeof(block\_header);

header->next = header->next->next;

}

}

void\* customalloc(size\_t query) {

if (query < MIN\_QUERY\_SIZE)//кладем в блок столько данных сколько надо

query = MIN\_QUERY\_SIZE;

block\_header\* header = (block\_header\*)start;

merge(header);

while (header->is\_free == false || header->capacity < query) {

merge(header);

header = header->next;

if (header == NULL) {

header = init();

break;

}

}

header->is\_free = false;

// отдадим остаток места другому блоку, если этот блок достаточно большой

if (header->capacity - query >= MIN\_QUERY\_SIZE + sizeof(block\_header)) {

block\_header\* new\_header = (block\_header\*)((uint8\_t\*)header + query + sizeof(block\_header));

new\_header->capacity = header->capacity - query - sizeof(block\_header);

header->capacity -= new\_header->capacity;

new\_header->is\_free = true;

header->next = new\_header;

}

return (void\*)((uint8\_t\*)header + sizeof(block\_header));

}

void customfree(void\* ptr) {

block\_header\* header = (block\_header\*)((uint64\_t)ptr - sizeof(block\_header));

header->is\_free = true;

merge(header);

}

int main() {

start = mmap(NULL, INIT\_BLOCK\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, 0, 0);

void\* smth = customalloc(200);

void\* smth\_else = customalloc(10);

void\* another\_thing = customalloc(15);

block\_header\* h = (block\_header\*)start;

while (h != NULL) {

printf("h\_cap = %ld\n", h->capacity);

h = h->next;

}//показывает все выделенные нам участки памяти

customfree(smth);

customfree(smth\_else);

customfree(another\_thing);

start = mmap(NULL, INIT\_BLOCK\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, 0, 0);

void\* finel = customalloc(25);

printf("-----------------------\n");

h = (block\_header\*) start;

while (h != NULL) {

printf("h\_cap = %ld\n", h->capacity);

h = h->next;

}

return 0;

}

Custom\_alloc.h

#ifndef \_CUS\_ALL\_H\_

#define \_CUS\_ALL\_H\_

#include <stdbool.h>

#include <stdlib.h>

#include <stdio.h>

#define MIN\_BLOCK\_SIZE 32

typedef struct chunk\_header chunk\_header;

char\* start;

struct \_\_attribute\_\_((packed)) chunk\_header {

chunk\_header\* next;

size\_t capacity;

bool is\_free;

};

void\* customalloc(size\_t size);

#endif