Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

# Лабораторная работа №4 по курсу

# «Операционные системы»

Группа: М8О-211Б-23

Студент: Воробьев Г. Я.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 18.11.24

Москва, 2024

# Постановка задачи

Цель работы

Приобретение практических навыков в:

1. Создании аллокаторов памяти и их анализу;
2. Создании динамических библиотек и программ, использующие динамические библиотеки.

Задание

Исследовать два аллокатора памяти: необходимо реализовать два алгоритма аллокации памяти и сравнить их по следующим характеристикам:

- Фактор использования
- Скорость выделения блоков
- Скорость освобождения блоков
- Простота использования аллокатора

Требуется создать две динамические библиотеки, реализующие два аллокатора, соответственно. Библиотеки загружаются в память с помощью интерфейса ОС (dlopen / LoadLibrary) для работы с динамическими библиотеками. Выбор библиотеки, реализующей аллокатор, осуществляется чтением первого аргумента при запуске программы (argv[1]). Этот аргумент должен содержать путь до динамической библиотеки (относительный или абсолютный). Если аргумент не передан или по переданному пути библиотеки не оказалось, то указатели на функции, реализующие API аллокатора ниже, должны быть присвоены функциям, которые оборачивают системный аллокатор ОС (mmap / VirtualAlloc) в этот API. Эти аварийные оберточные функции должны быть реализованы внутри программы, которая загружает динамические библиотеки (см. пример на GitHub Gist). Каждый аллокатор памяти должен иметь функции аналогичные стандартным функциям malloc и free (realloc, опционально). Перед работой каждый аллокатор инициализируется свободными страницами памяти, выделенными стандартными средствами ядра (mmap / VirtualAlloc). Необходимо самостоятельно разработать стратегию тестирования для определения ключевых характеристик аллокаторов памяти. При тестировании нужно свести к минимуму потери точности из-за накладных расходов при измерении ключевых характеристик, описанных выше. Каждый аллокатор должен обладать следующим интерфейсом:

- Allocator* allocator_create(void *const memory, const size_t size) (инициализация аллокатора на памяти memory размера size);
- void allocator_destroy(Allocator *const allocator)(деинициализация структуры аллокатора);
- void* allocator_alloc(Allocator *const allocator, constsize_t size) (выделение памяти аллокатором памяти размера size);
- void allocator_free(Allocator *const allocator, void *const memory) (возвращает выделенную память аллокатору);

## Алгоритм Мак-Кьюзика-Кэрелса

Аллокатор памяти на основе алгоритма Мак-Кьюзика-Кэрелса использует несколько списков свободных блоков для управления памятью. Каждый список соответствует определенному размеру блоков. Когда требуется выделить память, аллокатор ищет подходящий свободный блок в соответствующем списке. После использования блоки возвращаются обратно в список.

Основные компоненты и принципы работы: Списки свободных блоков представляют собой структуру данных (обычно односвязный список), где каждый элемент описывает свободный участок памяти.

**Основные компоненты:**

- Списки свободных блоков — структура данных (обычно связанный список), хранящая указатели на свободные участки памяти.
- Блоки памяти — участки памяти, которые аллокатор выделяет или освобождает. Каждый блок может содержать информацию о размере блока и указатель на следующий свободный блок.
- Функции выделения и освобождения памяти:
    1. Выделение: поиск подходящего свободного блока в соответствующем списке.
    2. Освобождение: возвращение блока в соответствующий список свободных.

**Операции:**

1. Инициализация свободных списков:
    a. Аллокатор получает блок памяти размером, достаточным для управления списками.
    b. Каждый список представляет собой блоки определенного размера.
    c. Инициализация списков происходит путем установки указателей на NULL.
2. Выделение памяти:
    a. При запросе памяти, аллокатор округляет размер запроса до ближайшего подходящего размера блока.
    b. Проверяется соответствующий список свободных блоков.
    c. Если в списке есть свободный блок, он используется для удовлетворения запроса.
    d. Если в списке нет свободного блока, выделяется новый блок из общей памяти аллокатора.
    e. В случае выделения памяти, блок удаляется из списка свободных блоков, а указатель на выделенную память возвращается в программу.
3. Освобождение памяти:
    a. Если блок памяти не нужен, то он освобождается и переходит в соответствующий список свободных блоков.
    b. Блок добавляется в начало списка, чтобы его можно было использовать для будущих запросов.

## Алгоритма двойников

Это способ динамического распределения памяти, который помогает эффективно управлять памятью, минимизируя фрагментацию и улучшая производительность при выделении и освобождении блоков памяти. Данный метод один из популярных методов аллокации памяти, который находит применение в операционных системах и системах с жесткими требованиями к производительности и эффективности.

**Основные компоненты и принципы работы:**

Метод использует массив или список для хранения блоков памяти по их размерам (показатели порядка или степени двойки). Каждый блок в списке представляет собой указатель на свободный блок определенного размера, и в случае выделения или освобождения памяти эти списки обновляются.

**Операции:**

1. Выделение памяти:
    a. При выделении памяти под определенный размер, алгоритм ищет минимальный блок, который подходит под требования. Если такой блок не удается найти, то алгоритм будет увеличивать блок до ближайшей степени двойки, пока блок не станет доступным.
    b. Таким же методом он может разделять большие блоки на более мелкие.
2. Освобождение памяти:
    a. Алгоритм проверяет, свободен ли соседний блок памяти. Если так, то 2 блока объединяются в 1 больший блок, который становится доступным.
3. Поиск и объединение блоков:
    a. После того, как произошло освобождение памяти и 2 блока сливаются в один, процесс может рекурсивно повторяться, если соседние блоки так же свободны.

## Тестирование

**Алгоритм Двойников**

Время выделения 100 байт: 0.000012 секунд
Выделено 100 байт по адресу 0x102ab00c8
Время выделения 200 байт: 0.000001 секунд
Выделено 200 байт по адресу 0x102ab01c8
Freed block of size 128 at order 7
Merged blocks to size 256 at order 8
Время освобождения block1: 0.000004 секунд
Освобожден block1 по адресу 0x102ab00c8
Время выделения 150 байт: 0.000001 секунд
Выделено 150 байт по адресу 0x102ab00c8
Block3 использовал память block1.
Freed block of size 256 at order 8
Время освобождения block2: 0.000001 секунд
Освобожден block2 по адресу 0x102ab01c8
Время выделения 50 байт: 0.000001 секунд
Выделено 50 байт по адресу 0x102ab00c8
Block4 использовал память block1.
Время выделения 300 байт: 0.000001 секунд
Выделено 300 байт по адресу 0x102ab02c8
Демонстрация работы аллокатора завершена.

**Алгоритм Мак-Кьюзика-Кэрелса**

Время выделения 100 байт: 0.000000 секунд
Выделено 100 байт по адресу 0x10228c060
Время выделения 200 байт: 0.000000 секунд
Выделено 200 байт по адресу 0x10228c0d4
Время освобождения block1: 0.000000 секунд
Освобожден block1 по адресу 0x10228c060
Время выделения 150 байт: 0.000000 секунд
Выделено 150 байт по адресу 0x10228c1ac
Block3 выделил новую память.
Время освобождения block2: 0.000000 секунд
Освобожден block2 по адресу 0x10228c0d4
Время выделения 50 байт: 0.000001 секунд
Выделено 50 байт по адресу 0x10228c252
Block4 выделил новую память.
Время выделения 300 байт: 0.000000 секунд
Выделено 300 байт по адресу 0x10228c294
Демонстрация работы аллокатора завершена.

Из результатов тестов видно, что алгоритм Мак-Кьюзика-Кэрелса значительно эффективнее алгоритма двойников: в алгоритме Мак-Кьюзика-Кэрелса операции выделения и освобождения памяти проще и требуют меньше вычислительных ресурсов, так как они сводятся к добавлению или удалению блоков из списков. В алгоритме двойников операции выделения и освобождения могут быть более сложными из-за необходимости деления и объединения блоков, что увеличивает время выполнения операций.

# Код программы

## allocator_1.h

```
#ifndef ALLOCATOR_1_H

#define ALLOCATOR_1_H



#include <stdio.h>

#include <stdlib.h>

#include <stddef.h>

#include <stdbool.h>

#include <string.h>



// Количество списков свободных блоков

#define NUM_FREE_LISTS 5



// Заголовок свободного блока

struct FreeBlock {

    size_t size;                // Размер блока

    struct FreeBlock* next;    // Указатель на следующий свободный блок

};





// Allocator structure

struct Allocator {

    void* base_memory;

    void* current_memory;

    size_t initial_size;

    size_t remaining_size;
```

```
    struct FreeBlock* freelist[NUM_FREE_LISTS + 1];
};

// Макрос для определения индекса в массиве freelist
#define NDX(size) \
    (size) > 128 \
        ? (size) > 256 ? 4 : 3 \
        : (size) > 64 \
            ? 2 \
            : (size) > 32 ? 1 : 0

// Инициализация списков свободных блоков
void allocator_init(struct Allocator* allocator);

//Инициализация аллокатора на памяти memory размера size
struct Allocator* allocator_create(void* memory, size_t size);

//Деинициализация структуры аллокатора
void allocator_destroy(struct Allocator* allocator);

//Выделение памяти аллокатором памяти размера size
void* allocator_alloc(struct Allocator* allocator, size_t size);

//Возвращает выделенную память аллокатору
void allocator_free(struct Allocator* allocator, void* ptr);

#endif // ALLOCATOR1_H
```

**allocator_2.h**

```
#ifndef ALLOCATOR2_H

#define ALLOCATOR2_H



#include <stddef.h>



#define NUM_FREE_LISTS 5



// Макрос для определения индекса в массиве freelist

#define NDX(size) \

   ((size) > 256 ? 4 : \

   (size) > 128 ? 3 : \

   (size) > 64 ? 2 : \

   (size) > 32 ? 1 : 0)



struct FreeBlock {

   size_t size;

   struct FreeBlock* next;

};



struct Allocator {

   void* base_memory;

   size_t memory_size;

   struct FreeBlock* freelists[NUM_FREE_LISTS + 1];

};



struct Allocator* allocator_create(void* memory, size_t size);

void allocator_destroy(struct Allocator* allocator);

void* allocator_alloc(struct Allocator* allocator, size_t size);

void allocator_free(struct Allocator* allocator, void* memory);
```

```
#endif // ALLOCATOR2_H
```

**allocator1.c**

```
// allocator1.c

#include "../inc/allocator_1.h"

#include <sys/mman.h>



void allocator_init(struct Allocator* allocator) {

   for (int i = 0; i < NUM_FREE_LISTS; i++) {

       allocator->freelist[i] = NULL;

   }

}



// allocator_create: Инициализация аллокатора на памяти memory размера size

struct Allocator* allocator_create(void* memory, size_t size) {

   struct Allocator* allocator = (struct Allocator*)memory;

   allocator->base_memory = memory;

   allocator->current_memory = (void*)((char*)memory + sizeof(struct Allocator));

   allocator->initial_size = size - sizeof(struct Allocator);

   allocator->remaining_size = allocator->initial_size;

   allocator_init(allocator);

   return allocator;

}



// allocator_destroy: Деинициализация структуры аллокатора

void allocator_destroy(struct Allocator* allocator) {

   // Сброс аллокатора в начальное состояние

   allocator->current_memory = allocator->base_memory;

   allocator->remaining_size = allocator->initial_size;

   // Очистка всех списков свободных блоков
```

```
    for (int i = 0; i < NUM_FREE_LISTS; i++) {
        allocator->freelist[i] = NULL;
    }
    // Освобождение памяти пула
    munmap(allocator->base_memory, allocator->initial_size + sizeof(struct Allocator));
}

// allocator_alloc: Выделение памяти аллокатором памяти размера size
void* allocator_alloc(struct Allocator* allocator, size_t size) {
    if (size == 0) {
        return NULL;
    }

    // Корректировка размера для включения FreeBlock и выравнивание, если необходимо
    size_t alloc_size = sizeof(struct FreeBlock) + size;

    // Поиск подходящего списка свободных блоков
    int index = NDX(size);
    struct FreeBlock* block = allocator->freelist[index];

    if (block != NULL) {
        // Использование первого блока в списке свободных блоков
        allocator->freelist[index] = block->next;
        // Установка размера в заголовке блока
        block->size = size;
        // Возврат памяти после заголовка
        return (void*)(block + 1);
    } else {
        // Выделение нового блока из пула памяти
        void* ptr = allocator->current_memory;
        if (ptr == NULL) {
```

```
            return NULL;
        }
        // Проверка, достаточно ли осталось памяти
        if (allocator->remaining_size < alloc_size) {
            return NULL;
        }
        // Выделение блока
        block = (struct FreeBlock*)ptr;
        block->size = size;
        // Обновление указателя на текущую память и оставшегося размера
        allocator->current_memory = (void*)((char*)allocator->current_memory +
alloc_size);
        allocator->remaining_size -= alloc_size;
        // Возврат памяти после заголовка
        return (void*)(block + 1);
    }
}

// allocator_free: Возвращает выделенную память аллокатору
void allocator_free(struct Allocator* allocator, void* ptr) {
    if (ptr == NULL) {
        return;
    }

    // Получение заголовка блока
    struct FreeBlock* block = (struct FreeBlock*)ptr - 1;
    size_t size = block->size;

    // Определение, какой список свободных блоков использовать
    int index = NDX(size);
```

```
    // Вставка блока в список свободных блоков
    block->next = allocator->freelist[index];
    allocator->freelist[index] = block;
}
```

**allocator2.c**

```
#include "../inc/allocator_2.h"
#include <math.h>
#include <stdio.h> // Для отладочной информации
#include <stdbool.h>

struct Allocator* allocator_create(void* memory, size_t size) {
    if (size < (1 << NUM_FREE_LISTS)) return NULL;

    struct Allocator* allocator = (struct Allocator*)memory;
    allocator->base_memory = (char*)memory + sizeof(struct Allocator);
    allocator->memory_size = size - sizeof(struct Allocator);

    for (int i = 0; i <= NUM_FREE_LISTS; i++) {
        allocator->freelists[i] = NULL;
    }

    size_t initial_index = NDX(size);

    allocator->freelists[initial_index] = (struct FreeBlock*)allocator->base_memory;
    allocator->freelists[initial_index]->size = size;
    allocator->freelists[initial_index]->next = NULL;

    return allocator;
}
```

```
void allocator_destroy(struct Allocator* allocator) {
   // Сброс аллокатора в начальное состояние
   allocator->base_memory = NULL;
   allocator->memory_size = 0;
   // Очистка всех списков свободных блоков
   for (int i = 0; i <= NUM_FREE_LISTS; i++) {
       allocator->freelists[i] = NULL;
   }
}

void* allocator_alloc(struct Allocator* allocator, size_t size) {
   if (size == 0) return NULL;

   // Вычисление порядка для запрашиваемого размера
   size_t index = NDX(size + sizeof(struct FreeBlock));
   if (index > NUM_FREE_LISTS) return NULL;

   // Поиск подходящего блока
   while (index <= NUM_FREE_LISTS) {
       if (allocator->freelists[index]) {
           struct FreeBlock* block = allocator->freelists[index];
           allocator->freelists[index] = block->next;

           // Разделение блока, если необходимо
           while (index > NDX(size + sizeof(struct FreeBlock))) {
               index--;
               size_t block_size = 1 << index;
               struct FreeBlock* buddy = (struct FreeBlock*)((char*)block + block_size);
               buddy->size = block_size;
               buddy->next = allocator->freelists[index];
```

```
                allocator->freelists[index] = buddy;

            }



            block->size = 1 << index;

            return (char*)block + sizeof(struct FreeBlock);

        }

        index++;

    }



    return NULL;

}



void allocator_free(struct Allocator* allocator, void* memory) {

    if (!memory) return;



    struct FreeBlock* block = (struct FreeBlock*)((char*)memory - sizeof(struct
FreeBlock));

    size_t index = NDX(block->size);



    // Добавляем блок в список свободных блоков

    block->next = allocator->freelists[index];

    allocator->freelists[index] = block;



    printf("Освобожден блок размером %zu на уровне %zu\n", block->size, index); //
Отладочная информация



    // Пытаемся объединить блок с его близнецом

    while (index < NUM_FREE_LISTS) {

        size_t block_size = 1 << index;

        size_t buddy_address = ((size_t)block - (size_t)allocator->base_memory) ^
block_size;
```

```
        struct FreeBlock* buddy = (struct FreeBlock*)((char*)allocator->base_memory +
buddy_address);



        // Проверяем, свободен ли близнец

        struct FreeBlock** list = &allocator->freelists[index];

        struct FreeBlock* prev = NULL;

        struct FreeBlock* curr = *list;



        bool buddy_found = false;

        while (curr) {

            if (curr == buddy) {

                // Удаляем близнеца из списка свободных блоков

                if (prev) {

                    prev->next = curr->next;

                } else {

                    *list = curr->next;

                }



                // Объединяем блоки

                if ((size_t)buddy < (size_t)block) {

                    block = buddy;

                }



                block->size = block_size * 2;

                index = NDX(block->size);

                block->next = allocator->freelists[index];

                allocator->freelists[index] = block;

                printf("Объединены блоки до размера %zu на уровне %zu\n", block->size,
index); // Отладочная информация

                buddy_found = true;

                break;
```

```
            }

            prev = curr;

            curr = curr->next;

        }



        if (!buddy_found) {

            break; // Близнец не найден, прекращаем объединение

        }

    }

}
```

**main.c**

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <unistd.h>

#include <dlfcn.h> // Для динамической загрузки

#include <time.h>  // Для замера времени

#include <string.h> // Для использования strlen



// Указатели на функции аллокатора

static struct Allocator* (*allocator_create_func)(void*, size_t);

static void (*allocator_destroy_func)(struct Allocator*);

static void* (*allocator_alloc_func)(struct Allocator*, size_t);

static void (*allocator_free_func)(struct Allocator*, void*);



void write_message(const char* message) {

    write(STDOUT_FILENO, message, strlen(message));

}
```

```
void write_error(const char* message) {
    write(STDERR_FILENO, message, strlen(message));
}

void write_time(const char* prefix, double time) {
    char buffer[256];
    snprintf(buffer, sizeof(buffer), "%s: %f секунд\n", prefix, time);
    write(STDOUT_FILENO, buffer, strlen(buffer));
}

int main(int argc, char **argv) {
    if (argc < 2) {
        write_error("Использование: <путь_к_библиотеке>\n");
        return EXIT_FAILURE;
    }

    // Загрузка динамической библиотеки
    void *library = dlopen(argv[1], RTLD_LOCAL | RTLD_NOW);
    if (!library) {
        write_error(dlerror());
        return EXIT_FAILURE;
    }

    // Загрузка функций аллокатора
    allocator_create_func = dlsym(library, "allocator_create");
    if (!allocator_create_func) {
        write_error(dlerror());
        return EXIT_FAILURE;
    }

    allocator_destroy_func = dlsym(library, "allocator_destroy");
```

```
    if (!allocator_destroy_func) {
        write_error(dlerror());
        return EXIT_FAILURE;
    }

    allocator_alloc_func = dlsym(library, "allocator_alloc");
    if (!allocator_alloc_func) {
        write_error(dlerror());
        return EXIT_FAILURE;
    }

    allocator_free_func = dlsym(library, "allocator_free");
    if (!allocator_free_func) {
        write_error(dlerror());
        return EXIT_FAILURE;
    }

    // Определение размера пула и выделение памяти
    size_t pool_size = 1024 * 1024; // Определение размера пула (например, 1MB)
    void* memory_pool = mmap(NULL, pool_size, PROT_READ | PROT_WRITE, MAP_ANONYMOUS |
MAP_PRIVATE, -1, 0);
    if (memory_pool == MAP_FAILED) {
        write_error("Ошибка mmap\n");
        return EXIT_FAILURE;
    }

    // Создание аллокатора
    struct Allocator* allocator = allocator_create_func(memory_pool, pool_size);
    if (!allocator) {
        write_error("Ошибка создания аллокатора\n");
        munmap(memory_pool, pool_size);
```

```
    return EXIT_FAILURE;
}

// Определение границ пула памяти
char* pool_start = (char*)memory_pool;
char* pool_end = pool_start + pool_size;

struct timespec start, end;
double alloc_time, free_time;

// Выделение 100 байт
clock_gettime(CLOCK_MONOTONIC, &start);
void* block1 = allocator_alloc_func(allocator, 100);
clock_gettime(CLOCK_MONOTONIC, &end);
alloc_time = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
write_time("Время выделения 100 байт", alloc_time);

if (!block1) {
    write_error("Не удалось выделить 100 байт.\n");
} else {
    char buffer[256];
    snprintf(buffer, sizeof(buffer), "Выделено 100 байт по адресу %p\n", block1);
    write_message(buffer);
    char* block1_addr = (char*)block1;
    if (block1_addr < pool_start || block1_addr >= pool_end) {
        write_error("Block1 находится за пределами пула памяти.\n");
    }
}

// Выделение 200 байт
clock_gettime(CLOCK_MONOTONIC, &start);
```

```
void* block2 = allocator_alloc_func(allocator, 200);

clock_gettime(CLOCK_MONOTONIC, &end);

alloc_time = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;

write_time("Время выделения 200 байт", alloc_time);



if (!block2) {

    write_error("Не удалось выделить 200 байт.\n");

} else {

    char buffer[256];

    snprintf(buffer, sizeof(buffer), "Выделено 200 байт по адресу %p\n", block2);

    write_message(buffer);

    char* block2_addr = (char*)block2;

    if (block2_addr < pool_start || block2_addr >= pool_end) {

        write_error("Block2 находится за пределами пула памяти.\n");

    }

}



// Освобождение block1

if (block1) {

    clock_gettime(CLOCK_MONOTONIC, &start);

    allocator_free_func(allocator, block1);

    clock_gettime(CLOCK_MONOTONIC, &end);

    free_time = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;

    write_time("Время освобождения block1", free_time);

    char buffer[256];

    snprintf(buffer, sizeof(buffer), "Освобожден block1 по адресу %p\n", block1);

    write_message(buffer);

}



// Выделение 150 байт

clock_gettime(CLOCK_MONOTONIC, &start);
```

```
void* block3 = allocator_alloc_func(allocator, 150);
clock_gettime(CLOCK_MONOTONIC, &end);
alloc_time = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
write_time("Время выделения 150 байт", alloc_time);

if (!block3) {
    write_error("Не удалось выделить 150 байт.\n");
} else {
    char buffer[256];
    snprintf(buffer, sizeof(buffer), "Выделено 150 байт по адресу %p\n", block3);
    write_message(buffer);
    char* block3_addr = (char*)block3;
    if (block3_addr < pool_start || block3_addr >= pool_end) {
        write_error("Block3 находится за пределами пула памяти.\n");
    }
    // Проверка, использовал ли block3 память block1
    if (block3 == block1) {
        write_message("Block3 использовал память block1.\n");
    } else {
        write_message("Block3 выделил новую память.\n");
    }
}

// Дополнительные тесты для проверки повторного использования памяти
// Освобождение block2
if (block2) {
    clock_gettime(CLOCK_MONOTONIC, &start);
    allocator_free_func(allocator, block2);
    clock_gettime(CLOCK_MONOTONIC, &end);
    free_time = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
    write_time("Время освобождения block2", free_time);
```

```
    char buffer[256];
    snprintf(buffer, sizeof(buffer), "Освобожден block2 по адресу %p\n", block2);
    write_message(buffer);
}

// Выделение 50 байт
clock_gettime(CLOCK_MONOTONIC, &start);
void* block4 = allocator_alloc_func(allocator, 50);
clock_gettime(CLOCK_MONOTONIC, &end);
alloc_time = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
write_time("Время выделения 50 байт", alloc_time);

if (!block4) {
    write_error("Не удалось выделить 50 байт.\n");
} else {
    char buffer[256];
    snprintf(buffer, sizeof(buffer), "Выделено 50 байт по адресу %p\n", block4);
    write_message(buffer);
    char* block4_addr = (char*)block4;
    if (block4_addr < pool_start || block4_addr >= pool_end) {
        write_error("Block4 находится за пределами пула памяти.\n");
    }
    // Проверка, использовал ли block4 память block1
    if (block4 == block1) {
        write_message("Block4 использовал память block1.\n");
    } else {
        write_message("Block4 выделил новую память.\n");
    }
}

// Выделение 300 байт
```

```
    clock_gettime(CLOCK_MONOTONIC, &start);

    void* block5 = allocator_alloc_func(allocator, 300);

    clock_gettime(CLOCK_MONOTONIC, &end);

    alloc_time = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;

    write_time("Время выделения 300 байт", alloc_time);



    if (!block5) {

        write_error("Не удалось выделить 300 байт.\n");

    } else {

        char buffer[256];

        snprintf(buffer, sizeof(buffer), "Выделено 300 байт по адресу %p\n", block5);

        write_message(buffer);

        char* block5_addr = (char*)block5;

        if (block5_addr < pool_start || block5_addr >= pool_end) {

            write_error("Block5 находится за пределами пула памяти.\n");

        }

    }



    // Уничтожение аллокатора

    allocator_destroy_func(allocator);

    munmap(memory_pool, pool_size);



    // Закрытие динамической библиотеки

    dlclose(library);



    write_message("Демонстрация работы аллокатора завершена.\n");



    return EXIT_SUCCESS;

}
```

# Протокол работы программы

**Dtruss:**

```
SYSCALL(args)            = return

Время выделения 100 байт: 0.000000 секунд

Выделено 100 байт по адресу 0x10046c060

Время выделения 200 байт: 0.000000 секунд

Выделено 200 байт по адресу 0x10046c0d4

Время освобождения block1: 0.000001 секунд

Освобожден block1 по адресу 0x10046c060

Время выделения 150 байт: 0.000000 секунд

Выделено 150 байт по адресу 0x10046c1ac

Block3 выделил новую память.

Время освобождения block2: 0.000000 секунд

Освобожден block2 по адресу 0x10046c0d4

Время выделения 50 байт: 0.000000 секунд

Выделено 50 байт по адресу 0x10046c252

Block4 выделил новую память.

Время выделения 300 байт: 0.000000 секунд

Выделено 300 байт по адресу 0x10046c294

Демонстрация работы аллокатора завершена.

munmap(0x10043C000, 0x84000)             = 0 0

munmap(0x1004C0000, 0x8000)              = 0 0

munmap(0x1004C8000, 0x4000)              = 0 0

munmap(0x1004CC000, 0x4000)              = 0 0

munmap(0x1004D0000, 0x48000)             = 0 0

munmap(0x100518000, 0x4C000)             = 0 0

crossarch_trap(0x0, 0x0, 0x0)            = -1 Err#45

open(".\0", 0x100000, 0x0)               = 3 0

fcntl(0x3, 0x32, 0x16F9F30F8)            = 0 0
```

```
close(0x3)              = 0 0

fsgetpath(0x16F9F3108, 0x400, 0x16F9F30E8)              = 58 0

fsgetpath(0x16F9F3118, 0x400, 0x16F9F30F8)              = 14 0

csrctl(0x0, 0x16F9F351C, 0x4)           = -1 Err#1

__mac_syscall(0x19D89BC12, 0x2, 0x16F9F3460)            = 0 0

csrctl(0x0, 0x16F9F350C, 0x4)           = -1 Err#1

__mac_syscall(0x19D898A45, 0x5A, 0x16F9F34A0)           = 0 0

sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16F9F2A08, 0x16F9F2A00, 0x19D89A738, 0xD)
= 0 0

sysctl([CTL_KERN, 150, 0, 0, 0, 0] (2), 0x16F9F2AB8, 0x16F9F2AB0, 0x0, 0x0)              = 0
0

open("/\0", 0x20100000, 0x0)            = 3 0

openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0)             = 4 0

dup(0x4, 0x0, 0x0)              = 5 0

fstatat64(0x4, 0x16F9F2591, 0x16F9F2500)                = 0 0

openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0)            = 6 0

fcntl(0x6, 0x32, 0x16F9F2590)           = 0 0

dup(0x6, 0x0, 0x0)              = 7 0

dup(0x5, 0x0, 0x0)              = 8 0

close(0x3)              = 0 0

close(0x5)              = 0 0

close(0x4)              = 0 0

close(0x6)              = 0 0

__mac_syscall(0x19D89BC12, 0x2, 0x16F9F2F80)            = 0 0

shared_region_check_np(0x16F9F2BA0, 0x0, 0x0)           = 0 0

fsgetpath(0x16F9F3120, 0x400, 0x16F9F3048)              = 82 0

fcntl(0x8, 0x32, 0x16F9F3120)           = 0 0

close(0x8)              = 0 0

close(0x7)              = 0 0

getfsstat64(0x0, 0x0, 0x2)              = 11 0

getfsstat64(0x10040C050, 0x5D28, 0x2)           = 11 0
```

```
getattrlist("/\0", 0x16F9F3060, 0x16F9F2FD0)             = 0 0

stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared_cache_arm64e\0",
0x16F9F33C0, 0x0)          = 0 0

dtrace: error on enabled probe ID 1696 (ID 845: syscall::stat64:return): invalid address
(0x0) in action #11 at DIF offset 12

stat64("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/core\0", 0x16F9F2870, 0x0)
= 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/core\0", 0x0, 0x0)            = 3
0

mmap(0x0, 0x85D8, 0x1, 0x40002, 0x3, 0x0)                = 0x10040C000 0

fcntl(0x3, 0x32, 0x16F9F2988)            = 0 0

close(0x3)               = 0 0

munmap(0x10040C000, 0x85D8)              = 0 0

stat64("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/core\0", 0x16F9F2DE0, 0x0)
= 0 0

open("@rpath/liballocator_1.dylib\0", 0x0, 0x0)          = -1 Err#2

open("@rpath\0", 0x100000, 0x0)          = -1 Err#2

stat64("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_1.dylib\0",
0x16F9F1C00, 0x0)          = 0 0

stat64("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_1.dylib\0",
0x16F9F1630, 0x0)          = 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_1.dylib\0", 0x0, 0x0)
= 3 0

mmap(0x0, 0x8338, 0x1, 0x40002, 0x3, 0x0)                = 0x10040C000 0

fcntl(0x3, 0x32, 0x16F9F1748)            = 0 0

close(0x3)               = 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_1.dylib\0", 0x0, 0x0)
= 3 0

fstat64(0x3, 0x16F9F0DF0, 0x0)           = 0 0

fcntl(0x3, 0x61, 0x16F9F13E8)            = 0 0

fcntl(0x3, 0x62, 0x16F9F13E8)            = 0 0

mmap(0x100418000, 0x4000, 0x5, 0x40012, 0x3, 0x0)               = 0x100418000 0

mmap(0x10041C000, 0x4000, 0x3, 0x40012, 0x3, 0x4000)            = 0x10041C000 0

mmap(0x100420000, 0x4000, 0x1, 0x40012, 0x3, 0x8000)            = 0x100420000 0
```

```
close(0x3)              = 0 0

munmap(0x10040C000, 0x8338)             = 0 0

stat64("/usr/lib/libSystem.B.dylib\0", 0x16F9F1D30, 0x0)                = -1 Err#2

stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0", 0x16F9F1CE0, 0x0)
= -1 Err#2

stat64("/usr/lib/libSystem.B.dylib\0", 0x16F9F1C60, 0x0)                = -1 Err#2

stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0", 0x16F9F1C10, 0x0)
= -1 Err#2

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/core\0", 0x0, 0x0)           = 3
0

__mac_syscall(0x19D89BC12, 0x2, 0x16F9F0420)            = 0 0

map_with_linking_np(0x16F9F02E0, 0x1, 0x16F9F0310)              = 0 0

close(0x3)              = 0 0

mprotect(0x100400000, 0x4000, 0x1)              = 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_1.dylib\0", 0x0, 0x0)
= 3 0

__mac_syscall(0x19D89BC12, 0x2, 0x16F9F0420)            = 0 0

map_with_linking_np(0x16F9F0340, 0x1, 0x16F9F0370)              = 0 0

close(0x3)              = 0 0

mprotect(0x10041C000, 0x4000, 0x1)              = 0 0

open("/dev/dtracehelper\0", 0x2, 0x0)           = 3 0

ioctl(0x3, 0x80086804, 0x16F9EF968)             = 0 0

close(0x3)              = 0 0

shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)            = 0 0

access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0)            = -1 Err#2

bsdthread_register(0x19DB9E0F4, 0x19DB9E0E8, 0x4000)            = 1073746399 0

getpid(0x0, 0x0, 0x0)           = 1203 0

shm_open(0x19DA35F41, 0x0, 0xFFFFFFFFAB5FA000)          = 3 0

fstat64(0x3, 0x16F9F0030, 0x0)          = 0 0

mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0)               = 0x100424000 0

close(0x3)              = 0 0

csops(0x4B3, 0x0, 0x16F9F016C)          = 0 0
```

```
ioctl(0x2, 0x4004667A, 0x16F9F00DC)              = 0 0

mprotect(0x100434000, 0x4000, 0x0)               = 0 0

mprotect(0x100440000, 0x4000, 0x0)               = 0 0

mprotect(0x100444000, 0x4000, 0x0)               = 0 0

mprotect(0x100450000, 0x4000, 0x0)               = 0 0

mprotect(0x100454000, 0x4000, 0x0)               = 0 0

mprotect(0x100460000, 0x4000, 0x0)               = 0 0

mprotect(0x10042C000, 0xC8, 0x1)                 = 0 0

mprotect(0x10042C000, 0xC8, 0x3)                 = 0 0

mprotect(0x10042C000, 0xC8, 0x1)                 = 0 0

mprotect(0x100414000, 0x4000, 0x1)               = 0 0

mprotect(0x100464000, 0xC8, 0x1)                 = 0 0

mprotect(0x100464000, 0xC8, 0x3)                 = 0 0

mprotect(0x100464000, 0xC8, 0x1)                 = 0 0

mprotect(0x10042C000, 0xC8, 0x3)                 = 0 0

mprotect(0x10042C000, 0xC8, 0x1)                 = 0 0

mprotect(0x100414000, 0x4000, 0x3)               = 0 0

mprotect(0x100414000, 0x4000, 0x1)               = 0 0

issetugid(0x0, 0x0, 0x0)                 = 0 0

getentropy(0x16F9EF718, 0x20, 0x0)               = 0 0

getattrlist("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/core\0", 0x16F9EFFD0,
0x16F9EFFEC)             = 0 0

access("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build\0", 0x4, 0x0)               = 0
0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build\0", 0x0, 0x0)                 = 3
0

fstat64(0x3, 0x14F6044A0, 0x0)           = 0 0

csrctl(0x0, 0x16F9F01BC, 0x4)            = 0 0

fcntl(0x3, 0x32, 0x16F9EFEB8)            = 0 0

close(0x3)               = 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/Info.plist\0", 0x0, 0x0)
= -1 Err#2
```

```
proc_info(0x2, 0x4B3, 0xD)               = 64 0

csops_audittoken(0x4B3, 0x10, 0x16F9F0240)               = 0 0

sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16F9F0598, 0x16F9F0590, 0x1A12C8D3A, 0x15)
= 0 0

sysctl([CTL_KERN, 148, 0, 0, 0, 0] (2), 0x16F9F0628, 0x16F9F0620, 0x0, 0x0)              = 0
0

open("./liballocator_1.dylib\0", 0x0, 0x0)               = 3 0

fcntl(0x3, 0x32, 0x16FA01A08)            = 0 0

close(0x3)               = 0 0

stat64("./liballocator_1.dylib\0", 0x16FA01570, 0x0)             = 0 0

mmap(0x0, 0x100000, 0x3, 0x41002, 0xFFFFFFFFFFFFFFFF, 0x0)               = 0x10046C000 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\262\321\213\320\264\320\265\320\273\320\265\320\275\320\270\321\217 100
\320\261\320\260\320\271\321\202: 0.000000
\321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)               = 66 0

write(0x1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275\320\276 100
\320\261\320\260\320\271\321\202 \320\277\320\276
\320\260\320\264\321\200\320\265\321\201\321\203 0x10046c060\n\0", 0x3C)             = 60 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\262\321\213\320\264\320\265\320\273\320\265\320\275\320\270\321\217 200
\320\261\320\260\320\271\321\202: 0.000000
\321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)               = 66 0

write(0x1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275\320\276 200
\320\261\320\260\320\271\321\202 \320\277\320\276
\320\260\320\264\321\200\320\265\321\201\321\203 0x10046c0d4\n\0", 0x3C)             = 60 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\276\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275\320\270\321\
217 block1: 0.000001 \321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)
= 66 0

write(0x1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275
block1 \320\277\320\276 \320\260\320\264\321\200\320\265\321\201\321\203 0x10046c060\n\0",
0x3A)           = 58 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\262\321\213\320\264\320\265\320\273\320\265\320\275\320\270\321\217 150
\320\261\320\260\320\271\321\202: 0.000000
\321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)               = 66 0

write(0x1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275\320\276 150
\320\261\320\260\320\271\321\202 \320\277\320\276
\320\260\320\264\321\200\320\265\321\201\321\203 0x10046c1ac\n\0", 0x3C)             = 60 0
```

```
write(0x1, "Block3 \320\262\321\213\320\264\320\265\320\273\320\270\320\273
\320\275\320\276\320\262\321\203\321\216
\320\277\320\260\320\274\321\217\321\202\321\214.\n\0", 0x2F)     = 47 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\276\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275\320\270\321\
217 block2: 0.000000 \321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)
= 66 0

write(0x1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275
block2 \320\277\320\276 \320\260\320\264\321\200\320\265\321\201\321\203 0x10046c0d4\n\0",
0x3A)           = 58 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\262\321\213\320\264\320\265\320\273\320\265\320\275\320\270\321\217 50
\320\261\320\260\320\271\321\202: 0.000000
\321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x41)                = 65 0

write(0x1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275\320\276 50
\320\261\320\260\320\271\321\202 \320\277\320\276
\320\260\320\264\321\200\320\265\321\201\321\203 0x10046c252\n\0", 0x3B)              = 59 0

write(0x1, "Block4 \320\262\321\213\320\264\320\265\320\273\320\270\320\273
\320\275\320\276\320\262\321\203\321\216
\320\277\320\260\320\274\321\217\321\202\321\214.\n\0", 0x2F)     = 47 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\262\321\213\320\264\320\265\320\273\320\265\320\275\320\270\321\217 300
\320\261\320\260\320\271\321\202: 0.000000
\321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)               = 66 0

write(0x1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275\320\276 300
\320\261\320\260\320\271\321\202 \320\277\320\276
\320\260\320\264\321\200\320\265\321\201\321\203 0x10046c294\n\0", 0x3C)             = 60 0

munmap(0x10046C000, 0x100000)            = 0 0

munmap(0x10046C000, 0x100000)            = 0 0

write(0x1,
"\320\224\320\265\320\274\320\276\320\275\321\201\321\202\321\200\320\260\321\206\320\270\321
\217 \321\200\320\260\320\261\320\276\321\202\321\213
\320\260\320\273\320\273\320\276\320\272\320\260\321\202\320\276\321\200\320\260
\320\267\320\260\320\262\320\265\321\200\321\210\320\265\320\275\320\260.\n\0", 0x4F)
= 79 0
```

```
glebvorobev@Glebs-MacBook-Air build % sudo dtruss ./core ./liballocator_2.dylib

SYSCALL(args)            = return
```

```
Время выделения 100 байт: 0.000005 секунд

Выделено 100 байт по адресу 0x1020c00c8

Время выделения 200 байт: 0.000000 секунд

Выделено 200 байт по адресу 0x1020c01c8

Освобожден блок размером 128 на уровне 7

Объединены блоки до размера 256 на уровне 8

Время освобождения block1: 0.000025 секунд

Освобожден block1 по адресу 0x1020c00c8

Время выделения 150 байт: 0.000000 секунд

Выделено 150 байт по адресу 0x1020c00c8

Block3 использовал память block1.

Освобожден блок размером 256 на уровне 8

Время освобождения block2: 0.000005 секунд

Освобожден block2 по адресу 0x1020c01c8

Время выделения 50 байт: 0.000000 секунд

Выделено 50 байт по адресу 0x1020c00c8

Block4 использовал память block1.

Время выделения 300 байт: 0.000001 секунд

Выделено 300 байт по адресу 0x1020c02c8

Демонстрация работы аллокатора завершена.

munmap(0x102100000, 0x84000)             = 0 0

munmap(0x102184000, 0x8000)              = 0 0

munmap(0x10218C000, 0x4000)              = 0 0

munmap(0x102190000, 0x4000)              = 0 0

munmap(0x102194000, 0x48000)             = 0 0

munmap(0x1021DC000, 0x4C000)             = 0 0

crossarch_trap(0x0, 0x0, 0x0)            = -1 Err#45

open(".\0", 0x100000, 0x0)               = 3 0

fcntl(0x3, 0x32, 0x16DDB70F8)            = 0 0

close(0x3)               = 0 0

fsgetpath(0x16DDB7108, 0x400, 0x16DDB70E8)              = 58 0
```

```
fsgetpath(0x16DDB7118, 0x400, 0x16DDB70F8)              = 14 0

csrctl(0x0, 0x16DDB751C, 0x4)           = -1 Err#1

__mac_syscall(0x19D89BC12, 0x2, 0x16DDB7460)            = 0 0

csrctl(0x0, 0x16DDB750C, 0x4)           = -1 Err#1

__mac_syscall(0x19D898A45, 0x5A, 0x16DDB74A0)           = 0 0

sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16DDB6A08, 0x16DDB6A00, 0x19D89A738, 0xD)
= 0 0

sysctl([CTL_KERN, 150, 0, 0, 0, 0] (2), 0x16DDB6AB8, 0x16DDB6AB0, 0x0, 0x0)             = 0
0

open("/\0", 0x20100000, 0x0)            = 3 0

openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0)             = 4 0

dup(0x4, 0x0, 0x0)              = 5 0

fstatat64(0x4, 0x16DDB6591, 0x16DDB6500)                = 0 0

openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0)            = 6 0

fcntl(0x6, 0x32, 0x16DDB6590)           = 0 0

dup(0x6, 0x0, 0x0)              = 7 0

dup(0x5, 0x0, 0x0)              = 8 0

close(0x3)              = 0 0

close(0x5)              = 0 0

close(0x4)              = 0 0

close(0x6)              = 0 0

__mac_syscall(0x19D89BC12, 0x2, 0x16DDB6F80)            = 0 0

shared_region_check_np(0x16DDB6BA0, 0x0, 0x0)           = 0 0

fsgetpath(0x16DDB7120, 0x400, 0x16DDB7048)              = 82 0

fcntl(0x8, 0x32, 0x16DDB7120)           = 0 0

close(0x8)              = 0 0

close(0x7)              = 0 0

getfsstat64(0x0, 0x0, 0x2)              = 11 0

getfsstat64(0x102048050, 0x5D28, 0x2)           = 11 0

getattrlist("/\0", 0x16DDB7060, 0x16DDB6FD0)            = 0 0

stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared_cache_arm64e\0",
0x16DDB73C0, 0x0)          = 0 0
```

```
dtrace: error on enabled probe ID 1696 (ID 845: syscall::stat64:return): invalid address
(0x0) in action #11 at DIF offset 12

stat64("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/core\0", 0x16DDB6870, 0x0)
= 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/core\0", 0x0, 0x0)            = 3
0

mmap(0x0, 0x85D8, 0x1, 0x40002, 0x3, 0x0)                = 0x102048000 0

fcntl(0x3, 0x32, 0x16DDB6988)            = 0 0

close(0x3)               = 0 0

munmap(0x102048000, 0x85D8)              = 0 0

stat64("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/core\0", 0x16DDB6DE0, 0x0)
= 0 0

open("@rpath/liballocator_1.dylib\0", 0x0, 0x0)          = -1 Err#2

open("@rpath\0", 0x100000, 0x0)          = -1 Err#2

stat64("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_1.dylib\0",
0x16DDB5C00, 0x0)          = 0 0

stat64("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_1.dylib\0",
0x16DDB5630, 0x0)          = 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_1.dylib\0", 0x0, 0x0)
= 3 0

mmap(0x0, 0x8338, 0x1, 0x40002, 0x3, 0x0)                = 0x102048000 0

fcntl(0x3, 0x32, 0x16DDB5748)            = 0 0

close(0x3)               = 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_1.dylib\0", 0x0, 0x0)
= 3 0

fstat64(0x3, 0x16DDB4DF0, 0x0)           = 0 0

fcntl(0x3, 0x61, 0x16DDB53E8)            = 0 0

fcntl(0x3, 0x62, 0x16DDB53E8)            = 0 0

mmap(0x102054000, 0x4000, 0x5, 0x40012, 0x3, 0x0)                = 0x102054000 0

mmap(0x102058000, 0x4000, 0x3, 0x40012, 0x3, 0x4000)             = 0x102058000 0

mmap(0x10205C000, 0x4000, 0x1, 0x40012, 0x3, 0x8000)             = 0x10205C000 0

close(0x3)               = 0 0

munmap(0x102048000, 0x8338)              = 0 0

stat64("/usr/lib/libSystem.B.dylib\0", 0x16DDB5D30, 0x0)                 = -1 Err#2
```

```
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0", 0x16DDB5CE0, 0x0)
= -1 Err#2

stat64("/usr/lib/libSystem.B.dylib\0", 0x16DDB5C60, 0x0)                 = -1 Err#2

stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0", 0x16DDB5C10, 0x0)
= -1 Err#2

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/core\0", 0x0, 0x0)            = 3
0

__mac_syscall(0x19D89BC12, 0x2, 0x16DDB4420)             = 0 0

map_with_linking_np(0x16DDB42E0, 0x1, 0x16DDB4310)               = 0 0

close(0x3)               = 0 0

mprotect(0x10203C000, 0x4000, 0x1)               = 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_1.dylib\0", 0x0, 0x0)
= 3 0

__mac_syscall(0x19D89BC12, 0x2, 0x16DDB4420)             = 0 0

map_with_linking_np(0x16DDB4340, 0x1, 0x16DDB4370)               = 0 0

close(0x3)               = 0 0

mprotect(0x102058000, 0x4000, 0x1)               = 0 0

open("/dev/dtracehelper\0", 0x2, 0x0)            = 3 0

ioctl(0x3, 0x80086804, 0x16DDB3968)              = 0 0

close(0x3)               = 0 0

shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)             = 0 0

access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0)             = -1 Err#2

bsdthread_register(0x19DB9E0F4, 0x19DB9E0E8, 0x4000)             = 1073746399 0

getpid(0x0, 0x0, 0x0)            = 1914 0

shm_open(0x19DA35F41, 0x0, 0xFFFFFFFFAB5FA000)           = 3 0

fstat64(0x3, 0x16DDB4030, 0x0)           = 0 0

mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0)                = 0x102060000 0

close(0x3)               = 0 0

csops(0x77A, 0x0, 0x16DDB416C)           = 0 0

ioctl(0x2, 0x4004667A, 0x16DDB40DC)              = 0 0

mprotect(0x102070000, 0x4000, 0x0)               = 0 0

mprotect(0x10207C000, 0x4000, 0x0)               = 0 0
```

```
mprotect(0x102080000, 0x4000, 0x0)                = 0 0

mprotect(0x10208C000, 0x4000, 0x0)                = 0 0

mprotect(0x102090000, 0x4000, 0x0)                = 0 0

mprotect(0x10209C000, 0x4000, 0x0)                = 0 0

mprotect(0x102068000, 0xC8, 0x1)                  = 0 0

mprotect(0x102068000, 0xC8, 0x3)                  = 0 0

mprotect(0x102068000, 0xC8, 0x1)                  = 0 0

mprotect(0x102050000, 0x4000, 0x1)                = 0 0

mprotect(0x1020A0000, 0xC8, 0x1)                  = 0 0

mprotect(0x1020A0000, 0xC8, 0x3)                  = 0 0

mprotect(0x1020A0000, 0xC8, 0x1)                  = 0 0

mprotect(0x102068000, 0xC8, 0x3)                  = 0 0

mprotect(0x102068000, 0xC8, 0x1)                  = 0 0

mprotect(0x102050000, 0x4000, 0x3)                = 0 0

mprotect(0x102050000, 0x4000, 0x1)                = 0 0

issetugid(0x0, 0x0, 0x0)                  = 0 0

getentropy(0x16DDB3718, 0x20, 0x0)                = 0 0

getattrlist("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/core\0", 0x16DDB3FD0,
0x16DDB3FEC)              = 0 0

access("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build\0", 0x4, 0x0)                = 0
0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build\0", 0x0, 0x0)                  = 3
0

fstat64(0x3, 0x14EE044A0, 0x0)            = 0 0

csrctl(0x0, 0x16DDB41BC, 0x4)             = 0 0

fcntl(0x3, 0x32, 0x16DDB3EB8)             = 0 0

close(0x3)                = 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/Info.plist\0", 0x0, 0x0)
= -1 Err#2

proc_info(0x2, 0x77A, 0xD)                = 64 0

csops_audittoken(0x77A, 0x10, 0x16DDB4240)                = 0 0

sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16DDB4598, 0x16DDB4590, 0x1A12C8D3A, 0x15)
= 0 0
```

```
sysctl([CTL_KERN, 148, 0, 0, 0, 0] (2), 0x16DDB4628, 0x16DDB4620, 0x0, 0x0)              = 0
0

open("./liballocator_2.dylib\0", 0x0, 0x0)               = 3 0

fcntl(0x3, 0x32, 0x16DDC5A08)            = 0 0

close(0x3)               = 0 0

stat64("./liballocator_2.dylib\0", 0x16DDC5570, 0x0)             = 0 0

stat64("./liballocator_2.dylib\0", 0x16DDC4FA0, 0x0)             = 0 0

open("./liballocator_2.dylib\0", 0x0, 0x0)               = 3 0

mmap(0x0, 0x8328, 0x1, 0x40002, 0x3, 0x0)                = 0x1020A8000 0

fcntl(0x3, 0x32, 0x16DDC50B8)            = 0 0

close(0x3)               = 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_2.dylib\0", 0x0, 0x0)
= 3 0

fstat64(0x3, 0x16DDC4760, 0x0)           = 0 0

fcntl(0x3, 0x61, 0x16DDC4D58)            = 0 0

fcntl(0x3, 0x62, 0x16DDC4D58)            = 0 0

mmap(0x1020B4000, 0x4000, 0x5, 0x40012, 0x3, 0x0)                = 0x1020B4000 0

mmap(0x1020B8000, 0x4000, 0x3, 0x40012, 0x3, 0x4000)             = 0x1020B8000 0

mmap(0x1020BC000, 0x4000, 0x1, 0x40012, 0x3, 0x8000)             = 0x1020BC000 0

close(0x3)               = 0 0

munmap(0x1020A8000, 0x8328)              = 0 0

open("/Users/glebvorobev/Documents/Projects/OS/LAB_4/build/liballocator_2.dylib\0", 0x0, 0x0)
= 3 0

close(0x3)               = 0 0

mprotect(0x1020B8000, 0x4000, 0x1)               = 0 0

mmap(0x0, 0x100000, 0x3, 0x41002, 0xFFFFFFFFFFFFFFFF, 0x0)               = 0x1020C0000 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\262\321\213\320\264\320\265\320\273\320\265\320\275\320\270\321\217 100
\320\261\320\260\320\271\321\202: 0.000005
\321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)               = 66 0

write(0x1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275\320\276 100
\320\261\320\260\320\271\321\202 \320\277\320\276
\320\260\320\264\321\200\320\265\321\201\321\203 0x1020c00c8\n\0", 0x3C)             = 60 0
```

```
write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\262\321\213\320\264\320\265\320\273\320\265\320\275\320\270\321\217 200
\320\261\320\260\320\271\321\202: 0.000000
\321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)               = 66 0

write(0x1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275\320\276 200
\320\261\320\260\320\271\321\202 \320\277\320\276
\320\260\320\264\321\200\320\265\321\201\321\203 0x1020c01c8\n\0", 0x3C)             = 60 0

fstat64(0x1, 0x16DDC6C10, 0x0)           = 0 0

ioctl(0x1, 0x4004667A, 0x16DDC6C5C)              = 0 0

write_nocancel(0x1,
"\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275
\320\261\320\273\320\276\320\272
\321\200\320\260\320\267\320\274\320\265\321\200\320\276\320\274 128 \320\275\320\260
\321\203\321\200\320\276\320\262\320\275\320\265 7\n\0", 0x47)             = 71 0

write_nocancel(0x1,
"\320\236\320\261\321\212\320\265\320\264\320\270\320\275\320\265\320\275\321\213
\320\261\320\273\320\276\320\272\320\270 \320\264\320\276
\321\200\320\260\320\267\320\274\320\265\321\200\320\260 256 \320\275\320\260
\321\203\321\200\320\276\320\262\320\275\320\265 8\n\0", 0x4C)            = 76 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\276\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275\320\270\321\
217 block1: 0.000025 \321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)
= 66 0

write(0x1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275
block1 \320\277\320\276 \320\260\320\264\321\200\320\265\321\201\321\203 0x1020c00c8\n\0",
0x3A)           = 58 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\262\321\213\320\264\320\265\320\273\320\265\320\275\320\270\321\217 150
\320\261\320\260\320\271\321\202: 0.000000
\321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)               = 66 0

write(0x1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275\320\276 150
\320\261\320\260\320\271\321\202 \320\277\320\276
\320\260\320\264\321\200\320\265\321\201\321\203 0x1020c00c8\n\0", 0x3C)             = 60 0

write(0x1, "Block3
\320\270\321\201\320\277\320\276\320\273\321\214\320\267\320\276\320\262\320\260\320\273
\320\277\320\260\320\274\321\217\321\202\321\214 block1.\n\0", 0x33)       = 51 0

write_nocancel(0x1,
"\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275
\320\261\320\273\320\276\320\272
\321\200\320\260\320\267\320\274\320\265\321\200\320\276\320\274 256 \320\275\320\260
\321\203\321\200\320\276\320\262\320\275\320\265 8\n\0", 0x47)             = 71 0
```

```
write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\276\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275\320\270\321\
217 block2: 0.000005 \321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)
= 66 0

write(0x1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275
block2 \320\277\320\276 \320\260\320\264\321\200\320\265\321\201\321\203 0x1020c01c8\n\0",
0x3A)           = 58 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\262\321\213\320\264\320\265\320\273\320\265\320\275\320\270\321\217 50
\320\261\320\260\320\271\321\202: 0.000000
\321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x41)                = 65 0

write(0x1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275\320\276 50
\320\261\320\260\320\271\321\202 \320\277\320\276
\320\260\320\264\321\200\320\265\321\201\321\203 0x1020c00c8\n\0", 0x3B)              = 59 0

write(0x1, "Block4
\320\270\321\201\320\277\320\276\320\273\321\214\320\267\320\276\320\262\320\260\320\273
\320\277\320\260\320\274\321\217\321\202\321\214 block1.\n\0", 0x33)       = 51 0

write(0x1, "\320\222\321\200\320\265\320\274\321\217
\320\262\321\213\320\264\320\265\320\273\320\265\320\275\320\270\321\217 300
\320\261\320\260\320\271\321\202: 0.000001
\321\201\320\265\320\272\321\203\320\275\320\264\n\0", 0x42)               = 66 0

write(0x1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275\320\276 300
\320\261\320\260\320\271\321\202 \320\277\320\276
\320\260\320\264\321\200\320\265\321\201\321\203 0x1020c02c8\n\0", 0x3C)             = 60 0

munmap(0x1020C0000, 0x100000)            = 0 0

munmap(0x1020B4000, 0xC000)              = 0 0

write(0x1,
"\320\224\320\265\320\274\320\276\320\275\321\201\321\202\321\200\320\260\321\206\320\270\321
\217 \321\200\320\260\320\261\320\276\321\202\321\213
\320\260\320\273\320\273\320\276\320\272\320\260\321\202\320\276\321\200\320\260
\320\267\320\260\320\262\320\265\321\200\321\210\320\265\320\275\320\260.\n\0", 0x4F)
= 79 0
```

# Вывод

В ходе написания данной лабораторной работы я узнал об устройстве аллокаторов. Научился создавать, подключать и использовать динамические библиотеки. Были реализованы два алгоритма аллокации памяти, работающие через один API и подключаемые через динамические библиотеки. В ходе написания данной лабораторной работы я узнал об устройстве аллокаторов. Научился создавать, подключать и использовать динамические библиотеки. Были реализованы два алгоритма аллокации памяти, работающие через один API и подключаемые через динамические библиотеки.