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: Малеев В.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 25.12.24

Москва, 2024

Постановка задачи

Вариант 5.

Исследовать два аллокатора памяти: Алгоритм Мак-Кьюзика-Кэрелса и алгоритм двойников. Необходимо реализовать два алгоритма аллокации памяти и сравнить их не используя malloc, free, calloc, realloc, new, delete.

Общий метод и алгоритм решения

Использованные системные вызовы:

1. **int* munmap(void addr, size_t length); -** Удаляет отображения, созданные с помощью mmap.
2. **int* dlclose(void handle); -** Закрывает динамическую библиотеку, открытую с помощью dlopen, и освобождает ресурсы, связанные с этим дескриптором.
3. **void exit(int status); -** Завершает выполнение программы и возвращает статус выхода в операционную систему.
4. **char* dlerror(void); -** Возвращает строку, описывающую последнюю ошибку, возникшую при вызове функций dlopen, dlsym, dlclose.
5. **void** dlopen(const char filename, int flag); -** Открывает динамическую библиотеку и возвращает дескриптор для последующего использования.
6. **void* mmap(void addr, size_t length, int prot, int flags, int fd, off_t offset);** – создает новое отображение памяти или изменяет существующее**.**

Описание лабораторной работы

В рамках лабораторной работы была разработана программа, которая демонстрирует работу двух аллокаторов работающих по разным алгоритмам.

Цель лабораторной работы

Приобретение практических навыков в:

1) Создании аллокаторов памяти и их анализу;

2) Создании динамических библиотек и программ, использующие динамические библиотеки**.**

Описание программы

Программа состоит из трех частей:

1. **Вызов аллокатора (main.c)**: выделяет и освобождает определенное количество памяти при помощи переданного в качестве параметра при вызове аллокатора.

2. **Реализация алгоритма Мак-Кьюзика-Кэрелса (liballocator1.c)**: Память управляется с помощью списка свободных блоков. При выделении памяти алгоритм тщательно ищет наиболее подходящий свободный блок, чтобы минимизировать количество неиспользуемой памяти.

3. **Реализация алгоритма двойников (liballocator2.c)**: Память выделяется блоками, размеры которых являются степенями двойки. Это делает процесс управления выделением более структурированным, но может приводить к специфическим нюансам в использовании памяти.

Код программы

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <sys/mman.h>
#include <time.h>

typedef struct {
    void *(*allocator_create)(void *const memory, const size_t size);
    void (*allocator_destroy)(void *const allocator);
    void *(*allocator_alloc)(void *const allocator, const size_t size);
    void (*allocator_free)(void *const allocator, void *const memory);
} AllocatorAPI;

void* fallback_allocator_create(void *const memory, const size_t size)
    { printf("%li\n", size);
    return memory;
}

void fallback_allocator_destroy(void *const allocator) {
    if (allocator)
        printf("\n");
}

void* fallback_allocator_alloc(void *const allocator, const size_t size)
    { printf("%li\n", size);
    if (allocator)
        printf("\n");
    return mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS,
-1, 0);
}

void fallback_allocator_free(void *const allocator, void *const memory)
    { munmap(memory, 4096);
    if (allocator)
        printf("\n");
}
```

```
int main(int argc, char *argv[]) {
    void *handle = NULL;
    AllocatorAPI api;

    if (argc > 1) {
        handle = dlopen(argv[1], RTLD_LAZY);
        if (!handle) {
            fprintf(stderr, "%s\n", dlerror());
            exit(EXIT_FAILURE);
        }

        api.allocator_create = dlsym(handle, "allocator_create");
        api.allocator_destroy = dlsym(handle, "allocator_destroy");
        api.allocator_alloc = dlsym(handle, "allocator_alloc");
        api.allocator_free = dlsym(handle, "allocator_free");

        if (!api.allocator_create || !api.allocator_destroy || !api.allocator_alloc
|| !api.allocator_free) {
            fprintf(stderr, "%s\n", dlerror());
            exit(EXIT_FAILURE);
        }
    } else {
        api.allocator_create = fallback_allocator_create;
        api.allocator_destroy = fallback_allocator_destroy;
        api.allocator_alloc = fallback_allocator_alloc;
        api.allocator_free = fallback_allocator_free;
    }

    size_t memory_size = 4096;
    size_t size_data = 1327;

    void *memory = mmap(NULL, memory_size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);
    if (memory == MAP_FAILED)
        { perror("mmap");
        exit(EXIT_FAILURE);
    }
    void *allocator = api.allocator_create(memory, memory_size);

    clock_t start, end;

    double cpu_time_used;

    start = clock();
    void *ptr1 = api.allocator_alloc(allocator, size_data);
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time to allocate %li bytes: %f seconds\n", size_data, cpu_time_used);

    start = clock();
    api.allocator_free(allocator, ptr1);
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time to free %li bytes: %f seconds\n", size_data, cpu_time_used);

    api.allocator_destroy(allocator);

    munmap(memory, memory_size);
```

```
    if (handle) {
        dlclose(handle);
    }

    return 0;
}
```

liballocator1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>

#define PAGE_SIZE 4096
#define MIN_BLOCK_SIZE 16

typedef struct {
    unsigned char *memory;
    size_t total_size;
    size_t num_blocks;
    unsigned char *bitmap;
    size_t bitmap_size;
} McKusickKarelsAllocator;

static size_t calculate_bitmap_size(size_t num_blocks) {
    return (num_blocks + 7) / 8;
}

McKusickKarelsAllocator *allocator_create(size_t size) {
    size = (size + PAGE_SIZE - 1) & ~(PAGE_SIZE - 1);
    size_t num_blocks = size / MIN_BLOCK_SIZE;
    size_t bitmap_size = calculate_bitmap_size(num_blocks);

    McKusickKarelsAllocator *allocator = mmap(NULL,
sizeof(McKusickKarelsAllocator),
                                              PROT_READ | PROT_WRITE,
                                              MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

    if (allocator == MAP_FAILED) {
        perror("mmap");
        return NULL;
    }

    allocator->memory = mmap(NULL, size, PROT_READ | PROT_WRITE,
                             MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

    if (allocator->memory == MAP_FAILED)
        { perror("mmap");
        munmap(allocator, sizeof(McKusickKarelsAllocator));
        return NULL;
    }
```

```
    allocator->bitmap = mmap(NULL, bitmap_size, PROT_READ | PROT_WRITE,
                             MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

    if (allocator->bitmap == MAP_FAILED)
        { perror("mmap");
        munmap(allocator->memory, size);
        munmap(allocator, sizeof(McKusickKarelsAllocator));
        return NULL;
    }

    allocator->total_size = size;
    allocator->num_blocks = num_blocks;
    allocator->bitmap_size = bitmap_size;

    memset(allocator->bitmap, 0xFF, bitmap_size);

    return allocator;
}

void *allocator_alloc(McKusickKarelsAllocator *allocator, size_t size) {
    size_t blocks_needed = (size + MIN_BLOCK_SIZE - 1) / MIN_BLOCK_SIZE;

    for (size_t i = 0; i < allocator->num_blocks - blocks_needed + 1;) {
        size_t j;
        for (j = 0; j < blocks_needed; j++) {
            if (!(allocator->bitmap[(i + j) / 8] & (1 << ((i + j) % 8))))
                { break;
            }
        }

        if (j == blocks_needed) {
            for (j = 0; j < blocks_needed; j++) {
                allocator->bitmap[(i + j) / 8] &= ~(1 << ((i + j) % 8));
            }
            return allocator->memory + i * MIN_BLOCK_SIZE;
        }
        i += j + 1;
    }

    return NULL;
}

void allocator_free(McKusickKarelsAllocator *allocator, void *ptr, size_t size)
    { size_t blocks_to_free = (size + MIN_BLOCK_SIZE - 1) / MIN_BLOCK_SIZE;
    size_t start_index = ((unsigned char *)ptr - allocator->memory) /
MIN_BLOCK_SIZE;

    for (size_t i = start_index; i < start_index + blocks_to_free; i++) {
        allocator->bitmap[i / 8] |= (1 << (i % 8));
    }
}

void allocator_destroy(McKusickKarelsAllocator *allocator) {
    munmap(allocator->bitmap, allocator->bitmap_size);
    munmap(allocator->memory, allocator->total_size);
    munmap(allocator, sizeof(McKusickKarelsAllocator));
}
```

liballocator2.c

```
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>

#define SIZE_DATA 32

typedef struct BuddyBlock {
    size_t size;
    struct BuddyBlock *next;
    struct BuddyBlock *prev;
    int is_free;
} BuddyBlock;

typedef struct BuddyAllocator
    { void *memory;
    size_t total_size;
    BuddyBlock *free_lists[SIZE_DATA];
} BuddyAllocator;

void *allocator_create(void *const memory, const size_t size)
    { BuddyAllocator *allocator = (BuddyAllocator *)mmap(NULL,
sizeof(BuddyAllocator), PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, -1,
0);
    if (allocator == MAP_FAILED) {
        perror("mmap");
        return NULL;
    }

    allocator->memory = memory;
    allocator->total_size = size;

    for (int i = 0; i < SIZE_DATA; i++)
        allocator->free_lists[i] = NULL;

    BuddyBlock *initial_block = (BuddyBlock *)memory;
    initial_block->size = size;
    initial_block->next = NULL;
    initial_block->prev = NULL;
    initial_block->is_free = 1;

    int order = 0;
    size_t block_size = 1;
    while (block_size < size) {
        block_size <<= 1;
        order++;
    }

    initial_block->next = allocator->free_lists[order];
    if (allocator->free_lists[order]) {
        allocator->free_lists[order]->prev = initial_block;
    }
    allocator->free_lists[order] = initial_block;
```

```
    return allocator;
}

void allocator_destroy(void *const allocator) {
    if (munmap(allocator, sizeof(BuddyAllocator)) == -1)
        perror("munmap");
}

void *allocator_alloc(void *const allocator, const size_t size)
    { BuddyAllocator *buddy_allocator = (BuddyAllocator *)allocator;
    size_t block_size = 1;
    int order = 0;

    while (block_size < size) {
        block_size <<= 1;
        order++;
    }

    BuddyBlock *block = NULL;
    for (int i = order; i < SIZE_DATA; i++) {
        if (buddy_allocator->free_lists[i]) {
            block = buddy_allocator->free_lists[i];
            order = i;
            break;
        }
    }
    if (!block) return NULL;

    if (block->next)

        block->next->prev = block->prev;
    if (block->prev)
        block->prev->next = block->next;
    else
        buddy_allocator->free_lists[order] = block->next;

    while (order > 0 && block_size > size) {
        order--;
        block_size >>= 1;

        BuddyBlock *buddy = (BuddyBlock *)((char *)block + block_size);
        buddy->size = block_size;
        buddy->next = buddy_allocator->free_lists[order];
        buddy->prev = NULL;
        buddy->is_free = 1;

        if (buddy_allocator->free_lists[order])
            buddy_allocator->free_lists[order]->prev = buddy;
        buddy_allocator->free_lists[order] = buddy;
    }

    block->is_free = 0;
    return block;
}

void allocator_free(void *const allocator, void *const memory)
    { BuddyAllocator *buddy_allocator = (BuddyAllocator *)allocator;
    BuddyBlock *block = (BuddyBlock *)memory;
    block->is_free = 1;
```

```
    while (1) {
        size_t block_size = block->size;
        char *block_addr = (char *)block;
        ptrdiff_t offset = (block_addr - (char *)buddy_allocator->memory);
        char *buddy_addr = (char *)buddy_allocator->memory + (offset ^ block_size);

        if (buddy_addr < (char *)buddy_allocator->memory || buddy_addr >= (char
*)buddy_allocator->memory + buddy_allocator->total_size)
            break;

        BuddyBlock *buddy = (BuddyBlock *)buddy_addr;

        if (buddy->is_free && buddy->size == block_size)
            { if (buddy->next)
                buddy->next->prev = buddy->prev;
            if (buddy->prev)
                buddy->prev->next = buddy->next;
            else {
                int order = 0;
                size_t temp_size = block_size;
                while (temp_size >>= 1) order++;
                buddy_allocator->free_lists[order] = buddy->next;
            }

            if (block < buddy)
                { block->size <<=
                1;
            } else {
                buddy->size <<= 1;
                block = buddy;
            }
        } else {
            break;
        }
    }

    int order = 0;
    size_t block_size = block->size;
    while (block_size >>= 1) order++;

    block->next = buddy_allocator->free_lists[order];
    block->prev = NULL;
    if (buddy_allocator->free_lists[order])
        buddy_allocator->free_lists[order]->prev = block;
    buddy_allocator->free_lists[order] = block;
}
```

Протокол работы программы

Время для алгоритма двойников:

```
Time to allocate 1327 bytes: 0.000004 seconds

Time to free 1327 bytes: 0.000002 seconds
```

Для алгоритма Мак-Кьюзика-Кэрелса:

```
Time to allocate 1327 bytes: 0.000008 seconds

Time to free 1327 bytes: 0.000024 seconds
```

Сравнение алгоритмов аллокаторов: алгоритм Мак-Кьюзика-Кэрелса и алгоритм двойников

1) Фактор использования памяти

Алгоритм Мак-Кьюзика-Кэрелса:

Управление памятью осуществляется через списки свободных блоков. Для каждого размера выделяемого блока поддерживается отдельный список, из которого выбирается наиболее подходящий блок при запросе на выделение памяти.

Высокий уровень эффективности при использовании памяти путем минимизации внутренней фрагментации. Блоки подбираются так, чтобы размер максимально соответствовал запросу, что ведет к оптимальному использованию доступной памяти.

Алгоритм двойников:

Память выделяется блоками, размерами которых являются степени двойки. Это подразумевает использование системы "buddy's" (двойников), чтобы управлять памятью и объединять блоки при их освобождении.

Склонен к значительной внутренней фрагментации, поскольку выделенные блоки часто превышают необходимый размер. Однако уровень внешней фрагментации обычно ниже, так как память управляется более структурированно.

2) Скорость выделения блоков

Алгоритм Мак-Кьюзика-Кэрелса:

Может быть замедлена из-за необходимости поиска списка свободных блоков для нахождения наиболее подходящего блока.

Алгоритм двойников:

Обычно выше, так как необходимо лишь выбрать ближайший размер блока, соответствующий степени двойки.

3) Скорость освобождения блоков

Алгоритм Мак-Кьюзика-Кэрелса:

Может потребовать больше времени из-за необходимости обновления списков и потенциального объединения смежных свободных блоков.

Алгоритм двойников:

Освобождение происходит быстро, поскольку освобожденный блок просто возвращается в соответствующий список двойников, а объединение выполняется по простой логике.

4) Простота использования аллокатора

Алгоритм Мак-Кьюзика-Кэрелса:

Преимущественно прост в реализации, так как основная логика заключается в поддержке списков и управлении ими.

Алгоритм двойников:

Сложнее в реализации и использовании из-за необходимости поддержки системы двойников и более сложного управления процессом объединения блоков.

Dtruss:

```
SYSCALL(args)           = return
Time to allocate 1327 bytes: 0.000003 seconds
Time to free 1327 bytes: 0.000012 seconds
dtrace: error on enabled probe ID 1690 (ID 845: syscall::stat64:return): invalid
address (0x0) in action #11 at DIF offset 12
munmap(0x100DE8000, 0x94000)        = 0 0
munmap(0x100E7C000, 0x8000)         = 0 0
munmap(0x100E84000, 0x4000)         = 0 0
munmap(0x100E88000, 0x4000)         = 0 0
munmap(0x100E8C000, 0x5C000)        = 0 0
crossarch_trap(0x0, 0x0, 0x0)       = -1 Err#45
open(".\0", 0x100000, 0x0)          = 3 0
fcntl(0x3, 0x32, 0x16F2EB268)       = 0 0
close(0x3)        = 0 0
fsgetpath(0x16F2EB278, 0x400, 0x16F2EB258)            = 53 0
fsgetpath(0x16F2EB288, 0x400, 0x16F2EB268)            = 14 0
csrctl(0x0, 0x16F2EB68C, 0x4)       = -1 Err#1
__mac_syscall(0x18C115E36, 0x2, 0x16F2EB5E0)          = 0 0
csrctl(0x0, 0x16F2EB6AC, 0x4)       = -1 Err#1
__mac_syscall(0x18C112C8E, 0x5A, 0x16F2EB640)         = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16F2EABA8, 0x16F2EABA0, 0x18C1148DF, 0xD)
      = 0 0
sysctl([CTL_KERN, 147, 0, 0, 0, 0] (2), 0x16F2EAC58, 0x16F2EAC50, 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, 0x16F2EA731, 0x16F2EA6A0)        = 0 0
openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0)         = 6 0
fcntl(0x6, 0x32, 0x16F2EA730)       = 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
shared_region_check_np(0x16F2EAD40, 0x0, 0x0)         = 0 0
fsgetpath(0x16F2EB290, 0x400, 0x16F2EB1E8)            = 82 0
fcntl(0x8, 0x32, 0x16F2EB290)       = 0 0
close(0x8)        = 0 0
close(0x7)        = 0 0
getfsstat64(0x0, 0x0, 0x2)          = 11 0
getfsstat64(0x100F26AA0, 0x5D28, 0x2)           = 11 0
getattrlist("/\0", 0x16F2EB1C0, 0x16F2EB130)          = 0 0
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/
dyld_shared_cache_arm64e\0", 0x16F2EB520, 0x0)        = 0 0
stat64("/Users/vladislavmaleev/OperatingSystem/Lab4/src/main\0", 0x16F2EA9D0, 0x0)
      = 0 0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src/main\0", 0x0, 0x0)       = 3
0
mmap(0x0, 0x8568, 0x1, 0x40002, 0x3, 0x0)       = 0x100F68000 0
fcntl(0x3, 0x32, 0x16F2EAAE8)       = 0 0
close(0x3)        = 0 0
```

```
munmap(0x100F68000, 0x8568)          = 0 0
stat64("/Users/vladislavmaleev/OperatingSystem/Lab4/src/main\0", 0x16F2EAF40, 0x0)
      = 0 0
stat64("/usr/lib/libSystem.B.dylib\0", 0x16F2E9ED0, 0x0)           = -1 Err#2
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0",
0x16F2E9E80, 0x0)       = -1 Err#2
stat64("/usr/lib/system/libdispatch.dylib\0", 0x16F2E7AE0, 0x0)          = -1 Err#2
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/system/libdispatch.dylib\0",
0x16F2E7A90, 0x0)       = -1 Err#2
stat64("/usr/lib/system/libdispatch.dylib\0", 0x16F2E7AE0, 0x0)          = -1 Err#2
open("/dev/dtracehelper\0", 0x2, 0x0)            = 3 0
ioctl(0x3, 0x80086804, 0x16F2E9B28)        = 0 0
close(0x3)        = 0 0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src/main\0", 0x0, 0x0)       = 3
0
__mac_syscall(0x18C115E36, 0x2, 0x16F2E91B0)           = 0 0
map_with_linking_np(0x16F2E9060, 0x1, 0x16F2E9090)           = 0 0
close(0x3)        = 0 0
mprotect(0x100B18000, 0x4000, 0x1)         = 0 0
shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)         = 0 0
mprotect(0x100F24000, 0x40000, 0x1)        = 0 0
access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0)         = -1 Err#2
bsdthread_register(0x18C3FDE34, 0x18C3FDE28, 0x4000)         = 1073742303 0
getpid(0x0, 0x0, 0x0)          = 1202 0
shm_open(0x18C29BF51, 0x0, 0x636A626F)           = 3 0
fstat64(0x3, 0x16F2E9FB0, 0x0)             = 0 0
mmap(0x0, 0x4000, 0x1, 0x40001, 0x3, 0x0)        = 0x100F70000 0
close(0x3)        = 0 0
ioctl(0x2, 0x4004667A, 0x16F2EA05C)        = -1 Err#25
ioctl(0x2, 0x40487413, 0x16F2EA060)        = -1 Err#25
mprotect(0x100F7C000, 0x4000, 0x0)         = 0 0
mprotect(0x100F88000, 0x4000, 0x0)         = 0 0
mprotect(0x100F8C000, 0x4000, 0x0)         = 0 0
mprotect(0x100F98000, 0x4000, 0x0)         = 0 0
mprotect(0x100F9C000, 0x4000, 0x0)         = 0 0
mprotect(0x100FA8000, 0x4000, 0x0)         = 0 0
mprotect(0x100F74000, 0x98, 0x1)           = 0 0
mprotect(0x100F74000, 0x98, 0x3)           = 0 0
mprotect(0x100F74000, 0x98, 0x1)           = 0 0
mprotect(0x100FAC000, 0x4000, 0x1)         = 0 0
mprotect(0x100FB0000, 0x98, 0x1)           = 0 0
mprotect(0x100FB0000, 0x98, 0x3)           = 0 0
mprotect(0x100FB0000, 0x98, 0x1)           = 0 0
mprotect(0x100F74000, 0x98, 0x3)           = 0 0
mprotect(0x100F74000, 0x98, 0x1)           = 0 0
mprotect(0x100FAC000, 0x4000, 0x3)         = 0 0
mprotect(0x100FAC000, 0x4000, 0x1)         = 0 0
mprotect(0x100F24000, 0x40000, 0x3)        = 0 0
mprotect(0x100F24000, 0x40000, 0x1)        = 0 0
objc_bp_assist_cfg_np(0x18C03D400, 0x80000018001C1048, 0x0)        = -1 Err#5
issetugid(0x0, 0x0, 0x0)             = 0 0
mprotect(0x100F24000, 0x40000, 0x3)        = 0 0
getentropy(0x16F2E97C8, 0x20, 0x0)         = 0 0
mprotect(0x100F24000, 0x40000, 0x1)        = 0 0
mprotect(0x100F24000, 0x40000, 0x3)        = 0 0
```

```
mprotect(0x100F24000, 0x40000, 0x1)        = 0 0
getattrlist("/Users/vladislavmaleev/OperatingSystem/Lab4/src/main\0", 0x16F2E9F40,
0x16F2E9F58)            = 0 0
access("/Users/vladislavmaleev/OperatingSystem/Lab4/src\0", 0x4, 0x0)          = 0
0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src\0", 0x0, 0x0)            = 3
0
fstat64(0x3, 0x11D6044C0, 0x0)             = 0 0
csrctl(0x0, 0x16F2EA17C, 0x4)        = 0 0
fcntl(0x3, 0x32, 0x16F2E9E28)        = 0 0
close(0x3)        = 0 0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src/Info.plist\0", 0x0, 0x0)
= -1 Err#2
proc_info(0x2, 0x4B2, 0xD)           = 64 0
csops_audittoken(0x4B2, 0x10, 0x16F2EA1B0)             = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16F2EA508, 0x16F2EA500, 0x18F7ADD3D, 0x15)
      = 0 0
sysctl([CTL_KERN, 145, 0, 0, 0, 0] (2), 0x16F2EA598, 0x16F2EA590, 0x0, 0x0)
= 0 0
csops(0x4B2, 0x0, 0x16F2EA63C)             = 0 0
mprotect(0x100F24000, 0x40000, 0x3)        = 0 0
open("liballocator1.so\0", 0x0, 0x0)             = 3 0
fcntl(0x3, 0x32, 0x16F2EA7D8)        = 0 0
close(0x3)        = 0 0
stat64("liballocator1.so\0", 0x16F2EA350, 0x0)         = 0 0
stat64("liballocator1.so\0", 0x16F2E9D80, 0x0)         = 0 0
open("liballocator1.so\0", 0x0, 0x0)             = 3 0
mmap(0x0, 0x83B0, 0x1, 0x40002, 0x3, 0x0)        = 0x100B20000 0
fcntl(0x3, 0x32, 0x16F2E9E98)        = 0 0
close(0x3)        = 0 0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src/liballocator1.so\0", 0x0, 0x0)
      = 3 0
fcntl(0x3, 0x61, 0x16F2E9B48)        = 0 0
fcntl(0x3, 0x62, 0x16F2E9B48)        = 0 0
mmap(0x100B2C000, 0x4000, 0x5, 0x40012, 0x3, 0x0)            = 0x100B2C000 0
mmap(0x100B30000, 0x4000, 0x3, 0x40012, 0x3, 0x4000)         = 0x100B30000 0
mmap(0x100B34000, 0x4000, 0x1, 0x40012, 0x3, 0x8000)         = 0x100B34000 0
close(0x3)        = 0 0
munmap(0x100B20000, 0x83B0)          = 0 0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src/liballocator1.so\0", 0x0, 0x0)
      = 3 0
close(0x3)        = 0 0
mprotect(0x100B30000, 0x4000, 0x1)         = 0 0
mmap(0x0, 0x1000, 0x3, 0x41002, 0xFFFFFFFFFFFFFFFF, 0x0)           = 0x100B20000 0
mmap(0x0, 0x28, 0x3, 0x41002, 0xFFFFFFFFFFFFFFFF, 0x0)             = 0x100B24000 0
mmap(0x0, 0x20, 0x3, 0x41002, 0xFFFFFFFFFFFFFFFF, 0x0)             = 0x100B28000 0
getrusage(0x0, 0x16F2EB600, 0x0)           = 0 0
getrusage(0x0, 0x16F2EB600, 0x0)           = 0 0
getrlimit(0x1008, 0x16F2EB448, 0x0)        = 0 0
fstat64(0x1, 0x16F2EB440, 0x0)             = 0 0
getrusage(0x0, 0x16F2EB600, 0x0)           = 0 0
getrusage(0x0, 0x16F2EB600, 0x0)           = 0 0
munmap(0x100B28000, 0x20)            = 0 0
munmap(0x100B20000, 0x1000)          = 0 0
munmap(0x100B24000, 0x28)            = 0 0
munmap(0x100B20000, 0x1000)          = 0 0
```

```
munmap(0x100B2C000, 0xC000)          = 0 0
write_nocancel(0x1, "Time to allocate 1327 bytes: 0.000003 seconds\nTime to free
1327 bytes: 0.000012 seconds\n\0", 0x58)         = 88 0
```

```
SYSCALL(args)           = return
Time to allocate 1327 bytes: 0.000004 seconds
Time to free 1327 bytes: 0.000002 seconds
dtrace: error on enabled probe ID 1690 (ID 845: syscall::stat64:return): invalid
address (0x0) in action #11 at DIF offset 12
munmap(0x100B0C000, 0x94000)         = 0 0
munmap(0x100BA0000, 0x8000)          = 0 0
munmap(0x100BA8000, 0x4000)          = 0 0
munmap(0x100BAC000, 0x4000)          = 0 0
munmap(0x100BB0000, 0x5C000)         = 0 0
crossarch_trap(0x0, 0x0, 0x0)        = -1 Err#45
open(".\0", 0x100000, 0x0)           = 3 0
fcntl(0x3, 0x32, 0x16F617268)        = 0 0
close(0x3)        = 0 0
fsgetpath(0x16F617278, 0x400, 0x16F617258)             = 53 0
fsgetpath(0x16F617288, 0x400, 0x16F617268)             = 14 0
csrctl(0x0, 0x16F61768C, 0x4)        = -1 Err#1
__mac_syscall(0x18C115E36, 0x2, 0x16F6175E0)           = 0 0
csrctl(0x0, 0x16F6176AC, 0x4)        = -1 Err#1
__mac_syscall(0x18C112C8E, 0x5A, 0x16F617640)          = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16F616BA8, 0x16F616BA0, 0x18C1148DF, 0xD)
      = 0 0
sysctl([CTL_KERN, 147, 0, 0, 0, 0] (2), 0x16F616C58, 0x16F616C50, 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, 0x16F616731, 0x16F6166A0)         = 0 0
openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0)         = 6 0
fcntl(0x6, 0x32, 0x16F616730)        = 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
shared_region_check_np(0x16F616D40, 0x0, 0x0)          = 0 0
fsgetpath(0x16F617290, 0x400, 0x16F6171E8)             = 82 0
fcntl(0x8, 0x32, 0x16F617290)        = 0 0
close(0x8)        = 0 0
close(0x7)        = 0 0
getfsstat64(0x0, 0x0, 0x2)           = 11 0
getfsstat64(0x100BFAAA0, 0x5D28, 0x2)            = 11 0
getattrlist("/\0", 0x16F6171C0, 0x16F617130)           = 0 0
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/
dyld_shared_cache_arm64e\0", 0x16F617520, 0x0)         = 0 0
stat64("/Users/vladislavmaleev/OperatingSystem/Lab4/src/main\0", 0x16F6169D0, 0x0)
      = 0 0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src/main\0", 0x0, 0x0)       = 3
0
mmap(0x0, 0x8568, 0x1, 0x40002, 0x3, 0x0)        = 0x100C3C000 0
fcntl(0x3, 0x32, 0x16F616AE8)        = 0 0
close(0x3)        = 0 0
munmap(0x100C3C000, 0x8568)          = 0 0
stat64("/Users/vladislavmaleev/OperatingSystem/Lab4/src/main\0", 0x16F616F40, 0x0)
      = 0 0
```

```
stat64("/usr/lib/libSystem.B.dylib\0", 0x16F615ED0, 0x0)           = -1 Err#2
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0",
0x16F615E80, 0x0)       = -1 Err#2
stat64("/usr/lib/system/libdispatch.dylib\0", 0x16F613AE0, 0x0)          = -1 Err#2
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/system/libdispatch.dylib\0",
0x16F613A90, 0x0)       = -1 Err#2
stat64("/usr/lib/system/libdispatch.dylib\0", 0x16F613AE0, 0x0)          = -1 Err#2
open("/dev/dtracehelper\0", 0x2, 0x0)            = 3 0
ioctl(0x3, 0x80086804, 0x16F615B28)       = 0 0
close(0x3)        = 0 0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src/main\0", 0x0, 0x0)       = 3
0
__mac_syscall(0x18C115E36, 0x2, 0x16F6151B0)           = 0 0
map_with_linking_np(0x16F615060, 0x1, 0x16F615090)           = 0 0
close(0x3)        = 0 0
mprotect(0x1007EC000, 0x4000, 0x1)         = 0 0
shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)         = 0 0
mprotect(0x100BF8000, 0x40000, 0x1)        = 0 0
access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0)         = -1 Err#2
bsdthread_register(0x18C3FDE34, 0x18C3FDE28, 0x4000)         = 1073742303 0
getpid(0x0, 0x0, 0x0)          = 1462 0
shm_open(0x18C29BF51, 0x0, 0x636A626F)           = 3 0
fstat64(0x3, 0x16F615FB0, 0x0)             = 0 0
mmap(0x0, 0x4000, 0x1, 0x40001, 0x3, 0x0)        = 0x100C44000 0
close(0x3)        = 0 0
ioctl(0x2, 0x4004667A, 0x16F61605C)        = -1 Err#25
ioctl(0x2, 0x40487413, 0x16F616060)        = -1 Err#25
mprotect(0x100C50000, 0x4000, 0x0)         = 0 0
mprotect(0x100C5C000, 0x4000, 0x0)         = 0 0
mprotect(0x100C60000, 0x4000, 0x0)         = 0 0
mprotect(0x100C6C000, 0x4000, 0x0)         = 0 0
mprotect(0x100C70000, 0x4000, 0x0)         = 0 0
mprotect(0x100C7C000, 0x4000, 0x0)         = 0 0
mprotect(0x100C48000, 0x98, 0x1)           = 0 0
mprotect(0x100C48000, 0x98, 0x3)           = 0 0
mprotect(0x100C48000, 0x98, 0x1)           = 0 0
mprotect(0x100C80000, 0x4000, 0x1)         = 0 0
mprotect(0x100C84000, 0x98, 0x1)           = 0 0
mprotect(0x100C84000, 0x98, 0x3)           = 0 0
mprotect(0x100C84000, 0x98, 0x1)           = 0 0
mprotect(0x100C48000, 0x98, 0x3)           = 0 0
mprotect(0x100C48000, 0x98, 0x1)           = 0 0
mprotect(0x100C80000, 0x4000, 0x3)         = 0 0
mprotect(0x100C80000, 0x4000, 0x1)         = 0 0
mprotect(0x100BF8000, 0x40000, 0x3)        = 0 0
mprotect(0x100BF8000, 0x40000, 0x1)        = 0 0
objc_bp_assist_cfg_np(0x18C03D400, 0x80000018001C1048, 0x0)        = -1 Err#5
issetugid(0x0, 0x0, 0x0)             = 0 0
mprotect(0x100BF8000, 0x40000, 0x3)        = 0 0
getentropy(0x16F6157C8, 0x20, 0x0)         = 0 0
mprotect(0x100BF8000, 0x40000, 0x1)        = 0 0
mprotect(0x100BF8000, 0x40000, 0x3)        = 0 0
mprotect(0x100BF8000, 0x40000, 0x1)        = 0 0
getattrlist("/Users/vladislavmaleev/OperatingSystem/Lab4/src/main\0", 0x16F615F40,
0x16F615F58)            = 0 0
```

```
access("/Users/vladislavmaleev/OperatingSystem/Lab4/src\0", 0x4, 0x0)          = 0
0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src\0", 0x0, 0x0)            = 3
0
fstat64(0x3, 0x1586044C0, 0x0)             = 0 0
csrctl(0x0, 0x16F61617C, 0x4)        = 0 0
fcntl(0x3, 0x32, 0x16F615E28)        = 0 0
close(0x3)         = 0 0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src/Info.plist\0", 0x0, 0x0)
= -1 Err#2
proc_info(0x2, 0x5B6, 0xD)           = 64 0
csops_audittoken(0x5B6, 0x10, 0x16F6161B0)             = 0 0
sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16F616508, 0x16F616500, 0x18F7ADD3D, 0x15)
      = 0 0
sysctl([CTL_KERN, 145, 0, 0, 0, 0] (2), 0x16F616598, 0x16F616590, 0x0, 0x0)
= 0 0
csops(0x5B6, 0x0, 0x16F61663C)             = 0 0
mprotect(0x100BF8000, 0x40000, 0x3)        = 0 0
open("liballocator2.so\0", 0x0, 0x0)              = 3 0
fcntl(0x3, 0x32, 0x16F6167D8)        = 0 0
close(0x3)         = 0 0
stat64("liballocator2.so\0", 0x16F616350, 0x0)         = 0 0
stat64("liballocator2.so\0", 0x16F615D80, 0x0)         = 0 0
open("liballocator2.so\0", 0x0, 0x0)              = 3 0
mmap(0x0, 0x8350, 0x1, 0x40002, 0x3, 0x0)         = 0x1007F4000 0
fcntl(0x3, 0x32, 0x16F615E98)        = 0 0
close(0x3)         = 0 0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src/liballocator2.so\0", 0x0, 0x0)
      = 3 0
fcntl(0x3, 0x61, 0x16F615B48)        = 0 0
fcntl(0x3, 0x62, 0x16F615B48)        = 0 0
mmap(0x100800000, 0x4000, 0x5, 0x40012, 0x3, 0x0)             = 0x100800000 0
mmap(0x100804000, 0x4000, 0x3, 0x40012, 0x3, 0x4000)          = 0x100804000 0
mmap(0x100808000, 0x4000, 0x1, 0x40012, 0x3, 0x8000)          = 0x100808000 0
close(0x3)         = 0 0
munmap(0x1007F4000, 0x8350)          = 0 0
open("/Users/vladislavmaleev/OperatingSystem/Lab4/src/liballocator2.so\0", 0x0, 0x0)
      = 3 0
close(0x3)         = 0 0
mprotect(0x100804000, 0x4000, 0x1)         = 0 0
mmap(0x0, 0x1000, 0x3, 0x41002, 0xFFFFFFFFFFFFFFFF, 0x0)            = 0x1007F4000 0
mmap(0x0, 0x110, 0x3, 0x41002, 0xFFFFFFFFFFFFFFFF, 0x0)             = 0x1007F8000 0
getrusage(0x0, 0x16F617600, 0x0)           = 0 0
getrusage(0x0, 0x16F617600, 0x0)           = 0 0
getrlimit(0x1008, 0x16F617448, 0x0)        = 0 0
fstat64(0x1, 0x16F617440, 0x0)             = 0 0
getrusage(0x0, 0x16F617600, 0x0)           = 0 0
getrusage(0x0, 0x16F617600, 0x0)           = 0 0
munmap(0x1007F8000, 0x110)           = 0 0
munmap(0x1007F4000, 0x1000)          = 0 0
munmap(0x100800000, 0xC000)          = 0 0
write_nocancel(0x1, "Time to allocate 1327 bytes: 0.000004 seconds\nTime to free
1327 bytes: 0.000002 seconds\n\0", 0x58)         = 88 0
```

Вывод

В рамках лабораторной работы была разработана программа, демонстрирующая работу аллокатора передаваемого в качестве аргумента при вызове программы. Был проведен всесторонний анализ работы двух разных аллокаторов.