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-210Б-23

Студент: Горячев А.В.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 07.01.25

Москва, 2024

**Постановка задачи**

**Вариант 5.**

Реализовать два алгоритма аллокации памяти и сравнить их по ряду характеристик. Виды аллокаторов:

1. Алгоритм Мак-Кьюзика-Кэрелса и алгоритм двойников.
2. Алгоритм двойников (buddy allocator).

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* mmap() – выделение анонимной памяти (файловый дескриптор -1)
* munmap() – удаление отображения памяти
* dlopen() – загрузка динамической библиотеки
* dlsym() – получение указателя на символ в библиотеке
* dlclose() – выгрузка динамической библиотеки

**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:

1.1. Аллокатор получает блок памяти размером, достаточным для управления списками.

1.2. Каждый список представляет собой блоки определенного размера.

1.3. Инициализация списков происходит путем установки указателей на NULL.

2. Выделение памяти:

2.1. При запросе памяти, аллокатор округляет размер запроса до ближайшего подходящего размера блока.

2.2. Проверяется соответствующий список свободных блоков.

2.3. Если в списке есть свободный блок, он используется для удовлетворения запроса.

2.4. Если в списке нет свободного блока, выделяется новый блок из общей памяти аллокатора.

2.5. В случае выделения памяти, блок удаляется из списка свободных блоков, а указатель на выделенную память возвращается в программу.

3. Освобождение памяти:

3.1. Если блок памяти не нужен, то он освобождается и переходит в соответствующий список свободных блоков.

3.2. Блок добавляется в начало списка, чтобы его можно было использовать для будущих запросов.

**Алгоритм двойников (buddy allocator):**

Алгоритм аллокации методом двойников (Buddy Allocator) – это усовершенствованная версия аллокатора на основе степеней двойки. Его основной особенностью является возможность объединения ("слияния") двух свободных блоков одинакового размера, если они являются "соседними" (buddy). Такой подход позволяет эффективно управлять памятью, уменьшая внешнюю фрагментацию.

Данный алгоритм обеспечивает не только разбиение больших блоков на более мелкие для выделения блоков, наиболее близких к размеру запрашиваемой памяти (для уменьшения внутренней фрагментации), но также реализует и объединение двух соседних свободных блоков одного размера в один большой блок.

Инициализация аллокатора (create):

1. Аллокатору выделяется большой блок памяти (через mmap), который делится на блоки степеней двойки.
2. Для управления памятью создаются списки свободных блоков (free\_lists), где каждый индекс отвечает за размер блоков, равный 2^i байт.
3. Начальный блок (самый большой) добавляется в соответствующий список свободных блоков.

Алгоритм выделения памяти (alloc):

1. Пользователь запрашивает блок памяти.
2. Размер запрашиваемого блока округляется до ближайшей степени двойки.
3. Аллокатор проверяет, есть ли свободный блок нужного размера в free\_lists.
4. Если подходящий блок найден, он выделяется.
5. Если свободного блока нужного размера нет, аллокатор ищет больший блок:
   1. Если найден блок большего размера, он разделяется на две части:
      1. Первый блок выделяется пользователю.
      2. Второй блок (его buddy) добавляется в список свободных блоков меньшего размера.
6. Процесс повторяется до тех пор, пока не будет найден или создан блок нужного размера.

Алгоритм освобождения памяти (free):

1. Пользователь освобождает блок памяти.
2. Аллокатор добавляет освобожденный блок в список свободных блоков соответствующего размера.
3. Затем аллокатор проверяет, можно ли объединить этот блок с его "buddy":
   1. Если блок и его buddy свободны, они объединяются в один блок большего размера.
   2. Новый блок помещается в список свободных блоков большего размера.Процесс объединения продолжается до тех пор, пока блок не станет максимально возможного размера или его buddy занят.

С одной стороны, объединение блоков меньшего размера в большой блок является преимуществом buddy allocator’а по сравнению с аллокатором степени двойки. Но уменьшение фрагментации происходит за счет усложнения алгоритма и влияет на эффективность.

**Тестирование**

При тестировании было замечено, что аллокаторы по скорости работают одинаково на удалении одного блока памяти. Алгоритм двойников работает чуть дольше при выделении одного блока, т.к. приходится проходиться по всем уровням и делить большие блок на более маленькие, но он работает чуть быстрее на серии из 200 аллокаций и удалений, т.к. блоки выделяются и освобождаются без необходимости поиска подходящего размера в списках.

Сравнение использования аллокаторов по времени, в секундах:

|  |  |  |  |
| --- | --- | --- | --- |
| **Вид аллокатора** | **Аллокация**  **(2000 байт)** | **Удаление**  **(2000 байт)** | **Серия из 200 аллокаций**  **и освобождений** |
| Mac-Cusic allocator | 0. 000004 | 0. 000001 | 0.000113 |
| buddy allocator | 0.000008 | 0. 000001 | 0.000053 |

**Код программы**

**Аллокатор** **Мак-Кьюзика-Кэрелса:**

**carels\_allocator.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

#define MAX\_BLOCK\_SIZE 2048

#define NUM\_PAGES 8  // по 16, 32, 64, .. , 2048

typedef struct FreeBlock {

    struct FreeBlock \*next;

} FreeBlock;

typedef struct {

    unsigned char \*memory;

    size\_t total\_size;

    FreeBlock \*free\_lists[NUM\_PAGES];

} Allocator;

static size\_t round\_up\_to\_power\_of\_two(size\_t size) {

    size\_t rounded = MIN\_BLOCK\_SIZE;

    while (rounded < size && rounded <= MAX\_BLOCK\_SIZE) {

        rounded <<= 1;

    }

    return rounded;

}

static int get\_size\_index(size\_t size) {

    size\_t block\_size = MIN\_BLOCK\_SIZE;

    int index = 0;

    while (block\_size < size && block\_size <= MAX\_BLOCK\_SIZE) {

        block\_size <<= 1;

        index++;

    }

    return index;

}

Allocator \*allocator\_create(void \*const memory, size\_t size) {

    size = (size + PAGE\_SIZE - 1) & ~(PAGE\_SIZE - 1); // Round up to page size

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

    allocator->memory = (unsigned char \*)memory + sizeof(Allocator);

    allocator->total\_size = size - sizeof(Allocator);

    memset(allocator->free\_lists, 0, sizeof(allocator->free\_lists));

    return allocator;

}

void \*allocator\_alloc(Allocator \*allocator, size\_t size) {

    if (size < 0 || size > MAX\_BLOCK\_SIZE) {

        return NULL;

    }

    size = round\_up\_to\_power\_of\_two(size);

    int index = get\_size\_index(size);

    // если в странице есть свободные блоки, то убираем его из списка свободных и возвращаем его

    if (allocator->free\_lists[index]) {

        FreeBlock \*block = allocator->free\_lists[index];

        allocator->free\_lists[index] = block->next;

        return (void \*)block;

    }

    // если свободных блоков нет, делаем новую страницу

    size\_t blocks\_per\_page = PAGE\_SIZE / size;

    unsigned char \*new\_page = mmap(NULL, PAGE\_SIZE, PROT\_READ | PROT\_WRITE,

                                   MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);

    if (new\_page == MAP\_FAILED) {

        perror("mmap");

        return NULL;

    }

    // разбиваем новую страницу на блоки

    for (size\_t i = 1; i < blocks\_per\_page; i++) {

        FreeBlock \*block = (FreeBlock \*)(new\_page + i \* size);

        block->next = allocator->free\_lists[index];

        allocator->free\_lists[index] = block;

    }

    return (void \*)new\_page;

}

void allocator\_free(Allocator \*allocator, void \*memory) {

    unsigned char \*block\_start = (unsigned char \*)memory;

    size\_t offset = block\_start - allocator->memory;

    // определение размера блока по его выравниванию

    size\_t size = MIN\_BLOCK\_SIZE;

    int index = 0;

    while (index < NUM\_PAGES && (offset % size != 0 || size > MAX\_BLOCK\_SIZE)) {

        size <<= 1;

        index++;

    }

    if (index >= NUM\_PAGES || size > MAX\_BLOCK\_SIZE) {

        fprintf(stderr, "Error: the specified pointer does not match the selected block.\n");

        return;

    }

    FreeBlock \*block = (FreeBlock \*)memory;

    block->next = allocator->free\_lists[index];

    allocator->free\_lists[index] = block;

}

void allocator\_destroy(Allocator \*allocator) {

    if (!allocator) return;

    munmap(allocator->memory, allocator->total\_size);

    munmap(allocator, sizeof(Allocator));

}

**Buddy allocator:**

**buddy\_allocator.c:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <stdbool.h>

#include <sys/mman.h>

#include <unistd.h>

#define MAX\_LEVELS 16

typedef struct Block {

    bool is\_free;

    struct Block\* next;

    size\_t level;

} Block;

typedef struct Allocator {

    void\* memory;

    size\_t total\_size;

    size\_t min\_block\_size;

    Block\* free\_lists[MAX\_LEVELS];

} Allocator;

size\_t get\_level(size\_t size, size\_t total\_size, size\_t min\_block\_size) {

    size\_t level = 0;

    while (min\_block\_size \* (1 << level) < size && total\_size >= min\_block\_size \* (1 << (level + 1))) {

        level++;

    }

    return level;

}

Allocator\* allocator\_create(void\* const memory, const size\_t size) {

    if (size < 0) return NULL;

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

    allocator->memory = (unsigned char \*)memory + sizeof(Allocator);

    allocator->total\_size = size;

    allocator->min\_block\_size = 16;

    for (int i = 0; i < MAX\_LEVELS; i++) {

        allocator->free\_lists[i] = NULL;

    }

    Block\* initial\_block = (Block\*)allocator->memory;

    initial\_block->is\_free = true;

    initial\_block->next = NULL;

    allocator->free\_lists[MAX\_LEVELS - 1] = initial\_block;

    return allocator;

}

void allocator\_destroy(Allocator\* const allocator) {

    if (!allocator) return;

    munmap(allocator->memory, allocator->total\_size);

    munmap(allocator, sizeof(Allocator));

}

void split\_block(Allocator\* allocator, size\_t level) {

    if (!allocator || level >= MAX\_LEVELS || !allocator->free\_lists[level]) return;

    Block\* block = allocator->free\_lists[level];

    allocator->free\_lists[level] = block->next;

    size\_t block\_size = allocator->min\_block\_size \* (1 << level);

    Block\* buddy = (Block\*)((char\*)block + block\_size / 2);

    block->is\_free = true;

    buddy->is\_free = true;

    buddy->next = allocator->free\_lists[level - 1];

    allocator->free\_lists[level - 1] = buddy;

    block->next = allocator->free\_lists[level - 1];

    allocator->free\_lists[level - 1] = block;

}

void\* allocator\_alloc(Allocator\* const allocator, const size\_t size) {

    if (!allocator || size == 0 || size > allocator->total\_size) return NULL;

    size\_t level = get\_level(size, allocator->total\_size, allocator->min\_block\_size);

    for (size\_t i = level; i < MAX\_LEVELS; i++) {

        if (allocator->free\_lists[i]) {

            while (i > level) {

                split\_block(allocator, i);

                i--;

            }

            Block\* block = allocator->free\_lists[level];

            allocator->free\_lists[level] = block->next;

            block->is\_free = false;

            return (void\*)block;

        }

    }

    return NULL;

}

void merge\_buddies(Allocator\* allocator, Block\* block) {

    if (!allocator || !block) return;

    while (block->level < MAX\_LEVELS - 1) {

        size\_t block\_size = allocator->min\_block\_size \* (1 << block->level);

        size\_t block\_offset = (char\*)block - (char\*)allocator->memory;

        size\_t buddy\_offset = block\_offset ^ block\_size;

        Block\* buddy = (Block\*)((char\*)allocator->memory + buddy\_offset);

        if ((char\*)buddy < (char\*)allocator->memory ||

            (char\*)buddy >= (char\*)allocator->memory + allocator->total\_size) {

            break;

        }

        if (!buddy->is\_free) {

            break;

        }

        Block\*\* list = &allocator->free\_lists[buddy->level];

        while (\*list && \*list != buddy) {

            list = &(\*list)->next;

        }

        if (\*list == buddy) {

            \*list = buddy->next;

        }

        if ((char\*)block > (char\*)buddy) {

            block = buddy;

        }

        block->level++;

        block->is\_free = true;

    }

    block->next = allocator->free\_lists[block->level];

    allocator->free\_lists[block->level] = block;

}

void allocator\_free(Allocator\* const allocator, void\* const memory) {

    if (!allocator || !memory) return;

    if ((char\*)memory < (char\*)allocator->memory ||

        (char\*)memory >= (char\*)allocator->memory + allocator->total\_size) {

        return;

    }

    Block\* block = (Block\*)memory;

    block->is\_free = true;

    merge\_buddies(allocator, block);

}

**main.c:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dlfcn.h>

#include <sys/mman.h>

#include <time.h>

typedef struct Allocator Allocator;

typedef Allocator\* (\*allocator\_create\_t)(void\* const memory, const size\_t size);

typedef void (\*allocator\_destroy\_t)(Allocator\* const allocator);

typedef void\* (\*allocator\_alloc\_t)(Allocator\* const allocator, const size\_t size);

typedef void (\*allocator\_free\_t)(Allocator\* const allocator, void\* const memory);

Allocator\* fallback\_allocator\_create(void\* const memory, const size\_t size) {

    printf("Using fallback allocator\n");

    return (Allocator\*)memory;

}

void fallback\_allocator\_destroy(Allocator\* const allocator) {

    printf("Fallback allocator destroyed\n");

}

void\* fallback\_allocator\_alloc(Allocator\* const allocator, const size\_t size) {

    printf("Fallback allocator allocating %zu bytes\n", size);

    return mmap(NULL, size, PROT\_READ | PROT\_WRITE, MAP\_ANONYMOUS | MAP\_PRIVATE, -1, 0);

}

void fallback\_allocator\_free(Allocator\* const allocator, void\* const memory) {

    printf("Fallback allocator freeing memory\n");

    munmap(memory, 0);

}

#define ARRAY\_SIZE 200

void get\_range(int index, int \*start, int \*end, int min\_value, int max\_value) {

    \*start = (1 << (4 + index));

    \*end = (1 << (5 + index)) - 1;

    if (\*end > max\_value) \*end = max\_value;

}

void fill\_array(int \*array, int size, int num\_ranges, int min\_value, int max\_value) {

    int numbers\_per\_range = size / num\_ranges;

    int index = 0;

    for (int range = 0; range < num\_ranges; range++) {

        int start, end;

        get\_range(range, &start, &end, min\_value, max\_value);

        for (int i = 0; i < numbers\_per\_range; i++) {

            array[index++] = start + rand() % (end - start + 1);

        }

    }

    while (index < size) {

        int start, end;

        get\_range(rand() % num\_ranges, &start, &end, min\_value, max\_value);

        array[index++] = start + rand() % (end - start + 1);

    }

    for (int i = size - 1; i > 0; i--) {

        int j = rand() % (i + 1);

        int temp = array[i];

        array[i] = array[j];

        array[j] = temp;

    }

}

double calculate\_time(struct timespec start, struct timespec end) {

    return (end.tv\_sec - start.tv\_sec) + (end.tv\_nsec - start.tv\_nsec) / 1e9;

}

int main(int argc, char\* argv[]) {

    allocator\_create\_t allocator\_create = fallback\_allocator\_create;

    allocator\_destroy\_t allocator\_destroy = fallback\_allocator\_destroy;

    allocator\_alloc\_t allocator\_alloc = fallback\_allocator\_alloc;

    allocator\_free\_t allocator\_free = fallback\_allocator\_free;

    void \*library\_handle = NULL;

    if (argc > 1) {

        const char\* library\_path = argv[1];

        library\_handle = dlopen(library\_path, RTLD\_LAZY);

        if (library\_handle) {

            printf("Loaded library: %s\n", library\_path);

            allocator\_create = (allocator\_create\_t)dlsym(library\_handle, "allocator\_create");

            allocator\_destroy = (allocator\_destroy\_t)dlsym(library\_handle, "allocator\_destroy");

            allocator\_alloc = (allocator\_alloc\_t)dlsym(library\_handle, "allocator\_alloc");

            allocator\_free = (allocator\_free\_t)dlsym(library\_handle, "allocator\_free");

            if (!allocator\_create || !allocator\_destroy || !allocator\_alloc || !allocator\_free) {

                printf("Failed to load symbols from library. Using fallback allocator.\n");

                allocator\_create = fallback\_allocator\_create;

                allocator\_destroy = fallback\_allocator\_destroy;

                allocator\_alloc = fallback\_allocator\_alloc;

                allocator\_free = fallback\_allocator\_free;

            }

        } else {

            printf("Failed to load library: %s. Using fallback allocator.\n", library\_path);

        }

    } else {

        printf("No library path provided. Using fallback allocator.\n");

    }

    const size\_t memory\_size = 1024 \* 1024; // 1 мб

    void\* memory = mmap(NULL, memory\_size, PROT\_READ | PROT\_WRITE, MAP\_ANONYMOUS | MAP\_PRIVATE, -1, 0);

    if (memory == MAP\_FAILED) {

        perror("mmap failed");

        return 1;

    }

    Allocator\* allocator = allocator\_create(memory, memory\_size);

    if (!allocator) {

        printf("Failed to create allocator\n");

        munmap(memory, memory\_size);

        return 1;

    }

    // TEST

    // \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

    struct timespec start, end;

    double time\_used;

    clock\_gettime(CLOCK\_MONOTONIC, &start);

    void\* ptr = allocator\_alloc(allocator, 2000);

    clock\_gettime(CLOCK\_MONOTONIC, &end);

    time\_used = calculate\_time(start, end);

    if (ptr) {

        printf("Time of allocation: %f seconds\n", time\_used);

    } else {

        printf("Allocation failed\n");

    }

    // \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

    clock\_gettime(CLOCK\_MONOTONIC, &start);

    allocator\_free(allocator, ptr);

    clock\_gettime(CLOCK\_MONOTONIC, &end);

    time\_used = calculate\_time(start, end);

    printf("Time of freeing: %f seconds\n", time\_used);

    // \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

        int sizes[ARRAY\_SIZE];

        srand(42);

        fill\_array(sizes, ARRAY\_SIZE, 8, 16, 2048);

        void \*test[ARRAY\_SIZE];

        clock\_gettime(CLOCK\_MONOTONIC, &start);

        for (int i = 0; i < ARRAY\_SIZE; i++) {

            test[i] = allocator\_alloc(allocator, sizes[i]);

            if (!test[i]) {

                fprintf(stderr, "Error: allocation error.\n");

                return 1;

            }

        }

        for (int i = 0; i < ARRAY\_SIZE; i++) {

            allocator\_free(allocator, test[i]);

        }

        clock\_gettime(CLOCK\_MONOTONIC, &end);

        time\_used = calculate\_time(start, end);

        printf("Time of allocations and freeings: %f seconds\n", time\_used);

    // \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

    allocator\_destroy(allocator);

    munmap(memory, memory\_size);

    if (library\_handle) {

        dlclose(library\_handle);

    }

}

**Протокол работы программы**

execve("./main", ["./main", "./buddy\_allocator\_lib.so"], 0x7ffd712e8f68 /\* 35 vars \*/) = 0

brk(NULL) = 0x3e9ee000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe2c8cd4f0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f8527155000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/glibc-hwcaps/x86-64-v3/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/glibc-hwcaps/x86-64-v3", 0x7ffe2c8cc710, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/glibc-hwcaps/x86-64-v2/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/glibc-hwcaps/x86-64-v2", 0x7ffe2c8cc710, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64/x86\_64", 0x7ffe2c8cc710, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64", 0x7ffe2c8cc710, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/x86\_64", 0x7ffe2c8cc710, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/tls", 0x7ffe2c8cc710, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64/x86\_64", 0x7ffe2c8cc710, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64", 0x7ffe2c8cc710, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/x86\_64", 0x7ffe2c8cc710, 0) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/local/cuda-11.0/lib64", {st\_mode=S\_IFDIR|0755, st\_size=4096, ...}, 0) = 0

openat(AT\_FDCWD, "glibc-hwcaps/x86-64-v3/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "glibc-hwcaps/x86-64-v2/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "tls/x86\_64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "tls/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "tls/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "tls/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "x86\_64/x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "x86\_64/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=37071, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 37071, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f852714b000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...}, AT\_EMPTY\_PATH) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f8526f22000

mprotect(0x7f8526f4a000, 2023424, PROT\_NONE) = 0

mmap(0x7f8526f4a000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f8526f4a000

mmap(0x7f85270df000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f85270df000

mmap(0x7f8527138000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f8527138000

mmap(0x7f852713e000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f852713e000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f8526f1f000

arch\_prctl(ARCH\_SET\_FS, 0x7f8526f1f740) = 0

set\_tid\_address(0x7f8526f1fa10) = 100061

set\_robust\_list(0x7f8526f1fa20, 24) = 0

rseq(0x7f8526f200e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f8527138000, 16384, PROT\_READ) = 0

mprotect(0x403000, 4096, PROT\_READ) = 0

mprotect(0x7f852718f000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7f852714b000, 37071) = 0

getrandom("\x4e\xf5\xbf\x81\x38\x93\x25\x2e", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x3e9ee000

brk(0x3ea0f000) = 0x3ea0f000

openat(AT\_FDCWD, "./buddy\_allocator\_lib.so", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=15576, ...}, AT\_EMPTY\_PATH) = 0

getcwd("/home/alexandr/sem3/osi/lab4", 128) = 29

mmap(NULL, 16456, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f8527150000

mmap(0x7f8527151000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x7f8527151000

mmap(0x7f8527152000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f8527152000

mmap(0x7f8527153000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f8527153000

close(3) = 0

mprotect(0x7f8527153000, 4096, PROT\_READ) = 0

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

write(1, "Loaded library: ./buddy\_allocato"..., 41Loaded library: ./buddy\_allocator\_lib.so

) = 41

mmap(NULL, 1048576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f8526e1f000

write(1, "Time of allocation: 0.000009 sec"..., 37Time of allocation: 0.000009 seconds

) = 37

write(1, "Time of freeing: 0.000001 second"..., 34Time of freeing: 0.000001 seconds

) = 34

write(1, "Time of allocations and freeings"..., 51Time of allocations and freeings: 0.000052 seconds

) = 51

munmap(0x7f8526e1f098, 1048576) = -1 EINVAL (Invalid argument)

munmap(0x7f8526e1f000, 152) = 0

munmap(0x7f8526e1f000, 1048576) = 0

munmap(0x7f8527150000, 16456) = 0

exit\_group(0) = ?

+++ exited with 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, подключаемые через динамические библиотеки. Я сталкивался с трудностями осознания, как работает адресная арифметика в алгоритме двойников, какие адреса занимают близнецы. Также освежил в памяти битовые операции. Интересно было рассуждать, какие структуры хранения выбрать для реализации алгоритма для удобного оперирования над ними.