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: Косов В.В. Преподаватель: Бахарев В.Д. Оценка:

Дата: 06.01.25

Москва, 2024

# Постановка задачи

**Вариант 9.**

Требуется создать две динамические библиотеки, реализующие два аллокатора памяти: списки свободных блоков (наиболее подходящее) и алгоритм двойников;

# Общий метод и алгоритм решения

Использованные системные вызовы:

* void \* mmap(void \*start, size\_t length, int prot , int flags, int fd, off\_t offset); – отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым дескриптором fd, в память, начиная с адреса start.
* int munmap(void \*start, size\_t length); – удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти".
* ssize\_t write(int fd, const void \*buf, size\_t count); - Записывает данные в файл или файловый дескриптор.
* void \*dlopen(const char \*filename, int flag); - Открывает динамическую библиотеку.
* void \*dlsym(void \*handle, const char \*symbol); - Извлекает адрес функции или переменной symbol из открытой библиотеки handle.
* int dlclose(void \*handle); - Закрывает динамическую библиотеку handle.

# Описание программы

## main.c

Открывает динамические библиотеки, извлекает нужные функции из них. Если функции не найдены, то используются функции-заглушки, чтобы избежать ошибок во время исполнения программы.

## library.h

Подключает сторонние библиотеки и объявляет фукнции, которые реализовывает аллокатор

## buddys.c

Файл в котором реализована логика работы аллокатора на методе двойников.

**Инициализация**:

* Память делится на блоки, размеры которых являются степенями двойки (32, 64, 128 и т. д.).
* Вся память представлена как один большой блок (наивысшая степень двойки).

**Разделение блоков**:

* Если запрашиваемый размер меньше текущего блока, он делится пополам, образуя два "двойника". Этот процесс повторяется до тех пор, пока не будет найден блок подходящего размера.

**Выделение памяти**:

* Для каждого запроса выбирается минимальный блок подходящего размера.
* Аллокатор использует битовую карту (*bitmap*), которая хранится в начале выделенной памяти, для отслеживания статуса каждого блока (занят/свободен).

**Освобождение памяти**:

* Освобожденный блок помечается как свободный в битовой карте.
* Если его "двойник" также свободен, они объединяются в один более крупный блок. Процесс повторяется рекурсивно для более крупных блоков.

**Ограничение размеров**:

* Запросы округляются до ближайшей степени двойки. Например, запрос 50 байт преобразуется в 64 байта.

**Граничные условия**:

* Если запрос превышает размер доступной памяти , то выделение завершается с ошибкой.

## freebloks.c

Файл в котором реализована логика работы аллокатора на списке свободных блоков.

**Инициализация**:

* При создании аллокатора вся доступная память разбивается на один большой свободный блок.

**Список свободных блоков**:

* Используется односвязный список для отслеживания всех свободных блоков.
* Размеры блоков могут быть произвольными, что позволяет гибко использовать память.

**Выделение памяти**:

* Происходит поиск наименьшего свободного блока, подходящего под запрос.
* Если найденный блок больше, чем необходимо, он разделяется на два: первый блок удовлетворяет запрос, второй остаётся в списке свободных блоков.

**Освобождение памяти**:

* Освобожденный блок добавляется обратно в список свободных.
* Если соседние блоки также свободны, они объединяются в один более крупный блок для уменьшения фрагментации.

**Объединение блоков**:

* После освобождения блок проверяет своих соседей. Если они также свободны, блоки объединяются, чтобы минимизировать количество фрагментов памяти.

**Граничные условия**:

* Если размер запроса меньше минимального блока, выделяется минимально допустимый блок.
* В случае исчерпания памяти аллокатор возвращает ошибку.

# Код программы

**main.c**

**#include "library.h"**

**#define MEMORY\_POOL\_SIZE 1024**

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

**const char msg[] = "allocator\_create: Function not found, using mmap\n";**

**write(STDERR\_FILENO, msg, sizeof(msg) - 1);**

**void \*mapped\_memory = mmap(memory, size, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS | MAP\_FIXED, -1, 0);**

**if (mapped\_memory == MAP\_FAILED) {**

**const char err\_msg[] = "allocator\_create: mmap failed\n";**

**write(STDERR\_FILENO, err\_msg, sizeof(err\_msg) - 1);**

**return NULL;**

**}**

**return (Allocator \*)mapped\_memory;**

**}**

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

**const char msg[] = "allocator\_destroy: Function not found, using munmap\n";**

**write(STDERR\_FILENO, msg, sizeof(msg) - 1);**

**if (allocator) {**

**if (munmap(allocator, MEMORY\_POOL\_SIZE) == -1) {**

**const char err\_msg[] = "allocator\_destroy: munmap failed\n";**

**write(STDERR\_FILENO, err\_msg, sizeof(err\_msg) - 1);**

**}**

**}**

**}**

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

**const char msg[] = "allocator\_alloc: Function not found, using mmap\n";**

**write(STDERR\_FILENO, msg, sizeof(msg) - 1);**

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

**if (mapped\_memory == MAP\_FAILED) {**

**const char err\_msg[] = "allocator\_alloc: mmap failed\n";**

**write(STDERR\_FILENO, err\_msg, sizeof(err\_msg) - 1);**

**return NULL;**

**}**

**return mapped\_memory;**

**}**

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

**const char msg[] = "allocator\_free: Function not found, using munmap\n";**

**write(STDERR\_FILENO, msg, sizeof(msg) - 1);**

**if (memory && munmap(memory, sizeof(memory)) == -1) {**

**const char err\_msg[] = "allocator\_free: munmap failed\n";**

**write(STDERR\_FILENO, err\_msg, sizeof(err\_msg) - 1);**

**}**

**}**

**static allocator\_create\_f \*allocator\_create;**

**static allocator\_destroy\_f \*allocator\_destroy;**

**static allocator\_alloc\_f \*allocator\_alloc;**

**static allocator\_free\_f \*allocator\_free;**

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

**if (argc < 2) {**

**const char msg[] = "Usage: ./Main <library\_path>\n";**

**write(STDERR\_FILENO, msg, sizeof(msg));**

**return EXIT\_FAILURE;**

**}**

**void \*library = dlopen(argv[1], RTLD\_LOCAL | RTLD\_NOW);**

**if (!library) {**

**const char msg[] = "Failed to load library\n";**

**write(STDERR\_FILENO, msg, sizeof(msg));**

**return EXIT\_FAILURE;**

**}**

**argc++;**

**if (argc > 2) {**

**allocator\_create = dlsym(library, "allocator\_create");**

**allocator\_destroy = dlsym(library, "allocator\_destroy");**

**allocator\_alloc = dlsym(library, "allocator\_alloc");**

**allocator\_free = dlsym(library, "allocator\_free");**

**if (!allocator\_create) {**

**allocator\_create = allocator\_create\_stub;**

**}**

**if (!allocator\_destroy) {**

**allocator\_destroy = allocator\_destroy\_stub;**

**}**

**if (!allocator\_alloc) {**

**allocator\_alloc = allocator\_alloc\_stub;**

**}**

**if (!allocator\_free) {**

**allocator\_free = allocator\_free\_stub;**

**}**

**} else {**

**const char msg[] = "error: failed to open custom library\n";**

**write(STDERR\_FILENO, msg, sizeof(msg));**

**return EXIT\_FAILURE;**

**}**

**// Teсты библиотеки**

**size\_t size = MEMORY\_POOL\_SIZE;**

**void \*addr = mmap(NULL, size, PROT\_READ | PROT\_WRITE,**

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

**if (addr == MAP\_FAILED) {**

**dlclose(library);**

**char message[] = "mmap failed\n";**

**write(STDERR\_FILENO, message, sizeof(message) - 1);**

**return EXIT\_FAILURE;**

**}**

**Allocator \*allocator = allocator\_create(addr, MEMORY\_POOL\_SIZE);**

**if (!allocator) {**

**const char msg[] = "Failed to initialize allocator\n";**

**write(STDERR\_FILENO, msg, sizeof(msg));**

**munmap(addr, size);**

**dlclose(library);**

**return EXIT\_FAILURE;**

**}**

**int \*int\_block = (int \*)allocator\_alloc(allocator, sizeof(int));**

**if (int\_block) {**

**\*int\_block = 42;**

**const char msg[] = "Allocated int\_block with value 42\n";**

**write(STDOUT\_FILENO, msg, sizeof(msg));**

**} else {**

**const char msg[] = "Failed to allocate memory for int\_block\n";**

**write(STDERR\_FILENO, msg, sizeof(msg));**

**}**

**float \*float\_block = (float \*)allocator\_alloc(allocator, sizeof(float));**

**if (float\_block) {**

**\*float\_block = 3.14f;**

**const char msg[] = "Allocated float\_block with value 3.14\n";**

**write(STDOUT\_FILENO, msg, sizeof(msg));**

**} else {**

**const char msg[] = "Failed to allocate memory for float\_block\n";**

**write(STDERR\_FILENO, msg, sizeof(msg));**

**}**

**if (int\_block) {**

**allocator\_free(allocator, int\_block);**

**const char msg[] = "Freed int\_block\n";**

**write(STDOUT\_FILENO, msg, sizeof(msg));**

**}**

**if (float\_block) {**

**allocator\_free(allocator, float\_block);**

**const char msg[] = "Freed float\_block\n";**

**write(STDOUT\_FILENO, msg, sizeof(msg));**

**}**

**allocator\_destroy(allocator);**

**const char msg[] = "Allocator destroyed\n";**

**write(STDOUT\_FILENO, msg, sizeof(msg));**

**if (library) dlclose(library);**

**munmap(addr, size);**

**return EXIT\_SUCCESS;**

**library.h**

**#ifndef ALLOCATOR\_H**

**#define ALLOCATOR\_H**

**#include <dlfcn.h>**

**#include <stdint.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <fcntl.h>**

**#include <dlfcn.h>**

**#include <sys/mman.h>**

**#include <stddef.h>**

**#include <string.h>**

**#include <stdbool.h>**

**#ifdef \_MSC\_VER**

**#define EXPORT \_\_declspec(dllexport)**

**#else**

**#define EXPORT**

**#endif**

**typedef struct Allocator Allocator;**

**typedef struct Block Block;**

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

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

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

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

**#endifvsevolod@DESKTOP-K08EACJ:~/os\_labs/laba\_4/task$**

**freeblocks.c**

**#include "library.h"**

**#define MIN\_BLOCK\_SIZE 32**

**typedef struct Block {**

**size\_t size;**

**struct Block \*next;**

**bool is\_free;**

**} Block;**

**typedef struct Allocator {**

**Block \*free\_list;**

**void \*memory\_start;**

**size\_t total\_size;**

**} Allocator;**

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

**if (!memory || size < sizeof(Allocator)) {**

**return NULL;**

**}**

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

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

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

**allocator->free\_list = (Block \*)allocator->memory\_start;**

**allocator->free\_list->size = allocator->total\_size - sizeof(Block);**

**allocator->free\_list->next = NULL;**

**allocator->free\_list->is\_free = true;**

**return allocator;**

**}**

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

**if (allocator) {**

**memset(allocator, 0, allocator->total\_size);**

**}**

**}**

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

**if (!allocator || size == 0) {**

**return NULL;**

**}**

**size = (size + MIN\_BLOCK\_SIZE - 1) / MIN\_BLOCK\_SIZE \* MIN\_BLOCK\_SIZE;**

**Block \*best = NULL;**

**Block \*prev\_best = NULL;**

**Block \*current = allocator->free\_list;**

**Block \*prev = NULL;**

**while (current) {**

**if (current->is\_free && current->size >= size) {**

**if (best == NULL || current->size < best->size) {**

**best = current;**

**prev\_best = prev;**

**}**

**}**

**prev = current;**

**current = current->next;**

**}**

**if (best) {**

**size\_t remain\_size = best->size - size;**

**if (remain\_size >= sizeof(Block) + MIN\_BLOCK\_SIZE) {**

**Block \*new\_block =**

**(Block \*)((char \*)best + sizeof(Block) +**

**size);**

**new\_block->size = remain\_size - sizeof(Block);**

**new\_block->is\_free = true;**

**new\_block->next = best->next;**

**best->next = new\_block;**

**best->size = size;**

**}**

**best->is\_free = false;**

**if (prev\_best == NULL) {**

**allocator->free\_list = best->next;**

**} else {**

**prev\_best->next = best->next;**

**}**

**return (void \*)((char \*)best + sizeof(Block));**

**}**

**return NULL;**

**}**

**EXPORT void allocator\_free(Allocator \*allocator, void \*ptr\_to\_memory) {**

**if (!allocator || !ptr\_to\_memory) {**

**return;**

**}**

**Block \*head = (Block \*)((char \*)ptr\_to\_memory - sizeof(Block));**

**if (!head) return;**

**head->next = allocator->free\_list;**

**head->is\_free = true;**

**allocator->free\_list = head;**

**Block \*current = allocator->free\_list;**

**while (current && current->next) {**

**if (((char \*)current + sizeof(Block) + current->size) ==**

**(char \*)current->next) {**

**current->size += current->next->size + sizeof(Block);**

**current->next = current->next->next;**

**} else {**

**current = current->next;**

**}**

**}**

**buddys.c**

#include "library.h"

typedef struct Allocator {

void \*memory;

size\_t size;

uint8\_t \*bitmap;

size\_t block\_size;

} Allocator;

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

if (!memory || size == 0) return NULL;

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

allocator->memory = (void \*)((uint8\_t \*)memory + sizeof(Allocator));

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

allocator->block\_size = 32;

allocator->bitmap = (uint8\_t \*)allocator->memory;

size\_t bitmap\_size = allocator->size / allocator->block\_size / 8;

memset(allocator->bitmap, 0, bitmap\_size); // Все блоки свободны

allocator->memory = (uint8\_t \*)allocator->bitmap + bitmap\_size;

return allocator;

}

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

if (allocator) {

memset(allocator, 0, allocator->size);

}

}

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

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

size\_t blocks\_needed = (size + allocator->block\_size - 1) / allocator->block\_size;

size\_t total\_blocks = allocator->size / allocator->block\_size;

uint8\_t \*bitmap = allocator->bitmap;

size\_t free\_blocks = 0;

for (size\_t i = 0; i < total\_blocks; ++i) {

if (!(bitmap[i / 8] & (1 << (i % 8)))) {

++free\_blocks;

if (free\_blocks == blocks\_needed) {

size\_t start\_block = i - blocks\_needed + 1;

for (size\_t j = start\_block; j <= i; ++j) {

bitmap[j / 8] |= (1 << (j % 8)); // Помечаем блоки как занятые

}

return (uint8\_t \*)allocator->memory + start\_block \* allocator->block\_size;

}

} else {

free\_blocks = 0;

}

}

// Если не найдено достаточное количество свободных блоков, пытаемся выполнить деление:

size\_t block\_size = allocator->block\_size;

while (block\_size > 1) { // Продолжаем делить блоки, пока не дойдем до минимального размера

block\_size /= 2;

free\_blocks = 0;

for (size\_t i = 0; i < total\_blocks; ++i) {

if (!(bitmap[i / 8] & (1 << (i % 8)))) {

++free\_blocks;

if (free\_blocks == blocks\_needed) {

size\_t start\_block = i - blocks\_needed + 1;

for (size\_t j = start\_block; j <= i; ++j) {

bitmap[j / 8] |= (1 << (j % 8)); // Помечаем блоки как занятые

}

return (uint8\_t \*)allocator->memory + start\_block \* allocator->block\_size;

}

} else {

free\_blocks = 0;

}

}

}

// Если не удалось выделить память, возвращаем NULL

return NULL;

}

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

if (!allocator || !memory) return;

size\_t offset = (uint8\_t \*)memory - (uint8\_t \*)allocator->memory;

if (offset % allocator->block\_size != 0) return;

size\_t block\_index = offset / allocator->block\_size;

// Помечаем блок как свободный в битовой карте

allocator->bitmap[block\_index / 8] &= ~(1 << (block\_index % 8));

// Начинаем проверку для объединения "двойников"

size\_t current\_block\_size = allocator->block\_size;

// Пока блок можно "объединять", проверяем его двойника

while (current\_block\_size < allocator->size) {

size\_t buddy\_index = block\_index ^ (current\_block\_size / 2); // Индекс двойника

if (buddy\_index < allocator->size / current\_block\_size &&

!(allocator->bitmap[buddy\_index / 8] & (1 << (buddy\_index % 8)))) {

// Если двойник свободен, объединяем блоки

allocator->bitmap[block\_index / 8] &= ~(1 << (block\_index % 8));

allocator->bitmap[buddy\_index / 8] &= ~(1 << (buddy\_index % 8));

block\_index = block\_index / 2; // Переходим к родительскому блоку

current\_block\_size \*= 2;

} else {

break;

}

}

}

# Протокол работы программы Процесс тестирования:

Для тестирования использовались следующие сценарии:

1. **Массовое выделение и освобождение памяти**: Выделение памяти разного размера с последующим освобождением.
2. **Проверка объединения блоков**: Выделение нескольких блоков и освобождение их в произвольном порядке для проверки корректности объединения.
3. **Измерение производительности**: Сравнение времени выполнения операций выделения и освобождения памяти.
4. **Измерение фрагментации**: Оценка степени использования памяти.

# Обоснование подхода тестирования

Тесты разработаны для проверки следующих характеристик:

1. **Эффективность выделения памяти**: Важно для приложений, интенсивно использующих динамическую память.
2. **Корректность работы**: Объединение блоков и освобождение должны работать без ошибок.
3. **Производительность**: Аллокатор должен минимизировать накладные расходы.
4. **Фрагментация**: Важно для долгосрочной работы без истощения памяти.

# Результаты тестирования

# **vsevolod@DESKTOP-K08EACJ:~/os\_labs/laba\_4/task$ ./Main ./buddys.so**

# **Allocated int\_block with value 42**

# **Allocated float\_block with value 3.14**

# **Freed int\_block**

# **Freed float\_block**

# **Allocator destroyed**

# **vsevolod@DESKTOP-K08EACJ:~/os\_labs/laba\_4/task$ ./Main ./freeblocks.so**

# **Allocated int\_block with value 42**

# **Allocated float\_block with value 3.14**

# **Freed int\_block**

# **Freed float\_block**

# **Allocator destroyed**

# **vsevolod@DESKTOP-K08EACJ:~/os\_labs/laba\_4/task$**

**Метод свободных блоков**

**Производительность**: Быстрое выделение памяти, но медленное объединение блоков при освобождении.

* + **Фрагментация**: Минимальная.
  + **Память**: Эффективное использование памяти для запросов любого размера.

**Метод двойников**

* **Производительность**: Высокая скорость выделения и освобождения памяти.
* **Фрагментация**: Заметная внутренняя фрагментация из-за округления размеров запросов.
* **Память**: Эффективен для запросов, кратных степени двойки.

**Strace:**

strace ./Main ./buddys.so

execve("./Main", ["./Main", "./buddys.so"], 0x7ffcb6baa4d8 /\* 35 vars \*/) = 0 brk(NULL) = 0x55e54677c000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffacdc9c40) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1,

0) = 0x7f6534bac000

access("/etc/ld.so.preload", R\_OK) = -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=18823, ...}, AT\_EMPTY\_PATH) = 0 mmap(NULL, 18823, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f6534ba7000

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) = 0x7f653497e000

mprotect(0x7f65349a6000, 2023424, PROT\_NONE) = 0

mmap(0x7f65349a6000, 1658880, PROT\_READ|PROT\_EXEC,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f65349a6000

mmap(0x7f6534b3b000, 360448, PROT\_READ,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f6534b3b000

mmap(0x7f6534b94000, 24576, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f6534b94000

mmap(0x7f6534b9a000, 52816, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f6534b9a000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1,

0) = 0x7f653497b000

arch\_prctl(ARCH\_SET\_FS, 0x7f653497b740) = 0 set\_tid\_address(0x7f653497ba10) = 105236

set\_robust\_list(0x7f653497ba20, 24) = 0

rseq(0x7f653497c0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f6534b94000, 16384, PROT\_READ) = 0

mprotect(0x55e5241b6000, 4096, PROT\_READ) = 0

mprotect(0x7f6534be6000, 8192, PROT\_READ) = 0

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

= 0

munmap(0x7f6534ba7000, 18823) = 0

getrandom("\x90\x06\xe1\xf7\x2b\xd6\x44\xd0", 8, GRND\_NONBLOCK) = 8 brk(NULL) = 0x55e54677c000

brk(0x55e54679d000) = 0x55e54679d000

**openat(AT\_FDCWD, "./buddys.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=15632, ...}, AT\_EMPTY\_PATH) = 0 getcwd("/home/pablo/Main/3,4 SEM/LabOs1/LabOS4/src", 128) = 43

**mmap(NULL, 16432, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) =**

**0x7f6534ba7000mmap(0x7f6534ba8000, 4096, PROT\_READ|PROT\_EXEC,**

**MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x7f6534ba8000**

**mmap(0x7f6534ba9000, 4096, PROT\_READ,**

**MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f6534ba9000**

**mmap(0x7f6534baa000, 8192, PROT\_READ|PROT\_WRITE,**

**MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f6534baa000**

**close(3) = 0**

mprotect(0x7f6534baa000, 4096, PROT\_READ) = 0

**mmap(NULL, 1024, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS,**

**-1, 0) = 0x7f6534be5000**

write(1, "Allocated int\_block with value 4"..., 35Allocated int\_block with value 42

) = 35

write(1, "Allocated float\_block with value"..., 39Allocated float\_block with value 3.14

) = 39

write(1, "Freed int\_block\n\0", 17Freed int\_block

) = 17

write(1, "Freed float\_block\n\0", 19Freed float\_block

) = 19

write(1, "Allocator destroyed\n\0", 21Allocator destroyed

) = 21

munmap(0x7f6534ba7000, 16432) = 0

**munmap(0x7f6534be5000, 1024) = 0**

exit\_group(0) = ?

+++ exited with 0 +++

# Вывод

В процессе выполнения этой лабораторной работы я освоил работу с динамическими библиотеками, новыми системными вызовами, предназначенными для работы с динамическими библиотеками, и написанием собственного аллокатора памяти в языке С. Я научился писать собственные динамические библиотеки, подключать, обрабатывать ошибки, связанные с их подключением, и использовать их. Главная сложность работы возникла при написании собственного аллокатора памяти, поскольку материал был новый для меня и информацию про алгоритмы аллокаторов приходилась искать в книгах и интернете.