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: Коростин Н.А.

Преподаватель: Бахарев В.Д.

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

Дата: 16.11..24

Москва, 2024

**Постановка задачи**

**Вариант 6.**

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

1. Блоки по 2^n.
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 Блоки по 2^n:**

Работа аллокаторов такого вида заключается в выделении блоков памяти, размер которых всегда равен степени двойки. Такой подход позволяет упростить управление памятью и повысить производительность. В начале аллокатору выделяется большой блок памяти (через mmap). Вся выделенная память рассматривается как единый блок самой большой степени двойки, который может быть использован. Для управления памятью создаются списки свободных блоков (free\_lists), где каждый индекс соответствует блоку памяти определенного размера (например, 2⁰, 2¹, 2² и т.д.). При освобождении блока памяти указатель на блок помещается в список свободных блоков соответствующего размера. Далее, если нужно снова выделить блок памяти такого же размера, то блок сразу берется из списка свободных блоков.

Алгоритм выделения памяти (alloc):

1. Пользователь запрашивает блок памяти определенного размера.
2. Размер округляется до ближайшей степени двойки, равной или больше запрашиваемого (например, запрос на 13 байт округляется до 16 байт).
3. Если подходящий блок есть в соответствующем списке свободных блоков (free\_lists), он выделяется.
4. Если подходящего блока нет, аллокатор ищет блок большего размера в следующих списках. Если такой блок найден:
   1. Блок делится на два равных меньших блока.
   2. Один из этих меньших блоков возвращается пользователю, а второй добавляется в список свободных блоков для меньшего размера.
5. Если блок большего размера также отсутствует, возвращается ошибка (недостаточно памяти).

Алгоритм освобождения памяти (free):

1. Пользователь освобождает ранее выделенный блок памяти.
2. Освобожденный блок добавляется в соответствующий список свободных блоков (free\_lists).

Данный алгоритм обладает несколькими преимуществами, такими как, например, простота реализации: структура свободных списков и логика деления блоков довольно тривиальны и эффективны. Другое преимущество: работа с фрагментацией. Внутренняя фрагментация (пустое место в выделенных блоках) уменьшается, так как блоки кратны ближайшей степени двойки.

Внешняя фрагментация минимизируется, так как память разделяется строго по фиксированным размерам и переиспользуется при освобождении блока.

По этим причинам данный алгоритм лежит в основе других видов аллокаторов памяти, таких как, например, buddy allocator.

**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**

Из таблицы видно, что алгоритм двойников выигрывает алгоритм степени двойки на больших значениях количества аллокации, и примерно сравним с ним при единичных аллокациях и освобождениях памяти. Аварийная обертка на mmap проигрывает обоим этим аллокаторам и во время аллокации памяти, и во время освобождения.

Сравнение использования аллокаторов по времени, в секундах:

| **Вид аллокатора** | **Память 1030 байт**  **(аллокация)** | **Память 10567**  **(удаление)** | **Серия из 1000 аллокаций**  **и освобождений** |
| --- | --- | --- | --- |
| 2^n allocator | 0.000933 | 0.000063 | 0.005763 |
| buddy allocator | 0.000744 | 0.000086 | 0.002568 |
| emergency (mmap) | 0.008086 | 0.000091 | 0.046574 |

**Код программы**

**Аллокатор 2^n:**

**power\_two\_allocator.cpp:**

#include "buddy\_allocator.h"

#include <iostream>

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

size\_t power = 1;

while (power < size) {

power <<= 1;

}

return power;

}

int get\_power\_of\_two(size\_t size) {

return (int)log2(size);

}

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

if (!memory || size < sizeof(Block)) {

return NULL;

}

BuddyAllocator\* allocator =

(BuddyAllocator\*)mmap(NULL, sizeof(BuddyAllocator), PROT\_READ | PROT\_WRITE,

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

if (allocator == MAP\_FAILED) {

return NULL;

}

allocator->memory = memory;

allocator->total\_size = size;

for (size\_t i = 0; i < NUM\_SIZES; i++) {

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

}

size\_t block\_size = round\_to\_power\_of\_two(size);

int power\_exponent = get\_power\_of\_two(block\_size);

if (power\_exponent >= NUM\_SIZES) {

munmap(allocator, sizeof(BuddyAllocator));

return NULL;

}

Block\* initial\_block = (Block\*)memory;

initial\_block->size = block\_size;

initial\_block->is\_free = 1;

initial\_block->next = NULL;

allocator->free\_lists[power\_exponent] = initial\_block;

return allocator;

}

void allocator\_destroy(void\* const buddy\_allocator) {

if (!buddy\_allocator) return;

BuddyAllocator\* allocator = (BuddyAllocator\*) buddy\_allocator;

munmap(allocator, sizeof(BuddyAllocator));

}

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

if (!buddy\_allocator || size == 0) return NULL;

BuddyAllocator\* allocator = (BuddyAllocator\*)buddy\_allocator;

size\_t block\_size = round\_to\_power\_of\_two(size);

int power\_exponent = get\_power\_of\_two(block\_size);

for (int i = power\_exponent; i < NUM\_SIZES; i++) {

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

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

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

while (i > power\_exponent) {

i--;

size\_t smaller\_block\_size = 1 << i;

Block\* buddy = (Block\*)((char\*)block + smaller\_block\_size);

buddy->size = smaller\_block\_size;

buddy->is\_free = 1;

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

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

block->size = smaller\_block\_size;

}

block->is\_free = 0;

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

}

}

return NULL;

}

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

if (!buddy\_allocator || !memory) return;

BuddyAllocator\* allocator = (BuddyAllocator\*)buddy\_allocator;

Block\* block = (Block\*)((char\*)memory - sizeof(Block));

block->is\_free = 1;

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

int power\_exponent = get\_power\_of\_two(block\_size);

while (power\_exponent < NUM\_SIZES - 1) {

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 ||

!buddy->is\_free || buddy->size != block\_size) {

break;

}

Block\*\* list = &allocator->free\_lists[power\_exponent];

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

list = &(\*list)->next;

}

if (\*list) {

\*list = buddy->next;

}

if (buddy\_offset < block\_offset) {

block = buddy;

}

block\_size \*= 2;

block->size = block\_size;

power\_exponent++;

}

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

allocator->free\_lists[power\_exponent] = block;

}

**power\_two\_allocator.h:**

#ifndef \_2N\_ALLOCATOR\_

#define \_2N\_ALLOCATOR\_

#ifdef \_\_cplusplus

extern "C" {

#endif

#include <unistd.h>

#include <math.h>

#include <cstddef>

#include <sys/mman.h>

#define MIN\_BLOCK\_SIZE 8

#define MAX\_BLOCK\_SIZE 1024

#define MAX\_BLOCK\_INDEX 22

typedef struct Block {

struct Block\* next;

size\_t size;

} Block;

typedef struct TwonAllocator {

Block\* free\_lists[MAX\_BLOCK\_INDEX];

void\* memory;

size\_t total\_size;

} TwonAllocator;

size\_t round\_to\_power\_of\_two(size\_t);

int get\_power\_of\_two(size\_t);

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

void allocator\_destroy(void\* const);

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

void allocator\_free(void\* const, void\* const);

#ifdef \_\_cplusplus

}

#endif

#endif // \_2N\_ALLOCATOR\_

**Buddy allocator:**

**buddy\_allocator.cpp:**

#include "power\_two\_allocator.h"

#include <sys/mman.h>

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

size\_t power = MIN\_BLOCK\_SIZE;

while (power < size) {

power <<= 1;

}

return power;

}

int get\_power\_of\_two(size\_t size) {

return (int)log2(size);

}

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

if (!memory || size < MIN\_BLOCK\_SIZE) {

return NULL;

}

TwonAllocator\* allocator = (TwonAllocator\*)mmap(

NULL, sizeof(TwonAllocator), PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);

if (allocator == MAP\_FAILED) {

return NULL;

}

size\_t total\_size = round\_to\_power\_of\_two(size);

size\_t max\_size = 1 << (MAX\_BLOCK\_INDEX - 1);

if (total\_size > max\_size) {

munmap(allocator, sizeof(TwonAllocator));

return NULL;

}

allocator->memory = memory;

allocator->total\_size = total\_size;

for (size\_t i = 0; i < MAX\_BLOCK\_INDEX; i++) {

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

}

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

initial\_block->size = total\_size;

initial\_block->next = NULL;

int index = get\_power\_of\_two(total\_size) - get\_power\_of\_two(MIN\_BLOCK\_SIZE);

allocator->free\_lists[index] = initial\_block;

return allocator;

}

void allocator\_destroy(void\* const twon\_allocator) {

if (!twon\_allocator) return;

TwonAllocator\* allocator = (TwonAllocator\*)twon\_allocator;

munmap(allocator, sizeof(TwonAllocator));

}

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

if (!twon\_allocator || size == 0) return NULL;

TwonAllocator\* allocator = (TwonAllocator\*)twon\_allocator;

size\_t block\_size = round\_to\_power\_of\_two(size);

size\_t max\_size = 1 << (MAX\_BLOCK\_INDEX - 1);

if (block\_size > max\_size) {

return NULL;

}

int index = get\_power\_of\_two(block\_size) - get\_power\_of\_two(MIN\_BLOCK\_SIZE);

if (index < 0 || index >= MAX\_BLOCK\_INDEX) {

return NULL;

}

while (index < MAX\_BLOCK\_INDEX && !allocator->free\_lists[index]) {

index++;

}

if (index >= MAX\_BLOCK\_INDEX) {

return NULL;

}

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

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

while (block->size > block\_size) {

size\_t new\_size = block->size >> 1;

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

buddy->size = new\_size;

buddy->next =

allocator->free\_lists[get\_power\_of\_two(new\_size) - get\_power\_of\_two(MIN\_BLOCK\_SIZE)];

allocator->free\_lists[get\_power\_of\_two(new\_size) - get\_power\_of\_two(MIN\_BLOCK\_SIZE)] =

buddy;

block->size = new\_size;

}

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

}

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

if (!twon\_allocator || !memory) return;

TwonAllocator\* allocator = (TwonAllocator\*)twon\_allocator;

Block\* block = (Block\*)((char\*)memory - sizeof(Block));

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

int index = get\_power\_of\_two(block\_size) - get\_power\_of\_two(MIN\_BLOCK\_SIZE);

if (index < 0 || index >= MAX\_BLOCK\_INDEX) return;

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

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

}

**buddy\_allocator.h:**

#ifndef \_BUDDY\_ALLOCATOR\_

#define \_BUDDY\_ALLOCATOR\_

#ifdef \_\_cplusplus

extern "C" {

#endif

#include <sys/mman.h>

#include <math.h>

#include <unistd.h>

#include <cstddef>

#define NUM\_SIZES 30

typedef struct Block {

Block\* next;

size\_t size;

int is\_free;

} Block;

typedef struct BuddyAllocator {

void\* memory;

size\_t total\_size;

Block\* free\_lists[NUM\_SIZES];

} BuddyAllocator;

size\_t round\_to\_power\_of\_two(size\_t);

int get\_power\_of\_two(size\_t);

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

void allocator\_destroy(void\* const);

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

void allocator\_free(void\* const, void\* const);

#ifdef \_\_cplusplus

}

#endif

#endif // \_BUDDY\_ALLOCATOR\_

**main.cpp:**

#include <dlfcn.h>

#include <sys/mman.h>

#include <time.h>

#include <unistd.h>

#include <cstddef>

#include <ctime>

#include <map>

#include <string>

#include <iostream>

#include "../include/errors.hpp"

#include "../include/io.hpp"

typedef void\* (\*create\_func)(void\* const, const size\_t);

typedef void (\*destroy\_func)(void\* const);

typedef void\* (\*alloc\_func)(void\* const, const size\_t);

typedef void (\*free\_func)(void\* const, void\* const);

static create\_func allocator\_create = nullptr;

static destroy\_func allocator\_destroy = nullptr;

static alloc\_func allocator\_alloc = nullptr;

static free\_func allocator\_free = nullptr;

std::map<void\*, size\_t> allocated\_blocks;

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

return memory;

}

void stub\_allocator\_destroy(void\* const allocator) {

write\_to\_file(STDOUT\_FILENO, "Emergency allocator destroyed\n");

return;

}

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

void\* ptr = mmap(nullptr, size, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANON,

-1, 0);

if (ptr == MAP\_FAILED) {

return nullptr;

}

allocated\_blocks[ptr] = size;

return ptr;

}

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

munmap(memory, allocated\_blocks[memory]);

allocated\_blocks.erase(memory);

}

void load\_emergency\_functions() {

allocator\_create = stub\_allocator\_create;

allocator\_destroy = stub\_allocator\_destroy;

allocator\_alloc = stub\_allocator\_alloc;

allocator\_free = stub\_allocator\_free;

}

void load\_library\_functions(char\* lib\_path, void\*\* cur\_library) {

void\* library = dlopen(lib\_path, RTLD\_LAZY);

if (!library) {

std::cerr << "Error: " << dlerror() << std::endl;

return;

}

allocator\_create = (create\_func)dlsym(library, "allocator\_create");

if (!allocator\_create) {

write\_to\_file(STDERR\_FILENO, "Failed to load 'allocator\_create'\n");

}

allocator\_destroy = (destroy\_func)dlsym(library, "allocator\_destroy");

if (!allocator\_destroy) {

write\_to\_file(STDERR\_FILENO, "Failed to load 'allocator\_destroy'\n");

}

allocator\_alloc = (alloc\_func)dlsym(library, "allocator\_alloc");

if (!allocator\_alloc) {

write\_to\_file(STDERR\_FILENO, "Failed to load 'allocator\_alloc'\n");

}

allocator\_free = (free\_func)dlsym(library, "allocator\_free");

if (!allocator\_free) {

write\_to\_file(STDERR\_FILENO, "Failed to load 'allocator\_free'\n");

}

\*cur\_library = library;

}

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) {

void\* library = NULL;

if (argc < 2) {

log\_errors(WRONG\_NUMBER\_OF\_PARAMS);

write\_to\_file(STDOUT\_FILENO, "Using Emergency Implementations\n");

load\_emergency\_functions();

} else {

load\_library\_functions(argv[1], &library);

if (!allocator\_create || !allocator\_destroy || !allocator\_alloc ||

!allocator\_free) {

log\_errors(LIBRARY\_OPEN\_ERROR);

write\_to\_file(STDOUT\_FILENO, "Using Emergency Implementations\n");

load\_emergency\_functions();

}

write\_to\_file(STDOUT\_FILENO, "Library Loaded Successfully\n");

}

size\_t memory\_size = 1024 \* 1024;

void\* memory = mmap(NULL, memory\_size, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE |

MAP\_ANON, -1, 0);

if (memory == MAP\_FAILED) {

log\_errors(MEMORY\_MAPPING\_FAILED);

return 1;

}

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

if (!allocator) {

log\_errors(ALLOCATOR\_CREATION\_ERROR);

munmap(memory, memory\_size);

return 1;

}

print\_to\_stdout("Allocator created successfully\n");

// Тестинг

struct timespec start, end;

double time\_used;

size\_t data\_size = 1030;

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

void\* ptr1 = allocator\_alloc(allocator, data\_size);

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

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

if (ptr1) {

print\_to\_stdout("Time of allocation of " + std::to\_string(data\_size) +

" bytes: " + std::to\_string(time\_used) + " second\n");

} else {

print\_to\_stdout("Allocation failed\n");

}

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

allocator\_free(allocator, ptr1);

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

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

print\_to\_stdout("Time of freeing of " + std::to\_string(data\_size) +

" bytes: " + std::to\_string(time\_used) + " second\n");

/////////////////////////////////////////////////////////////////////////////

data\_size = 10567;

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

void\* ptr2 = allocator\_alloc(allocator, data\_size);

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

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

if (ptr1) {

print\_to\_stdout("Time of allocation of " + std::to\_string(data\_size) +

" bytes: " + std::to\_string(time\_used) + " second\n");

} else {

print\_to\_stdout("Allocation failed\n");

}

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

allocator\_free(allocator, ptr2);

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

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

print\_to\_stdout("Time of freeing of " + std::to\_string(data\_size) +

" bytes: " + std::to\_string(time\_used) + " second\n");

/////////////////////////////////////////////////////////////////////

void \* test[1000];

size\_t size = 1234;

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

for (int i = 0; i < 1000; i++) {

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

}

for (int i = 0; i < 1000; i++) {

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

}

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

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

print\_to\_stdout("Time of 1000 allocations and freeings of " +

std::to\_string(data\_size) +

" bytes: " + std::to\_string(time\_used) + " second\n");

allocator\_destroy(allocator);

munmap(memory, memory\_size);

if (library) {

dlclose(library);

}

return 0;

}

**Протокол работы программы**

strace ./main\_exec

/home/nikita/operation\_systems/lab4/lib\_buddy\_allocator/libbuddy\_allocator.so

execve("./main\_exec", ["./main\_exec", "/home/nikita/operation\_systems/l"...], 0x7ffdd3540a88

/\* 56 vars \*/) = 0

brk(NULL) = 0x5dc2cc64a000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x750458a0f000

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

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=70647, ...}) = 0

mmap(NULL, 70647, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7504589fd000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libstdc++.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\0\0\0\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=2592224, ...}) = 0

mmap(NULL, 2609472, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x750458600000

mmap(0x75045869d000, 1343488, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x9d000) = 0x75045869d000

mmap(0x7504587e5000, 552960, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e5000) =

0x7504587e5000

mmap(0x75045886c000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x26b000) = 0x75045886c000

mmap(0x75045887a000, 12608, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1,

0) = 0x75045887a000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libgcc\_s.so.1", 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

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=183024, ...}) = 0

mmap(NULL, 185256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7504589cf000

mmap(0x7504589d3000, 147456, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x4000) = 0x7504589d3000

mmap(0x7504589f7000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) =

0x7504589f7000

mmap(0x7504589fb000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x2b000) = 0x7504589fb000

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\0\220\243\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

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 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, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x750458200000

mmap(0x750458228000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x28000) = 0x750458228000

mmap(0x7504583b0000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) =

0x7504583b0000

mmap(0x7504583ff000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x1fe000) = 0x7504583ff000

mmap(0x750458405000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1,

0) = 0x750458405000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.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\0\0\0\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=952616, ...}) = 0

mmap(NULL, 950296, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7504588e6000

mmap(0x7504588f6000, 520192, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x10000) = 0x7504588f6000

mmap(0x750458975000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x8f000) =

0x750458975000

mmap(0x7504589cd000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0xe7000) = 0x7504589cd000

close(3) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7504588e4000

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7504588e1000

arch\_prctl(ARCH\_SET\_FS, 0x7504588e1740) = 0

set\_tid\_address(0x7504588e1a10) = 11163

set\_robust\_list(0x7504588e1a20, 24) = 0

rseq(0x7504588e2060, 0x20, 0, 0x53053053) = 0

mprotect(0x7504583ff000, 16384, PROT\_READ) = 0

mprotect(0x7504589cd000, 4096, PROT\_READ) = 0

mprotect(0x7504589fb000, 4096, PROT\_READ) = 0

mprotect(0x75045886c000, 45056, PROT\_READ) = 0

mprotect(0x5dc2a0b73000, 4096, PROT\_READ) = 0

mprotect(0x750458a47000, 8192, PROT\_READ) = 0

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

munmap(0x7504589fd000, 70647) = 0

futex(0x75045887a7bc, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

getrandom("\xed\xb0\x6c\xc4\xe1\x2f\x75\x9f", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x5dc2cc64a000

brk(0x5dc2cc66b000) = 0x5dc2cc66b000

openat(AT\_FDCWD,

"/home/nikita/operation\_systems/lab4/lib\_buddy\_allocator/libbuddy\_allocator.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

fstat(3, {st\_mode=S\_IFREG|0775, st\_size=40552, ...}) = 0

mmap(NULL, 16448, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x750458a0a000

mmap(0x750458a0b000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x1000) = 0x750458a0b000

mmap(0x750458a0c000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) =

0x750458a0c000

mmap(0x750458a0d000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x2000) = 0x750458a0d000

close(3) = 0

mprotect(0x750458a0d000, 4096, PROT\_READ) = 0

write(1, "Library Loaded Successfully\n", 28Library Loaded Successfully

) = 28

mmap(NULL, 1048576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x750458500000

mmap(NULL, 256, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x750458a09000

write(1, "Allocator created successfully\n", 31Allocator created successfully

) = 31

clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, {tv\_sec=0, tv\_nsec=1164157}) = 0

clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, {tv\_sec=0, tv\_nsec=1175232}) = 0

write(1, "Time of allocation of 1030 bytes"..., 50Time of allocation of 1030 bytes: 0.000011

second

) = 50

clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, {tv\_sec=0, tv\_nsec=1198094}) = 0

clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, {tv\_sec=0, tv\_nsec=1201703}) = 0

write(1, "Time of freeing of 1030 bytes: 0"..., 47Time of freeing of 1030 bytes: 0.000003

second

) = 47

clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, {tv\_sec=0, tv\_nsec=1210822}) = 0

clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, {tv\_sec=0, tv\_nsec=1214372}) = 0

write(1, "Time of allocation of 2031 bytes"..., 50Time of allocation of 2031 bytes: 0.000004

second

) = 50

clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, {tv\_sec=0, tv\_nsec=1223016}) = 0

clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, {tv\_sec=0, tv\_nsec=1226474}) = 0

write(1, "Time of freeing of 2031 bytes: 0"..., 47Time of freeing of 2031 bytes: 0.000003

second

) = 47

munmap(0x750458a09000, 256) = 0

munmap(0x750458500000, 1048576) = 0

munmap(0x750458a0a000, 16448) = 0

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В ходе выполнения лабораторной работы я получил навыки в создании динамических библиотек и реализовал программу, использующую динамическую библиотеку. Также были изучены принципы работы с памятью, создания аллокаторов различных типов и их анализа. Для выполнения лабораторной я разобрался в алгоритме buddy allocator, а также в более простых алгоритмах аллокации памяти, что помогло сравнить два реализованных мной аллокатора и оценить их эффективность.