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-214Б-23

Студент: Гайдуков А.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 26.12.24

Москва, 2024

Постановка задачи

Реализовать два алгоритма аллокации и сравнить их.

Вариант 1.

Списки свободных блоков (первое подходящее) и блоки по 2^n;

Общий метод и алгоритм решения

Использованные системные вызовы:

- void mmap(void addr, size_t length, int prot, int flags, int fd, off_t offset);
- int munmap(void addr, size_t length);
- void dlopen(const char filename, int flags);
- void *dlsym(void handle, const char symbol);
- int dlclose(void handle);

Алгоритм аллокаторов:

1. ***Списочный***

1. **Инициализация**
   - Выделяется память с помощью `mmap`.
   - Инициализируется структура `Allocator` и первый блок памяти.
   - Первый блок занимает всю выделенную память, его размер равен размеру выделенной памяти минус размер структуры `Allocator`.
   - Устанавливается указатель `next` первого блока в `NULL`.
2. **Выделение памяти**
   - Производится поиск свободного блока, размер которого достаточен для запроса.
   - Если найден подходящий блок:
     - Блок разделяется на два: один выделяется под запрос, другой остается свободным.
     - Размер выделенного блока равен запрошенному размеру.
     - Размер свободного блока уменьшается на размер выделенного блока и размер структуры `Block`.
   - Если подходящего блока нет, возвращается `NULL` (возможность расширения памяти закомментирована).
   - Возвращается указатель на выделенную память.
3. **Освобождение памяти**
   - Находится блок, соответствующий переданному указателю.
   - Блок помечается как свободный:
     - Если блок находится в середине списка, он удаляется из списка, и соседние блоки связываются напрямую.
     - Если блок является первым, его размер устанавливается в `0`.
   - Если блок не найден, программа завершается с ошибкой.
4. **Уничтожение аллокатора**
   - Вся выделенная память освобождается с помощью `munmap`.
   - Процесс завершается.

Ключевые моменты

- Управление памятью осуществляется через связный список блоков.

- Каждый блок содержит информацию о своем размере и указатель на следующий блок.
- Память выделяется и освобождается на уровне операционной системы с помощью `mmap` и `munmap`.

2. Со степенями двойки

1. **Инициализация**
   - Выделяется память с помощью `mmap` для каждого уровня (всего `POWERS_AMOUNT` уровней).
   - Для каждого уровня создается структура `Header`, которая управляет блоками памяти.
   - Первый блок каждого уровня инициализируется, его размер равен размеру выделенной памяти минус размер структуры `Header` и `Block`.
   - Устанавливается указатель `next` первого блока в `NULL`.
2. **Выделение памяти**
   - Определяется уровень (степень двойки) для запрошенного размера с помощью `ceil(log2(size))`.
   - В соответствующем уровне ищется свободный блок, размер которого достаточен для запроса.
   - Если найден подходящий блок:
     - Блок разделяется на два: один выделяется под запрос, другой остается свободным.
     - Размер выделенного блока равен ближайшей степени двойки, большей или равной запрошенному размеру.
     - Размер свободного блока уменьшается на размер выделенного блока и размер структуры `Block`.
   - Если подходящего блока нет, возвращается `NULL`.
   - Возвращается указатель на выделенную память.
3. **Освобождение памяти**
   - Определяется уровень (степень двойки) для освобождаемого блока.
   - В соответствующем уровне ищется блок, соответствующий переданному указателю.
   - Блок помечается как свободный:
     - Если блок находится в середине списка, он удаляется из списка, и соседние блоки связываются напрямую.
     - Если блок является первым, его размер устанавливается в `0`.
   - Если блок не найден, программа завершается с ошибкой.
4. **Уничтожение аллокатора**
   - Для каждого уровня освобождается память с помощью `munmap`.
   - Процесс завершается.

Ключевые моменты

- Память разделена на уровни, каждый уровень соответствует степени двойки.
- Управление памятью осуществляется через связные списки блоков на каждом уровне.
- Каждый блок содержит информацию о своем размере и указатель на следующий блок.
- Память выделяется и освобождается на уровне операционной системы с помощью `mmap` и `munmap`.

Код программы

main.c (тестирование на время)

```
#include <unistd.h>
#include <string.h>
#include <dlfcn.h>
#include <time.h>

#include "myio.h"
#include "intstr.h"

void* (*allocate)(const size_t size);

void (*deallocate)(const void* memory);


int init_lstlib(){
    void *hdl = dlopen("./lib/liblstalloc.so", RTLD_LAZY);
    if(hdl == NULL){
        return 0;
    }
    allocate = (void* (*)(const size_t))dlsym(hdl, "allocate");
    if(allocate == NULL){
        return 0;
    }
    deallocate = (void (*)(const void*))dlsym(hdl, "deallocate");
    if(deallocate == NULL){
        return 0;
    }
    return 1;
}

int init_tbllib(){
    void *hdl = dlopen("./lib/libtblalloc.so", RTLD_LAZY);
    if(hdl == NULL){
        return 0;
    }
    allocate = (void* (*)(const size_t))dlsym(hdl, "allocate");
    if(allocate == NULL){
        return 0;
    }
    deallocate = (void (*)(const void*))dlsym(hdl, "deallocate");
    if(deallocate == NULL){
        return 0;
    }
    return 1;
```

```
}

int init_stdlib(){
    void *hdl = dlopen("./lib/libstd.so", RTLD_LAZY);
    if(hdl == NULL){
        return 0;
    }
    allocate = (void* (*)(const size_t))dlsym(hdl, "custom_malloc");
    if(allocate == NULL){
        return 0;
    }
    deallocate = (void (*)(const void*))dlsym(hdl, "custom_free");
    if(deallocate == NULL){
        return 0;
    }
    return 1;
}

int main(int argc, char const *argv[])
{
    char allocator_name[100];
    if(argc != 2){
        if(init_stdlib() != 1){
            my_write("Library not found!\n");
            return -1;
        }
        strcpy(allocator_name, "Standart");
    }
    else{
        if(!strcmp("List", argv[1])){
            if(init_lstlib() != 1){
                my_write("Library not found!\n");
                return -1;
            }
        }
        else if(!strcmp("Table", argv[1])){
            if(init_tbllib() != 1){
                my_write("Library not found!\n");
                return -1;
            }
        }
        else{
            my_write("Unknown allocator <"); my_write(argv[1]); my_write(">. Choose
between <List> or <Table> allocators...\n");
            return -1;
        }
        strcpy(allocator_name, argv[1]);
    }
```

```
    int** array = (int**)allocate(sizeof(int*) * 50000);
    int iter = 0;

    clock_t start = clock();
    for (size_t i = 1; i <= 10; i++)
    {
        for (size_t j = 0; j < 5000; j++)
        {
            array[iter++] = allocate(sizeof(int) * i);
            if(array[iter - 1] == NULL){
                for (size_t k = 0; k < iter; k++)
                {
                    deallocate(array[k]);
                }
                deallocate(array);
                my_write("Error while allocating!\n");
                return -1;
            }
        }
    }
    clock_t end = clock();
    my_write("Memory allocating with library <"); my_write(allocator_name); my_write("> lasts: "); print_int((end - start) / 1000); my_write(" ms\n");
    print_int(50000);my_write(" int* allocated.\n");
    start = clock();
    for (int i = iter - 1; i >= 0; i--)
    {
        deallocate(array[i]);
    }
    deallocate(array);
    end = clock();
    my_write("Memory deallocating with library <"); my_write(allocator_name); my_write("> lasts: "); print_int((end - start) / 1000); my_write(" ms\n");
    print_int(50000);my_write(" int* deallocated.\n");
    return 0;
}
```

intstr.c

```
#include <pthread.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <math.h>

#include "intstr.h"
#include "myio.h"
```

```
int hasOnlyNums(char* num){
    for (size_t i = 0; i < strlen(num) && num[i] != '\n'; i++)
    {
        if(!isdigit(num[i]) || (num[i] == '-' && i != 0)){
            return 0;
        }
    }
    return 1;
}

char* double_to_str(double num, char* str){
    int integer_num = (int)num;
    str = int_to_str(integer_num, str);
    num -= integer_num;
    num *= 100000000;
    integer_num = (int)num;

    char rev_num[100];
    int len = 0;
    for (int i = 0; integer_num != 0; i++)
    {
        rev_num[i] = '0' + integer_num % 10;
        integer_num /= 10;
        len = i + 1;
    }

    char* tmp;
    tmp = (char*)realloc(str, sizeof(char) * (sizeof(str) + len + 1));
    if(tmp == NULL){
        free(str);
        return NULL;
    }

    str = tmp;
    str[strlen(str)] = '.';
    int stln = strlen(str);
    for (int i = len - 1; i >= 0; i--, stln++)
    {
        str[stln] = rev_num[i];
    }
    str[stln] = '\0';
    return str;
}

int str_to_int(char* num){
    int res = 0, beg = 0;
    if(!hasOnlyNums(num)){
```

```
        return 0;
    }
    else if(num[0] == '-'){
        ++beg;
    }
    for (int i = beg; i < strlen(num) && num[i] != '\n'; i++)
    {
        res = (res * 10) + num[i] - '0';
    }
    return res;
}

char* int_to_str(int number, char* string){
    char rev_num[100];
    int len = 0;
    for (int i = 0; number != 0; i++)
    {
        rev_num[i] = '0' + number % 10;
        number /= 10;
        len = i + 1;
    }
    string = (char*)malloc(sizeof(char) * (len + 1));
    if(string == NULL){
        return " ";
    }
    for (int i = len - 1, j = 0; i >= 0; i--, j++)
    {
        string[j] = rev_num[i];
    }
    string[len] = '\0';
    return string;
}

void print_int(const int num){
    char* c = int_to_str(num, c);
    my_write(c);
    free(c);
    return;
}

void print_double(const double num){
    char *c = double_to_str(num, c);
    my_write(c);
    free(c);
    return;
}
```

myio.c

```
#include <string.h>
#include <ctype.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define BUFSIZ 8192

ssize_t my_write(char* str){
    return write(STDOUT_FILENO, str, strlen(str));
}

ssize_t my_read(char* buf){
    return read(STDIN_FILENO, buf, BUFSIZ);
}

int file_open(char* filename){
    return open(filename, O_CREAT | O_TRUNC | O_RDWR , S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
}

int write_to_file(int descriptor, char* str){
    if(descriptor == -1){
        my_write("File error!\n");
        return 0;
    }
    write(descriptor, str, strlen(str));
    return 1;
}

int file_close(int descriptor){
    close(descriptor);
    return 0;
}
```

(списочный аллокатор)

```
#include "intstr.h"
#include "myio.h"
#include "VirtualAlloc.h"
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <stdlib.h>
```

```
static Allocator* allocator = NULL;
#define MAP_ANONYMOUS 0x20


Allocator* allocator_create(void* memory, const size_t size){
    char* memo;
    if(memory == NULL){
        memo = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS,
-1, 0);
        if(memo == MAP_FAILED){
            perror();
            return NULL;
        }
    }
    else{
        memo = memory;
    }
    allocator = (Allocator*)memo;
    allocator->memory = (void*)((char*)memo + sizeof(Allocator));
    allocator->size = size - sizeof(Allocator);
    allocator->head_of_blocks = (struct Block*)allocator->memory;

    allocator->head_of_blocks->size = allocator->size;
    allocator->head_of_blocks->next = NULL;
    atexit(allocator_destroy_exit);
    return allocator;
}

void allocator_destroy(const Allocator* allocator){
    if(allocator == NULL){
        return;
    }
    munmap(allocator->memory - sizeof(Allocator), allocator->size + sizeof(Allocator));
    return;
}

void allocator_destroy_exit(){
    if(allocator == NULL){
        return;
    }
    munmap(allocator->memory - sizeof(Allocator), allocator->size + sizeof(Allocator));
    return;
}

/* Allocator* allocator_resize(const Allocator* allocator, const size_t size){
    if(allocator == NULL){
        return NULL;
    }
    void* memory = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);
```

```
    if(memory == NULL){
        return NULL;
    }
    Allocator* allocator_new = (Allocator*)memory;
    allocator_new->memory = (void*)((char*)memory + sizeof(Allocator));
    allocator_new->size = size - sizeof(Allocator);
    allocator_new->head_of_blocks = (struct Block*)allocator_new->memory;

    allocator_new->head_of_blocks->size = allocator_new->size;
    allocator_new->head_of_blocks->next = NULL;
    memcpy(allocator_new->memory, allocator->memory, allocator->size);
    allocator_destroy(allocator);
    return allocator_new;
} */

void* allocator_alloc(const Allocator* allocator, const size_t size){
    if(allocator == NULL){
        return NULL;
    }
    struct Block* empty = allocator->head_of_blocks, *prev = empty;
    int mode = 0;
    size_t allocated = 0, empty_memo_between = 0;

    int res = (char*)empty->next - (char*)empty - sizeof(struct Block);

    if(empty->size == 0 && (char*)empty->next - (char*)empty - sizeof(struct Block) >=
size){
        empty->size = size;
        return (void*)((char*)empty + sizeof(struct Block));
    }

    while(empty->next != NULL){
        prev = empty;

        allocated += (prev->size + sizeof(struct Block));

        empty = empty->next;

        size_t delta = empty - prev - sizeof(struct Block) - prev->size;
        empty_memo_between += delta <= 0? 0: delta;

        mode = empty->next == NULL ? 0 : 1;
        if(empty != prev && (char*)empty - (char*)prev - sizeof(struct Block) -
prev->size >= size){
            break;
        }
    }
    if(mode){
        struct Block *delta_block;
        delta_block = (struct Block*)((char*)prev + prev->size + sizeof(struct Block));
```

```
        prev->next = delta_block;
        delta_block->size = size;
        delta_block->next = empty;
        return (void*)((char*)delta_block + sizeof(struct Block));
    }
    else{

        if(allocator->size - allocated - sizeof(struct Block) - empty_memo_between <=
size){
            allocator = allocator_resize(allocator, allocator->size * 2 - allocated <
size ? allocator->size * 2 + size : allocator->size * 2);
        }

        struct Block *next_block;
        next_block = (struct Block*)((char*)empty + sizeof(struct Block) + size);
        size_t empty_memory = empty->size;
        empty->size = size;
        empty->next = next_block;
        next_block->size = empty_memory - sizeof(struct Block) - size;
        next_block->next = NULL;
        return (void*)((char*)empty + sizeof(struct Block));
    }
}

void allocator_free(const Allocator* allocator, const void* memory){
    if(allocator == NULL){
        return;
    }
    else if(allocator->head_of_blocks->next == NULL){
        my_write("The memory that should have been deleted not found!\n");
        exit(-1);
        return;
    }
    struct Block* delete_block = (struct Block*)((char*)memory - sizeof(struct Block)),
*search = (struct Block*)allocator->head_of_blocks, *prev = search; //а если
head_of_blocks освободили уже???
    while(search != delete_block && search != NULL){
        prev = search;
        search = search->next;
    }
    if(search == NULL){
        my_write("The memory that should have been deleted not found!\n");
        exit(-1);
        return;
    }
    else if(search == allocator->head_of_blocks)
    {
        if(search->next != NULL){
            search->size = 0;
            return;
        }else{
```

```
            return;
        }
    }
    else{
        prev->next = search->next;
    }
    return;
}

void* allocate(const size_t size){
    if(allocator == NULL){
        allocator = allocator_create(NULL, sysconf(_SC_PAGE_SIZE) *
sysconf(_SC_PAGE_SIZE));
        if(allocator == NULL){
            return NULL;
        }
    }
    void* ret_memory = allocator_alloc(allocator, size);
    if(ret_memory == NULL){
        return NULL;
    }
    return ret_memory;
}

void deallocate(const void* memory){
    if(allocator == NULL){
        return;
    }
    allocator_free(allocator, memory);
    return;
}
```

(аллокатор со степенями двойки)

```
#include "myio.h"
#include "intstr.h"
#include <unistd.h>
#include <math.h>
#include <sys/mman.h>
#include "VirtualAllocTwo.h"

#include <stdlib.h>

#define POWERS_AMOUNT 20


static Allocator* allocator = NULL;

#define MP_ANONYMOUS 0x20
Allocator* allocator_create(const size_t size ){
```

```
    Allocator* allocator;
    for (size_t i = 0; i < POWERS_AMOUNT; i++)
    {
        void* memory = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MP_ANONYMOUS, -1, 0);
        if(memory == MAP_FAILED){
            for (size_t j = i; j >= 0; j--)
            {
                munmap(allocator->head[i]->memory, size);
            }
            return NULL;
        }

        if(i == 0){

            allocator = (Allocator*)memory;

            allocator->head[i] = (struct Header*)((char*)memory + sizeof(Allocator));

            allocator->head[i]->memory = (void*)((char*)memory + sizeof(Allocator) +
sizeof(struct Header));

            allocator->head[i]->size = size - sizeof(struct Header);
            allocator->head[i]->power = i;

            struct Block* block = (struct Block*)allocator->head[i]->memory;
            block->next = NULL;
            block->size = allocator->head[i]->size - sizeof(struct Block);
            allocator->head[i]->head = block;
            continue;
        }

        allocator->head[i] = (struct Header*)memory;
        allocator->head[i]->memory = (void*)((char*)memory + sizeof(struct Header));
        allocator->head[i]->size = size - sizeof(struct Header);
        allocator->head[i]->power = i;

        struct Block* block = (struct Block*)allocator->head[i]->memory;
        block->next = NULL;
        block->size = allocator->head[i]->size - sizeof(struct Block);
        allocator->head[i]->head = block;
    }
    allocator->size_of_block = size;
    atexit(allocator_destroy_exit);
    return allocator;
}

void allocator_destroy(const Allocator* allocator){
    for (int j = POWERS_AMOUNT - 1; j >= 0; j--)
    {
```

```
        if(j == 0){
            munmap((void*)((char*)allocator->head[j]->memory - sizeof(struct Header) -
sizeof(Allocator)), allocator->size_of_block);
            continue;
        }
        munmap((void*)((char*)allocator->head[j]->memory - sizeof(struct Header)),
allocator->size_of_block);
    }
    return;
}

void allocator_destroy_exit(){
    for (int j = POWERS_AMOUNT - 1; j >= 0; j--)
    {
        if(j == 0){
            munmap((void*)((char*)allocator->head[j]->memory - sizeof(struct Header) -
sizeof(Allocator)), allocator->size_of_block);
            continue;
        }
        munmap((void*)((char*)allocator->head[j]->memory - sizeof(struct Header)),
allocator->size_of_block);
    }
    return;
}



void* allocator_alloc(const Allocator* allocator, const size_t size){
    if(allocator == NULL){
        return NULL;
    }
    size_t power = (size_t)ceil(log2(size));


    struct Block* empty = allocator->head[power]->head, *prev = empty;
    int mode = 0;
    size_t allocated = 0, empty_memo_between = 0;

    if(empty->size == 0 && (char*)empty->next - (char*)empty - sizeof(struct Block) >=
1 << power){
        empty->size = 1 << (int)ceil(log2(size));
        return (void*)((char*)empty + sizeof(struct Block));
    }

    while(empty->next != NULL){
        prev = empty;

        allocated += (prev->size + sizeof(struct Block));

        empty = empty->next;

        size_t delta = empty - prev - sizeof(struct Block) - prev->size;
```

```
        empty_memo_between += delta <= 0? 0: delta;

        mode = empty->next == NULL ? 0 : 1;
        if(empty != prev && (char*)empty - (char*)prev - sizeof(struct Block) -
prev->size >= /* size */ 1 << power){
            break;
        }
    }
    if(mode){
        struct Block *delta_block;
        delta_block = (struct Block*)((char*)prev + prev->size + sizeof(struct Block));
        prev->next = delta_block;
        delta_block->size = 1 << power;
        delta_block->next = empty;
        return (void*)((char*)delta_block + sizeof(struct Block));
    }
    else{

        struct Block *next_block;
        next_block = (struct Block*)((char*)empty + sizeof(struct Block) + (1 <<
power));
        size_t empty_memory = empty->size;
        empty->size = 1 << power;
        empty->next = next_block;
        next_block->size = empty_memory - sizeof(struct Block) - (1 << power);
        next_block->next = NULL;
        return (void*)((char*)empty + sizeof(struct Block));
    }

}

void allocator_free(const Allocator* allocator, const void* memory){
    struct Block* block = (struct Block*)((char*)memory - sizeof(struct Block));

    if(allocator == NULL){
        return;
    }

    size_t power = (size_t)ceil(log2(block->size));

    struct Block* delete_block = (struct Block*)((char*)memory - sizeof(struct Block)),
*search = (struct Block*)allocator->head[power]->head/* allocator->head_of_blocks */,
*prev = search; //а если head_of_blocks освободили уже???
    if(search->next == NULL){
        my_write("The memory that should have been deleted not found!\n");
        exit(-1);
        return;
    }
    while(search != delete_block && search != NULL){
        prev = search;
        search = search->next;
```

```
    }
    if(search == NULL){
        my_write("The memory that should have been deleted not found!\n");
        exit(-1);
        return;
    }
    else if(search == allocator->head[power]->head)
    {
        if(search->next != NULL){
            search->size = 0;
            return;
        }else{
            my_write("The memory that should have been deleted not found!\n");
            exit(-1);
            return;
        }
    }
    else{
        prev->next = search->next;
    }
    return;
}

void* allocate(const size_t size){
    if(allocator == NULL){
        allocator = allocator_create(sysconf(_SC_PAGE_SIZE) * sysconf(_SC_PAGE_SIZE));
        if(allocator == NULL){
            return NULL;
        }
    }
    void* ret_memory = allocator_alloc(allocator, size);
    if(ret_memory == NULL){
        return NULL;
    }
    return ret_memory;
}

void deallocate(const void* memory){
    if(allocator == NULL){
        return;
    }
    allocator_free(allocator, memory);
    return;
}
```

(стандартный аллокатор)

```
#include <stdlib.h>
```

```
void* custom_malloc(const size_t size){
    return malloc(size);
}

void custom_free(const void* memory){
    free(memory);}
```

Протокол работы программы

Тестирование:

```
gaalex@gaalex-HP-ProBook-445-G7:~/Programs/OS/Fourth_lab$ ./output/main List
Memory allocating with library <List> lasts: 4149 ms
50000 int* allocated.
Memory deallocating with library <List> lasts: 1574 ms
50000 int* deallocated.
gaalex@gaalex-HP-ProBook-445-G7:~/Programs/OS/Fourth_lab$ ./output/main Table
Memory allocating with library <Table> lasts: 1036 ms
50000 int* allocated.
Memory deallocating with library <Table> lasts: 439 ms
50000 int* deallocated.
gaalex@gaalex-HP-ProBook-445-G7:~/Programs/OS/Fourth_lab$ ./output/main
Memory allocating with library <Standart> lasts: 3 ms
50000 int* allocated.
Memory deallocating with library <Standart> lasts: 1 ms
50000 int* deallocated.
```

```
gaalex@gaalex-HP-ProBook-445-G7:~/Programs/OS/Fourth_lab$ ./output/main List && ./output/main Table && ./output/main
Memory allocating with library <List> lasts: 3843 ms
50000 int* allocated.
Memory deallocating with library <List> lasts: 1576 ms
50000 int* deallocated.
Memory allocating with library <Table> lasts: 1060 ms
50000 int* allocated.
Memory deallocating with library <Table> lasts: 456 ms
50000 int* deallocated.
Memory allocating with library <Standart> lasts: 3 ms
50000 int* allocated.
Memory deallocating with library <Standart> lasts: 1 ms
50000 int* deallocated.
```

```
gaalex@gaalex-HP-ProBook-445-G7:~/Programs/OS/Fourth_lab$ ./output/main List && ./output/main Table && ./output/main
Memory allocating with library <List> lasts: 3932 ms
50000 int* allocated.
Memory deallocating with library <List> lasts: 1586 ms
50000 int* deallocated.
Memory allocating with library <Table> lasts: 1080 ms
50000 int* allocated.
Memory deallocating with library <Table> lasts: 451 ms
50000 int* deallocated.
Memory allocating with library <Standart> lasts: 1 ms
50000 int* allocated.
Memory deallocating with library <Standart> lasts:  ms
50000 int* deallocated.
```

Процесс и обоснование тестирования:

В процессе тестирования были совершены попытки выделить память под разные типы данных, очищать данные в разном порядке и добавлять новые данные. Это связано с тем, что оба алгоритма для преодоления фрагментации находят пустые области памяти между занятыми блоками и в случае если места достаточно, выделяют память в этом фрагменте. Тестирование прошло успешно.

Для того чтобы оценить выполнение алгоритмов по времени я отдельно замерил выделение памяти под 50000 указателей на int разного размера и очищение всех этих данных.

Результаты замеров показали, что самый неэффективный алгоритм это списочный (List), далее идет алгоритм с таблицей степеней двоек (Table). Самым эффективным алгоритмом оказалась стандартная реализация malloc и free.

Для списочного аллокатора в большинстве случаев характерно перемещение в конец списка при аллокации нового блока памяти, если между нодами списка нет места. Таким образом, сложность выделения памяти - O(n). При очищении поиск области памяти тоже сложностью O(n).

Аллокатор с таблицами степеней двоек, исходя из замеров, оказался эффективнее списочного. При аллокации блока памяти происходит выравнивание блока памяти под степень двойки и добавление в список фрагментов памяти одинакового размера по алгоритму списочного аллокатора. Сложность такого алгоритма O(n / m), где m - количество ячеек в таблице степеней двойки. То же самое при очищении памяти. Таким образом, этот алгоритм является чуть более оптимизированным в сравнении со списочным аллокатором.

Strace:

```
$ strace ./output/main List

execve("./output/main", ["./output/main", "List"], 0x7ffefe4c69f8 /* 49 vars */) = 0

brk(NULL)                               = 0x561127009000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcb8281260) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb34b6e0000

access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=85427, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 85427, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb34b6cb000

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) = 0x7fb34b4a2000

mprotect(0x7fb34b4ca000, 2023424, PROT_NONE) = 0

mmap(0x7fb34b4ca000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7fb34b4ca000

mmap(0x7fb34b65f000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7fb34b65f000

mmap(0x7fb34b6b8000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7fb34b6b8000

mmap(0x7fb34b6be000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7fb34b6be000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb34b49f000

arch_prctl(ARCH_SET_FS, 0x7fb34b49f740) = 0

set_tid_address(0x7fb34b49fa10)         = 10053

set_robust_list(0x7fb34b49fa20, 24)     = 0

rseq(0x7fb34b4a00e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fb34b6b8000, 16384, PROT_READ) = 0

mprotect(0x56111570f000, 4096, PROT_READ) = 0

mprotect(0x7fb34b71a000, 8192, PROT_READ) = 0
```

```
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fb34b6cb000, 85427)           = 0
getrandom("\xb7\x99\x89\x64\x93\x39\x5c\xb0", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x561127009000
brk(0x56112702a000)                     = 0x56112702a000
openat(AT_FDCWD, "./lib/liblstalloc.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|0775, st_size=16296, ...}, AT_EMPTY_PATH) = 0
getcwd("/home/gaalex/Programs/OS/Fourth_lab", 128) = 36
mmap(NULL, 16536, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fb34b6db000
mmap(0x7fb34b6dc000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1000) = 0x7fb34b6dc000
mmap(0x7fb34b6dd000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) =
0x7fb34b6dd000
mmap(0x7fb34b6de000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x2000) = 0x7fb34b6de000
close(3)                                = 0
mprotect(0x7fb34b6de000, 4096, PROT_READ) = 0
mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb34a49f000
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1404427}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=4, tv_nsec=81759985}) = 0
write(1, "Memory allocating with library <", 32Memory allocating with library <) = 32
write(1, "List", 4List)                 = 4
write(1, "> lasts: ", 9> lasts: )            = 9
write(1, "4080", 44080)                 = 4
write(1, " ms\n", 4 ms
)                   = 4
write(1, "50000", 550000)                = 5
```

```
write(1, " int* allocated.\n", 17 int* allocated.
)      = 17
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=4, tv_nsec=81859484}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=5, tv_nsec=664722051}) = 0
write(1, "Memory deallocating with library"..., 34Memory deallocating with library <) =
34
write(1, "List", 4List)                    = 4
write(1, "> lasts: ", 9> lasts: )               = 9
write(1, "1582", 41582)                    = 4
write(1, " ms\n", 4 ms
)                    = 4
write(1, "50000", 550000)                    = 5
write(1, " int* deallocated.\n", 19 int* deallocated.
)    = 19
munmap(0x7fb34a49f000, 16777216)         = 0
exit_group(0)                            = ?
+++ exited with 0 +++
```

```
$ strace ./output/main Table
execve("./output/main", ["./output/main", "Table"], 0x7ffe232bc828 /* 49 vars */) = 0
brk(NULL)                                = 0x558be620b000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc8cac14c0) = -1 EINVAL (Недопустимый аргумент)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f316aa000
access("/etc/ld.so.preload", R_OK)       = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=85427, ...}, AT_EMPTY_PATH) = 0
```

```
mmap(NULL, 85427, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f8f31695000

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"..., 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) = 0x7f8f3146c000

mprotect(0x7f8f31494000, 2023424, PROT_NONE) = 0

mmap(0x7f8f31494000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f8f31494000

mmap(0x7f8f31629000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7f8f31629000

mmap(0x7f8f31682000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7f8f31682000

mmap(0x7f8f31688000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7f8f31688000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f31469000

arch_prctl(ARCH_SET_FS, 0x7f8f31469740) = 0

set_tid_address(0x7f8f31469a10)         = 10087

set_robust_list(0x7f8f31469a20, 24)     = 0

rseq(0x7f8f3146a0e0, 0x20, 0, 0x53053053) = 0
```

```
mprotect(0x7f8f31682000, 16384, PROT_READ) = 0
mprotect(0x558baf7c0000, 4096, PROT_READ) = 0
mprotect(0x7f8f316e4000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f8f31695000, 85427)           = 0
getrandom("\x1d\x12\x0a\x0e\xdf\xf0\xe6\xc4", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x558be620b000
brk(0x558be622c000)                     = 0x558be622c000
openat(AT_FDCWD, "./lib/libtblalloc.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|0775, st_size=16240, ...}, AT_EMPTY_PATH) = 0
getcwd("/home/gaalex/Programs/OS/Fourth_lab", 128) = 36
mmap(NULL, 16520, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f8f316a5000
mmap(0x7f8f316a6000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1000) = 0x7f8f316a6000
mmap(0x7f8f316a7000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) =
0x7f8f316a7000
mmap(0x7f8f316a8000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x2000) = 0x7f8f316a8000
close(3)                                = 0
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=85427, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 85427, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f8f31454000
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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=940560, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 942344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f8f3136d000
```

mmap(0x7f8f3137b000, 507904, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe000) = 0x7f8f3137b000

mmap(0x7f8f313f7000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8a000) = 0x7f8f313f7000

mmap(0x7f8f31452000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe4000) = 0x7f8f31452000

close(3) = 0

mprotect(0x7f8f31452000, 4096, PROT_READ) = 0

mprotect(0x7f8f316a8000, 4096, PROT_READ) = 0

munmap(0x7f8f31454000, 85427) = 0

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f3036d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f2f36d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f2e36d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f2d36d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f2c36d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f2b36d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f2a36d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f2936d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f2836d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f2736d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8f2636d000

```
mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f2536d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f2436d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f2336d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f2236d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f2136d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f2036d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f1f36d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f1e36d000

mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8f1d36d000

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1851757}) = 0

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=981457219}) = 0

write(1, "Memory allocating with library <", 32Memory allocating with library <) = 32

write(1, "Table", 5Table)                    = 5

write(1, "> lasts: ", 9> lasts: )               = 9

write(1, "979", 3979)                      = 3

write(1, " ms\n", 4 ms

)                    = 4

write(1, "50000", 550000)                    = 5

write(1, " int* allocated.\n", 17 int* allocated.

)      = 17

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=981495841}) = 0

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=1, tv_nsec=435449833}) = 0
```

```
write(1, "Memory deallocating with library"..., 34Memory deallocating with library <) =
34

write(1, "Table", 5Table)                      = 5

write(1, "> lasts: ", 9> lasts: )                = 9

write(1, "453", 3453)                        = 3

write(1, " ms\n", 4 ms

)                    = 4

write(1, "50000", 550000)                      = 5

write(1, " int* deallocated.\n", 19 int* deallocated.

)    = 19

munmap(0x7f8f1d36d000, 16777216)         = 0

munmap(0x7f8f1e36d000, 16777216)         = 0

munmap(0x7f8f1f36d000, 16777216)         = 0

munmap(0x7f8f2036d000, 16777216)         = 0

munmap(0x7f8f2136d000, 16777216)         = 0

munmap(0x7f8f2236d000, 16777216)         = 0

munmap(0x7f8f2336d000, 16777216)         = 0

munmap(0x7f8f2436d000, 16777216)         = 0

munmap(0x7f8f2536d000, 16777216)         = 0

munmap(0x7f8f2636d000, 16777216)         = 0

munmap(0x7f8f2736d000, 16777216)         = 0

munmap(0x7f8f2836d000, 16777216)         = 0

munmap(0x7f8f2936d000, 16777216)         = 0

munmap(0x7f8f2a36d000, 16777216)         = 0

munmap(0x7f8f2b36d000, 16777216)         = 0

munmap(0x7f8f2c36d000, 16777216)         = 0

munmap(0x7f8f2d36d000, 16777216)         = 0

munmap(0x7f8f2e36d000, 16777216)         = 0
```

```
munmap(0x7f8f2f36d000, 16777216)        = 0
munmap(0x7f8f3036d000, 16777216)        = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

```
$ strace ./output/main
execve("./output/main", ["./output/main"], 0x7ffcf763a520 /* 49 vars */) = 0
brk(NULL)                               = 0x55bc3504c000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffffadc7560) = -1 EINVAL (Недопустимый аргумент)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7ff23cfbd000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=85427, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 85427, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff23cfa8000
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) = 0x7ff23cd7f000
mprotect(0x7ff23cda7000, 2023424, PROT_NONE) = 0
mmap(0x7ff23cda7000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7ff23cda7000
mmap(0x7ff23cf3c000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7ff23cf3c000
mmap(0x7ff23cf95000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7ff23cf95000
mmap(0x7ff23cf9b000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ff23cf9b000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff23cd7c000
arch_prctl(ARCH_SET_FS, 0x7ff23cd7c740) = 0
set_tid_address(0x7ff23cd7ca10)         = 10162
set_robust_list(0x7ff23cd7ca20, 24)     = 0
rseq(0x7ff23cd7d0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7ff23cf95000, 16384, PROT_READ) = 0
mprotect(0x55bc16ab7000, 4096, PROT_READ) = 0
mprotect(0x7ff23cff7000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7ff23cfa8000, 85427)           = 0
getrandom("\xb7\x8b\xff\xc1\xfc\xe8\xbc\xf7", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55bc3504c000
brk(0x55bc3506d000)                     = 0x55bc3506d000
openat(AT_FDCWD, "./lib/libstd.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|0775, st_size=15592, ...}, AT_EMPTY_PATH) = 0
getcwd("/home/gaalex/Programs/OS/Fourth_lab", 128) = 36
mmap(NULL, 16440, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff23cfb8000
```

```
mmap(0x7ff23cfb9000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7ff23cfb9000

mmap(0x7ff23cfba000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7ff23cfba000

mmap(0x7ff23cfbb000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7ff23cfbb000

close(3)                                = 0

mprotect(0x7ff23cfbb000, 4096, PROT_READ) = 0

mmap(NULL, 401408, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff23cd1a000

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1307671}) = 0

brk(0x55bc3508e000)                     = 0x55bc3508e000

brk(0x55bc350af000)                     = 0x55bc350af000

brk(0x55bc350d0000)                     = 0x55bc350d0000

brk(0x55bc350f1000)                     = 0x55bc350f1000

brk(0x55bc35112000)                     = 0x55bc35112000

brk(0x55bc35133000)                     = 0x55bc35133000

brk(0x55bc35154000)                     = 0x55bc35154000

brk(0x55bc35175000)                     = 0x55bc35175000

brk(0x55bc35196000)                     = 0x55bc35196000

brk(0x55bc351b7000)                     = 0x55bc351b7000

brk(0x55bc351d8000)                     = 0x55bc351d8000

brk(0x55bc351f9000)                     = 0x55bc351f9000

brk(0x55bc3521a000)                     = 0x55bc3521a000

brk(0x55bc3523b000)                     = 0x55bc3523b000

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=4509518}) = 0

write(1, "Memory allocating with library <", 32Memory allocating with library <) = 32

write(1, "Standart", 8Standart)                 = 8

write(1, "> lasts: ", 9> lasts: )                = 9
```

```
write(1, "3", 13)                       = 1
write(1, " ms\n", 4 ms
)                   = 4
write(1, "50000", 550000)                   = 5
write(1, " int* allocated.\n", 17 int* allocated.
)      = 17
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=4604146}) = 0
munmap(0x7ff23cd1a000, 401408)          = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=5937177}) = 0
write(1, "Memory deallocating with library"..., 34Memory deallocating with library <) =
34
write(1, "Standart", 8Standart)                 = 8
write(1, "> lasts: ", 9> lasts: )                = 9
write(1, "1", 11)                       = 1
write(1, " ms\n", 4 ms
)                   = 4
write(1, "50000", 550000)                   = 5
write(1, " int* deallocated.\n", 19 int* deallocated.
)    = 19
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Я реализовал два алгоритма аллокации памяти: списочный и со степенями двойки. Во время выполнения лабораторный работы с какими-либо сложностями кроме дебага я не столкнулся.