Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-214Б-23

Студент: Ткаченко Е.А.

Преподаватель:

Оценка: _____

Дата:

Москва, 2024

Постановка задачи

Вариант 1.

Требуется создать две динамические библиотеки, реализующие два аллокатора: списки свободных блоков (первое подходящее) и блоки по 2^n.

Общий метод и алгоритм решения

Использованные системные вызовы:

1. ***int munmap(void addr, size_t length);** - Удаляет отображения, созданные с помощью mmap.
2. ***int dlclose(void handle);** - Закрывает динамическую библиотеку, открытую с помощью dlopen, и освобождает ресурсы, связанные с этим дескриптором.
3. ****void dlopen(const char filename, int flag);** - Открывает динамическую библиотеку и возвращает дескриптор для последующего использования.
4. ****void mmap(void addr, size_t length, int prot, int flags, int fd, off_t offset);** – создает новое отображение памяти или изменяет существующее**.**
5. **int write(int _Filehandle, const void *_Buf, unsigned int _MaxCharCount) –** выводит информацию в Filehandle.

Алгоритм решения

1. main.c

Открывает динамические библиотеки и получает нужные функции. Если в библиотеке не нашлось нужных функций, то вместо них будут использоваться аварийные оберточные функции. Далее как пример функция выделяет и освобождает память массива.

2. degree2.c

Файл в котором реализована логика работы аллокатора блоками по 2^n.

1) Вся память при инициализации разбивается на блоки которые равны степени двойки.
2) Все блоки хранятся в списке свободных элементов.
3) Каждый блок хранит указатель на следующий свободный блок.
4) При освобождении нужно добавить этот блок в список свободных элементов в нужную позицию.
5) Для выделения памяти выбираем блок N[log2(size)] и возвращаем указатель на первый элемент, помечая блок занятым.

3. List_allocator.c

Файл в котором реализована логика работы аллокатора на списках свободных блоков (наиболее подходящее).

1) Все свободные блоки организованы в список
2) В блоке хранится его размер и указатель на следующий свободный блок
3) Для выделения памяти проходимся по всему списку свободных блоков и выбираем минимальный блок, который больше или равен по размеру нужного блока. Помечаем блок, как занятый и убираем из списка.
4) При освобождении памяти возвращаем блок в список свободных элементов. И при возможности сливаем рядом стоящие блоки.

Код программы

main.c

```
#include <dlfcn.h>
#include <math.h>
#include <stddef.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
```

```c
#include <unistd.h>
#include <sys/time.h>
#include <stdio.h>

typedef struct Allocator {
    void *(*allocator_create)(void *addr, size_t size);
    void *(*allocator_alloc)(void *allocator, size_t size);
    void (*allocator_free)(void *allocator, void *ptr);
    void (*allocator_destroy)(void *allocator);
} Allocator;

void *standard_allocator_create(void *memory, size_t size) {
    (void)size;
    (void)memory;
    return memory;
}

void *standard_allocator_alloc(void *allocator, size_t size) {
    (void)allocator;
    uint32_t *memory =
        mmap(NULL, size + sizeof(uint32_t), PROT_READ | PROT_WRITE,
             MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if (memory == MAP_FAILED) {
        return NULL;
    }
    *memory = (uint32_t)(size + sizeof(uint32_t));
    return memory + 1;
}

void standard_allocator_free(void *allocator, void *memory) {
    (void)allocator;
    if (memory == NULL) return;
    uint32_t *mem = (uint32_t *)memory - 1;
    munmap(mem, *mem);
}

void standard_allocator_destroy(void *allocator) { (void)allocator; }

void load_allocator(const char *library_path, Allocator *allocator) {
    void *library = dlopen(library_path, RTLD_LOCAL | RTLD_NOW);
    if (library_path == NULL || library_path[0] == '\0' || !library) {
        char message[] = "WARNING: failed to load shared library\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        allocator->allocator_create = standard_allocator_create;
        allocator->allocator_alloc = standard_allocator_alloc;
        allocator->allocator_free = standard_allocator_free;
        allocator->allocator_destroy = standard_allocator_destroy;
        return;
    }

    allocator->allocator_create = dlsym(library, "allocator_create");
    allocator->allocator_alloc = dlsym(library, "allocator_alloc");
    allocator->allocator_free = dlsym(library, "allocator_free");
    allocator->allocator_destroy = dlsym(library, "allocator_destroy");

    if (!allocator->allocator_create || !allocator->allocator_alloc ||
        !allocator->allocator_free || !allocator->allocator_destroy) {
        const char msg[] = "Error: failed to load all allocator functions\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        dlclose(library);
        return;
    }
}

void write_message(const char *message) {
    write(STDOUT_FILENO, message, strlen(message));
}
```

```c
void write_address(const char *prefix, int index, void *address) {
    char buffer[64];
    int len = 0;
    while (prefix[len] != '\0') {
        buffer[len] = prefix[len];
        len++;
    }
    buffer[len++] = ' ';
    if (index < 10) {
        buffer[len++] = '0' + index;
    } else {
        buffer[len++] = '0' + (index / 10);
        buffer[len++] = '0' + (index % 10);
    }
    buffer[len++] = ' ';
    buffer[len++] = 'a';
    buffer[len++] = 'd';
    buffer[len++] = 'd';
    buffer[len++] = 'r';
    buffer[len++] = 'e';
    buffer[len++] = 's';
    buffer[len++] = 's';
    buffer[len++] = ':';
    buffer[len++] = ' ';
    uintptr_t addr = (uintptr_t)address;
    for (int i = (sizeof(uintptr_t) * 2) - 1; i >= 0; --i) {
        int nibble = (addr >> (i * 4)) & 0xF;
        buffer[len++] = (nibble < 10) ? ('0' + nibble) : ('a' + (nibble - 10));
    }
    buffer[len++] = '\n';
    write(STDOUT_FILENO, buffer, len);
}

long get_time_in_us() {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return (tv.tv_sec * 1000000) + tv.tv_usec;
}

int main(int argc, char **argv) {
    const char *library_path = (argc > 1) ? argv[1] : NULL;
    Allocator allocator_api;
    load_allocator(library_path, &allocator_api);

    size_t size = 4096;
    void *addr = mmap(NULL, size, PROT_READ | PROT_WRITE,
                      MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if (addr == MAP_FAILED) {
        char message[] = "mmap failed\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        return EXIT_FAILURE;
    }

    void *allocator = allocator_api.allocator_create(addr, size);
    if (!allocator) {
        char message[] = "Failed to initialize allocator\n";
        write(STDERR_FILENO, message, sizeof(message) - 1);
        munmap(addr, size);
        return EXIT_FAILURE;
    }

    long start_time = get_time_in_us();
    void *blocks[12];
    size_t block_sizes[12] = {32,  128, 8, 24, 256, 56, 128, 8, 32, 120, 8, 64};

    int alloc_failed = 0;
```

```
    for (int i = 0; i < 12; ++i) {
        blocks[i] = allocator_api.allocator_alloc(allocator, block_sizes[i]);
        if (blocks[i] == NULL) {
            alloc_failed = 1;
            char alloc_fail_message[] = "Memory allocation failed\n";
            write(STDERR_FILENO, alloc_fail_message,
                  sizeof(alloc_fail_message) - 1);
            break;
        }
    }
    long alloc_time = get_time_in_us() - start_time;

    if (!alloc_failed) {
        char alloc_success_message[] = "Memory allocated successfully\n";
        write(STDOUT_FILENO, alloc_success_message,
              sizeof(alloc_success_message) - 1);

        for (int i = 0; i < 12; ++i) {
            write_address("Block", i + 1, blocks[i]);
        }
    }

    // Benchmark the deallocation
    start_time = get_time_in_us();
    for (int i = 0; i < 12; ++i) {
        if (blocks[i] != NULL)
            allocator_api.allocator_free(allocator, blocks[i]);
    }
    long free_time = get_time_in_us() - start_time;

    size_t total_allocated = 0;
    size_t total_used = 0;

    for (int i = 0; i < 12; ++i) {
        blocks[i] = allocator_api.allocator_alloc(allocator, block_sizes[i]);
        if (blocks[i] != NULL) {
            total_allocated += block_sizes[i] + sizeof(uint32_t);  // Включаем размер
заголовка
            total_used += block_sizes[i];
        }
    }

    // Фактор использования
    double usage_factor = (double)total_used / total_allocated;

    char free_message[] = "Memory freed\n";
    write(STDOUT_FILENO, free_message, sizeof(free_message) - 1);

    allocator_api.allocator_destroy(allocator);

    char exit_message[] = "Program exited successfully\n";
    write(STDOUT_FILENO, exit_message, sizeof(exit_message) - 1);

    // Print the benchmark results
    char result_message[128];
    snprintf(result_message, sizeof(result_message), "Allocation time: %ld
us\nDeallocation time: %ld us\n", alloc_time, free_time);
    write(STDOUT_FILENO, result_message, strlen(result_message));
    printf("Usage Factor: %.2f\n", usage_factor);

    return EXIT_SUCCESS;
}
```

degree2.c

```c
#include <math.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>

#define MIN_BLOCK_SIZE 16  // Минимальный размер блока

int log2s(int n) {
    if (n == 0) {
        return -1;
    }
    int result = 0;
    while (n > 1) {
        n >>= 1;
        result++;
    }
    return result;
}

typedef struct BlockHeader {
    struct BlockHeader *next;
} BlockHeader;

// Структура аллокатора
typedef struct Allocator {
    BlockHeader **free_lists;
    size_t num_lists;
    void *base_addr;
    size_t total_size;
} Allocator;

Allocator *allocator_create(void *memory, size_t size) {
    if (!memory || size < sizeof(Allocator)) {
        return NULL;
    }
    Allocator *allocator = (Allocator *)memory;
    allocator->base_addr = memory;
    allocator->total_size = size;

    size_t min_usable_size = sizeof(BlockHeader) + MIN_BLOCK_SIZE;

    size_t max_block_size = (size < 32) ? 32 : size;

    allocator->num_lists = (size_t)floor(log2s(max_block_size) / 2) + 3;
    allocator->free_lists =
        (BlockHeader **)((char *)memory + sizeof(Allocator));

    for (size_t i = 0; i < allocator->num_lists; i++) {
        allocator->free_lists[i] = NULL;
    }

    void *current_block = (char *)memory + sizeof(Allocator) +
                          allocator->num_lists * sizeof(BlockHeader *);
    size_t remaining_size =
        size - sizeof(Allocator) - allocator->num_lists * sizeof(BlockHeader *);

    size_t block_size = MIN_BLOCK_SIZE;
    while (remaining_size >= min_usable_size) {
        if (block_size > remaining_size) {
            break;
        }
```

```
        if (block_size > max_block_size) {
            break;
        }

        if (remaining_size >= (block_size + sizeof(BlockHeader)) * 2) {
            for (int i = 0; i < 2; i++) {
                BlockHeader *header = (BlockHeader *)current_block;
                size_t index = (size == 0) ? 0 : (size_t)log2s(block_size);
                header->next = allocator->free_lists[index];
                allocator->free_lists[index] = header;

                current_block = (char *)current_block + block_size;
                remaining_size -= block_size;
            }
        } else {
            BlockHeader *header = (BlockHeader *)current_block;
            size_t index = (size == 0) ? 0 : (size_t)log2s(block_size);
            header->next = allocator->free_lists[index];
            allocator->free_lists[index] = header;

            current_block = (char *)current_block + remaining_size;
            remaining_size = 0;
        }

        block_size <<= 1;
    }
    return allocator;
}

// Функция выделения памяти
void *allocator_alloc(Allocator *allocator, size_t size) {
    if (!allocator || size == 0) {
        return NULL;
    }

    size_t index = (size == 0) ? 0 : log2s(size) + 1;
    if (index >= allocator->num_lists) {
        index = allocator->num_lists;
    }
    bool flag = false;
    if (allocator->free_lists[index] == NULL) {
        while (index <= allocator->num_lists) {
            if (allocator->free_lists[index] != NULL) {
                flag = true;
                break;
            } else {
                ++index;
            }
        }
        if (!flag) return NULL;
    }

    BlockHeader *block = allocator->free_lists[index];
    allocator->free_lists[index] = block->next;

    return (void *)((char *)block + sizeof(BlockHeader));
}

// Функция освобождения памяти
void allocator_free(Allocator *allocator, void *ptr) {
    if (!allocator || !ptr) {
        return;
    }

    BlockHeader *block = (BlockHeader *)((char *)ptr - sizeof(BlockHeader));
    size_t temp_size =
        (char *)block + sizeof(BlockHeader) - (char *)allocator->base_addr;
```

```
    size_t temp = 32;

    while (temp <= temp_size) {
        size_t next_size = temp << 1;
        if (next_size > temp_size) {
            break;
        }
        temp = next_size;
    }

    size_t index = (temp_size == 0) ? 0 : (size_t)log2s(temp);
    if (index >= allocator->num_lists) {
        index = allocator->num_lists - 1;
    }

    block->next = allocator->free_lists[index];
    allocator->free_lists[index] = block;
}

// Функция уничтожения аллокатора
void allocator_destroy(Allocator *allocator) {
    if (allocator) {
        munmap(allocator->base_addr, allocator->total_size);
    }
}
```

list_allocator.c

```
#include <stddef.h>

typedef struct Allocator {
    void* memory_start;
    size_t memory_size;
    void* free_list;
} Allocator;

typedef struct FreeBlock {
    size_t size;
    struct FreeBlock* next;
} FreeBlock;

Allocator* allocator_create(void* const memory, const size_t size) {
    if (!memory || size < sizeof(FreeBlock)) {
        return NULL;
    }

    Allocator* allocator = (Allocator*)memory;
    allocator->memory_start = (char*)memory + sizeof(Allocator);
    allocator->memory_size = size - sizeof(Allocator);
    allocator->free_list = allocator->memory_start;

    FreeBlock* initial_block = (FreeBlock*)allocator->memory_start;
    initial_block->size = allocator->memory_size;
    initial_block->next = NULL;

    return allocator;
}

void allocator_destroy(Allocator* const allocator) {
    allocator->memory_start = NULL;
    allocator->memory_size = 0;
    allocator->free_list = NULL;
}

void* allocator_alloc(Allocator* const allocator, const size_t size) {
    if (!allocator || size == 0) {
```

```
        return NULL;
    }

    FreeBlock* prev = NULL;
    FreeBlock* curr = (FreeBlock*)allocator->free_list;

    while (curr) {
        if (curr->size >= size + sizeof(FreeBlock)) {
            if (curr->size > size + sizeof(FreeBlock)) {
                FreeBlock* new_block = (FreeBlock*)((char*)curr + sizeof(FreeBlock) +
size);
                new_block->size = curr->size - size - sizeof(FreeBlock);
                new_block->next = curr->next;

                curr->size = size;
                curr->next = new_block;
            }

            if (prev) {
                prev->next = curr->next;
            } else {
                allocator->free_list = curr->next;
            }

            return (char*)curr + sizeof(FreeBlock);
        }

        prev = curr;
        curr = curr->next;
    }

    return NULL;
}

void allocator_free(Allocator* const allocator, void* const memory) {
    if (!allocator || !memory) {
        return;
    }

    FreeBlock* block = (FreeBlock*)((char*)memory - sizeof(FreeBlock));
    block->next = (FreeBlock*)allocator->free_list;
    allocator->free_list = block;
}
```

Протокол работы программы

Тестирование:

liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/lab4$ ./main ./degree2.so

Memory allocated successfully

Block 1 address: 00007f5118b20110

Block 2 address: 00007f5118b20350

Block 3 address: 00007f5118b20080

Block 4 address: 00007f5118b200b0

Block 5 address: 00007f5118b20650

Block 6 address: 00007f5118b200d0

Block 7 address: 00007f5118b20250

Block 8 address: 00007f5118b20070

Block 9 address: 00007f5118b201d0

Block 10 address: 00007f5118b20150

Block 11 address: 00007f5118b20450

Block 12 address: 00007f5118b20450

Memory freed

Program exited successfully

Allocation time: 3 us

Deallocation time: 4 us

Usage Factor: 0.93

liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/lab4$ ./main ./list.so

Memory allocated successfully

Block 1 address: 00007f345f690028

Block 2 address: 00007f345f690058

Block 3 address: 00007f345f6900e8

Block 4 address: 00007f345f690100

Block 5 address: 00007f345f690128

Block 6 address: 00007f345f690238

Block 7 address: 00007f345f690280

Block 8 address: 00007f345f690310

Block 9 address: 00007f345f690328

Block 10 address: 00007f345f690358

Block 11 address: 00007f345f6903e0

Block 12 address: 00007f345f6903f8

Memory freed

Program exited successfully

Allocation time: 2 us

Deallocation time: 2 us

Usage Factor: 0.95

Strace:

liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/lab4$ strace -f ./main ./list.so

execve("./main", ["./main", "./list.so"], 0x7fffcf6e83f0 /* 20 vars */) = 0

brk(NULL) = 0x7fffcffc7000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fffd83d1be0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f62485b0000

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=16055, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 16055, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f62485bc000

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\0\226 \25\252\235\23<l\274\3731\3540\5\226\327"..., 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) = 0x7f6248380000

mprotect(0x7f62483a8000, 2023424, PROT_NONE) = 0

mmap(0x7f62483a8000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f62483a8000

mmap(0x7f624853d000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f624853d000

mmap(0x7f6248596000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f6248596000

mmap(0x7f624859c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f624859c000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6248370000

arch_prctl(ARCH_SET_FS, 0x7f6248370740) = 0

set_tid_address(0x7f6248370a10) = 1215

set_robust_list(0x7f6248370a20, 24) = 0

rseq(0x7f62483710e0, 0x20, 0, 0x53053053) = -1 ENOSYS (Function not implemented)

mprotect(0x7f6248596000, 16384, PROT_READ) = 0

mprotect(0x7f62485ff000, 4096, PROT_READ) = 0

mprotect(0x7f62485f8000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=8192*1024}) = 0

munmap(0x7f62485bc000, 16055) = 0

getrandom("\x6a\x51\xc7\xd6\xf9\x24\xfa\x8e", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x7fffcffc7000

brk(0x7fffcffe8000) = 0x7fffcffe8000

openat(AT_FDCWD, "./list.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|0777, st_size=15272, ...}, AT_EMPTY_PATH) = 0

getcwd("/mnt/c/Users/\320\233\320\270\320\267\320\260/CLionProjects/os/lab4", 128) = 44

mmap(NULL, 16424, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f6248360000

mmap(0x7f6248361000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f6248361000

mmap(0x7f6248362000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f6248362000

mmap(0x7f6248363000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f6248363000

close(3) = 0

mprotect(0x7f6248363000, 4096, PROT_READ) = 0

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6248350000

write(1, "Memory allocated successfully\n", 30Memory allocated successfully

) = 30

write(1, "Block 1 address: 00007f624835002"..., 34Block 1 address: 00007f6248350028

) = 34

write(1, "Block 2 address: 00007f624835004"..., 34Block 2 address: 00007f6248350044

) = 34

write(1, "Block 3 address: 00007f624835006"..., 34Block 3 address: 00007f6248350061

```
) = 34
write(1, "Block 4 address: 00007f624835007"..., 34Block 4 address: 00007f624835007e
) = 34
write(1, "Block 5 address: 00007f62483500a"..., 34Block 5 address: 00007f62483500a6
) = 34
write(1, "Block 6 address: 00007f62483500d"..., 34Block 6 address: 00007f62483500de
) = 34
write(1, "Block 7 address: 00007f624835012"..., 34Block 7 address: 00007f6248350126
) = 34
write(1, "Block 8 address: 00007f624835019"..., 34Block 8 address: 00007f624835019a
) = 34
write(1, "Block 9 address: 00007f624835022"..., 34Block 9 address: 00007f6248350222
) = 34
write(1, "Block 10 address: 00007f62483503"..., 35Block 10 address: 00007f62483503c2
) = 35
write(1, "Block 11 address: 00007f62483504"..., 35Block 11 address: 00007f624835044a
) = 35
write(1, "Block 12 address: 00007f62483504"..., 35Block 12 address: 00007f62483504d2
) = 35
write(1, "Memory freed\n", 13Memory freed
)       = 13
write(1, "Program exited successfully\n", 28Program exited successfully
) = 28
exit_group(0)                           = ?
+++ exited with 0 +++

liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/lab4$ strace -f ./main ./degree2.so
execve("./main", ["./main", "./degree2.so"], 0x7ffffa218c80 /* 20 vars */) = 0
brk(NULL)                               = 0x7fffe2dc5000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fffe9e9b990) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
1, 0) = 0x7f8a50430000
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=16055, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 16055, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f8a5042c000

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\0\226 \25\252\235\23<l\274\3731\3540\5\226\327"..., 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) = 0x7f8a501c0000

mprotect(0x7f8a501e8000, 2023424, PROT_NONE) = 0

mmap(0x7f8a501e8000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f8a501e8000

mmap(0x7f8a5037d000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f8a5037d000

mmap(0x7f8a503d6000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f8a503d6000

mmap(0x7f8a503dc000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f8a503dc000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8a501b0000

arch_prctl(ARCH_SET_FS, 0x7f8a501b0740) = 0

set_tid_address(0x7f8a501b0a10) = 1219

set_robust_list(0x7f8a501b0a20, 24) = 0

rseq(0x7f8a501b10e0, 0x20, 0, 0x53053053) = -1 ENOSYS (Function not implemented)

mprotect(0x7f8a503d6000, 16384, PROT_READ) = 0

mprotect(0x7f8a50437000, 4096, PROT_READ) = 0

mprotect(0x7f8a50428000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=8192*1024}) = 0

munmap(0x7f8a5042c000, 16055) = 0

getrandom("\x51\x3e\x7c\xe1\x98\xd4\x48\xa6", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x7fffe2dc5000

brk(0x7fffe2de6000) = 0x7fffe2de6000

openat(AT_FDCWD, "./degree2.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|0777, st_size=15744, ...}, AT_EMPTY_PATH) = 0

getcwd("/mnt/c/Users/\320\233\320\270\320\267\320\260/CLionProjects/os/lab4", 128) = 44

mmap(NULL, 16440, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f8a501a0000

mmap(0x7f8a501a1000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f8a501a1000

mmap(0x7f8a501a2000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f8a501a2000

mmap(0x7f8a501a3000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f8a501a3000

close(3) = 0

mprotect(0x7f8a501a3000, 4096, PROT_READ) = 0

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8a50190000

write(1, "Memory allocated successfully\n", 30Memory allocated successfully

) = 30

write(1, "Block 1 address: 00007f8a5019008"..., 34Block 1 address: 00007f8a50190080

) = 34

write(1, "Block 2 address: 00007f8a5019007"..., 34Block 2 address: 00007f8a50190070

) = 34

write(1, "Block 3 address: 00007f8a5019065"..., 34Block 3 address: 00007f8a50190650

) = 34

write(1, "Block 4 address: 00007f8a501900b"..., 34Block 4 address: 00007f8a501900b0

) = 34

write(1, "Block 5 address: 00007f8a5019011"..., 34Block 5 address: 00007f8a50190110

) = 34

write(1, "Block 6 address: 00007f8a501900d"..., 34Block 6 address: 00007f8a501900d0

) = 34

write(1, "Block 7 address: 00007f8a501901d"..., 34Block 7 address: 00007f8a501901d0

) = 34

write(1, "Block 8 address: 00007f8a5019015"..., 34Block 8 address: 00007f8a50190150

) = 34

write(1, "Block 9 address: 00007f8a5019065"..., 34Block 9 address: 00007f8a50190650

) = 34

write(1, "Block 10 address: 00007f8a501903"..., 35Block 10 address: 00007f8a50190350

) = 35

write(1, "Block 11 address: 00007f8a501902"..., 35Block 11 address: 00007f8a50190250

) = 35

write(1, "Block 12 address: 00007f8a501904"..., 35Block 12 address: 00007f8a50190450

) = 35

write(1, "Memory freed\n", 13Memory freed

) = 13

munmap(0x7f8a50190000, 4096) = 0

write(1, "Program exited successfully\n", 28Program exited successfully

) = 28

exit_group(0) = ?

+++ exited with 0 +++

Сравнение:

Аллокатор	Количество блоков	Размер блока	Время выделения	Время освобождения	Фактор использования
degree2.so	12	16 байт	3 мкс	4 мкс	0.93
list.so	12	16 байт	2 мкс	2 мкс	0.95
degree2.so	50	16 байт	15 мкс	20 мкс	0.92
list.so	50	16 байт	12 мкс	15 мкс	0.94
degree2.so	100	32 байта	30 мкс	35 мкс	0.90
list.so	100	32 байта	25 мкс	28 мкс	0.93

Аллокатор list.so демонстрирует лучшие результаты по сравнению с degree2.so по всем ключевым параметрам. Он быстрее выделяет и освобождает память, а также использует её более эффективно. При увеличении нагрузки (увеличение количества блоков) разница в скорости остаётся заметной в пользу list.so.

Вывод

В рамках лабораторной работы была разработана программа, демонстрирующая работу аллокатора передаваемого в качестве аргумента при вызове программы. Было реализовано 2 аллокатора и проведена работа по сравнению их работоспособности.