

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-213Б-23

Студент: Петров М.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_

Дата: 01.12.24

Москва, 2024

# Постановка задачи

## Вариант 4.

Алгоритм Мак-Кьюзика-Кэрелса и блоки по  $2^n$ ;

## Общий метод и алгоритм решения

Использованные системные вызовы:

1. `dlopen` — используется для открытия динамической библиотеки.
2. `dlsym` — используется для получения адреса символа (функции или переменной) из динамически загружаемой библиотеки.
3. `dlclose` — используется для закрытия дескриптора динамической библиотеки.
4. `mmap` — используется для отображения файла или устройства памяти в адресное пространство процесса.
5. `munmap` — используется для отмены отображения ранее отображенного файла или области памяти в адресное пространство процесса.
6. `write` — используется для записи данных в файл или устройство. В данном случае он используется для записи сообщений об ошибках в стандартный поток ошибок.

Программа загружает динамическую библиотеку с помощью функции ``dlopen``, затем получает адреса необходимых функций из этой библиотеки через ``dlsym``. После этого она использует функцию ``mmap`` для отображения блока памяти заданного размера в адресное пространство процесса. Затем создается объект аллокатора с использованием загруженной из библиотеки функции ``allocator_create``. Далее программа пытается выделить блок памяти определенного размера с помощью функции ``allocator_alloc`` и, если это удастся, выводит его адрес. Потом освобождает выделенный блок с помощью функции ``allocator_free``. В конце программа уничтожает объект аллокатора и освобождает память с помощью ``allocator_destroy`` и ``munmap``.

## Код программы

### main.c

```
#include <dlfcn.h>
#include <sys/mman.h>
#include <unistd.h>
#include <string.h>

#define MEMORY_SIZE 1024 * 1024

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

Allocator* allocator_create(void* const memory, const size_t size);
void allocator_destroy(Allocator* const allocator);
void* allocator_alloc(Allocator* const allocator, const size_t size);
void allocator_free(Allocator* const allocator, void* const memory);

void my_write(const char* message) {
    write(STDERR_FILENO, message, strlen(message));
}
```

```

void my_write_hex(void* ptr) {
    char buffer[64];
    unsigned long addr = (unsigned long)ptr;
    size_t i;
    for (i = 0; i < sizeof(buffer) - 1 && addr; ++i) {
        unsigned char byte = addr & 0xF;
        buffer[i] = (byte < 10) ? '0' + byte : 'a' + (byte - 10);
        addr >>= 4;
    }
    buffer[i] = '\\0';
    my_write(buffer);
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        my_write("Usage: <path_to_allocator_library>\\n");
        return 1;
    }

    void* handle = dlopen(argv[1], RTLD_LAZY);
    if (!handle) {
        my_write("Failed to load library: ");
        my_write(dlerror());
        my_write("\\n");
        return 1;
    }

    Allocator* (*allocator_create)(void*, size_t) = dlsym(handle, "allocator_create");
    void (*allocator_destroy)(Allocator*) = dlsym(handle, "allocator_destroy");
    void* (*allocator_alloc)(Allocator*, size_t) = dlsym(handle, "allocator_alloc");
    void (*allocator_free)(Allocator*, void*) = dlsym(handle, "allocator_free");

    char* error;
    if ((error = dlerror()) != NULL) {
        my_write("Error resolving symbols: ");
        my_write(error);
        my_write("\\n");
        dlclose(handle);
        return 1;
    }

    void* memory = mmap(NULL, MEMORY_SIZE, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS,
-1, 0);
    if (memory == MAP_FAILED) {
        my_write("mmap failed\\n");
        dlclose(handle);
        return 1;
    }

    Allocator* allocator = allocator_create(memory, MEMORY_SIZE);
    if (!allocator) {
        my_write("Failed to create allocator\\n");
        munmap(memory, MEMORY_SIZE);
        dlclose(handle);
        return 1;
    }
}

```

```

}

void* block = allocator_alloc(allocator, 128);
if (block) {
    my_write("Allocated block at ");
    my_write_hex(block);
    my_write("\n");
} else {
    my_write("Failed to allocate block\n");
}

allocator_free(allocator, block);
my_write("Freed block\n");

allocator_destroy(allocator);
munmap(memory, MEMORY_SIZE);
dlclose(handle);

return 0;
}

```

### allocator.c

```

#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>

#ifdef _MSC_VER
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif

#define MAX_BLOCK_SIZES 4

typedef struct
{
    size_t block_size;      // Размер одного блока
    size_t block_count;     // Количество блоков
    uint8_t *bitmap;        // Битовая карта для управления блоками
    uint8_t *memory_start;  // Указатель на начало памяти для этого пула
} MemoryPool;

typedef struct
{
    void *memory;            // Указатель на переданную память
    size_t memory_size;      // Размер памяти
    MemoryPool pools[MAX_BLOCK_SIZES]; // Пулы для разных размеров блоков
} Allocator;

static void write_message(const char *message)
{
    write(STDERR_FILENO, message, strlen(message));
}

```

```

}

EXPORT Allocator *allocator_create(void *mem, size_t mem_size)
{
    Allocator *allocator = (Allocator *)malloc(sizeof(Allocator));
    if (!allocator)
        return NULL;

    allocator->memory = mem;
    allocator->memory_size = mem_size;

    size_t block_sizes[MAX_BLOCK_SIZES] = {16, 32, 64, 128};

    uint8_t *current_memory = mem;
    for (int i = 0; i < MAX_BLOCK_SIZES; i++)
    {
        size_t block_size = block_sizes[i];
        size_t block_count = mem_size / block_size / MAX_BLOCK_SIZES;

        allocator->pools[i].block_size = block_size;
        allocator->pools[i].block_count = block_count;
        allocator->pools[i].bitmap = current_memory;
        allocator->pools[i].memory_start = current_memory + block_count / 8;

        memset(allocator->pools[i].bitmap, 0, block_count / 8);

        current_memory += block_count / 8 + block_count * block_size;
    }

    return allocator;
}

EXPORT void *allocator_alloc(Allocator *allocator, size_t size)
{
    for (int i = 0; i < MAX_BLOCK_SIZES; i++)
    {
        MemoryPool *pool = &allocator->pools[i];

        if (size > pool->block_size)
            continue;

        for (size_t j = 0; j < pool->block_count; j++)
        {
            size_t byte_index = j / 8;
            size_t bit_index = j % 8;

            if (!(pool->bitmap[byte_index] & (1 << bit_index)))
            {
                pool->bitmap[byte_index] |= (1 << bit_index);
                return pool->memory_start + j * pool->block_size;
            }
        }
    }

    return NULL;
}

```

```

EXPORT void allocator_free(Allocator *allocator, void *ptr)
{
    for (int i = 0; i < MAX_BLOCK_SIZES; i++)
    {
        MemoryPool *pool = &allocator->pools[i];

        if (ptr >= (void *)pool->memory_start && ptr < (void *) (pool->memory_start + pool->block_count * pool->block_size))
        {
            size_t offset = (uint8_t *)ptr - pool->memory_start;
            size_t index = offset / pool->block_size;
            size_t byte_index = index / 8;
            size_t bit_index = index % 8;

            pool->bitmap[byte_index] &= ~(1 << bit_index);
            return;
        }
    }
}

EXPORT void allocator_destroy(Allocator *allocator)
{
    if (allocator)
    {
        if (munmap(allocator->memory, allocator->memory_size) == -1)
        {
            const char error_msg[] = "Error: munmap failed\n";
            write_message(error_msg);
            exit(EXIT_FAILURE);
        }
        free(allocator);
    }
}

```

## 2.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);
    }
}

```

## Протокол работы программы

### Тестирование:

markvolkov@MacBook-Air-Mark-2 LAB4 % ./main alloc.so

Allocated block at 00fc11501

Freed block

markvolkov@MacBook-Air-Mark-2 LAB4 % ./main 2.so

Allocated block at 073cf7201

Freed block

markvolkov@MacBook-Air-Mark-2 LAB4 % ./main shf.so

Failed to load library: dlopen(shf.so, 0x0001): tried: 'shf.so' (no such file),  
 '/System/Volumes/Preboot/Cryptexes/OS/shf.so' (no such file), '/usr/lib/shf.so' (no such  
 file, not in dyld cache), 'shf.so' (no such file)

### Strace:

strace -f ./main ./2.so

execve("./main", ["/main", "/2.so"], 0x7fff371b4f0 /\* 46 vars \*/) = 0

brk(NULL) = 0x62d2f6d8f000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffd787f7cf0) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1,  
 0) = 0x7b0d2ec8c000

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=58047, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 58047, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7b0d2ec7d000

```

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\3\0>\0\1\0\0\0\0\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"..., 784, 64) = 784
pread64(3, "\4\0\0\0\0\0\0\5\0\0\0GNU\02\0\0\300\4\0\0\0\3\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I17\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"..., 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7b0d2ea00000
mprotect(0x7b0d2ea28000, 2023424, PROT_NONE) = 0
mmap(0x7b0d2ea28000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7b0d2ea28000
mmap(0x7b0d2ebbd000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7b0d2ebbd000
mmap(0x7b0d2ec16000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7b0d2ec16000
mmap(0x7b0d2ec1c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7b0d2ec1c000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7b0d2ec7a000
arch_prctl(ARCH_SET_FS, 0x7b0d2ec7a740) = 0
set_tid_address(0x7b0d2ec7aa10) = 2173
set_robust_list(0x7b0d2ec7aa20, 24) = 0
rseq(0x7b0d2ec7b0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7b0d2ec16000, 16384, PROT_READ) = 0
mprotect(0x62d2e073e000, 4096, PROT_READ) = 0
mprotect(0x7b0d2ecc6000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0
munmap(0x7b0d2ec7d000, 58047) = 0
getrandom("\xf9\x16\x59\x4e\x90\xde\x5a\x6c", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x62d2f6d8f000
brk(0x62d2f6db0000) = 0x62d2f6db0000
openat(AT_FDCWD, "/.2.so", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0775, st_size=15736, ...}, AT_EMPTY_PATH) = 0
getcwd("/home/artemdelgray/Telegram Desktop", 128) = 53
mmap(NULL, 16440, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7b0d2ec87000
mmap(0x7b0d2ec88000, 4096, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7b0d2ec88000
mmap(0x7b0d2ec89000, 4096, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7b0d2ec89000
mmap(0x7b0d2ec8a000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7b0d2ec8a000
close(3) = 0
mprotect(0x7b0d2ec8a000, 4096, PROT_READ) = 0
mmap(NULL, 1048576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7b0d2e900000
write(2, "Allocated block at ", 19Allocated block at ) = 19

```

```
write(2, "073009e2d0b7", 12073009e2d0b7)    = 12
write(2, "\n", 1
)          = 1
write(2, "Freed block\n", 12Freed block
)          = 12
munmap(0x7b0d2e900000, 1048576)    = 0
munmap(0x7b0d2e900000, 1048576)    = 0
munmap(0x7b0d2ec87000, 16440)      = 0
exit_group(0)                      = ?
+++ exited with 0 +++
```

## Вывод

**Программа демонстрирует работу с динамическими библиотеками, управление памятью с помощью `mmap`, создание и использование собственного аллокатора, а также включает функции для отладки и логирования.**