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: Нугаев М. Э. Преподаватель: Бахарев В.Д.

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

Дата: 07.01.25

Москва, 2025

# Постановка задачи

**Вариант 2.**Списки свободных блоков (первое подходящее) и алгоритм Мак-КьюзиКэрелса.

# Общий метод и алгоритм решения

Использованные системные вызовы:

* ssize\_t write(int \_\_fd, const void \*\_\_buf, size\_t \_\_n); – записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
* void exit(int \_\_status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
* void \* mmap(void \*start, size\_t length, int prot , int flags, int fd, off\_t offset); – отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым дескриптором fd, в память, начиная с адреса start.
* int munmap(void \*start, size\_t length); – удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку

"неправильное обращение к памяти".

* void \*dlopen(const char \*filename, int flag); – загружает динамическую библиотеку, имя которой указано в строке filename, и возвращает прямой указатель на начало динамической библиотеки.
* void \*dlsym(void \*handle, char \*symbol); – использует указатель на динамическую библиотеку, возвращаемую dlopen, и оканчивающееся нулем символьное имя, а затем возвращает адрес, указывающий, откуда загружается этот символ. Если символ не найден, то возвращаемым значением dlsym является NULL;
* int dlclose(void \*handle); – уменьшает на единицу счетчик ссылок на указатель динамической библиотеки.

# Код программы

## main.c

#include <stdio.h>

#include <stdlib.h>

#include <dlfcn.h>

#include <sys/mman.h>

#include <time.h>

typedef struct AllocatorAPI {

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

    void (\*allocator\_destroy)(void\*);

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

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

} AllocatorAPI;

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

    return memory;

}

void default\_allocator\_destroy(void\* allocator) {

}

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

    if (allocator) {

        return (void\*)((char\*)allocator + sizeof(size\_t));

    }

    return NULL;

}

void default\_allocator\_free(void\* allocator, void\* memory) {

}

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

    AllocatorAPI api;

    void\* library\_handle = NULL;

    api.allocator\_create = default\_allocator\_create;

    api.allocator\_destroy = default\_allocator\_destroy;

    api.allocator\_alloc = default\_allocator\_alloc;

    api.allocator\_free = default\_allocator\_free;

    if (argc > 1) {

        library\_handle = dlopen(argv[1], RTLD\_LAZY);

        if (!library\_handle) {

            fprintf(stderr, "Ошибка при загрузке библиотеки: %s\n", dlerror());

            return 1;

        }

        api.allocator\_create = dlsym(library\_handle, "allocator\_create");

        api.allocator\_destroy = dlsym(library\_handle, "allocator\_destroy");

        api.allocator\_alloc = dlsym(library\_handle, "allocator\_alloc");

        api.allocator\_free = dlsym(library\_handle, "allocator\_free");

        if (!api.allocator\_create || !api.allocator\_destroy || !api.allocator\_alloc || !api.allocator\_free) {

            fprintf(stderr, "Ошибка при загрузке функций из библиотеки: %s\n", dlerror());

            dlclose(library\_handle);

            return 1;

        }

    } else {

        fprintf(stderr, "Библиотека не указана. Используется стандартный аллокатор.\n");

    }

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

    void\* memory = mmap(NULL, pool\_size, PROT\_READ | PROT\_WRITE, MAP\_ANONYMOUS | MAP\_PRIVATE, -1, 0);

    if (memory == MAP\_FAILED) {

        perror("Ошибка mmap");

        return 1;

    }

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

    if (!allocator) {

        fprintf(stderr, "Ошибка при создании аллокатора.\n");

        munmap(memory, pool\_size);

        return 1;

    }

    struct timespec start, end;

    double time\_taken;

    for (int i = 0; i < 3; i++) {

        size\_t block\_size = 1024 \* (i + 1);

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

        void\* ptr = api.allocator\_alloc(allocator, block\_size);

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

        if (!ptr) {

            fprintf(stderr, "Ошибка при выделении памяти для блока размера %zu байт.\n", block\_size);

            break;

        }

        time\_taken = (end.tv\_sec - start.tv\_sec) + (end.tv\_nsec - start.tv\_nsec) / 1e9;

        printf("Выделен блок: %p, размер: %zu байт, время: %.11f секунд\n", ptr, block\_size, time\_taken);

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

        api.allocator\_free(allocator, ptr);

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

        time\_taken = (end.tv\_sec - start.tv\_sec) + (end.tv\_nsec - start.tv\_nsec) / 1e9;

        printf("Освобожден блок: %p, время: %.9f секунд\n", ptr, time\_taken);

    }

    api.allocator\_destroy(allocator);

    if (munmap(memory, pool\_size) == -1) {

        perror("Ошибка при освобождении памяти с помощью munmap");

        return 1;

    }

    if (library\_handle) {

        dlclose(library\_handle);

    }

    return 0;

}

## alloc1.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

typedef struct Block {

    size\_t size;

    struct Block\* next;

} Block;

typedef struct Allocator{

    void\* memory;

    size\_t size;

    Block\* free\_list;

} Allocator;

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

    Allocator\* allocator = (Allocator\*)mmap(NULL, sizeof(Allocator), PROT\_READ | PROT\_WRITE, MAP\_ANONYMOUS | MAP\_PRIVATE, -1, 0);

    allocator->memory = memory;

    allocator->size = size;

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

    allocator->free\_list->size = size;

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

    return allocator;

}

void allocator\_destroy(Allocator\* allocator) {

    munmap(allocator, sizeof(Allocator));

}

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

    Block\* prev = NULL;

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

    while (curr != NULL) {

        if (curr->size >= size) {

            if (curr->size > size + sizeof(Block)) {

                Block\* new\_block = (Block\*)((char\*)curr + sizeof(Block) + size);

                new\_block->size = curr->size - size - sizeof(Block);

                new\_block->next = curr->next;

                curr->size = size;

                curr->next = new\_block;

            }

            if (prev == NULL) {

                allocator->free\_list = curr->next;

            } else {

                prev->next = curr->next;

            }

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

        }

        prev = curr;

        curr = curr->next;

    }

    return NULL;

}

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

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

    block->next = allocator->free\_list;

    allocator->free\_list = block;

}

**alloc2.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#define ALIGN\_SIZE(size, alignment) (((size) + (alignment - 1)) & ~(alignment - 1))

#define FREE\_LIST\_ALIGNMENT 8

typedef struct Block {

    size\_t size;

    struct Block\* next;

} Block;

typedef struct Allocator{

    void\* memory;

    size\_t size;

    Block\* free\_list;

} Allocator;

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

    if (memory == NULL || size < sizeof(Allocator)) {

        return NULL;

    }

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

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

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

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

    if (allocator->free\_list != NULL) {

        allocator->free\_list->size = allocator->size;

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

    }

    return allocator;

}

void allocator\_destroy(Allocator\* allocator) {

    if (allocator == NULL) {

        return;

    }

    allocator->memory = NULL;

    allocator->size = 0;

    allocator->free\_list = NULL;

}

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

    if (allocator == NULL || size == 0) {

        return NULL;

    }

    size\_t aligned\_size = ALIGN\_SIZE(size, FREE\_LIST\_ALIGNMENT);

    Block\* prev = NULL;

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

    while (curr != NULL) {

        if (curr->size >= aligned\_size) {

            if (prev != NULL) {

                prev->next = curr->next;

            } else {

                allocator->free\_list = curr->next;

            }

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

        }

        prev = curr;

        curr = curr->next;

    }

    return NULL;

}

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

    if (allocator == NULL || memory == NULL) {

        return;

    }

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

    block->next = allocator->free\_list;

    allocator->free\_list = block;

}

# Strace:

# Allocated block: 0x7f1a926e9010, size: 1024 bytes, time: 0.000000200 seconds

# Freed block: 0x7f1a926e9010 (id=1, name=Object 1, value=123.45), time: 0.000000100 seconds

# Allocated block: 0x7f1a926e9420, size: 2048 bytes, time: 0.000000200 seconds

# Freed block: 0x7f1a926e9420 (id=2, name=Object 2, value=246.90), time: 0.000000100 seconds

# Allocated block: 0x7f1a926e9c30, size: 3072 bytes, time: 0.000003600 seconds

# Freed block: 0x7f1a926e9c30 (id=3, name=Object 3, value=370.35), time: 0.000000000 seconds

# miron@DESKTOP-GD72A05:~/LABS/lab 4$ strace -f ./main ./liballoc2.so

# execve("./main", ["./main", "./liballoc2.so"], 0x7ffd10f27ff0 /\* 26 vars \*/) = 0

# brk(NULL) = 0x56529f81f000

# arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffdb0716d50) = -1 EINVAL (Invalid argument)

# 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=33773, ...}) = 0

# mmap(NULL, 33773, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f73c3cfb000

# close(3) = 0

# openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libdl.so.2", 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 \22\0\0\0\0\0\0"..., 832) = 832

# fstat(3, {st\_mode=S\_IFREG|0644, st\_size=18848, ...}) = 0

# mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f73c3cf9000

# mmap(NULL, 20752, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f73c3cf3000

# mmap(0x7f73c3cf4000, 8192, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x7f73c3cf4000

# mmap(0x7f73c3cf6000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7f73c3cf6000

# mmap(0x7f73c3cf7000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7f73c3cf7000

# 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\300A\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\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

# pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357\_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68

# fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2029592, ...}) = 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

# pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

# pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357\_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68

# mmap(NULL, 2037344, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f73c3b01000

# mmap(0x7f73c3b23000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x22000) = 0x7f73c3b23000

# mmap(0x7f73c3c9b000, 319488, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19a000) = 0x7f73c3c9b000

# mmap(0x7f73c3ce9000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f73c3ce9000

# mmap(0x7f73c3cef000, 13920, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f73c3cef000

# close(3) = 0

# mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f73c3afe000

# arch\_prctl(ARCH\_SET\_FS, 0x7f73c3afe740) = 0

# mprotect(0x7f73c3ce9000, 16384, PROT\_READ) = 0

# mprotect(0x7f73c3cf7000, 4096, PROT\_READ) = 0

# mprotect(0x56529edc6000, 4096, PROT\_READ) = 0

# mprotect(0x7f73c3d31000, 4096, PROT\_READ) = 0

# munmap(0x7f73c3cfb000, 33773) = 0

# brk(NULL) = 0x56529f81f000

# brk(0x56529f840000) = 0x56529f840000

# openat(AT\_FDCWD, "./liballoc2.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@\20\0\0\0\0\0\0"..., 832) = 832

# fstat(3, {st\_mode=S\_IFREG|0755, st\_size=15776, ...}) = 0

# getcwd("/home/miron/LABS/lab 4", 128) = 23

# mmap(NULL, 16424, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f73c3cff000

# mmap(0x7f73c3d00000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x7f73c3d00000

# mmap(0x7f73c3d01000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f73c3d01000

# mmap(0x7f73c3d02000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f73c3d02000

# close(3) = 0

# mprotect(0x7f73c3d02000, 4096, PROT\_READ) = 0

# mmap(NULL, 1048576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f73c39fe000

# fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

# write(1, "Allocated block: 0x7f73c39fe028,"..., 77Allocated block: 0x7f73c39fe028, size: 1024 bytes, time: 0.000000200 seconds

# ) = 77

# write(1, "Freed block: 0x7f73c39fe028 (id="..., 91Freed block: 0x7f73c39fe028 (id=1, name=Object 1, value=123.45), time: 0.000000300 seconds

# ) = 91

# write(1, "Allocated block: 0x7f73c39fe028,"..., 77Allocated block: 0x7f73c39fe028, size: 2048 bytes, time: 0.000000300 seconds

# ) = 77

# write(1, "Freed block: 0x7f73c39fe028 (id="..., 91Freed block: 0x7f73c39fe028 (id=2, name=Object 2, value=246.90), time: 0.000000300 seconds

# ) = 91

# write(1, "Allocated block: 0x7f73c39fe028,"..., 77Allocated block: 0x7f73c39fe028, size: 3072 bytes, time: 0.000000200 seconds

# ) = 77

# write(1, "Freed block: 0x7f73c39fe028 (id="..., 91Freed block: 0x7f73c39fe028 (id=3, name=Object 3, value=370.35), time: 0.000000300 seconds

# ) = 91

# munmap(0x7f73c39fe000, 1048576) = 0

# munmap(0x7f73c3cff000, 16424) = 0

# exit\_group(0) = ?

# +++ exited with 0 +++

**Сравнение алгоритмов**

Аллокатор с обычным списком свободных блоков:

Выделен блок: 0x7f740795a010, размер: 1024 байт, время: 0.00000010000 секунд

Освобожден блок: 0x7f740795a010, время: 0.000000100 секунд

Выделен блок: 0x7f740795a420, размер: 2048 байт, время: 0.00000010000 секунд

Освобожден блок: 0x7f740795a420, время: 0.000000100 секунд

Выделен блок: 0x7f740795ac30, размер: 3072 байт, время: 0.00000240000 секунд

Освобожден блок: 0x7f740795ac30, время: 0.000000000 секунд

Аллокатор на основе алгоритма Мак-Кьюзика-Кэрелса:

Выделен блок: 0x7fd39939d028, размер: 1024 байт, время: 0.00000010000 секунд

Освобожден блок: 0x7fd39939d028, время: 0.000000000 секунд

Выделен блок: 0x7fd39939d028, размер: 2048 байт, время: 0.00000010000 секунд

Освобожден блок: 0x7fd39939d028, время: 0.000000100 секунд

Выделен блок: 0x7fd39939d028, размер: 3072 байт, время: 0.00000000000 секунд

Освобожден блок: 0x7fd39939d028, время: 0.000000000 секунд

Аллокатор с обычным списком свободных блоков: эффективно работает для небольших блоков данных, но неэффективен при фрагментации.

Аллокатор на основе алгоритма Мак-Кьюзика-Кэрелса: лучше управляется с памятью, предотвращаю фрагментацию. Работает быстрее для больших блоков данных. Использование памяти с учетом размера и выравнивания, что снижает вероятность ошибок при работе с памятью.

# Вывод

В ходе написания данной лабораторной работы я узнал об устройстве аллокаторов а также научился использовать динамические библиотеки.