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: Морозов Захар Олегович Преподаватель: Бахарев В.Д.

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

Дата: 07.01.25

Москва, 2025

# Постановка задачи

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

# 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;

}

## ffallocator.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;

}

**bdallocator.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <math.h>

#define MIN\_BLOCK\_SIZE 16

typedef struct Allocator {

    void\* memory;

    size\_t size;

    void\* free\_lists[32];

} 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;

    for (int i = 0; i < 32; i++) {

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

    }

    size\_t block\_size = 1 << (int)log2(size);

    \*(void\*\*)memory = NULL;

    allocator->free\_lists[(int)log2(block\_size)] = memory;

    return allocator;

}

void allocator\_destroy(Allocator\* allocator) {

    munmap(allocator, sizeof(Allocator));

}

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

    size\_t required\_size = size + sizeof(size\_t);

    int index = (int)ceil(log2(required\_size));

    if (index < log2(MIN\_BLOCK\_SIZE)) {

        index = (int)log2(MIN\_BLOCK\_SIZE);

    }

    while (index < 32 && !allocator->free\_lists[index]) {

        index++;

    }

    if (index >= 32) {

        return NULL;

    }

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

    allocator->free\_lists[index] = \*(void\*\*)block;

    while (index > log2(required\_size)) {

        index--;

        void\* buddy = (char\*)block + (1 << index);

        \*(void\*\*)buddy = allocator->free\_lists[index];

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

    }

    \*(size\_t\*)block = size;

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

}

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

    void\* block = (char\*)memory - sizeof(size\_t);

    size\_t size = \*(size\_t\*)block;

    int index = (int)log2(size + sizeof(size\_t));

    while (1) {

        void\* buddy = (void\*)((size\_t)block ^ (1 << index));

        void\* list = allocator->free\_lists[index];

        while (list) {

            if (list == buddy) {

                \*(void\*\*)block = \*(void\*\*)buddy;

                \*(void\*\*)buddy = NULL;

                block = (block < buddy) ? block : buddy;

                index++;

                break;

            }

            list = \*(void\*\*)list;

        }

        if (!list) {

            \*(void\*\*)block = allocator->free\_lists[index];

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

            break;

        }

    }

}

# Strace:

# strace -f ./a.out ./ffallocator.so

# execve("./a.out", ["./a.out", "./ffallocator.so"], 0x7ffc837eec80 /\* 22 vars \*/) = 0

# brk(NULL) = 0x556d5ea80000

# mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd57ece3000

# 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=19871, ...}) = 0

# mmap(NULL, 19871, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fd57ecde000

# close(3) = 0

# openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

# read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

# pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

# fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 0

# pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

# mmap(NULL, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fd57eacc000

# mmap(0x7fd57eaf4000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7fd57eaf4000

# mmap(0x7fd57ec7c000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7fd57ec7c000

# mmap(0x7fd57eccb000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7fd57eccb000

# mmap(0x7fd57ecd1000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fd57ecd1000

# close(3) = 0

# mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd57eac9000

# arch\_prctl(ARCH\_SET\_FS, 0x7fd57eac9740) = 0

# set\_tid\_address(0x7fd57eac9a10) = 5620

# set\_robust\_list(0x7fd57eac9a20, 24) = 0

# rseq(0x7fd57eaca060, 0x20, 0, 0x53053053) = 0

# mprotect(0x7fd57eccb000, 16384, PROT\_READ) = 0

# mprotect(0x556d5dca3000, 4096, PROT\_READ) = 0

# mprotect(0x7fd57ed1b000, 8192, PROT\_READ) = 0

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

# munmap(0x7fd57ecde000, 19871) = 0

# getrandom("\x2c\xcf\x62\xb4\x53\x87\x2b\x32", 8, GRND\_NONBLOCK) = 8

# brk(NULL) = 0x556d5ea80000

# brk(0x556d5eaa1000) = 0x556d5eaa1000

# openat(AT\_FDCWD, "./ffallocator.so", O\_RDONLY|O\_CLOEXEC) = 3

# read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

# fstat(3, {st\_mode=S\_IFREG|0775, st\_size=15656, ...}) = 0

# getcwd("/home/zakhar/lr4", 128) = 17

# mmap(NULL, 16416, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fd57ecde000

# mmap(0x7fd57ecdf000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x7fd57ecdf000

# mmap(0x7fd57ece0000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7fd57ece0000

# mmap(0x7fd57ece1000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7fd57ece1000

# close(3) = 0

# mprotect(0x7fd57ece1000, 4096, PROT\_READ) = 0

# mmap(NULL, 1048576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd57e9c9000

# mmap(NULL, 24, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd57e9c8000

# fstat(1, {st\_mode=S\_IFCHR|0600, st\_rdev=makedev(0x88, 0), ...}) = 0

# write(1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275 \320\261\320\273\320\276\320\272: 0x7fd57"..., 109Выделен блок: 0x7fd57e9c9010, размер: 1024 байт, время: 0.00000030000 секунд

# ) = 109

# write(1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275 \320\261\320\273\320\276\320\272: 0"..., 84Освобожден блок: 0x7fd57e9c9010, время: 0.000000500 секунд

# ) = 84

# write(1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275 \320\261\320\273\320\276\320\272: 0x7fd57"..., 109Выделен блок: 0x7fd57e9c9010, размер: 2048 байт, время: 0.00000030000 секунд

# ) = 109

# write(1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275 \320\261\320\273\320\276\320\272: 0"..., 84Освобожден блок: 0x7fd57e9c9010, время: 0.000000500 секунд

# ) = 84

# write(1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275 \320\261\320\273\320\276\320\272: 0x7fd57"..., 109Выделен блок: 0x7fd57e9c9010, размер: 3072 байт, время: 0.00000030000 секунд

# ) = 109

# write(1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\320\265\320\275 \320\261\320\273\320\276\320\272: 0"..., 84Освобожден блок: 0x7fd57e9c9010, время: 0.000000300 секунд

# ) = 84

# munmap(0x7fd57e9c8000, 24) = 0

# munmap(0x7fd57e9c9000, 1048576) = 0

# munmap(0x7fd57ecde000, 16416) = 0

# exit\_group(0) = ?

# +++ exited with 0 +++

**Сравнение алгоритмов**

**Аллокатор со списком свободных блоков (первое подходящее):**

Выделен блок: 0x7fcc58169010, размер: 1024 байт, время: 0.00000020000 секунд

Освобожден блок: 0x7fcc58169010, время: 0.000000200 секунд

Выделен блок: 0x7fcc58169010, размер: 2048 байт, время: 0.00000000000 секунд

Освобожден блок: 0x7fcc58169010, время: 0.000000000 секунд

Выделен блок: 0x7fcc58169010, размер: 3072 байт, время: 0.00000010000 секунд

Освобожден блок: 0x7fcc58169010, время: 0.000000000 секунд

**Аллокатор на блоках по 2^n:**

Выделен блок: 0x7f1d1183b008, размер: 1024 байт, время: 0.00001360000 секунд

Освобожден блок: 0x7f1d1183b008, время: 0.000000400 секунд

Выделен блок: 0x7f1d1183b008, размер: 2048 байт, время: 0.00000020000 секунд

Освобожден блок: 0x7f1d1183b008, время: 0.000000100 секунд

Выделен блок: 0x7f1d1183b008, размер: 3072 байт, время: 0.00000020000 секунд

Освобожден блок: 0x7f1d1183b008, время: 0.000000100 секунд

Аллокатор «Первое подходящее»: Может страдать от фрагментации. Медленнее, чем аллокатор на блоках 2^n, из-за поиска подходящего блока. Более гибко выделяет память.

Аллокатор на блоках 2^n: Быстрее, так как использует предсказуемые размеры блоков. Меньше фрагментации, но может быть неэффективен для небольших запросов. Неэффективно использует память, так как выделяет больше чем необходимо.

# Вывод

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