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-211Б-23

Студент: Косов В.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 06.01.25

Москва, 2024

Постановка задачи

Вариант 9.

Требуется создать две динамические библиотеки, реализующие два аллокатора памяти: списки свободных блоков (наиболее подходящее) и алгоритм двойников;

Общий метод и алгоритм решения

Использованные системные вызовы:

- 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".
- ssize_t write(int fd, const void *buf, size_t count); - Записывает данные в файл или файловый дескриптор.
- void *dlopen(const char *filename, int flag); - Открывает динамическую библиотеку.
- void *dlsym(void *handle, const char *symbol); - Извлекает адрес функции или переменной symbol из открытой библиотеки handle.
- int dlclose(void *handle); - Закрывает динамическую библиотеку handle.

Описание программы

main.c

Открывает динамические библиотеки, извлекает нужные функции из них. Если функции не найдены, то используются функции-заглушки, чтобы избежать ошибок во время исполнения программы.

library.h

Подключает сторонние библиотеки и объявляет фукнции, которые реализовывает аллокатор

buddys.c

Файл в котором реализована логика работы аллокатора на методе двойников.

Инициализация:

- Память делится на блоки, размеры которых являются степенями двойки (32, 64, 128 и т. д.).
- Вся память представлена как один большой блок (наивысшая степень двойки).

Разделение блоков:

- Если запрашиваемый размер меньше текущего блока, он делится пополам, образуя два "двойника". Этот процесс повторяется до тех пор, пока не будет найден блок подходящего размера.

Выделение памяти:

- Для каждого запроса выбирается минимальный блок подходящего размера.

- Аллокатор использует битовую карту (*bitmap*), которая хранится в начале выделенной памяти, для отслеживания статуса каждого блока (занят/свободен).

Освобождение памяти:

- Освобожденный блок помечается как свободный в битовой карте.
- Если его "двойник" также свободен, они объединяются в один более крупный блок. Процесс повторяется рекурсивно для более крупных блоков.

Ограничение размеров:

- Запросы округляются до ближайшей степени двойки. Например, запрос 50 байт преобразуется в 64 байта, а 1 байт в 32, т.к. минимальный размер блока 32 байта.

Граничные условия:

- Если запрос превышает размер доступной памяти или не соответствует правилам кратности, выделение завершается с ошибкой.

freebloks.c

Файл в котором реализована логика работы аллокатора на списке свободных блоков.

Инициализация:

- При создании аллокатора вся доступная память разбивается на один большой свободный блок.

Список свободных блоков:

- Используется односвязный список для отслеживания всех свободных блоков.
- Размеры блоков могут быть произвольными, что позволяет гибко использовать память.

Выделение памяти:

- Происходит поиск наименьшего свободного блока, подходящего под запрос.
- Если найденный блок больше, чем необходимо, он разделяется на два: первый блок удовлетворяет запрос, второй остаётся в списке свободных блоков.

Освобождение памяти:

- Освобожденный блок добавляется обратно в список свободных.
- Если соседние блоки также свободны, они объединяются в один более крупный блок для уменьшения фрагментации.

Объединение блоков:

- После освобождения блок проверяет своих соседей. Если они также свободны, блоки объединяются, чтобы минимизировать количество фрагментов памяти.

Граничные условия:

- Если размер запроса меньше минимального блока, выделяется минимально допустимый блок.
- В случае исчерпания памяти аллокатор возвращает ошибку.

Код программы

main.c

```
#include "library.h"



#define MEMORY_POOL_SIZE 1024


static Allocator *allocator_create_stub(void *const memory, const size_t size) {
    const char msg[] = "allocator_create: Function not found, using mmap\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);

    void *mapped_memory = mmap(memory, size, PROT_READ | PROT_WRITE,
MAP_PRIVATE | MAP_ANONYMOUS | MAP_FIXED, -1, 0);
    if (mapped_memory == MAP_FAILED) {
        const char err_msg[] = "allocator_create: mmap failed\n";
        write(STDERR_FILENO, err_msg, sizeof(err_msg) - 1);
        return NULL;
    }

    return (Allocator *)mapped_memory;
}


static void allocator_destroy_stub(Allocator *const allocator) {
    const char msg[] = "allocator_destroy: Function not found, using munmap\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);

    if (allocator) {
        if (munmap(allocator, MEMORY_POOL_SIZE) == -1) {
            const char err_msg[] = "allocator_destroy: munmap failed\n";
```

```
                write(STDERR_FILENO, err_msg, sizeof(err_msg) - 1);
            }
        }
}

static void *allocator_alloc_stub(Allocator *const allocator, const size_t size) {
        const char msg[] = "allocator_alloc: Function not found, using mmap\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);

        void *mapped_memory = mmap(NULL, size, PROT_READ | PROT_WRITE,
MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
        if (mapped_memory == MAP_FAILED) {
            const char err_msg[] = "allocator_alloc: mmap failed\n";
            write(STDERR_FILENO, err_msg, sizeof(err_msg) - 1);
            return NULL;
        }

        return mapped_memory;
}

static void allocator_free_stub(Allocator *const allocator, void *const memory) {
        const char msg[] = "allocator_free: Function not found, using munmap\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);

        if (memory && munmap(memory, sizeof(memory)) == -1) {
            const char err_msg[] = "allocator_free: munmap failed\n";
            write(STDERR_FILENO, err_msg, sizeof(err_msg) - 1);
        }
}



static allocator_create_f *allocator_create;
static allocator_destroy_f *allocator_destroy;
static allocator_alloc_f *allocator_alloc;
```

```
static allocator_free_f *allocator_free;

int main(int argc, char **argv) {
    if (argc < 2) {
        const char msg[] = "Usage: ./Main <library_path>\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        return EXIT_FAILURE;
    }

    void *library = dlopen(argv[1], RTLD_LOCAL | RTLD_NOW);
    argc++;
    if (argc > 2 && library) {
        if (!library) {
            const char msg[] = "Failed to load library\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            return EXIT_FAILURE;
        }

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

        if (!allocator_create) {
            allocator_create = allocator_create_stub;
        }
        if (!allocator_destroy) {
            allocator_destroy = allocator_destroy_stub;
        }
        if (!allocator_alloc) {
            allocator_alloc = allocator_alloc_stub;
        }
        if (!allocator_free) {
```

```
            allocator_free = allocator_free_stub;
        }

    } else {
        const char msg[] = "error: failed to open custom library\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        return EXIT_FAILURE;
    }


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

    Allocator *allocator = allocator_create(addr, MEMORY_POOL_SIZE);

    if (!allocator) {
        const char msg[] = "Failed to initialize allocator\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        munmap(addr, size);
        dlclose(library);

        return EXIT_FAILURE;
    }

    int *int_block = (int *)allocator_alloc(allocator, sizeof(int));
```

```
if (int_block) {
    *int_block = 42;
    const char msg[] = "Allocated int_block with value 42\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
} else {
    const char msg[] = "Failed to allocate memory for int_block\n";
    write(STDERR_FILENO, msg, sizeof(msg));
}

float *float_block = (float *)allocator_alloc(allocator, sizeof(float));
if (float_block) {
    *float_block = 3.14f;
    const char msg[] = "Allocated float_block with value 3.14\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
} else {
    const char msg[] = "Failed to allocate memory for float_block\n";
    write(STDERR_FILENO, msg, sizeof(msg));
}




if (int_block) {
    allocator_free(allocator, int_block);
    const char msg[] = "Freed int_block\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
}

if (float_block) {
    allocator_free(allocator, float_block);
    const char msg[] = "Freed float_block\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
}
```

```
    allocator_destroy(allocator);
    const char msg[] = "Allocator destroyed\n";
    write(STDOUT_FILENO, msg, sizeof(msg));

    if (library) dlclose(library);
    munmap(addr, size);

    return EXIT_SUCCESS;
```

library.h

```
#ifndef ALLOCATOR_H
#define ALLOCATOR_H

#include <dlfcn.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <dlfcn.h>
#include <sys/mman.h>
#include <stddef.h>
#include <string.h>
#include <stdbool.h>

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

typedef struct Allocator Allocator;
typedef struct Block Block;

typedef Allocator *allocator_create_f(void *const memory, const size_t size);
typedef void allocator_destroy_f(Allocator *const allocator);
typedef void *allocator_alloc_f(Allocator *const allocator, const size_t size);
typedef void allocator_free_f(Allocator *const allocator, void *const memory);

#endifvsevolod@DESKTOP-K08EACJ:~/os_labs/laba_4/task$
```

freeblocks.c

```
#include "library.h"

#define MIN_BLOCK_SIZE 32

typedef struct Block {
    size_t size;
    struct Block *next;
    bool is_free;
} Block;

typedef struct Allocator {
    Block *free_list;
    void *memory_start;
    size_t total_size;
} Allocator;


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

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

    allocator->free_list->size = allocator->total_size - sizeof(Block);
    allocator->free_list->next = NULL;
    allocator->free_list->is_free = true;

    return allocator;
}

EXPORT void allocator_destroy(Allocator *const allocator) {
    if (allocator) {
        memset(allocator, 0, allocator->total_size);
    }
}

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

    size = (size + MIN_BLOCK_SIZE - 1) / MIN_BLOCK_SIZE * MIN_BLOCK_SIZE;
```

```
    Block *best = NULL;
    Block *prev_best = NULL;
    Block *current = allocator->free_list;
    Block *prev = NULL;

    while (current) {
        if (current->is_free && current->size >= size) {
            if (best == NULL || current->size < best->size) {
                best = current;
                prev_best = prev;
            }
        }
        prev = current;
        current = current->next;
    }

    if (best) {
        size_t remain_size = best->size - size;
        if (remain_size >= sizeof(Block) + MIN_BLOCK_SIZE) {
                Block *new_block =
                    (Block *)((char *)best + sizeof(Block) +
                              size);
                new_block->size = remain_size - sizeof(Block);
                new_block->is_free = true;
                new_block->next = best->next;

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

        best->is_free = false;
        if (prev_best == NULL) {
            allocator->free_list = best->next;
        } else {
            prev_best->next = best->next;
        }

        return (void *)((char *)best + sizeof(Block));
    }

    return NULL;
}

EXPORT void allocator_free(Allocator *allocator, void *ptr_to_memory) {
    if (!allocator || !ptr_to_memory) {
        return;
```

```
    }

    Block *head = (Block *)((char *)ptr_to_memory - sizeof(Block));
    if (!head) return;

    head->next = allocator->free_list;
    head->is_free = true;
    allocator->free_list = head;

    Block *current = allocator->free_list;
    while (current && current->next) {
        if (((char *)current + sizeof(Block) + current->size) ==
            (char *)current->next) {
            current->size += current->next->size + sizeof(Block);
            current->next = current->next->next;
        } else {
            current = current->next;
        }
    }
```

buddys.c

```
#include "library.h"

typedef struct Allocator {
    void *memory;
    size_t size;
    uint8_t *bitmap;
    size_t block_size;
} Allocator;

EXPORT Allocator* allocator_create(void *const memory, const size_t size) {
    if (!memory || size == 0) return NULL;

    Allocator *allocator = (Allocator *)memory;
    allocator->memory = (void *)((uint8_t *)memory + sizeof(Allocator));
    allocator->size = size - sizeof(Allocator);
    allocator->block_size = 32;
    allocator->bitmap = (uint8_t *)allocator->memory;

    size_t bitmap_size = allocator->size / allocator->block_size / 8;
    memset(allocator->bitmap, 0, bitmap_size); // Все блоки свободны
    allocator->memory = (uint8_t *)allocator->bitmap + bitmap_size;

    return allocator;
}

EXPORT void allocator_destroy(Allocator *const allocator) {
    if (allocator) {
        memset(allocator, 0, allocator->size);
    }
}
```

```
EXPORT void* allocator_alloc(Allocator *const allocator, const size_t size) {
    if (!allocator || size == 0 || size > allocator->size) return NULL;

    size_t blocks_needed = (size + allocator->block_size - 1) / allocator->block_size;
    size_t total_blocks = allocator->size / allocator->block_size;
    uint8_t *bitmap = allocator->bitmap;

    size_t free_blocks = 0;
    for (size_t i = 0; i < total_blocks; ++i) {
        if (!(bitmap[i / 8] & (1 << (i % 8)))) {
            ++free_blocks;
            if (free_blocks == blocks_needed) {
                size_t start_block = i - blocks_needed + 1;
                for (size_t j = start_block; j <= i; ++j) {
                    bitmap[j / 8] |= (1 << (j % 8));
                }
                return (uint8_t *)allocator->memory + start_block * allocator->block_size;
            }
        } else {
            free_blocks = 0;
        }
    }

    return NULL;
}

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

    size_t offset = (uint8_t *)memory - (uint8_t *)allocator->memory;
    if (offset % allocator->block_size != 0) return;

    size_t block_index = offset / allocator->block_size;
    allocator->bitmap[block_index / 8] &= ~(1 << (block_index % 8));
```

Протокол работы программы

Процесс тестирования:

Для тестирования использовались следующие сценарии:

1. **Массовое выделение и освобождение памяти**: Выделение памяти разного размера с последующим освобождением.
2. **Проверка объединения блоков**: Выделение нескольких блоков и освобождение их в произвольном порядке для проверки корректности объединения.
3. **Измерение производительности**: Сравнение времени выполнения операций выделения и освобождения памяти.
4. **Измерение фрагментации**: Оценка степени использования памяти.

Обоснование подхода тестирования

Тесты разработаны для проверки следующих характеристик:

1. **Эффективность выделения памяти**: Важно для приложений, интенсивно использующих динамическую память.
2. **Корректность работы**: Объединение блоков и освобождение должны работать без ошибок.
3. **Производительность**: Аллокатор должен минимизировать накладные расходы.
4. **Фрагментация**: Важно для долгосрочной работы без истощения памяти.

Результаты тестирования

vsevolod@DESKTOP-K08EACJ:~/os_labs/laba_4/task$ ./Main ./buddys.so

Allocated int_block with value 42

Allocated float_block with value 3.14

Freed int_block

Freed float_block

Allocator destroyed

vsevolod@DESKTOP-K08EACJ:~/os_labs/laba_4/task$ ./Main ./freeblocks.so

Allocated int_block with value 42

Allocated float_block with value 3.14

Freed int_block

Freed float_block

Allocator destroyed

vsevolod@DESKTOP-K08EACJ:~/os_labs/laba_4/task$

Метод свободных блоков

Производительность: Быстрое выделение памяти, но медленное объединение блоков при освобождении.

- **Фрагментация**: Минимальная.
- **Память**: Эффективное использование памяти для запросов любого размера.

Метод двойников

- **Производительность**: Высокая скорость выделения и освобождения памяти.
- **Фрагментация**: Заметная внутренняя фрагментация из-за округления размеров запросов.
- **Память**: Эффективен для запросов, кратных степени двойки.

Strace:

strace ./Main ./buddys.so

execve("./Main", ["./Main", "./buddys.so"], 0x7ffcb6baa4d8 /* 35 vars */) = 0

brk(NULL) = 0x55e54677c000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fffacdc9c40) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6534bac000

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=18823, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 18823, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6534ba7000

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) = 0x7f653497e000

mprotect(0x7f65349a6000, 2023424, PROT_NONE) = 0

mmap(0x7f65349a6000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f65349a6000

mmap(0x7f6534b3b000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f6534b3b000

mmap(0x7f6534b94000, 24576, PROT_READ|PROT_WRITE,

MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f6534b94000

mmap(0x7f6534b9a000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f6534b9a000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f653497b000

arch_prctl(ARCH_SET_FS, 0x7f653497b740) = 0

set_tid_address(0x7f653497ba10) = 105236

set_robust_list(0x7f653497ba20, 24) = 0

rseq(0x7f653497c0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f6534b94000, 16384, PROT_READ) = 0

mprotect(0x55e5241b6000, 4096, PROT_READ) = 0

mprotect(0x7f6534be6000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7f6534ba7000, 18823) = 0

getrandom("\x90\x06\xe1\xf7\x2b\xd6\x44\xd0", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x55e54677c000

brk(0x55e54679d000) = 0x55e54679d000

openat(AT_FDCWD, "./buddys.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|0755, st_size=15632, ...}, AT_EMPTY_PATH) = 0

getcwd("/home/pablo/Main/3,4 SEM/LabOs1/LabOS4/src", 128) = 43

mmap(NULL, 16432, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f6534ba7000

mmap(0x7f6534ba8000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f6534ba8000

mmap(0x7f6534ba9000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f6534ba9000

mmap(0x7f6534baa000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f6534baa000

close(3) = 0

mprotect(0x7f6534baa000, 4096, PROT_READ) = 0

mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6534be5000

write(1, "Allocated int_block with value 4"..., 35Allocated int_block with value 42

) = 35

write(1, "Allocated float_block with value"..., 39Allocated float_block with value 3.14

) = 39

write(1, "Freed int_block\n\0", 17Freed int_block

) = 17

write(1, "Freed float_block\n\0", 19Freed float_block

) = 19

write(1, "Allocator destroyed\n\0", 21Allocator destroyed

) = 21

munmap(0x7f6534ba7000, 16432) = 0

munmap(0x7f6534be5000, 1024) = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В процессе выполнения этой лабораторной работы я освоил работу с динамическими библиотеками, новыми системными вызовами, предназначенными для работы с динамическими библиотеками, и написанием собственного аллокатора памяти в языке С. Я научился писать собственные динамические библиотеки, подключать, обрабатывать ошибки, связанные с их подключением, и использовать их. Главная сложность работы возникла при написании собственного аллокатора памяти, поскольку материал был новый для меня и информацию про алгоритмы аллокаторов приходилась искать в книгах и интернете.