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: Коваль М.Р.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 15.02.24

Москва, 2024

Постановка задачи

Вариант 5.

Алгоритм Мак-Кьюзика-Кэрелса и алгоритм двойников

Общий метод и алгоритм решения

Использованные системные вызовы:

- *int munmap(void addr, size_t length); - Удаляет отображения, созданные с помощью mmap.
- *int dlclose(void handle); - Закрывает динамическую библиотеку, открытую с помощью dlopen, и освобождает ресурсы, связанные с этим дескриптором.
- void exit(int status); - Завершает выполнение программы и возвращает статус выхода в операционную систему.
- *char dlerror(void); - Возвращает строку, описывающую последнюю ошибку, возникшую при вызове функций dlopen, dlsym, dlclose.
- **void dlopen(const char filename, int flag); - Открывает динамическую библиотеку и возвращает дескриптор для последующего использования.
- **void mmap(void addr, size_t length, int prot, int flags, int fd, off_t offset); – создает новое отображение памяти или изменяет существующее.
- Allocator* allocator_create(void *const memory, const size_t size) (инициализация аллокатора на памяти memory размера size);
- void allocator_destroy(Allocator *const allocator)(деинициализация структуры аллокатора);
- void* allocator_alloc(Allocator *const allocator, constsize_t size) (выделение памяти аллокатором памяти размера size);
- void allocator_free(Allocator *const allocator, void *const memory) (возвращает выделенную память аллокатору);

В данной лабораторной работе я написал программу реализующая 2 аллокатора памяти, с целью приобретения практических навыков в:

- Создании аллокаторов памяти и их анализу.
- Создании динамических библиотек и программ, использующие динамические библиотеки.

Алгоритм McKusick-Karels — это алгоритм управления памятью, разработанный Маршаллом Кирком Макьюзиком (Marshall Kirk McKusick) и Майклом Дж. Карелсом (Michael J. Karels) для операционной системы BSD Unix. Этот алгоритм был предложен как улучшение стандартного подхода к управлению памятью в ядре Unix, чтобы сделать его более эффективным и масштабируемым.

1. Использование списков свободных блоков:
   - Память делится на блоки разных размеров, и для каждого размера поддерживается отдельный список свободных

блоков.
   - Это позволяет быстро находить блок подходящего размера.

2. Динамическое разделение и объединение блоков:
   - Если блок слишком большой для запроса, он разделяется на два меньших блока. Один из них используется, а другой

добавляется в соответствующий список свободных блоков.
   - Когда блок освобождается, он объединяется с соседними свободными блоками, чтобы

избежать фрагментации.

3. Иерархия размеров блоков:
   - Размеры блоков обычно выбираются как степени двойки или в соответствии с другой удобной схемой (например,

геометрическая прогрессия).
   - Это упрощает поиск подходящего блока и управление списками.

4. Эффективное управление памятью:
   - Алгоритм минимизирует фрагментацию и обеспечивает быстрое выделение и освобождение памяти.

Алгоритм двойников:
Все блоки памяти имеют размер степени двойки. Когда нужно выделить новый блок, алгоритм проходится по массиву списков из свободных блоков размера 2^n, где n – индекс в этом массиве, и находит ближайший блок к нужному размеру. Если найденный блок больше, то он делится пополам, пока он не достигнет ближайшего возможного размера. Ненужные половинки добавляются в массив списков свободных блоков.

Код программы

main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <sys/mman.h>
#include <stdint.h>
#include <stddef.h>
#include <time.h>

typedef struct Allocator {
    size_t size;
    void *memory;
} Allocator;

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);

static allocator_create_f allocator_create = NULL;
static allocator_destroy_f allocator_destroy = NULL;
static allocator_alloc_f allocator_alloc = NULL;
static allocator_free_f allocator_free = NULL;

Allocator* fallback_allocator_create(void *const memory, const size_t size) {
    Allocator *allocator = (Allocator*) mmap(NULL, sizeof(Allocator), PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANON, -1,
0);
    if (allocator != MAP_FAILED) {
        allocator->size = size;
        allocator->memory = memory;
    }
    return allocator;
}

void fallback_allocator_destroy(Allocator *const allocator) {
    munmap(allocator->memory, allocator->size);
    munmap(allocator, sizeof(Allocator));
}

void* fallback_allocator_alloc(Allocator *const allocator, const size_t size) {
    return mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANON, -1, 0);
}
```

```
void fallback_allocator_free(Allocator *const allocator, void *const memory) {
    munmap(memory, sizeof(memory));
}

void load_allocator_library(const char *path) {
    void *handle = dlopen(path, RTLD_LAZY);
    if (!handle) {
        fprintf(stderr, "Error loading library: %s\n", dlerror());
        return;
    }

    allocator_create = (allocator_create_f) dlsym(handle, "allocator_create");
    allocator_destroy = (allocator_destroy_f) dlsym(handle, "allocator_destroy");
    allocator_alloc = (allocator_alloc_f) dlsym(handle, "allocator_alloc");
    allocator_free = (allocator_free_f) dlsym(handle, "allocator_free");

    if (!allocator_create || !allocator_destroy || !allocator_alloc || !allocator_free) {
        fprintf(stderr, "Error loading functions from library\n");
        dlclose(handle);
    }
}

double measure_time_allocation(Allocator *allocator, size_t alloc_size, int num_allocs) {
    clock_t start = clock();
    for (int i = 0; i < num_allocs; ++i) {
        void *block = allocator_alloc(allocator, alloc_size);
        if (!block) {
            fprintf(stderr, "Allocation failed at iteration %d\n", i);
            break;
        }
        allocator_free(allocator, block);
    }
    clock_t end = clock();
    return (double)(end - start) / CLOCKS_PER_SEC;
}

double measure_time_free(Allocator *allocator, size_t alloc_size, int num_allocs) {
    void **blocks = malloc(num_allocs * sizeof(void*));
    for (int i = 0; i < num_allocs; ++i) {
        blocks[i] = allocator_alloc(allocator, alloc_size);
    }

    clock_t start = clock();
    for (int i = 0; i < num_allocs; ++i) {
        allocator_free(allocator, blocks[i]);
```

```
    }
    clock_t end = clock();
    free(blocks);
    return (double)(end - start) / CLOCKS_PER_SEC;
}

int main(int argc, char **argv) {
    if (argc < 2) {
        printf("No library path provided, using fallback allocator\n");
        allocator_create = (allocator_create_f) fallback_allocator_create;
        allocator_destroy = (allocator_destroy_f) fallback_allocator_destroy;
        allocator_alloc = (allocator_alloc_f) fallback_allocator_alloc;
        allocator_free = (allocator_free_f) fallback_allocator_free;
    } else {
        load_allocator_library(argv[1]);
        if (!allocator_create) {
            printf("Failed to load library, using fallback allocator\n");
            allocator_create = (allocator_create_f) fallback_allocator_create;
            allocator_destroy = (allocator_destroy_f) fallback_allocator_destroy;
            allocator_alloc = (allocator_alloc_f) fallback_allocator_alloc;
            allocator_free = (allocator_free_f) fallback_allocator_free;
        } else {
            printf("Library loaded successfully\n");
        }
    }

    size_t memory_size = 1024 * 1024 * 10;
    void *memory = mmap(NULL, memory_size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANON, -1, 0);
    if (memory == MAP_FAILED) {
        perror("mmap failed");
        return 1;
    }

    Allocator *allocator = allocator_create(memory, memory_size);
    if (!allocator) {
        fprintf(stderr, "Allocator creation failed\n");
        return 1;
    }

    printf("Measuring allocation time...\n");
    double alloc_time = measure_time_allocation(allocator, 128, 10000);
    printf("Allocation time for 10,000 allocations: %.6f seconds\n", alloc_time);

    printf("Measuring free time...\n");
    double free_time = measure_time_free(allocator, 128, 10000);
```

```
    printf("Free time for 10,000 deallocations: %.6f seconds\n", free_time);

    allocator_destroy(allocator);
    munmap(memory, memory_size);

    return 0;
}
```

mck_allocator.c

```
#include "mck_allocator.h"
#include <stddef.h>
#include <string.h>
#include <stdint.h>


#define MAX_CLASS 10
#define PAGE_SIZE 4096

typedef struct Block {
    struct Block *next;
} Block;

struct Allocator {
    Block* freelist[MAX_CLASS];
    size_t class_size[MAX_CLASS];
    void *memory;
    size_t size;
};

Allocator* allocator_create(void *memory, size_t size) {
    Allocator *allocator = (Allocator *)memory;
    if(size < sizeof(Allocator)){
        return NULL;
    }
    allocator->memory = (char*)memory + sizeof(Allocator);
    allocator->size = size - sizeof(Allocator);

    size_t block_size = 16;
    for(size_t i = 0; i < MAX_CLASS; i++){
        allocator->class_size[i] = block_size;
        allocator->freelist[i] = NULL;
        block_size *= 2;
    }
```

```
    return allocator;
}

void allocator_destroy(Allocator *allocator) {
    for(size_t i = 0; i < MAX_CLASS; i++){
        allocator->freelist[i] = NULL;
    }
}

static size_t find_class(size_t size, size_t* class_size, size_t num_classes){
    for(size_t i = 0; i < num_classes; i++){
        if(size <= class_size[i]){
            return i;
        }
    }
    return num_classes;
}

void* allocator_alloc(Allocator *allocator, size_t size) {
    size_t class_i = find_class(size, allocator->class_size, MAX_CLASS);
    if(class_i >= MAX_CLASS){
        return NULL;
    }

    Block* block = allocator->freelist[class_i];
    if(block){
        allocator->freelist[class_i];
        return (void*)block;
    }

    size_t block_size = allocator->class_size[class_i];
    if(allocator->size < block_size){
        return NULL;
    }

    void* memory = allocator->memory;
    allocator->memory = (char*)allocator->memory + block_size;
    allocator->size -= block_size;

    return memory;
}

void allocator_free(Allocator *allocator, void *memory) {
    if (!memory) {
        return;
```

```
    }

    uintptr_t address = (uintptr_t)memory - (uintptr_t)allocator;
    if(address >= allocator->size){
        return;
    }

    size_t class_i = 0;
    size_t block_size = 0;
    for(; class_i < MAX_CLASS; class_i++){
        block_size = allocator->class_size[class_i];
        if((uintptr_t)memory % block_size == 0){
            break;
        }
    }

    if(class_i >= MAX_CLASS){
        return;
    }

    Block* block = (Block*)memory;
    block->next = allocator->freelist[class_i];
    allocator->freelist[class_i] = block;
}
```

mck_allocator.h

```
#ifndef MCK_ALLOCATOR_H
#define MCK_ALLOCATOR_H

#include <stddef.h>

typedef struct Allocator Allocator;

Allocator* allocator_create(void *memory, size_t size);

void allocator_destroy(Allocator *allocator);

void* allocator_alloc(Allocator *allocator, size_t size);

void allocator_free(Allocator *allocator, void *memory);

#endif // MCK_ALLOCATOR_H
```

buddy_allocator.c

```c
#include "buddy_allocator.h"
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>

#define MIN_BLOCK_SIZE 64
#define MAX_ORDER 20

typedef struct Block {
    struct Block *next;
    size_t order;
} Block;

typedef struct Allocator {
    void *memory;
    size_t size;
    size_t order;
    struct Block *freelist[MAX_ORDER + 1];
} Allocator;

Allocator* allocator_create(void *memory, size_t size) {
    if (size < MIN_BLOCK_SIZE) return NULL;

    // Выделяем дополнительное место для структуры Allocator
    size_t alloc_size = sizeof(Allocator);
    if (size <= alloc_size) return NULL;

    Allocator *allocator = (Allocator *)memory;
    allocator->memory = mmap(NULL, size - alloc_size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if (allocator->memory == MAP_FAILED) {
        return NULL;
    }

    allocator->size = size - alloc_size;
    allocator->order = 0;

    while ((1UL << allocator->order) < allocator->size) {
        allocator->order++;
    }

    // Инициализация списков свободных блоков
    for (size_t i = 0; i <= MAX_ORDER; i++) {
```

```
        allocator->freelist[i] = NULL;
    }

    // Создаём первый блок
    Block *block = (Block*)allocator->memory;
    block->next = NULL;
    block->order = allocator->order;
    allocator->freelist[allocator->order] = block;

    return allocator;
}

void allocator_destroy(Allocator *allocator) {
    if (!allocator) return;
    munmap(allocator->memory, allocator->size);
}

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

    size_t order = 0;
    while ((1UL << order) < size + sizeof(Block)) { // Учитываем заголовок блока
        order++;
    }
    if (order > MAX_ORDER) return NULL;

    for (size_t i = order; i <= MAX_ORDER; i++) {
        if (allocator->freelist[i]) {
            Block *block = allocator->freelist[i];
            allocator->freelist[i] = block->next;

            while (i > order) {
                i--;
                Block *buddy = (Block*)((char*)block + (1UL << i));
                buddy->next = allocator->freelist[i];
                buddy->order = i;
                allocator->freelist[i] = buddy;
            }

            block->next = NULL;
            block->order = order;
            return (void*)((char*)block + sizeof(Block)); // Пропускаем заголовок
        }
    }
    return NULL;
```

```
}

void allocator_free(Allocator *allocator, void *ptr) {
    if (!allocator || !ptr) return;

    Block *block = (Block*)((char*)ptr - sizeof(Block));
    size_t order = block->order;

    while (order < allocator->order) {
        Block *buddy = (Block*)((char*)block + (1UL << order)); // Получаем адрес "бадди"

        // Проверяем, что бадди находится в свободном списке
        Block **prev = &allocator->freelist[order];
        Block *curr = allocator->freelist[order];
        while (curr) {
            if (curr == buddy && curr->order == order) {
                *prev = curr->next;
                block = (block < buddy) ? block : buddy; // Выбираем младший адрес
                order++;
                break;
            }
            prev = &curr->next;
            curr = curr->next;
        }
        if (!curr) break;
    }

    block->next = allocator->freelist[order];
    block->order = order;
    allocator->freelist[order] = block;
}
```

buddy_allocator.h

```
#ifndef BUDDY_ALLOCATOR_H
#define BUDDY_ALLOCATOR_H

#include <stddef.h>

typedef struct Allocator Allocator;

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

```
#endif // BUDDY_ALLOCATOR_H
```

Протокол работы программы

```
(base) lab@DESKTOP-KPRMFIO:~/mk/lr4$ gcc -shared -fPIC -o libbuddy.so buddy_allocator.c
(base) lab@DESKTOP-KPRMFIO:~/mk/lr4$ gcc -shared -fPIC -o libmck.so mck_allocator.c
(base) lab@DESKTOP-KPRMFIO:~/mk/lr4$ gcc main.c -o alloc_test -ldl
(base) lab@DESKTOP-KPRMFIO:~/mk/lr4$ ./alloc_test ./libbuddy.so
Library loaded successfully
Measuring allocation time...
Allocation failed at iteration 0
Allocation time for 10,000 allocations: 0.000016 seconds
Measuring free time...
Segmentation fault (core dumped)
(base) lab@DESKTOP-KPRMFIO:~/mk/lr4$ ./alloc_test ./libmck.so
Library loaded successfully
Measuring allocation time...
Allocation time for 10,000 allocations: 0.000516 seconds
Measuring free time...
Free time for 10,000 deallocations: 0.000334 seconds
```

Strace

execve("/usr/bin/uname", ["uname"], 0x7ffc6075d880 /* 65 vars */) = 0

brk(NULL) = 0x56376bb5a000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f05673f5000

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=45219, ...}) = 0

mmap(NULL, 45219, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05673e9000

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"..., 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) = 0x7f05671d7000

mmap(0x7f05671ff000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f05671ff000

mmap(0x7f0567387000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f0567387000

mmap(0x7f05673d6000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f05673d6000

mmap(0x7f05673dc000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f05673dc000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f05671d4000

arch_prctl(ARCH_SET_FS, 0x7f05671d4740) = 0

set_tid_address(0x7f05671d4a10) = 4835

set_robust_list(0x7f05671d4a20, 24) = 0

rseq(0x7f05671d5060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f05673d6000, 16384, PROT_READ) = 0

mprotect(0x56374706c000, 4096, PROT_READ) = 0

mprotect(0x7f056742d000, 8192, PROT_READ) = 0

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

munmap(0x7f05673e9000, 45219) = 0

getrandom("\xed\xb1\x3f\xf8\x8f\x6d\x36\x59", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x56376bb5a000

brk(0x56376bb7b000) = 0x56376bb7b000

openat(AT_FDCWD, "/usr/share/locale/locale.alias", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=2996, ...}) = 0

read(3, "# Locale name alias data base.\n#"..., 4096) = 2996

read(3, "", 4096) = 0

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_IDENTIFICATION", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_IDENTIFICATION", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_IDENTIFICATION", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_IDENTIFICATION", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=258, ...}) = 0

mmap(NULL, 258, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05673f4000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=27028, ...}) = 0

mmap(NULL, 27028, PROT_READ, MAP_SHARED, 3, 0) = 0x7f05673ed000

close(3) = 0

futex(0x7f05673db72c, FUTEX_WAKE_PRIVATE, 2147483647) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_MEASUREMENT", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_MEASUREMENT", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_MEASUREMENT", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_MEASUREMENT", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=23, ...}) = 0

mmap(NULL, 23, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05673ec000

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_TELEPHONE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_TELEPHONE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_TELEPHONE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_TELEPHONE", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=47, ...}) = 0

mmap(NULL, 47, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05673eb000

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_ADDRESS", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_ADDRESS", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_ADDRESS", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_ADDRESS", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=127, ...}) = 0

mmap(NULL, 127, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05673ea000

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_NAME", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_NAME", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_NAME", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_NAME", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=62, ...}) = 0

mmap(NULL, 62, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05673e9000

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_PAPER", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_PAPER", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_PAPER", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_PAPER", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=34, ...}) = 0

mmap(NULL, 34, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05671d3000

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_MESSAGES", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_MESSAGES", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_MESSAGES", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_MESSAGES", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0

close(3) = 0

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_MESSAGES/SYS_LC_MESSAGES", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=48, ...}) = 0

mmap(NULL, 48, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05671d2000

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_MONETARY", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_MONETARY", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_MONETARY", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_MONETARY", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=270, ...}) = 0

mmap(NULL, 270, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05671d1000

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_COLLATE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_COLLATE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No suchfile or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_COLLATE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_COLLATE", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=1406, ...}) = 0

mmap(NULL, 1406, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05671d0000

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_TIME", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_TIME", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_TIME", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_TIME", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=3360, ...}) = 0

mmap(NULL, 3360, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05671cf000

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_NUMERIC", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_NUMERIC", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_NUMERIC", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_NUMERIC", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=50, ...}) = 0

mmap(NULL, 50, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f05671ce000

close(3) = 0

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.UTF-8/LC_CTYPE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_CTYPE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/snap/code/183/usr/lib/locale/C.utf8/LC_CTYPE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_CTYPE", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=360460, ...}) = 0

mmap(NULL, 360460, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f0567175000

close(3) = 0

uname({sysname="Linux", nodename="DESKTOP-KPRMFIO", ...}) = 0

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...}) = 0

write(1, "Linux\n", 6Linux

) = 6

close(1) = 0

close(2) = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы я изучил аллокаторы. Научился создавать, подключать и использовать динамические библиотеки. Реализовал два алгоритма аллокации памяти, узнал как работать с гитхабом, ибо git pull выдавал ошибки, на решение которых также было потрачено много времени.