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: Гига М.Я.

Преподаватель: Бахарев В.Д

Оценка: _____

Дата: 12.12.24

Москва, 2024

Постановка задачи

Вариант 9.

Исследовать два аллокатора памяти: необходимо реализовать два алгоритма аллокации памяти и сравнить их по следующим характеристикам: –

- Фактор использования –
- Скорость выделения блоков –
- Скорость освобождения блоков –
- Простота использования аллокатора

Требуется создать две динамические библиотеки, реализующие два аллокатора, соответственно. Библиотеки загружаются в память с помощью интерфейса ОС (dlopen / LoadLibrary) для работы с динамическими библиотеками. Выбор библиотеки, реализующей аллокатор, осуществляется чтением первого аргумента при запуске программы (argv[1]). Этот аргумент должен содержать путь до динамической библиотеки (относительный или абсолютный). Аллигаторы – метод двойников и список свободных блоков (наиболее подходящий).

Общий метод и алгоритм решения

Использованные системные вызовы:

- int write(int fd, void* buf, size_t count); – записывает count байт из buf в fd.
- void *mmap(void addr, size_t length, int prot, int flags, int fd, off_t offset); – выполняет отображение файла или устройства на память.
- int munmap(void addr, size_t length); – удаляет отображение файла или устройства на память.

Программа main в функции load_dynamic загружает динамическую библиотеку по указанному пути используя dlopen. Если библиотеку загрузить не удалось, выводится сообщение об ошибке и указателям на функции присвоены указатели на функции оборачивающими mmap и munmap. Если загрузить библиотеку удалось, то программа пытается найти в библиотеке символ соответствующий функции и присвоить указатель на него указателю на функцию. Если символа нет, то соответствующему указателю на функцию присвоен указатель на функцию оборачивающую mmap или munmap. В функции main load_dynamic вызывается с параметром argv[1]. Далее демонстрируется работа загруженных функции на примере работы с массивами.

Библиотека list_allocator реализует аллокатор на основании неявного связного списка. В этом аллокаторе память выделяется блоками, в начале и конце каждого из которых содержится HEADER из 4 байт. В HEADER записаны размер блока в байтах – четное число, и является ли блок свободным – последний бит. Функцию allocator_create выделяет память с помощью mmap и записывает ее размер

и указатель на начало в структуру Allocator. Функция allocator_destroy освобождает память через munmap. В функции allocator_alloc создается переменная ptr = NULL, p = allocator.memory. Программа в цикле проходит по всем блокам, увеличивая p на количество байт в HEADER и присваивая ptr значение p, если блок свободен, размер блока больше или равен необходимому и размер блока минимален. Если по завершении цикла ptr == NULL, то возвращается NULL: аллокация не удалась. В противном случае размер в HEADER меняется. Если новый размер меньше размера блока, то создается новый блок, в его HEADER записываются его размеры. Функция возвращает указатель на ptr со сдвигом в 4 байта. В функции allocator_free в HEADER последний бит меняется на 0 – блок свободен. Если соседние блоки свободны, происходит объединение с ними – в их HEADER меняется размер блока.

Библиотека buddy_allocator реализует аллокатор на основании метода двойников. В этом аллокаторе память выделяется блоками, размером 2^n. Каждый блок содержит HEADER, в котором записана его степень n, его путь от корневого: является ли он левым или правым блоком, указатель на то, является ли он свободным и два указателя на следующий и предыдущий блок. Функция allocator_create выделяет память с помощью mmap и записывает ее размер и указатель на начало в структуру Allocator. Также в список free аллокатора записывается указатель на первый блок. Функция allocator_destroy освобождает память через munmap. В функции allocator_alloc находится степень n необходимая для хранения информации. Далее программа проходит по связному списку свободных блоков и выбирает из них блок с минимальной степенью большей чем необходимая. Если такого блока нет, то возвращается NULL. Найденный блок удаляется из списка free. Если степень найденного блока больше необходимой, то он в цикле делится надвое пока его степень не станет равной необходимой. Все созданные правые блоки добавляются в список free. Функция возвращает указатель на выделенный блок со сдвигом в sizeof(Block) байт. Функция allocator_free пытается в цикле объединить освобождаемый блок с двойниками. Если двойник существует (не верхний блок) и он свободен, то двойник удаляется из списка free. Полученный после цикла объединенный блок помечается как свободный и добавляется в список free.

Код программы

main.c

```
// Вариант 9: списки свободных блоков (наиболее подходящее) и алгоритм двойников
#include <stdint.h>
#include <dlfcn.h>
#include <unistd.h>

#include <stdio.h>
#include <string.h>

#include <sys/mman.h>
```

```
#include "allocator.h"

struct Allocator {
    size_t size;
    uint32_t *memory;
};

static allocator_create_func *allocator_create;
static allocator_destroy_func *allocator_destroy;
static allocator_alloc_func *allocator_alloc;
static allocator_free_func *allocator_free;

Allocator *allocator_create_impl(void *const memory, const size_t size) {
    (void)size;
    (void)memory;
    return NULL;
}

void allocator_destroy_impl(Allocator *const allocator) {
    (void)allocator;
}

void *allocator_alloc_impl(Allocator *const allocator, const size_t size) {
    (void)allocator;
    uint32_t *memory = mmap(NULL, size + 1, PROT_READ | PROT_WRITE,
                            MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if (memory == MAP_FAILED) {
        return NULL;
    }
    *memory = size + 1;
    return memory + 1;
}

void allocator_free_impl(Allocator *const allocator, void *const memory) {
    (void)allocator;
    if (memory == NULL)
        return;
    uint32_t *mem = memory;
    mem--;
    munmap(mem, *mem);
}

#define LOAD_FUNCTION(name)                                                    \
    {                                                                          \
        name = (name##_func *)dlsym(library, #name);                           \
        if (allocator_create == NULL) {                                        \
            char message[] =                                                   \
                "WARNING: failed to find " #name " function implementation\n"; \
            write(STDERR_FILENO, message, strlen(message));                    \
            name = name##_impl;                                                \
        }                                                                      \
    }

void load_dynamic(const char *path) {
    void *library = dlopen(path, RTLD_LOCAL | RTLD_NOW);
    if (path && library) {
        LOAD_FUNCTION(allocator_create);
        LOAD_FUNCTION(allocator_destroy);
        LOAD_FUNCTION(allocator_alloc);
        LOAD_FUNCTION(allocator_free);
    } else {
        char message[] = "WARNING: failed to load shared library\n";
        write(STDERR_FILENO, message, strlen(message));
        allocator_create = allocator_create_impl;
```

```
        allocator_destroy = allocator_destroy_impl;
        allocator_alloc = allocator_alloc_impl;
        allocator_free = allocator_free_impl;
    }
}

int main(int argc, char *argv[]) {
    (void)argc;
    load_dynamic(argv[1]);
    char mem[1024];
    Allocator *all = allocator_create(mem, sizeof(mem));
    int *a = allocator_alloc(all, 12 * sizeof(int));
    int *b = allocator_alloc(all, 12 * sizeof(int));
    for (int i = 0; i < 12; i++) {
        a[i] = i;
        b[12 - i - 1] = i;
    }
    for (int i = 0; i < 12; i++) {
        printf("a[%d] = %d; b[%d] = %d\n", i, a[i], i, b[i]);
    }
    allocator_free(all, a);
    allocator_free(all, b);
    int *c = allocator_alloc(all, 4 * sizeof(int));
    for (int i = 0; i < 4; i++) {
        printf("c[%d] = %d\n", i, c[i]);
    }
    allocator_destroy(all);
}
```

allocator.h

```
#pragma once

#include <stddef.h>

typedef struct Allocator Allocator;

typedef float fabsf_func(float x);
typedef Allocator *allocator_create_func(void *const memory, const size_t size);
typedef void allocator_destroy_func(Allocator *const allocator);
typedef void *allocator_alloc_func(Allocator *const allocator,
                                   const size_t size);
typedef void allocator_free_func(Allocator *const allocator,
                                 void *const memory);
```

list_allocator.c

```
#include <stdio.h>
#include <string.h>
#include <sys/mman.h>
#include <stdlib.h>

#include "allocator.h"

#ifndef BLOCK_COUNT
#define BLOCK_COUNT 128
#endif

struct Allocator {
    size_t size;
    uint32_t *memory;
};
```

```
Allocator *allocator_create(void *const memory, const size_t size) {
    if (memory == NULL || size < sizeof(Allocator))
        return NULL;
    Allocator *all = (Allocator *)memory;
    all->size = BLOCK_COUNT; // 128 4 byte blocks
    if (all->size % 2 == 1)
        all->size++;

    all->memory = mmap(NULL, all->size * sizeof(uint32_t), PROT_READ | PROT_WRITE,
                       MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if (all->memory == MAP_FAILED)
        return NULL;
    *all->memory = all->size;
    *(all->memory + all->size - 1) = all->size;
    return all;
}

void allocator_destroy(Allocator *const allocator) {
    munmap(allocator->memory, allocator->size * sizeof(uint32_t));
}

void *allocator_alloc(Allocator *const allocator, const size_t size) {
    size_t size_in_blocks = (size + 3) / 4;
    size_t full_size = size_in_blocks + 2;
    uint32_t *end = allocator->memory + allocator->size;

    size_t diff = -1;
    uint32_t *ptr = NULL;

    for (uint32_t *p = allocator->memory; p < end; p = p + (*p & ~1)) {
        if (*p & 1)
            continue;
        if (*p < full_size) // we need memory for headers
            continue;

        if (size - *p < diff) {
            diff = size - *p;
            ptr = p;
        }
    }
    if (!ptr)
        return NULL;

    size_t newsize =
        full_size % 2 == 0 ? full_size : full_size + 1; // round to even bytes

    size_t oldsize = *ptr & ~1;
    *ptr = newsize | 1;          // front header
    *(ptr + newsize - 1) = *ptr; // back header

    if (newsize < oldsize)
        *(ptr + newsize) = oldsize - newsize;
    return ptr + 1;
}

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

    uint32_t *ptr = (uint32_t *)memory - 1; // current pointer
    *ptr = *ptr & ~1;

    uint32_t *front = ptr;
```

```
    uint32_t *back = ptr + *ptr - 1;
    size_t size = *ptr;
    printf("size = %zu\n", size);

    uint32_t *next = ptr + *ptr;
    if ((*next & 1) == 0) {
        back += *next;
        size += *next;
    }

    uint32_t *prev_back = ptr - 1;
    if ((*prev_back & 1) == 0) {
        front -= *prev_back;
        size += *prev_back;
    }
    *front = size;
    *back = size;
}
```

list_allocator.c

```
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>
#include <sys/mman.h>

#include "allocator.h"

#ifndef BLOCK_COUNT
#define BLOCK_COUNT 128
#endif

typedef struct Block Block;
struct Block {
    bool taken;
    uint32_t path;
    uint8_t size_class;
    Block *next;
    Block *prev;
};

#define MIN_SIZE_CLASS                                                          \
    ((size_t)ceil(log2(sizeof(Block) / sizeof(uint32_t) + 1)))
#define MAX_SIZE_CLASS ((size_t)log2(BLOCK_COUNT))

struct Allocator {
    size_t size;
    char *mem;
    Block *free;
};

Block *buddy(Block *p) {
    uint32_t *ptr = (uint32_t *)p;
    if (p->path & (1 << p->size_class))
        ptr -= (1 << p->size_class);
    else
        ptr += (1 << p->size_class);
    return (Block *)ptr;
}

Allocator *allocator_create(void *const memory, const size_t size) {
    if (memory == NULL || size < sizeof(Allocator))
        return NULL;
    Allocator *all = (Allocator *)memory;
```

```
    all->size = 1 << MAX_SIZE_CLASS; // 128 4 byte blocks
    all->mem = mmap(NULL, all->size * sizeof(uint32_t), PROT_READ | PROT_WRITE,
                    MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if (all->mem == MAP_FAILED)
        return NULL;

    all->free = (Block *)all->mem;
    all->free->taken = false;
    all->free->size_class = MAX_SIZE_CLASS;
    all->free->next = NULL;
    all->free->prev = NULL;
    all->free->path = 0;
    return all;
}

void allocator_destroy(Allocator *const allocator) {
    munmap(allocator->free, allocator->size * sizeof(uint32_t));
}

void *allocator_alloc(Allocator *const all, const size_t size) {
    if (all == 0 || size == 0)
        return NULL;
    size_t size_in_u32 = (size + sizeof(Block) + 3) / 4;
    size_t size_class = ceil(log2(size_in_u32));
    if (size_class > MAX_SIZE_CLASS)
        return NULL;
    Block *p = all->free;
    for (Block *c = all->free; c; c = c->next) {

        if (c->taken || (uint8_t)c->size_class < size_class)
            continue;
        if (p->taken) {
            p = c;
            continue;
        }
        if (c->size_class < p->size_class)
            p = c;
    }
    if (p->taken)
        return NULL;

    all->free = all->free->next;

    while (p->size_class > size_class) {
        p->size_class = p->size_class - 1;
        Block *other = buddy(p);
        other->size_class = p->size_class;
        other->next = all->free;
        other->prev = NULL;
        other->taken = false;
        other->path = p->path | 1 << p->size_class;
        if (all->free)
            all->free->prev = other;
        all->free = other;
    }
    p->taken = true;
    return (char *)p + sizeof(Block);
}

void allocator_free(Allocator *const allocator, void *const memory) {
    if (!allocator || !memory)
        return;
    Block *p = (Block *)((char *)memory - sizeof(Block));
    while (p->size_class < MAX_SIZE_CLASS) {
```

```
        Block *bud = buddy(p);
        if (bud->taken)
            break;
        if (bud->prev)
            bud->prev->next = bud->next;
        if (bud->next)
            bud->next->prev = bud->prev;
        if (allocator->free == bud)
            allocator->free = bud->next;
        // if buddy is free we can remove him from the free list and combine
        // with current
        p = p->path & (1 << p->size_class) ? bud : p;
        p->size_class++;
    }
    p->taken = false;
    p->next = allocator->free;
    p->prev = NULL;
    allocator->free = p;
}
```

Протокол работы программы

Тестирование:

```
$ ./main.out
WARNING: failed to load shared library
a[0] = 0; b[0] = 11
a[1] = 1; b[1] = 10
a[2] = 2; b[2] = 9
a[3] = 3; b[3] = 8
a[4] = 4; b[4] = 7
a[5] = 5; b[5] = 6
a[6] = 6; b[6] = 5
a[7] = 7; b[7] = 4
a[8] = 8; b[8] = 3
a[9] = 9; b[9] = 2
a[10] = 10; b[10] = 1
a[11] = 11; b[11] = 0
c[0] = 0
c[1] = 0
c[2] = 0
c[3] = 0

$ ./main.out buddy.so
a[0] = 0; b[0] = 11
a[1] = 1; b[1] = 10
a[2] = 2; b[2] = 9
a[3] = 3; b[3] = 8
a[4] = 4; b[4] = 7
a[5] = 5; b[5] = 6
a[6] = 6; b[6] = 5
a[7] = 7; b[7] = 4
a[8] = 8; b[8] = 3
a[9] = 9; b[9] = 2
a[10] = 10; b[10] = 1
a[11] = 11; b[11] = 0
c[0] = 0
c[1] = 1
c[2] = 2
c[3] = 3
```

```
$ ./main.out list.so
a[0] = 0; b[0] = 11
a[1] = 1; b[1] = 10
a[2] = 2; b[2] = 9
a[3] = 3; b[3] = 8
a[4] = 4; b[4] = 7
a[5] = 5; b[5] = 6
a[6] = 6; b[6] = 5
a[7] = 7; b[7] = 4
a[8] = 8; b[8] = 3
a[9] = 9; b[9] = 2
a[10] = 10; b[10] = 1
a[11] = 11; b[11] = 0
c[0] = 0
c[1] = 1
c[2] = 2
c[3] = 3
```

Strace:

```
$ sudo dtruss -f ./main.out ./link.so
       PID/THRD  SYSCALL(args)           = return
 1481/0x4113:  fork()             = 0 0
 1481/0x4113:  munmap(0x1028CC000, 0x84000)          = 0 0
 1481/0x4113:  munmap(0x102950000, 0x8000)           = 0 0
 1481/0x4113:  munmap(0x102958000, 0x4000)           = 0 0
 1481/0x4113:  munmap(0x10295C000, 0x4000)           = 0 0
 1481/0x4113:  munmap(0x102960000, 0x48000)          = 0 0
 1481/0x4113:  munmap(0x1029A8000, 0x4C000)          = 0 0
 1481/0x4113:  crossarch_trap(0x0, 0x0, 0x0)         = -1 Err#45
 1481/0x4113:  open(".\0", 0x100000, 0x0)            = 3 0
 1481/0x4113:  fcntl(0x3, 0x32, 0x16D5C7098)         = 0 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  fsgetpath(0x16D5C70A8, 0x400, 0x16D5C7088)         = 49 0
 1481/0x4113:  fsgetpath(0x16D5C70B8, 0x400, 0x16D5C7098)         = 14 0
 1481/0x4113:  csrctl(0x0, 0x16D5C74BC, 0x4)         = -1 Err#1
 1481/0x4113:  __mac_syscall(0x19C94FD62, 0x2, 0x16D5C7400)              = 0 0
 1481/0x4113:  csrctl(0x0, 0x16D5C74AC, 0x4)         = -1 Err#1
 1481/0x4113:  __mac_syscall(0x19C94CB95, 0x5A, 0x16D5C7440)             = 0 0
 1481/0x4113:  sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16D5C69A8, 0x16D5C69A0,
0x19C94E888, 0xD)          = 0 0
 1481/0x4113:  sysctl([CTL_KERN, 157, 0, 0, 0, 0] (2), 0x16D5C6A58, 0x16D5C6A50, 0x0,
0x0)          = 0 0
 1481/0x4113:  open("/\0", 0x20100000, 0x0)          = 3 0
 1481/0x4113:  openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0)             = 4 0
 1481/0x4113:  dup(0x4, 0x0, 0x0)              = 5 0
 1481/0x4113:  fstatat64(0x4, 0x16D5C6531, 0x16D5C64A0)           = 0 0
 1481/0x4113:  openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0)            = 6 0
 1481/0x4113:  fcntl(0x6, 0x32, 0x16D5C6530)         = 0 0
 1481/0x4113:  dup(0x6, 0x0, 0x0)              = 7 0
 1481/0x4113:  dup(0x5, 0x0, 0x0)              = 8 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  close(0x5)         = 0 0
 1481/0x4113:  close(0x4)         = 0 0
 1481/0x4113:  close(0x6)         = 0 0
 1481/0x4113:  __mac_syscall(0x19C94FD62, 0x2, 0x16D5C6F20)              = 0 0
 1481/0x4113:  shared_region_check_np(0x16D5C6B40, 0x0, 0x0)             = 0 0
 1481/0x4113:  fsgetpath(0x16D5C70C0, 0x400, 0x16D5C6FE8)         = 82 0
 1481/0x4113:  fcntl(0x8, 0x32, 0x16D5C70C0)         = 0 0
 1481/0x4113:  close(0x8)         = 0 0
 1481/0x4113:  close(0x7)         = 0 0
 1481/0x4113:  getfsstat64(0x0, 0x0, 0x2)            = 10 0
 1481/0x4113:  getfsstat64(0x102838050, 0x54B0, 0x2)        = 10 0
 1481/0x4113:  getattrlist("/\0", 0x16D5C7000, 0x16D5C6F70)              = 0 0
```

```
 1481/0x4113:  stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/
dyld_shared_cache_arm64e\0", 0x16D5C7360, 0x0)              = 0 0
 1481/0x4113:  stat64("/Users/maxgiga/dev/mai/MAI_OS/lab04/src/main.out\0", 0x16D5C6810,
0x0)          = 0 0
 1481/0x4113:  open("/Users/maxgiga/dev/mai/MAI_OS/lab04/src/main.out\0", 0x0, 0x0)
= 3 0
 1481/0x4113:  mmap(0x0, 0x8888, 0x1, 0x40002, 0x3, 0x0)          = 0x102838000 0
 1481/0x4113:  fcntl(0x3, 0x32, 0x16D5C6928)         = 0 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  munmap(0x102838000, 0x8888)           = 0 0
 1481/0x4113:  stat64("/Users/maxgiga/dev/mai/MAI_OS/lab04/src/main.out\0", 0x16D5C6D80,
0x0)          = 0 0
 1481/0x4113:  stat64("/usr/lib/libSystem.B.dylib\0", 0x16D5C5CD0, 0x0)        = -1
Err#2
 1481/0x4113:  stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/
libSystem.B.dylib\0", 0x16D5C5C80, 0x0)        = -1 Err#2
 1481/0x4113:  stat64("/usr/lib/libSystem.B.dylib\0", 0x16D5C5CD0, 0x0)        = -1
Err#2
 1481/0x4113:  open("/Users/maxgiga/dev/mai/MAI_OS/lab04/src/main.out\0", 0x0, 0x0)
= 3 0
 1481/0x4113:  __mac_syscall(0x19C94FD62, 0x2, 0x16D5C41A0)              = 0 0
 1481/0x4113:  map_with_linking_np(0x16D5C4070, 0x1, 0x16D5C40A0)        = 0 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  mprotect(0x10282C000, 0x4000, 0x1)          = 0 0
 1481/0x4113:  open("/dev/dtracehelper\0", 0x2, 0x0)       = 3 0
 1481/0x4113:  ioctl(0x3, 0x80086804, 0x16D5C3528)         = 0 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)            = 0 0
 1481/0x4113:  access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0)            = -1
Err#2
 1481/0x4113:  bsdthread_register(0x19CC520F4, 0x19CC520E8, 0x4000)            =
1073746399 0
 1481/0x4113:  getpid(0x0, 0x0, 0x0)           = 1481 0
 1481/0x4113:  shm_open(0x19CAE9F41, 0x0, 0xFFFFFFFF9CC90000)           = 3 0
 1481/0x4113:  mmap(0x0, 0x8000, 0x1, 0x40001, 0x3, 0x0)          = 0x102840000 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  csops(0x5C9, 0x0, 0x16D5C3CDC)         = 0 0
 1481/0x4113:  ioctl(0x2, 0x4004667A, 0x16D5C3C4C)         = -1 Err#25
 1481/0x4113:  ioctl(0x2, 0x40487413, 0x16D5C3C50)         = -1 Err#25
 1481/0x4113:  mprotect(0x102850000, 0x4000, 0x0)          = 0 0
 1481/0x4113:  mprotect(0x10285C000, 0x4000, 0x0)          = 0 0
 1481/0x4113:  mprotect(0x102860000, 0x4000, 0x0)          = 0 0
 1481/0x4113:  mprotect(0x10286C000, 0x4000, 0x0)          = 0 0
 1481/0x4113:  mprotect(0x102870000, 0x4000, 0x0)          = 0 0
 1481/0x4113:  mprotect(0x10287C000, 0x4000, 0x0)          = 0 0
 1481/0x4113:  mprotect(0x102848000, 0xC8, 0x1)            = 0 0
 1481/0x4113:  mprotect(0x102848000, 0xC8, 0x3)            = 0 0
 1481/0x4113:  mprotect(0x102848000, 0xC8, 0x1)            = 0 0
 1481/0x4113:  mprotect(0x102880000, 0x4000, 0x1)          = 0 0
 1481/0x4113:  mprotect(0x102884000, 0xC8, 0x1)            = 0 0
 1481/0x4113:  mprotect(0x102884000, 0xC8, 0x3)            = 0 0
 1481/0x4113:  mprotect(0x102884000, 0xC8, 0x1)            = 0 0
 1481/0x4113:  mprotect(0x102848000, 0xC8, 0x3)            = 0 0
 1481/0x4113:  mprotect(0x102848000, 0xC8, 0x1)            = 0 0
 1481/0x4113:  mprotect(0x102880000, 0x4000, 0x3)          = 0 0
 1481/0x4113:  mprotect(0x102880000, 0x4000, 0x1)          = 0 0
 1481/0x4113:  issetugid(0x0, 0x0, 0x0)        = 0 0
 1481/0x4113:  getentropy(0x16D5C32B8, 0x20, 0x0)          = 0 0
 1481/0x4113:  getattrlist("/Users/maxgiga/dev/mai/MAI_OS/lab04/src/main.out\0",
0x16D5C3B40, 0x16D5C3B5C)         = 0 0
 1481/0x4113:  access("/Users/maxgiga/dev/mai/MAI_OS/lab04/src\0", 0x4, 0x0)          =
0 0
 1481/0x4113:  open("/Users/maxgiga/dev/mai/MAI_OS/lab04/src\0", 0x0, 0x0)            =
3 0
```

```
 1481/0x4113:  csrctl(0x0, 0x16D5C3D2C, 0x4)         = -1 Err#1
 1481/0x4113:  fgetattrlist(0x3, 0x16D5C3DD0, 0x16D5C3D50)        = 0 0
 1481/0x4113:  __mac_syscall(0x1A8BFA505, 0x2, 0x16D5C3D50)              = 0 0
 1481/0x4113:  fcntl(0x3, 0x32, 0x16D5C3A28)         = 0 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  open("/Users/maxgiga/dev/mai/MAI_OS/lab04/src/Info.plist\0", 0x0, 0x0)
= -1 Err#2
 1481/0x4113:  proc_info(0x2, 0x5C9, 0xD)            = 64 0
 1481/0x4113:  csops_audittoken(0x5C9, 0x10, 0x16D5C3DB0)         = 0 0
 1481/0x4113:  sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16D5C4108, 0x16D5C4100,
0x1A0363D3A, 0x15)         = 0 0
 1481/0x4113:  sysctl([CTL_KERN, 155, 0, 0, 0, 0] (2), 0x16D5C4198, 0x16D5C4190, 0x0,
0x0)          = 0 0
 1481/0x4113:  open("./list.so\0", 0x0, 0x0)         = 3 0
 1481/0x4113:  fcntl(0x3, 0x32, 0x16D5D5BC8)         = 0 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  open("./list.so\0", 0x0, 0x0)         = 3 0
 1481/0x4113:  mmap(0x0, 0x8528, 0x1, 0x40002, 0x3, 0x0)          = 0x10288C000 0
 1481/0x4113:  fcntl(0x3, 0x32, 0x16D5D5278)         = 0 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  open("/Users/maxgiga/dev/mai/MAI_OS/lab04/src/list.so\0", 0x0, 0x0)
= 3 0
 1481/0x4113:  fcntl(0x3, 0x61, 0x16D5D4F18)         = 0 0
 1481/0x4113:  fcntl(0x3, 0x62, 0x16D5D4F18)         = 0 0
 1481/0x4113:  mmap(0x102898000, 0x4000, 0x5, 0x40012, 0x3, 0x0)         = 0x102898000 0
 1481/0x4113:  mmap(0x10289C000, 0x4000, 0x3, 0x40012, 0x3, 0x4000)             =
0x10289C000 0
 1481/0x4113:  mmap(0x1028A0000, 0x4000, 0x1, 0x40012, 0x3, 0x8000)             =
0x1028A0000 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  munmap(0x10288C000, 0x8528)           = 0 0
 1481/0x4113:  open("/Users/maxgiga/dev/mai/MAI_OS/lab04/src/list.so\0", 0x0, 0x0)
= 3 0
 1481/0x4113:  close(0x3)         = 0 0
 1481/0x4113:  mprotect(0x10289C000, 0x4000, 0x1)           = 0 0
 1481/0x4113:  mmap(0x0, 0x200, 0x3, 0x41001, 0xFFFFFFFFFFFFFFFF, 0x0)         =
0x10288C000 0
 1481/0x4113:  munmap(0x10288C000, 0x200)            = 0 0
 1481/0x4113:  write_nocancel(0x1, "a[0] = 0; b[0] = 11\na[1] = 1; b[1] = 10\na[2] = 2;
b[2] = 9\na[3] = 3; b[3] = 8\na[4] = 4; b[4] = 7\na[5] = 5; b[5] = 6\na[6] = 6; b[6] =
5\na[7] = 7; b[7] = 4\na[8] = 8; b[8] = 3\na[9] = 9; b[9] = 2\na[10] = 10; b[10] =
1\na[11] = 11; b[11] = 0\nsize = 14\nsize = 14\n", 0x124)         = 292 0
```

Вывод

В ходе выполнения лабораторной работы была составлена и отлажена программа на языке C, осуществляющая загрузку динамической библиотеки по пути, переданному в аргументах командной строки. Также были изучены два различных аллокатора и написаны динамические библиотеки, реализующие их