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-214Б-23

Студент: Демидов М.С.

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 19.10.24

Москва, 2024

Постановка

задачи Вариант 5.

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).
Если аргумент не передан или по переданному пути библиотеки не оказалось, то указатели на функции, реализующие API аллокатора ниже, должны быть присвоены функциям, которые оборачивают системный аллокатор ОС (mmap / VirtualAlloc) в этот API. Эти аварийные оберточные функции должны быть реализованы внутри программы, которая загружает динамические библиотеки (см. пример на GitHub Gist).
Каждый аллокатор памяти должен иметь функции аналогичные стандартным функциям malloc и free (realloc, опционально). Перед работой каждый аллокатор инициализируется свободными страницами памяти, выделенными стандартными средствами ядра (mmap / VirtualAlloc). Необходимо самостоятельно разработать стратегию тестирования для определения ключевых характеристик аллокаторов памяти. При тестировании нужно свести к минимуму потери точности из-за накладных расходов при измерении ключевых характеристик, описанных выше.
Каждый аллокатор должен обладать следующим интерфейсом (могут быть отличия в зависимости от особенностей алгоритма):
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, const size_t size) (выделение памяти аллокатором памяти размера size);
void allocator_free(Allocator *const allocator, void *const memory) (возвращает выделенную память аллокатору);
В отчете необходимо отобразить следующее:
– Подробное описание каждого из исследуемых алгоритмов
– Процесс тестирования
– Обоснование подхода тестирования
– Результаты тестирования
Алгоритм Мак-Кьюзика-Кэрелса и алгоритм двойников;

Общий метод и алгоритм

решения Использованные системные вызовы:

- mmap — выделяет память.
- munmap — освобождает память.
- dlopen — загружает библиотеку.
- dlsym — ищет символ в библиотеке.
- dlclose — закрывает библиотеку.

Удалить винду, поставить линукс, стать человеком.
Делаем две библиотеки по алгоритмам из презентаций:

Алгоритм Мак-Кьюзика-Кэрелса

1. Является улучшенной версией алгоритма «Блоки по 2^n»
2. Храним массив страниц
   a) Страница может быть свободной и хранить ссылку на следующую свободную страницу
   b) Может быть разбита на блоки - в массиве содержится размер блока, на которые разбита страница
   c) Указатель на то, что страница является частью крупного блока памяти
3. Храним массив с указателями на списки свободных блоков
   - Блоки имеют размер равный степени 2
   - Блоки внутри одной страницы имеют одинаковый размер

Основные положения алгоритма

1. Бьем свободный пул до тех пор пока не получаем блок нужного размера
2. Части разделенного блока называют двойниками
3. В каждом блоке надо хранить тег «свободен»/ «занят»
4. Если известен адрес блока и его размер, то известен адрес двойника
   Адрес: 10110000, размер: 16 -> адрес двойника: 10100000
5. Если освобождается блок, и его двойник оказывается свободен, то двойников сливают. Полученный блок пытаются слить с его двойником. Блок, который не удалось слить добавляют в список свободных блоков
6. Свободные блоки хранятся в двусвязном списке

Алгоритм Мак-Кьюзика-Кэрелса

1. Является улучшенной версией алгоритма «Блоки по 2^n»
2. Храним массив страниц
   a) Страница может быть свободной и хранить ссылку на следующую свободную страницу
   b) Может быть разбита на блоки - в массиве содержится размер блока, на которые разбита страница
   c) Указатель на то, что страница является частью крупного блока памяти
3. Храним массив с указателями на списки свободных блоков
   - Блоки имеют размер равный степени 2
   - Блоки внутри одной страницы имеют одинаковый размер

Основные положения алгоритма

1. Бьем свободный пул до тех пор пока не получаем блок нужного размера
2. Части разделенного блока называют двойниками
3. В каждом блоке надо хранить тег «свободен»/ «занят»
4. Если известен адрес блока и его размер, то известен адрес двойника
   Адрес: 10110000, размер: 16 -> адрес двойника: 10100000
5. Если освобождается блок, и его двойник оказывается свободен, то двойников сливают. Полученный блок пытаются слить с его двойником. Блок, который не удалось слить добавляют в список свободных блоков
6. Свободные блоки хранятся в двусвязном списке

Ну а дальше просто создаем main, в котором получаем библиотеку, проверяем функции, замеры, все дела, готово.

Код программ

buddy_alloc.c

```
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include "buddy_alloc.h"

#define MIN_BLOCK_SIZE 8
#define MAX_LEVELS 20

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

```
size_t get_next_power_of_two(size_t size) {
    size_t power = 1;
    while (power < size) {
        power <<= 1;
    }
    return power;
}

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

    size_t actual_size =
get_next_power_of_two(size);
    allocator->memory = mmap(NULL,
actual_size, PROT_READ | PROT_WRITE,
MAP_ANONYMOUS | MAP_PRIVATE, -1,
0);
    if (allocator->memory == MAP_FAILED) {
        munmap(allocator, sizeof(Allocator));
        return NULL;
```

```
    }

    allocator->size = actual_size;
    allocator->bitmap =
(uint8_t*)mmap(NULL, actual_size /
MIN_BLOCK_SIZE, PROT_READ |
PROT_WRITE, MAP_ANONYMOUS |
MAP_PRIVATE, -1, 0);
    if (allocator->bitmap == MAP_FAILED) {
        munmap(allocator->memory,
actual_size);
        munmap(allocator, sizeof(Allocator));
        return NULL;
    }

    memset(allocator->bitmap, 0, actual_size /
MIN_BLOCK_SIZE);
    return allocator;
}

void allocator_destroy(Allocator *const
allocator) {
    if (allocator) {
        munmap(allocator->memory,
allocator->size);
        munmap(allocator->bitmap,
allocator->size / MIN_BLOCK_SIZE);
        munmap(allocator, sizeof(Allocator));
    }
```

```
}

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

    size_t block_size =
get_next_power_of_two(size);
    if (block_size < MIN_BLOCK_SIZE) {
        block_size = MIN_BLOCK_SIZE;
    }

    size_t index = 0;
    size_t level = 0;
    while (block_size < allocator->size) {
        block_size <<= 1;
        level++;
    }

    if (block_size > allocator->size) {
        return NULL;
    }

    for (size_t i = 0; i < (allocator->size /
block_size); i++) {
        if (!allocator->bitmap[index + i]) {
            allocator->bitmap[index + i] = 1;
```

```
            return
(void*)((uint8_t*)allocator->memory + (i *
block_size));
        }
    }

    return NULL;
}

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

    size_t offset = (uint8_t*)memory -
(uint8_t*)allocator->memory;
    size_t block_size = MIN_BLOCK_SIZE;
    size_t index = offset / block_size;

    while (block_size < allocator->size) {
        if (allocator->bitmap[index]) {
            allocator->bitmap[index] = 0;
            return;
        }
        block_size <<= 1;
        index /= 2;
    }
}
```

mc_karels_alloc.c

```
#include "mc_karels_alloc.h"
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

#define MAX_CLASSES 10
#define PAGE_SIZE 4096

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

typedef struct Allocator {
    Block* free_list[MAX_CLASSES];
    size_t class_sizes[MAX_CLASSES];
    void* memory_start;
    size_t memory_size;
} Allocator;

static size_t find_class(size_t size, size_t*
class_sizes, size_t num_classes) {
    for (size_t i = 0; i < num_classes; i++) {
        if (size <= class_sizes[i]) {
            return i;
        }
    }
    return num_classes; // Error case
}
```

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

    allocator->memory_start = (char*)memory
+ sizeof(Allocator);
    allocator->memory_size = size -
sizeof(Allocator);

    // Инициализация классов размеров
    size_t block_size = 16;
    for (size_t i = 0; i < MAX_CLASSES; i++) {
        allocator->class_sizes[i] = block_size;
        allocator->free_list[i] = NULL;
        block_size *= 2;
    }

    return allocator;
}

void allocator_destroy(Allocator* const
allocator) {
    // Очистка списков свободных блоков
    for (size_t i = 0; i < MAX_CLASSES; i++) {
        allocator->free_list[i] = NULL;
    }
}
```

```
void* allocator_alloc(Allocator* const
allocator, const size_t size) {
    size_t class_index = find_class(size,
allocator->class_sizes, MAX_CLASSES);
    if (class_index >= MAX_CLASSES) return
NULL;

    Block* block =
allocator->free_list[class_index];
    if (block) {
        allocator->free_list[class_index] =
block->next;
        return (void*)block;
    }

    // Если свободных блоков нет, выделяем
из общей памяти
    size_t block_size =
allocator->class_sizes[class_index];
    if (allocator->memory_size < block_size)
return NULL;

    void* memory = allocator->memory_start;
    allocator->memory_start =
(char*)allocator->memory_start + block_size;
    allocator->memory_size -= block_size;
    return memory;
}
```

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

    uintptr_t addr = (uintptr_t)memory -
(uintptr_t)allocator;
    if (addr >= allocator->memory_size)
return;

    size_t class_index = 0;
    size_t block_size = 0;
    for (; class_index < MAX_CLASSES;
class_index++) {
        block_size =
allocator->class_sizes[class_index];
        if ((uintptr_t)memory % block_size == 0)
{
            break;
        }
    }

    if (class_index >= MAX_CLASSES) return;

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

lab4.c

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

typedef struct Allocator {
    size_t size;   // Размер памяти,
выделенной для аллокатора
    void *memory;  // Указатель на
выделенную память
} Allocator;

// Определение типов для функций из
динамических библиотек
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);

// Указатели на функции, которые будут
```

```
загружаться динамически
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;

// Функции для fallback-аллокатора
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));
}

// Функция для загрузки библиотеки
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);
    }
}

// Функция для измерения времени работы
аллокатора
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;
}

// Функция для измерения времени
освобождения памяти
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; //
10MB
```

```
    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;
    }

    // Сравнение времени работы
аллокаторов
    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);

    // Уничтожение аллокатора
    allocator_destroy(allocator);
    munmap(memory, memory_size);

    return 0;
}
```

Протокол работы программы

```
root@DESKTOP-B8KA07G:/mnt/d/Users/lenovo/Desktop/osi/OSmai/Lab4/traitor#
strace ./lab4 ./mc_karels_library.so
execve("./lab4", ["./lab4", "./mc_karels_library.so"], 0x7ffc12826908 /* 27 vars */) = 0
brk(NULL)                               = 0x557477f16000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcc51fbac0) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcc12f2a000
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=22867, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 22867, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fcc12f24000
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) = 0x7fcc12cfb000
mprotect(0x7fcc12d23000, 2023424, PROT_NONE) = 0
mmap(0x7fcc12d23000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x7fcc12d23000
mmap(0x7fcc12eb8000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x7fcc12eb8000
mmap(0x7fcc12f11000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x7fcc12f11000
mmap(0x7fcc12f17000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
0) = 0x7fcc12f17000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcc12cf8000
arch_prctl(ARCH_SET_FS, 0x7fcc12cf8740) = 0
set_tid_address(0x7fcc12cf8a10)         = 632
set_robust_list(0x7fcc12cf8a20, 24)     = 0
rseq(0x7fcc12cf90e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7fcc12f11000, 16384, PROT_READ) = 0
mprotect(0x557438af6000, 4096, PROT_READ) = 0
mprotect(0x7fcc12f64000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
```

```
munmap(0x7fcc12f24000, 22867)           = 0
getrandom("\x30\xea\x84\xc8\xbe\x92\xc7\xd5", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x557477f16000
brk(0x557477f37000)                     = 0x557477f37000
openat(AT_FDCWD, "./mc_karels_library.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|0777, st_size=15312, ...}, AT_EMPTY_PATH) = 0
getcwd("/mnt/d/Users/lenovo/Desktop/osi/OSmai/Lab4/traitor", 128) = 51
mmap(NULL, 16424, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fcc12f25000
mmap(0x7fcc12f26000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1000) = 0x7fcc12f26000
mmap(0x7fcc12f27000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) =
0x7fcc12f27000
mmap(0x7fcc12f28000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x2000) = 0x7fcc12f28000
close(3)                                = 0
mprotect(0x7fcc12f28000, 4096, PROT_READ) = 0
newfstatat(1, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
write(1, "Library loaded successfully\n", 28Library loaded successfully
) = 28
mmap(NULL, 10485760, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fcc122f8000
write(1, "Measuring allocation time...\n", 29Measuring allocation time...
) = 29
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=2155100}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=2983100}) = 0
write(1, "Allocation time for 10,000 alloc"..., 57Allocation time for 10,000 allocations:
0.000828 seconds
) = 57
write(1, "Measuring free time...\n", 23Measuring free time...
) = 23
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=3212400}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=3310900}) = 0
write(1, "Free time for 10,000 deallocatio"..., 53Free time for 10,000 deallocations:
0.000098 seconds
) = 53
munmap(0x7fcc122f8000, 10485760)        = 0
exit_group(0)                           = ?
+++ exited with 0 +++
root@DESKTOP-B8KA07G:/mnt/d/Users/lenovo/Desktop/osi/OSmai/Lab4/traitor#

root@DESKTOP-B8KA07G:/mnt/d/Users/lenovo/Desktop/osi/OSmai/Lab4/traitor#
strace ./lab4 ./buddy_library.so
execve("./lab4", ["./lab4", "./buddy_library.so"], 0x7ffc21e7e678 /* 27 vars */) = 0
brk(NULL)                               = 0x563f85892000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffca77dca70) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f663b432000
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=22867, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 22867, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f663b42c000
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) = 0x7f663b203000
mprotect(0x7f663b22b000, 2023424, PROT_NONE) = 0
mmap(0x7f663b22b000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x7f663b22b000
mmap(0x7f663b3c0000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
```

```
0x7f663b3c0000
mmap(0x7f663b419000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x7f663b419000
mmap(0x7f663b41f000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
0) = 0x7f663b41f000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f663b200000
arch_prctl(ARCH_SET_FS, 0x7f663b200740) = 0
set_tid_address(0x7f663b200a10)         = 688
set_robust_list(0x7f663b200a20, 24)     = 0
rseq(0x7f663b2010e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f663b419000, 16384, PROT_READ) = 0
mprotect(0x563f57a03000, 4096, PROT_READ) = 0
mprotect(0x7f663b46c000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f663b42c000, 22867)           = 0
getrandom("\xa7\x36\x97\x29\x49\xde\x9a\x5b", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x563f85892000
brk(0x563f858b3000)                     = 0x563f858b3000
openat(AT_FDCWD, "./buddy_library.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|0777, st_size=15792, ...}, AT_EMPTY_PATH) = 0
getcwd("/mnt/d/Users/lenovo/Desktop/osi/OSmai/Lab4/traitor", 128) = 51
mmap(NULL, 16456, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f663b42d000
mmap(0x7f663b42e000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1000) = 0x7f663b42e000
mmap(0x7f663b42f000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) =
0x7f663b42f000
mmap(0x7f663b430000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x2000) = 0x7f663b430000
close(3)                                = 0
mprotect(0x7f663b430000, 4096, PROT_READ) = 0
newfstatat(1, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
write(1, "Library loaded successfully\n", 28Library loaded successfully
) = 28
mmap(NULL, 10485760, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f663a800000
mmap(NULL, 24, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f663b46b000
mmap(NULL, 16777216, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f6639800000
mmap(NULL, 2097152, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6639600000
write(1, "Measuring allocation time...\n", 29Measuring allocation time...
) = 29
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=3520100}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=3779900}) = 0
write(1, "Allocation time for 10,000 alloc"..., 57Allocation time for 10,000 allocations:
0.000259 seconds
) = 57
write(1, "Measuring free time...\n", 23Measuring free time...
) = 23
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=4289600}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=4335100}) = 0
write(1, "Free time for 10,000 deallocatio"..., 53Free time for 10,000 deallocations:
0.000046 seconds
) = 53
munmap(0x7f6639800000, 16777216)        = 0
munmap(0x7f6639600000, 2097152)         = 0
munmap(0x7f663b46b000, 24)              = 0
munmap(0x7f663a800000, 10485760)        = 0
exit_group(0)                           = ?
+++ exited with 0 +++
root@DESKTOP-B8KA07G:/mnt/d/Users/lenovo/Desktop/osi/OSmai/Lab4/traitor#
```

Вывод

Я сломал права доступа к файлам на винде и пощупал системное программирование на линуксе.