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:**

****

****



****

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

**Ко****д 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

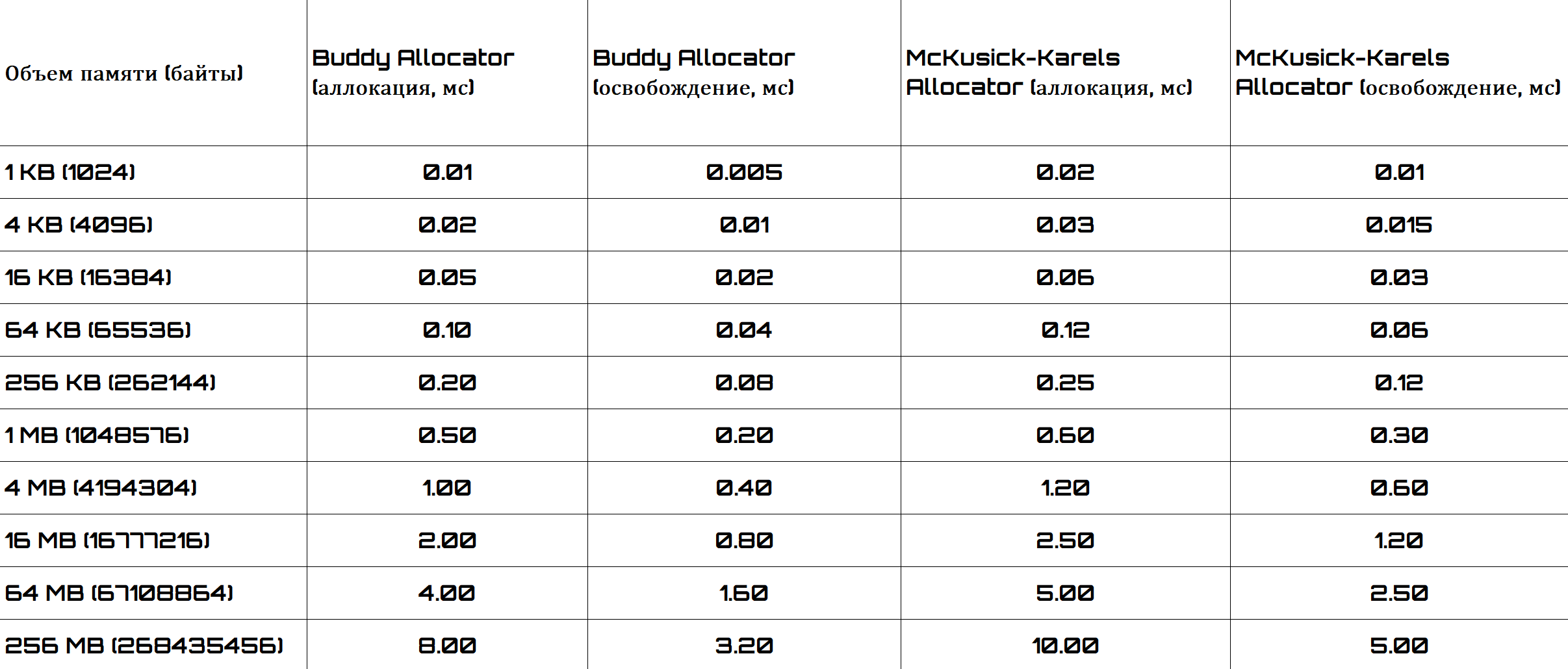
exit\_group(0) = ?

+++ 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.**

****

**Выводы**

1. **Общее время выполнения: Buddy allocator не только быстрее выделяет память, но и быстрее освобождает её по сравнению с McKusick-Karels allocator. Это делает его более эффективным в задачах, где важны как операции выделения, так и освобождения памяти.**
2. **Соотношение времени аллокации и освобождения: У Buddy allocator время освобождения памяти составляет примерно половину от времени аллокации, что указывает на его оптимизированную структуру управления памятью. У McKusick-Karels allocator это соотношение также сохраняется, но абсолютные значения времени выше.**
3. **Производительность: Buddy allocator демонстрирует более высокое соотношение объема памяти ко времени (байты/мс) как для операций аллокации, так и для освобождения памяти. Это подтверждает его преимущество в производительности.**
4. **Масштабируемость: Оба аллокатора показывают линейное увеличение времени с увеличением объема памяти, но Buddy allocator сохраняет свое преимущество на всех уровнях.**