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: Ефременко К.А.

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 01.11.24

Москва, 2024

Постановка задачи

Вариант 4.

Алгоритм Мак-Кьюзика-Кэрелса и блоки по 2^n;

Общий метод и алгоритм решения

Кратко опишите системные вызовы, которые вы использовали в лабораторной работе.

Использованные системные вызовы:

- **`write()`**
  - Используется для вывода данных в стандартные потоки (`STDOUT`, `STDERR`).
  - Прямо записывает данные в файловый дескриптор, минуя буферизацию стандартных библиотек.
  - Пример: вывод результатов тестирования.

- **`mmap()`**
  - Выделяет память напрямую из ОС.
  - Используется для инициализации области памяти, где аллокатор управляет блоками.
  - Пример: выделение 1 МБ для аллокатора.

- **`munmap()`**
  - Освобождает память, выделенную с помощью `mmap()`.
  - Пример: освобождение памяти в аварийном аллокаторе.

- **`dlopen()`**
  - Загружает динамическую библиотеку в память.
  - Пример: загрузка библиотек аллокаторов во время выполнения.

- **`dlsym()`**
  - Получает адрес функции из загруженной библиотеки.
  - Пример: нахождение функций `allocator_alloc` или `allocator_free`.

- **`dlclose()`**
  - Освобождает ресурсы, связанные с динамической библиотекой.
  - Пример: завершение работы с библиотекой после тестов.

- **Анализ требований**

- Определите интерфейс аллокаторов и задачи (две реализации, тестирование характеристик).

- **Реализация аллокаторов**

  - Разработайте структуры данных для каждого алгоритма.
  - Реализуйте основные функции: `allocator_create`, `allocator_alloc`, `allocator_free`, `allocator_destroy`.
  - Скомпилируйте аллокаторы в динамические библиотеки.

- **Разработка тестирующей программы**

  - Используйте системные вызовы `dlopen` и `dlsym` для загрузки аллокаторов.
  - Реализуйте аварийный аллокатор на основе `mmap`.
  - Напишите тесты для измерения времени выполнения и проверки корректности.

- **Отладка**

  - Добавьте минимальный вывод через `write()` для диагностики.
  - Проверьте работу с разными библиотеками и тестовыми размерами блоков.

Код программы

Allocator-mc.c

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

#define NUM_BUCKETS 32
#define MIN_BLOCK_SIZE 16

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

typedef struct Allocator
{
    void *memory;
    size_t size;
    Block *buckets[NUM_BUCKETS];
} Allocator;

static int get_bucket(size_t size)
{
    size_t adjusted_size = size > MIN_BLOCK_SIZE ? size : MIN_BLOCK_SIZE;
    int bucket = 0;
    while ((1U << bucket) < adjusted_size && bucket < NUM_BUCKETS - 1)
    {
```

```
        bucket++;
    }
    return bucket;
}

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

    Allocator *allocator = (Allocator *)memory;
    allocator->memory = (void *)((uintptr_t)memory + sizeof(Allocator));
    allocator->size = size - sizeof(Allocator);

    memset(allocator->buckets, 0, sizeof(allocator->buckets));

    int largest_bucket = get_bucket(allocator->size);
    Block *initial_block = (Block *)allocator->memory;
    initial_block->next = NULL;
    allocator->buckets[largest_bucket] = initial_block;

    return allocator;
}

void allocator_destroy(Allocator *const allocator)
{
    if (!allocator)
        return;
}

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

    int bucket = get_bucket(size);

    for (int i = bucket; i < NUM_BUCKETS; i++)
    {
        if (allocator->buckets[i])
        {
            Block *block = allocator->buckets[i];
            allocator->buckets[i] = block->next;

            size_t block_size = 1U << i;
            while (i > bucket)
            {
```

```
                i--;
                block_size >>= 1;
                Block *split_block = (Block *)((uintptr_t)block + block_size);
                split_block->next = allocator->buckets[i];
                allocator->buckets[i] = split_block;
            }

            return (void *)block;
        }
    }

    return NULL;
}

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

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

    size_t block_size = MIN_BLOCK_SIZE;
    int bucket = 0;

    while ((1U << bucket) < allocator->size && (offset % block_size == 0))
    {
        block_size <<= 1;
        bucket++;
    }
    bucket--;

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

Allocator-pow2.c

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

#define MIN_BLOCK_SIZE 16
#define MAX_BLOCK_SIZE 4096
```

```c
#define NUM_BUCKETS 8

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

typedef struct Allocator
{
    void *memory;
    size_t size;
    Block *buckets[NUM_BUCKETS];
} Allocator;

static int get_bucket(size_t size)
{
    size_t adjusted_size = size > MIN_BLOCK_SIZE ? size : MIN_BLOCK_SIZE;
    int bucket = 0;
    while ((1U << (bucket + 4)) < adjusted_size && bucket < NUM_BUCKETS - 1)
    {
        bucket++;
    }
    return bucket;
}

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

    Allocator *allocator = (Allocator *)memory;
    allocator->memory = (void *)((uintptr_t)memory + sizeof(Allocator));
    allocator->size = size - sizeof(Allocator);

    memset(allocator->buckets, 0, sizeof(allocator->buckets));

    Block *initial_block = (Block *)allocator->memory;
    initial_block->next = NULL;
    allocator->buckets[NUM_BUCKETS - 1] = initial_block;

    return allocator;
}

void allocator_destroy(Allocator *const allocator)
{
    if (!allocator)
        return;
}
```

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

    int bucket = get_bucket(size);

    for (int i = bucket; i < NUM_BUCKETS; i++)
    {
        if (allocator->buckets[i])
        {
            Block *block = allocator->buckets[i];
            allocator->buckets[i] = block->next;

            size_t block_size = 1U << (i + 4);
            while (i > bucket)
            {
                i--;
                block_size >>= 1;
                Block *split_block = (Block *)((uintptr_t)block + block_size);
                split_block->next = allocator->buckets[i];
                allocator->buckets[i] = split_block;
            }

            return (void *)block;
        }
    }

    return NULL;
}

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

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

    size_t block_size = MIN_BLOCK_SIZE;
    int bucket = 0;

    while ((1U << (bucket + 4)) < allocator->size && (offset % block_size == 0))
    {
```

```
        block_size <<= 1;
        bucket++;
    }
    bucket--;

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

Allocator-test.c

```
#define _GNU_SOURCE
#include <dlfcn.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <fcntl.h>

typedef struct Allocator Allocator;
typedef Allocator *(*allocator_create_t)(void *memory, size_t size);
typedef void (*allocator_destroy_t)(Allocator *allocator);
typedef void *(*allocator_alloc_t)(Allocator *allocator, size_t size);
typedef void (*allocator_free_t)(Allocator *allocator, void *memory);

static void *load_library(const char *path,
                          allocator_create_t *create,
                          allocator_destroy_t *destroy,
                          allocator_alloc_t *alloc,
                          allocator_free_t *free)
{
    void *lib_handle = dlopen(path, RTLD_NOW);
    if (!lib_handle)
    {
        write(STDERR_FILENO, dlerror(), strlen(dlerror()));
        return NULL;
    }

    *create = (allocator_create_t)dlsym(lib_handle, "allocator_create");
    *destroy = (allocator_destroy_t)dlsym(lib_handle, "allocator_destroy");
    *alloc = (allocator_alloc_t)dlsym(lib_handle, "allocator_alloc");
    *free = (allocator_free_t)dlsym(lib_handle, "allocator_free");

    if (!*create || !*destroy || !*alloc || !*free)
    {
        write(STDERR_FILENO, dlerror(), strlen(dlerror()));
        dlclose(lib_handle);
        return NULL;
```

```
    }

    return lib_handle;
}

static Allocator *emergency_allocator_create(void *memory, size_t size)
{
    return (Allocator *)memory;
}

static void emergency_allocator_destroy(Allocator *allocator) {}

static void *emergency_allocator_alloc(Allocator *allocator, size_t size)
{
    return mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS,
-1, 0);
}

static void emergency_allocator_free(Allocator *allocator, void *memory)
{
    munmap(memory, 0);
}

static void test_allocator(Allocator *allocator,
                           allocator_alloc_t alloc,
                           allocator_free_t free)
{
    size_t test_sizes[] = {16, 32, 64, 128, 256, 512, 1024};
    const int num_tests = sizeof(test_sizes) / sizeof(test_sizes[0]);

    void *allocations[num_tests];

    clock_t start, end;

    start = clock();
    for (int i = 0; i < num_tests; i++)
    {
        allocations[i] = alloc(allocator, test_sizes[i]);
    }
    end = clock();
    write(STDOUT_FILENO, "Allocation time: ", 17);
    dprintf(STDOUT_FILENO, "%f\n", (double)(end - start) / CLOCKS_PER_SEC);

    start = clock();
    for (int i = 0; i < num_tests; i++)
    {
        free(allocator, allocations[i]);
    }
    end = clock();
    write(STDOUT_FILENO, "Deallocation time: ", 19);
    dprintf(STDOUT_FILENO, "%f\n", (double)(end - start) / CLOCKS_PER_SEC);
```

```c
}

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        write(STDERR_FILENO, "Usage: ./allocator_test <library_path>\n", 40);
        return 1;
    }

    const size_t memory_size = 1 << 20; // 1 MB
    void *memory = mmap(NULL, memory_size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);
    if (memory == MAP_FAILED)
    {
        write(STDERR_FILENO, "Failed to allocate test memory\n", 31);
        return 1;
    }

    allocator_create_t allocator_create = NULL;
    allocator_destroy_t allocator_destroy = NULL;
    allocator_alloc_t allocator_alloc = NULL;
    allocator_free_t allocator_free = NULL;

    void *lib_handle = load_library(argv[1], &allocator_create,
&allocator_destroy, &allocator_alloc, &allocator_free);

    Allocator *allocator;
    if (!lib_handle)
    {
        write(STDERR_FILENO, "Falling back to emergency allocator\n", 37);
        allocator_create = emergency_allocator_create;
        allocator_destroy = emergency_allocator_destroy;
        allocator_alloc = emergency_allocator_alloc;
        allocator_free = emergency_allocator_free;
    }

    allocator = allocator_create(memory, memory_size);
    if (!allocator)
    {
        write(STDERR_FILENO, "Failed to initialize allocator\n", 31);
        return 1;
    }

    test_allocator(allocator, allocator_alloc, allocator_free);

    allocator_destroy(allocator);
    if (lib_handle)
    {
        dlclose(lib_handle);
    }
```

```
    munmap(memory, memory_size);
    return 0;
}
```

Протокол работы программы

```
execve("./allocator_test", ["./allocator_test", "./allocator_mc.so"], 0x7fff60c9d828
/* 31 vars */) = 0
brk(NULL)                               = 0x55699bdc7000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fdf5f959000
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=23179, ...}) = 0
mmap(NULL, 23179, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fdf5f953000
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\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) = 0x7fdf5f741000
mmap(0x7fdf5f769000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fdf5f769000
mmap(0x7fdf5f8f1000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7fdf5f8f1000
mmap(0x7fdf5f940000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7fdf5f940000
mmap(0x7fdf5f946000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fdf5f946000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fdf5f73e000
arch_prctl(ARCH_SET_FS, 0x7fdf5f73e740) = 0
set_tid_address(0x7fdf5f73ea10)         = 18866
set_robust_list(0x7fdf5f73ea20, 24)     = 0
rseq(0x7fdf5f73f060, 0x20, 0, 0x53053053) = 0
mprotect(0x7fdf5f940000, 16384, PROT_READ) = 0
mprotect(0x556999de2000, 4096, PROT_READ) = 0
mprotect(0x7fdf5f991000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fdf5f953000, 23179)           = 0
mmap(NULL, 1048576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fdf5f63e000
getrandom("\x4f\x29\x27\x9e\xc6\x6f\x38\xdc", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55699bdc7000
brk(0x55699bde8000)                     = 0x55699bde8000
openat(AT_FDCWD, "./allocator_mc.so", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\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
fstat(3, {st_mode=S_IFREG|0755, st_size=15648, ...}) = 0
getcwd("/home/pseudush/Lab_OC/lab4", 128) = 27
mmap(NULL, 16408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fdf5f954000
mmap(0x7fdf5f955000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1000) = 0x7fdf5f955000
mmap(0x7fdf5f956000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x2000) = 0x7fdf5f956000
```

```
mmap(0x7fdf5f957000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7fdf5f957000
close(3)                                = 0
mprotect(0x7fdf5f957000, 4096, PROT_READ) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1729300}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1769500}) = 0
write(1, "Allocation time: ", 17Allocation time: )       = 17
write(1, "0.000040\n", 90.000040
)               = 9
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1883600}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1915900}) = 0
write(1, "Deallocation time: ", 19Deallocation time: )     = 19
write(1, "0.000032\n", 90.000032
)               = 9
munmap(0x7fdf5f954000, 16408)           = 0
munmap(0x7fdf5f63e000, 1048576)         = 0
exit_group(0)                           = ?
+++ exited with 0 +++

execve("./allocator_test", ["./allocator_test", "./allocator_pow2.so"],
0x7ffd5f6cee78 /* 31 vars */) = 0
brk(NULL)                               = 0x559f05504000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fa87f32b000
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=23179, ...}) = 0
mmap(NULL, 23179, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fa87f325000
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\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) = 0x7fa87f113000
mmap(0x7fa87f13b000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fa87f13b000
mmap(0x7fa87f2c3000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7fa87f2c3000
mmap(0x7fa87f312000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7fa87f312000
mmap(0x7fa87f318000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fa87f318000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fa87f110000
arch_prctl(ARCH_SET_FS, 0x7fa87f110740) = 0
set_tid_address(0x7fa87f110a10)         = 19553
set_robust_list(0x7fa87f110a20, 24)     = 0
rseq(0x7fa87f111060, 0x20, 0, 0x53053053) = 0
mprotect(0x7fa87f312000, 16384, PROT_READ) = 0
mprotect(0x559f03a20000, 4096, PROT_READ) = 0
mprotect(0x7fa87f363000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fa87f325000, 23179)           = 0
mmap(NULL, 1048576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fa87f010000
getrandom("\x69\xf2\x39\x3e\xc6\xab\x8e\xfc", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x559f05504000
brk(0x559f05525000)                     = 0x559f05525000
openat(AT_FDCWD, "./allocator_pow2.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
fstat(3, {st_mode=S_IFREG|0755, st_size=15648, ...}) = 0
getcwd("/home/pseudush/Lab_OC/lab4", 128) = 27
mmap(NULL, 16408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fa87f326000
mmap(0x7fa87f327000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1000) = 0x7fa87f327000
mmap(0x7fa87f328000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x2000) = 0x7fa87f328000
mmap(0x7fa87f329000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7fa87f329000
close(3)                                = 0
mprotect(0x7fa87f329000, 4096, PROT_READ) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1562800}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1585000}) = 0
write(1, "Allocation time: ", 17Allocation time: )       = 17
write(1, "0.000023\n", 90.000023
)               = 9
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1675500}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=1697400}) = 0
write(1, "Deallocation time: ", 19Deallocation time: )     = 19
write(1, "0.000022\n", 90.000022
)               = 9
munmap(0x7fa87f326000, 16408)           = 0
munmap(0x7fa87f010000, 1048576)         = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Время аллоцирования и освобождения памяти

Block Size (Bytes)	Alloc Time Pow2 (s)	Free Time Pow2 (s)	Alloc Time McKusick (s)
2	0.0001317	0.0000028	0.0001539
4	0.0000056	0.0000004	0.0000033
8	0.0000012	0.0000003	0.0000026
16	0.0000012	0.0000003	0.0000008
32	0.0000012	0.0000003	0.0000011

Вывод

Во время выполнения данной лабораторной работы я узнал много полезного про системные вызовы и про создание отдельных процессов. Лучше узнал ОС Ubuntu. Основная сложность была в изучении необходимых материалов коих было не очень много. В будущем я бы хотел попробовать написать свой простой драйвер.