Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №4 по курсу

«Операционные системы»

Группа: М8О-210Б-23

Студент: Воронухин Н.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 10.12.24

Москва, 2024

Постановка задачи

Вариант 1.

Требуется создать 2 динамические библиотеки реализующие два аллокатора памяти: списки свободных блоков (первый подходящий) и алгоритм двойников.

Общий метод и алгоритм решения

Использованные системные вызовы:

- void exit(int status); - приводит к обычному завершению программы, и величина status возвращается процессу-родителю.
- void * mmap(void **start*, size_t *length*, int *prot* , int *flags*, int *fd*, off_t *offset*); - отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым описателем fd, в память, начиная с адреса start. Последний параметр (адрес) необязателен, и обычно бывает равен 0. Настоящее местоположение отраженных данных возвращается самой функцией mmap, и никогда не бывает равным 0.
- int munmap(void **start*, size_t *length*); - удаляет отображение для указанного адресного диапазона
- ssize_t write(int fd, const void buf[.count], size_t count); - пишет count байт из буфера buf в файл, на который ссылается файловый дескриптор fd.
- int munmap(void *addr, size_t len); - удаляет все отражения из заданной области памяти.
- void *dlopen(const char *filename, int flag); - загружает динамическую библиотеку, имя которой указано в строке filename, и возвращает прямой указатель на начало динамической библиотеки.
- void *dlsym(void *handle, char *symbol); - дает возможность получить по имени адрес символа, определенного в разделяемой библиотеке.
- int dlclose(void *handle); - закрывает динамическую библиотеку.

Реализованы 2 динамические библиотеки с аллокаторами памяти, первая со списком свободных блоков на основе односвязного списка, вторая – на основе алгоритма двойников с помощью массива списков, хранящих блоки разных размеров, и main-файл для их проверки.

Код программы

dl.h

```
#ifndef __LIBRARY_H
```

```
#define __LIBRARY_H

#include <dlfcn.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <dlfcn.h>
#include <sys/mman.h>
#include <stddef.h>
#include <string.h>
#include <stdbool.h>

#ifdef _MSC_VER
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif

typedef struct Allocator Allocator;
typedef struct Block Block;

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);

#endif
```

parent.c

```
#include "dl.h"
```

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


#define MEMORY_POOL_SIZE 1024 * 1024


static Allocator *allocator_create_stub(void *const memory, const size_t size)

{

    void *mapped_memory = mmap(memory, size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS | MAP_FIXED, -1, 0);

    if (mapped_memory == MAP_FAILED)

    {

        const char err_msg[] = "allocator_create: mmap failed\n";

        write(STDERR_FILENO, err_msg, sizeof(err_msg) - 1);

        return NULL;

    }

    return (Allocator *)mapped_memory;

}


static void allocator_destroy_stub(Allocator *const allocator)

{

    if (allocator && munmap(allocator, MEMORY_POOL_SIZE) == -1)

    {

        const char err_msg[] = "allocator_destroy: munmap failed\n";

        write(STDERR_FILENO, err_msg, sizeof(err_msg) - 1);

    }


}


static void *allocator_alloc_stub(Allocator *const allocator, const size_t size)

{

    void *mapped_memory = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);

    if (mapped_memory == MAP_FAILED)

    {

        const char err_msg[] = "allocator_alloc: mmap failed\n";
```

```
        write(STDERR_FILENO, err_msg, sizeof(err_msg) - 1);
        return NULL;
    }
    return mapped_memory;
}

static void allocator_free_stub(Allocator *const allocator, void *const memory)
{
    if (memory && munmap(memory, sizeof(memory)) == -1)
    {
        const char err_msg[] = "allocator_free: munmap failed\n";
        write(STDERR_FILENO, err_msg, sizeof(err_msg) - 1);
    }
}

static allocator_create_f *allocator_create;
static allocator_destroy_f *allocator_destroy;
static allocator_alloc_f *allocator_alloc;
static allocator_free_f *allocator_free;

int main(int argc, char **argv)
{
    void *library = NULL;

    if (argc == 2) {
        library = dlopen(argv[1], RTLD_LOCAL | RTLD_NOW);

        if (!library)
        {
            const char msg[] = "Failed to load library. Using stub.\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            allocator_create = allocator_create_stub;
            allocator_destroy = allocator_destroy_stub;
```

```
    allocator_alloc = allocator_alloc_stub;
    allocator_free = allocator_free_stub;
}
else
{
    allocator_create = dlsym(library, "allocator_create");
    allocator_destroy = dlsym(library, "allocator_destroy");
    allocator_alloc = dlsym(library, "allocator_alloc");
    allocator_free = dlsym(library, "allocator_free");

    if (!allocator_create)
    {
        const char msg[] = "Failed to load allocator_create. Using stub.\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        allocator_create = allocator_create_stub;
    }
    if (!allocator_destroy)
    {
        const char msg[] = "Failed to load allocator_destroy. Using stub.\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        allocator_destroy = allocator_destroy_stub;
    }
    if (!allocator_alloc)
    {
        const char msg[] = "Failed to load allocator_alloc. Using stub.\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        allocator_alloc = allocator_alloc_stub;
    }
    if (!allocator_free)
    {
        const char msg[] = "Failed to load allocator_free. Using stub.\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        allocator_free = allocator_free_stub;
```

```
        }
    }
}
else
{
    const char msg[] = "Incorrect use. Using stub.\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    allocator_create = allocator_create_stub;
    allocator_destroy = allocator_destroy_stub;
    allocator_alloc = allocator_alloc_stub;
    allocator_free = allocator_free_stub;
}

// TeCЃC,C‹ P±PëP±P»PëPsC,PµPєPë
size_t size = MEMORY_POOL_SIZE;
void *addr = mmap(NULL, size, PROT_READ | PROT_WRITE,
                  MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
if (addr == MAP_FAILED)
{
    dlclose(library);
    char message[] = "mmap failed\n";
    write(STDERR_FILENO, message, sizeof(message) - 1);
    return EXIT_FAILURE;
}

Allocator *allocator = allocator_create(addr, MEMORY_POOL_SIZE);

if (!allocator)
{
    const char msg[] = "Failed to initialize allocator\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    munmap(addr, size);
    dlclose(library);
```

```
    return EXIT_FAILURE;
}

int *int_block = (int *)allocator_alloc(allocator, sizeof(int));
if (int_block)
{
    *int_block = 42;
    const char msg[] = "Allocated int_block with value 42\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
}
else
{
    const char msg[] = "Failed to allocate memory for int_block\n";
    write(STDERR_FILENO, msg, sizeof(msg));
}

float *float_block = (float *)allocator_alloc(allocator, sizeof(float));
if (float_block)
{
    *float_block = 2.718f;
    const char msg[] = "Allocated float_block with value 2.718\n";
    write(STDOUT_FILENO, msg, sizeof(msg));
}
else
{
    const char msg[] = "Failed to allocate memory for float_block\n";
    write(STDERR_FILENO, msg, sizeof(msg));
}

if (int_block)
{
    allocator_free(allocator, int_block);
```

```
        const char msg[] = "Freed int_block\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
    }

    if (float_block)
    {
        allocator_free(allocator, float_block);
        const char msg[] = "Freed float_block\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
    }

    allocator_destroy(allocator);
    const char msg[] = "Allocator destroyed\n";
    write(STDOUT_FILENO, msg, sizeof(msg));

    if (library)
        dlclose(library);
    munmap(addr, size);

    return EXIT_SUCCESS;
}
```

childe.c

```
#include "dl.h"
#include <math.h>

#define MIN_BLOCK_SIZE 32
#define MAX_BLOCK_INDEX 24

typedef struct Block
{
    size_t size;
    struct Block *next;
```

```
} Block;


typedef struct Allocator
{
    void *memory;
    size_t total_size;
    Block *free_lists[MAX_BLOCK_INDEX];
} Allocator;


size_t round_to_power_of_two(size_t size)
{
    size_t power = MIN_BLOCK_SIZE;
    while (power < size)
    {
        power <<= 1;
    }
    return power;
}


int get_power_of_two(size_t size)
{
    return (int)(log2(size));
}


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

    Allocator *allocator = (Allocator*)memory; //(Allocator *)mmap(
```

```
            //NULL, sizeof(Allocator), PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);

        size_t total_size = round_to_power_of_two(size - sizeof(Allocator));
        size_t max_size = 1 << (MAX_BLOCK_INDEX - 1);

        if (total_size > max_size)
        {
            return NULL;
        }

        allocator->memory = memory - sizeof(Allocator);
        allocator->total_size = total_size;

        for (size_t i = 0; i < MAX_BLOCK_INDEX; i++)
        {
            allocator->free_lists[i] = NULL;
        }

        Block *initial_block = (Block *)allocator->memory;
        initial_block->size = total_size;
        initial_block->next = NULL;

        int index = get_power_of_two(total_size) - get_power_of_two(MIN_BLOCK_SIZE);
        allocator->free_lists[index] = initial_block;

        return allocator;
    }

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

```
    memset(allocator, 0, allocator->total_size + sizeof(Allocator));
}

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

    size_t block_size = round_to_power_of_two(size + sizeof(Block));
    size_t max_size = 1 << (MAX_BLOCK_INDEX - 1);

    if (block_size > max_size)
    {
        return NULL;
    }

    int index = get_power_of_two(block_size) - get_power_of_two(MIN_BLOCK_SIZE);
    if (index >= MAX_BLOCK_INDEX)
    {
        return NULL;
    }
    else if (index < 0)
    {
        index = 1;
    }

    while (index < MAX_BLOCK_INDEX && !allocator->free_lists[index])
    {
        index++;
    }

    if (index >= MAX_BLOCK_INDEX)
    {
```

```
            return NULL;
        }

        Block *block = allocator->free_lists[index];
        allocator->free_lists[index] = block->next;

        while (block->size > block_size)
        {
            size_t new_size = block->size >> 1;
            Block *buddy = (Block *)((char *)block + new_size);
            buddy->size = new_size;
            buddy->next =
                allocator->free_lists[get_power_of_two(new_size) -
get_power_of_two(MIN_BLOCK_SIZE)];
            allocator->free_lists[get_power_of_two(new_size) -
get_power_of_two(MIN_BLOCK_SIZE)] =
                buddy;
            block->size = new_size;
        }

        return (void *)((char *)block + sizeof(Block));
    }

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

        Block *block = (Block *)((char *)memory - sizeof(Block));

        size_t block_size = block->size;
        int index = get_power_of_two(block_size) - get_power_of_two(MIN_BLOCK_SIZE);
        if (index >= MAX_BLOCK_INDEX)
            return;
```

```
            block->next = allocator->free_lists[index];
            allocator->free_lists[index] = block;
        }
```

free_list.c

```
#include "dl.h"

#define MIN_BLOCK_SIZE 32

typedef struct Block
{
    size_t size;
    struct Block *next;
    bool is_free;
} Block;

typedef struct Allocator
{
    Block *free_list;
    void *memory_start;
    size_t total_size;
} Allocator;

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

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

```
    allocator->free_list = (Block *)allocator->memory_start;

    allocator->free_list->size = allocator->total_size - sizeof(Block);
    allocator->free_list->next = NULL;
    allocator->free_list->is_free = true;

    return allocator;
}

EXPORT void allocator_destroy(Allocator *allocator)
{
    if (allocator)
    {
        memset(allocator, 0, allocator->total_size + sizeof(Allocator));
    }
}

EXPORT void *allocator_alloc(Allocator *allocator, size_t size)
{
    if (!allocator || size < 1)
    {
        return NULL;
    }

    size = (size / MIN_BLOCK_SIZE
        + (size % MIN_BLOCK_SIZE) ? 1 : 0)
        * MIN_BLOCK_SIZE;

    Block *current = allocator->free_list;

    while (current)
    {
        if (current->is_free && current->size >= size) {
```

```
            size_t remain_size = current->size - size;
            if (remain_size >= sizeof(Block) + MIN_BLOCK_SIZE) {
                Block *new_block = (Block *)((char *)current + 2 * sizeof(Block) + size);
                new_block->size = remain_size - sizeof(Block);
                new_block->is_free = true;
                new_block->next = current->next;

                current->next = new_block;
                current->size = size;
            }
            current->is_free = false;
            return (void *)((char *)current + sizeof(Block));
        }
        else {
            current = current->next;
        }
    }

    return NULL;
}

EXPORT void allocator_free(Allocator *allocator, void *ptr_to_memory)
{
    if (!allocator || !ptr_to_memory) // bound
    {
        return;
    }

    Block *current = (Block *)((char *)ptr_to_memory - sizeof(Block));
    if (!current)
        return;

    current->is_free = true;
```

```
    allocator->free_list;
    while (current && current->next)
    {
        if (current->is_free && current->next->is_free)
        {
            current->size += current->next->size + sizeof(Block);
            current->next = current->next->next;
        }
        else
        {
            current = current->next;
        }
    }
}
```

Результаты тестирования

Метод свободных блоков.

- **Производительность:** Быстрое выделение памяти, но медленное объединение блоков.
- **Фрагментация:** присутствует как внутренняя, так и внешняя фрагментации, внутренняя зависит от размеров минимального блока, внешняя от последовательности запросов памяти.
- **Память:** Эффективен для запросов практически любого размера

Метод двойников.

- **Производительность:** Высокая скорость выделение памяти, и освобождения.
- **Фрагментация:** присутствует как сильная внутренняя, так и внешняя фрагментации, внутренняя зависит от размеров минимального блока и кратности степени 2 размеров запросов, внешняя от последовательности запросов памяти.
- **Память:** Эффективен для запросов кратных степени 2.

Протокол работы программы

```
vnadez@dbnicknv:~/Projects/OS_LABS/LAB_4$ ./main
/home/vnadez/Projects/OS_LABS/LAB_4/bd.so

Allocated int_block with value 42

Allocated float_block with value 2.718

Freed int_block

Freed float_block

Allocator destroyed


vnadez@dbnicknv:~/Projects/OS_LABS/LAB_4$ ./main

Incorrect use. Using stub.

Allocated int_block with value 42

Allocated float_block with value 2.718

Freed int_block

Freed float_block

Allocator destroyed
```

Strace:

```
16463 execve("./main", ["./main", "/home/vnadez/Projects/OS_LABS/LA"...],
0x7fff74df1b90 /* 64 vars */) = 0
```

```
     16463 brk(NULL)                           = 0x560e4d62c000

     16463 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7ff8977cc000

     16463 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

     16463 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

     16463 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=97022, ...}, AT_EMPTY_PATH) = 0

     16463 mmap(NULL, 97022, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff8977b4000

     16463 close(3)                            = 0

     16463 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

     16463 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"...,
832) = 832

     16463 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

     16463 newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1922136, ...}, AT_EMPTY_PATH) =
0

     16463 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

     16463 mmap(NULL, 1970000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff8975d3000

     16463 mmap(0x7ff8975f9000, 1396736, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7ff8975f9000

     16463 mmap(0x7ff89774e000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x17b000) = 0x7ff89774e000

     16463 mmap(0x7ff8977a1000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ce000) = 0x7ff8977a1000

     16463 mmap(0x7ff8977a7000, 53072, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ff8977a7000

     16463 close(3)                            = 0

     16463 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7ff8975d0000

     16463 arch_prctl(ARCH_SET_FS, 0x7ff8975d0740) = 0

     16463 set_tid_address(0x7ff8975d0a10)   = 16463

     16463 set_robust_list(0x7ff8975d0a20, 24) = 0

     16463 rseq(0x7ff8975d1060, 0x20, 0, 0x53053053) = 0

     16463 mprotect(0x7ff8977a1000, 16384, PROT_READ) = 0

     16463 mprotect(0x560e44fc1000, 4096, PROT_READ) = 0

     16463 mprotect(0x7ff8977fe000, 8192, PROT_READ) = 0

     16463 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0

     16463 munmap(0x7ff8977b4000, 97022)     = 0

     16463 getrandom("\x51\x50\xa6\x2b\x81\x1f\x3d\xcf", 8, GRND_NONBLOCK) = 8
```

```
     16463 brk(NULL)                           = 0x560e4d62c000

     16463 brk(0x560e4d64d000)                 = 0x560e4d64d000

     16463 openat(AT_FDCWD, "/home/vnadez/Projects/OS_LABS/LAB_4/fr.so",
O_RDONLY|O_CLOEXEC) = 3

     16463 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

     16463 newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=15440, ...}, AT_EMPTY_PATH) = 0

     16463 mmap(NULL, 16408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff8977c7000

     16463 mmap(0x7ff8977c8000, 4096, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7ff8977c8000

     16463 mmap(0x7ff8977c9000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x2000) = 0x7ff8977c9000

     16463 mmap(0x7ff8977ca000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7ff8977ca000

     16463 close(3)                            = 0

     16463 mprotect(0x7ff8977ca000, 4096, PROT_READ) = 0

     16463 mmap(NULL, 1048576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7ff8974d0000

     16463 write(1, "Allocated int_block with value 4"..., 35) = 35

     16463 write(1, "Allocated float_block with value"..., 40) = 40

     16463 write(1, "Freed int_block\n\0", 17) = 17

     16463 write(1, "Freed float_block\n\0", 19) = 19

     16463 write(1, "Allocator destroyed\n\0", 21) = 21

     16463 munmap(0x7ff8977c7000, 16408)     = 0

     16463 munmap(0x7ff8974d0000, 1048576)   = 0

     16463 exit_group(0)                       = ?

     16463 +++ exited with 0 +++
```

Вывод

В процессе выполнения лабораторной работы я освоил работу с динамическими библиотеками, новыми системными вызовами, предназначенными для работы с ними, написание собственного аллокатора памяти на языке C, а также проанализировал работу пары алгоритмов выделения памяти, самым проблемным из которых для меня оказался метод двойников.