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-211Б-23

Студент: Грищенко В.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 19.12.24

Москва, 2024

**Постановка задачи**

**Вариант 6.**

Исследовать два аллокатора памяти: buddy allocator и блоки по 2^n. Необходимо реализовать два алгоритма аллокации памяти и сравнить их не используя malloc, free, calloc, realloc, new, delete.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

1. **\*int munmap(void addr, size\_t length); -** Удаляет отображения, созданные с помощью mmap.
2. **\*int dlclose(void handle);** - Закрывает динамическую библиотеку, открытую с помощью dlopen, и освобождает ресурсы, связанные с этим дескриптором.
3. **void exit(int status);** - Завершает выполнение программы и возвращает статус выхода в операционную систему.
4. **\*char dlerror(void);** - Возвращает строку, описывающую последнюю ошибку, возникшую при вызове функций dlopen, dlsym, dlclose.
5. **\*\*void dlopen(const char filename, int flag);** - Открывает динамическую библиотеку и возвращает дескриптор для последующего использования.
6. **\*\*void mmap(void addr, size\_t length, int prot, int flags, int fd, off\_t offset);** – создает новое отображение памяти или изменяет существующее**.**

**Описание лабораторной работы**

В рамках лабораторной работы была разработана программа, которая демонстрирует работу двух аллокаторов работающих по разным алгоритмам.

**Цель лабораторной работы**

Приобретение практических навыков в:

1) Создании аллокаторов памяти и их анализу;

2) Создании динамических библиотек и программ, использующие динамические библиотеки**.Описание программы**

Программа состоит из трех частей:

1. **Вызов аллокатора (main.c)**: выделяет и освобождает определенное количество памяти при помощи переданного в качестве параметра при вызове аллокатора.
2. **Реализация алгоритма блоки по 2^n (liballocator1.c)**: Память выделяется блоками, размер которых является степенью двойки.
3. **Реализация алгоритма buddy allocator (liballocator2.c)**: Память управляется с помощью списка свободных блоков. При выделении памяти алгоритм ищет наиболее подходящий свободный блок.

**Код программы**

main.c

#include <stdio.h>

#include <stdlib.h>

#include <dlfcn.h>

#include <sys/mman.h>

#include <time.h>

typedef struct {

    void \*(\*allocator\_create)(void \*const memory, const size\_t size);

    void (\*allocator\_destroy)(void \*const allocator);

    void \*(\*allocator\_alloc)(void \*const allocator, const size\_t size);

    void (\*allocator\_free)(void \*const allocator, void \*const memory);

} AllocatorAPI;

void\* fallback\_allocator\_create(void \*const memory, const size\_t size) {

    printf("%li\n", size);

    return memory;

}

void fallback\_allocator\_destroy(void \*const allocator) {

    if (allocator)

        printf("\n");

}

void\* fallback\_allocator\_alloc(void \*const allocator, const size\_t size) {

    printf("%li\n", size);

    if (allocator)

        printf("\n");

    return mmap(NULL, size, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);

}

void fallback\_allocator\_free(void \*const allocator, void \*const memory) {

    munmap(memory, 4096);

    if (allocator)

        printf("\n");

}

int main(int argc, char \*argv[]) {

    void \*handle = NULL;

    AllocatorAPI api;

    if (argc > 1) {

        handle = dlopen(argv[1], RTLD\_LAZY);

        if (!handle) {

            fprintf(stderr, "%s\n", dlerror());

            exit(EXIT\_FAILURE);

        }

        api.allocator\_create = dlsym(handle, "allocator\_create");

        api.allocator\_destroy = dlsym(handle, "allocator\_destroy");

        api.allocator\_alloc = dlsym(handle, "allocator\_alloc");

        api.allocator\_free = dlsym(handle, "allocator\_free");

        if (!api.allocator\_create || !api.allocator\_destroy || !api.allocator\_alloc || !api.allocator\_free) {

            fprintf(stderr, "%s\n", dlerror());

            exit(EXIT\_FAILURE);

        }

    } else {

        api.allocator\_create = fallback\_allocator\_create;

        api.allocator\_destroy = fallback\_allocator\_destroy;

        api.allocator\_alloc = fallback\_allocator\_alloc;

        api.allocator\_free = fallback\_allocator\_free;

    }

    size\_t memory\_size = 4096;

    size\_t size\_data = 1327;

    void \*memory = mmap(NULL, memory\_size, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);

    if (memory == MAP\_FAILED) {

        perror("mmap");

        exit(EXIT\_FAILURE);

    }

    void \*allocator = api.allocator\_create(memory, memory\_size);

    clock\_t start, end;

    double cpu\_time\_used;

    start = clock();

    void \*ptr1 = api.allocator\_alloc(allocator, size\_data);

    end = clock();

    cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

    printf("Time to allocate %li bytes: %f seconds\n", size\_data, cpu\_time\_used);

    start = clock();

    api.allocator\_free(allocator, ptr1);

    end = clock();

    cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

    printf("Time to free %li bytes: %f seconds\n", size\_data, cpu\_time\_used);

    api.allocator\_destroy(allocator);

    munmap(memory, memory\_size);

    if (handle) {

        dlclose(handle);

    }

    return 0;

}

liballocator1.c

#include <stddef.h>

#include <stdio.h>

#include <string.h>

#include <sys/mman.h>

#include <unistd.h>

#define SIZE\_DATA 32

typedef struct BuddyBlock {

    size\_t size;

    struct BuddyBlock \*next;

    struct BuddyBlock \*prev;

    int is\_free;

} BuddyBlock;

typedef struct BuddyAllocator {

    void \*memory;

    size\_t total\_size;

    BuddyBlock \*free\_lists[SIZE\_DATA];

} BuddyAllocator;

void \*allocator\_create(void \*const memory, const size\_t size) {

    BuddyAllocator \*allocator = (BuddyAllocator \*)mmap(NULL, sizeof(BuddyAllocator), PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);

    if (allocator == MAP\_FAILED) {

        perror("mmap");

        return NULL;

    }

    allocator->memory = memory;

    allocator->total\_size = size;

    for (int i = 0; i < SIZE\_DATA; i++)

        allocator->free\_lists[i] = NULL;

    BuddyBlock \*initial\_block = (BuddyBlock \*)memory;

    initial\_block->size = size;

    initial\_block->next = NULL;

    initial\_block->prev = NULL;

    initial\_block->is\_free = 1;

    int order = 0;

    size\_t block\_size = 1;

    while (block\_size < size) {

        block\_size <<= 1;

        order++;

    }

    initial\_block->next = allocator->free\_lists[order];

    if (allocator->free\_lists[order]) {

        allocator->free\_lists[order]->prev = initial\_block;

    }

    allocator->free\_lists[order] = initial\_block;

    return allocator;

}

void allocator\_destroy(void \*const allocator) {

    if (munmap(allocator, sizeof(BuddyAllocator)) == -1)

        perror("munmap");

}

void \*allocator\_alloc(void \*const allocator, const size\_t size) {

    BuddyAllocator \*buddy\_allocator = (BuddyAllocator \*)allocator;

    size\_t block\_size = 1;

    int order = 0;

    while (block\_size < size) {

        block\_size <<= 1;

        order++;

    }

    BuddyBlock \*block = NULL;

    for (int i = order; i < SIZE\_DATA; i++) {

        if (buddy\_allocator->free\_lists[i]) {

            block = buddy\_allocator->free\_lists[i];

            order = i;

            break;

        }

    }

    if (!block) return NULL;

    if (block->next)

        block->next->prev = block->prev;

    if (block->prev)

        block->prev->next = block->next;

    else

        buddy\_allocator->free\_lists[order] = block->next;

    while (order > 0 && block\_size > size) {

        order--;

        block\_size >>= 1;

        BuddyBlock \*buddy = (BuddyBlock \*)((char \*)block + block\_size);

        buddy->size = block\_size;

        buddy->next = buddy\_allocator->free\_lists[order];

        buddy->prev = NULL;

        buddy->is\_free = 1;

        if (buddy\_allocator->free\_lists[order])

            buddy\_allocator->free\_lists[order]->prev = buddy;

        buddy\_allocator->free\_lists[order] = buddy;

    }

    block->is\_free = 0;

    return block;

}

void allocator\_free(void \*const allocator, void \*const memory) {

    BuddyAllocator \*buddy\_allocator = (BuddyAllocator \*)allocator;

    BuddyBlock \*block = (BuddyBlock \*)memory;

    block->is\_free = 1;

    while (1) {

        size\_t block\_size = block->size;

        char \*block\_addr = (char \*)block;

        ptrdiff\_t offset = (block\_addr - (char \*)buddy\_allocator->memory);

        char \*buddy\_addr = (char \*)buddy\_allocator->memory + (offset ^ block\_size);

        if (buddy\_addr < (char \*)buddy\_allocator->memory || buddy\_addr >= (char \*)buddy\_allocator->memory + buddy\_allocator->total\_size)

            break;

        BuddyBlock \*buddy = (BuddyBlock \*)buddy\_addr;

        if (buddy->is\_free && buddy->size == block\_size) {

            if (buddy->next)

                buddy->next->prev = buddy->prev;

            if (buddy->prev)

                buddy->prev->next = buddy->next;

            else {

                int order = 0;

                size\_t temp\_size = block\_size;

                while (temp\_size >>= 1) order++;

                buddy\_allocator->free\_lists[order] = buddy->next;

            }

            if (block < buddy) {

                block->size <<= 1;

            } else {

                buddy->size <<= 1;

                block = buddy;

            }

        } else {

            break;

        }

    }

    int order = 0;

    size\_t block\_size = block->size;

    while (block\_size >>= 1) order++;

    block->next = buddy\_allocator->free\_lists[order];

    block->prev = NULL;

    if (buddy\_allocator->free\_lists[order])

        buddy\_allocator->free\_lists[order]->prev = block;

    buddy\_allocator->free\_lists[order] = block;

}

liballocator2.c

#include <stddef.h>

#include <stdio.h>

#include <string.h>

#include <sys/mman.h>

#include <unistd.h>

#define MIN\_BLOCK\_SIZE 16

#define MAX\_ORDER 20

typedef struct BuddyAllocator {

    void \*memory;

    size\_t size;

    size\_t order;

    struct BuddyBlock \*free\_lists[MAX\_ORDER + 1];

} BuddyAllocator;

typedef struct BuddyBlock {

    struct BuddyBlock \*next;

    size\_t order;

} BuddyBlock;

void \*allocator\_create(void \*const memory, const size\_t size) {

    BuddyAllocator \*allocator = (BuddyAllocator \*)memory;

    allocator->memory = mmap(NULL, size, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);

    if (allocator->memory == MAP\_FAILED)

        return NULL;

    allocator->size = size;

    allocator->order = 0;

    while ((1UL << allocator->order) < size)

        allocator->order++;

    for (size\_t i = 0; i <= MAX\_ORDER; i++)

        allocator->free\_lists[i] = NULL;

    BuddyBlock \*block = (BuddyBlock \*)allocator->memory;

    block->next = NULL;

    block->order = allocator->order;

    allocator->free\_lists[allocator->order] = block;

    return allocator;

}

void allocator\_destroy(void \*const allocator) {

    BuddyAllocator \*buddy = (BuddyAllocator \*)allocator;

    munmap(buddy->memory, buddy->size);

}

void \*allocator\_alloc(void \*const allocator, const size\_t size) {

    BuddyAllocator \*buddy = (BuddyAllocator \*)allocator;

    size\_t order = 0;

    while ((1UL << order) < size)

        order++;

    for (size\_t i = order; i <= MAX\_ORDER; i++) {

        if (buddy->free\_lists[i]) {

            BuddyBlock \*block = buddy->free\_lists[i];

            buddy->free\_lists[i] = block->next;

            while (i > order) {

                i--;

                BuddyBlock \*buddy\_block = (BuddyBlock \*)((char \*)block + (1UL << i));

                buddy\_block->next = buddy->free\_lists[i];

                buddy\_block->order = i;

                buddy->free\_lists[i] = buddy\_block;

            }

            return block;

        }

    }

    return NULL;

}

void allocator\_free(void \*const allocator, void \*const memory) {

    BuddyAllocator \*buddy = (BuddyAllocator \*)allocator;

    BuddyBlock \*block = (BuddyBlock \*)memory;

    size\_t order = block->order;

    while (order < MAX\_ORDER) {

        BuddyBlock \*buddy\_block = (BuddyBlock \*)((char \*)block + (1UL << order));

        if (buddy\_block >= (BuddyBlock \*)buddy->memory && buddy\_block < (BuddyBlock \*)((char \*)buddy->memory + buddy->size) && buddy\_block->order == order) {

            buddy->free\_lists[order] = buddy\_block->next;

            order++;

            block = (BuddyBlock \*)((char \*)block - (1UL << (order - 1)));

        } else {

            break;

        }

    }

    block->next = buddy->free\_lists[order];

    block->order = order;

    buddy->free\_lists[order] = block;

}

**Протокол работы программы**

Время для алгоритма "блоки по 2^n":

Time to allocate 1327 bytes: 0.000889 seconds

Time to free 1327 bytes: 0.000529 seconds.

Для алгоритма "cписки свободных блоков":

Time to allocate 1327 bytes: 0.000518 seconds

Time to free 1327 bytes: 0.000322 seconds

(программа работает с valgrind поэтому так долго)

**Сравнение алгоритмов аллокаторов: buddy allocator и блоки по 2^n**

Аллокатор с блоками по 2n2n

Аллокатор с блоками по 2n2n выделяет память в блоках, размер которых является степенью двойки (например, 16, 32, 64, 128 байт и т.д.). Когда запрашивается память, аллокатор выбирает ближайший блок, размер которого больше или равен запрашиваемому размеру. Если такой блок не найден, аллокатор может разбить более крупный блок на два меньших блока, пока не будет найден подходящий размер. Освобождение памяти также происходит в блоках фиксированного размера, что упрощает управление памятью.

Buddy Allocator

Buddy Allocator также выделяет память в блоках, размер которых является степенью двойки, но с дополнительной особенностью: он объединяет соседние свободные блоки (buddy blocks) в более крупные блоки при освобождении памяти. Это помогает уменьшить внешнюю фрагментацию. Когда запрашивается память, аллокатор ищет свободный блок подходящего размера. Если такой блок не найден, он разбивает более крупный блок на два меньших блока (buddy blocks) и повторяет процесс, пока не будет найден подходящий размер. При освобождении памяти аллокатор проверяет, можно ли объединить освобождаемый блок с его buddy блоком, чтобы создать более крупный свободный блок.

Strace:

execve("./out/main", ["./out/main", "./out/liballocator2.so"], 0x7ffedfb9d208 /\* 28 vars \*/) = 0

brk(NULL) = 0x5625b2d25000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f82be352000

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=20335, ...}) = 0

**mmap**(NULL, 20335, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f82be34d000

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) = 0x7f82be13b000

**mmap**(0x7f82be163000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f82be163000

**mmap**(0x7f82be2eb000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7f82be2eb000

**mmap**(0x7f82be33a000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7f82be33a000

**mmap**(0x7f82be340000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f82be340000

close(3) = 0

**mmap**(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f82be138000

arch\_prctl(ARCH\_SET\_FS, 0x7f82be138740) = 0

set\_tid\_address(0x7f82be138a10) = 8456

set\_robust\_list(0x7f82be138a20, 24) = 0

rseq(0x7f82be139060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f82be33a000, 16384, PROT\_READ) = 0

mprotect(0x5625a5e20000, 4096, PROT\_READ) = 0

mprotect(0x7f82be38a000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

**munmap**(0x7f82be34d000, 20335) = 0

getrandom("\x56\x64\x46\xd0\xf4\x36\x98\x83", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x5625b2d25000

brk(0x5625b2d46000) = 0x5625b2d46000

openat(AT\_FDCWD, "./out/liballocator2.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=15784, ...}) = 0

getcwd("/home/user/test/Lab\_OCI/Lab\_4/src", 128) = 34

**mmap**(NULL, 16424, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f82be34d000

**mmap**(0x7f82be34e000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x7f82be34e000

**mmap**(0x7f82be34f000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f82be34f000

**mmap**(0x7f82be350000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7f82be350000

close(3) = 0

mprotect(0x7f82be350000, 4096, PROT\_READ) = 0

**mmap**(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f82be137000

**mmap**(NULL, 140199511420928, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = -1 ENOMEM (Cannot allocate memory)

dup(2) = 3

fcntl(3, F\_GETFL) = 0x8402 (flags O\_RDWR|O\_APPEND|O\_LARGEFILE)

fstat(3, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

write(3, "mmap: Cannot allocate memory\n", 29mmap: Cannot allocate memory

) = 29

close(3) = 0

clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, {tv\_sec=0, tv\_nsec=2301000}) = 0

--- SIGSEGV {si\_signo=SIGSEGV, si\_code=SEGV\_MAPERR, si\_addr=0x10} ---

+++ killed by SIGSEGV (core dumped) +++

Segmentation fault (core dumped)

execve("./out/parent", ["./out/parent"], 0x7ffe4804d3b0 /\* 27 vars \*/) = 0

brk(NULL) = 0x5654d7706000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f4b0c84b000

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=20335, ...}) = 0

mmap(NULL, 20335, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f4b0c846000

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) = 0x7f4b0c634000

mmap(0x7f4b0c65c000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f4b0c65c000

mmap(0x7f4b0c7e4000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7f4b0c7e4000

mmap(0x7f4b0c833000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7f4b0c833000

mmap(0x7f4b0c839000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f4b0c839000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f4b0c631000

arch\_prctl(ARCH\_SET\_FS, 0x7f4b0c631740) = 0

set\_tid\_address(0x7f4b0c631a10) = 34278

set\_robust\_list(0x7f4b0c631a20, 24) = 0

rseq(0x7f4b0c632060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f4b0c833000, 16384, PROT\_READ) = 0

mprotect(0x5654bdf69000, 4096, PROT\_READ) = 0

mprotect(0x7f4b0c883000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7f4b0c846000, 20335) = 0

shmget(0x4d2, 1028, IPC\_CREAT|0666) = 12

shmat(12, NULL, 0) = 0x7f4b0c84a000

read(0, "hello world\nthis is a test\n", 1024) = 27

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f4b0c631a10) = 34279

wait4(34279, NULL, 0, NULL) = 34279

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=34279, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f4b0c631a10) = 34280

wait4(34280, NULL, 0, NULL) = 34280

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=34280, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=1 /\* 0.01 s \*/} ---

fstat(1, {st\_mode=S\_IFREG|0600, st\_size=0, ...}) = 0

getrandom("\x83\x68\x3b\x9f\x41\x03\xf8\xb9", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x5654d7706000

brk(0x5654d7727000) = 0x5654d7727000

shmdt(0x7f4b0c84a000) = 0

shmctl(12, IPC\_RMID, NULL) = 0

write(1, "HELLO\_WORLD\nTHIS\_IS\_A\_TEST\n\n", 28) = 28

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В рамках лабораторной работы была разработана программа, демонстрирующая работу аллокатора передаваемого в качестве аргумента при вызове программы. Был проведен всесторонний анализ работы двух разных аллокаторов.