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: Жаворонков Н. Д.

Преподаватель: Бахарев В. Д.

Оценка: _____

Дата: 27.12.24

Москва
2024

Постановка задачи

Вариант 1.

Цель работы

Приобретение практических навыков в:

1) Создании аллокаторов памяти и их анализу;

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

Задание

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.

Списки свободных блоков (первое подходящее) и блоки по 2^n;

Общий метод и алгоритм решения

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
3. **void dlopen(const char filename, int flag); - Открывает динамическую библиотеку
4. **void mmap(void addr, size_t length, int prot, int flags, int fd, off_t offset); – создает новое отображение памяти или изменяет существующее.

Чтение файла: Родительский процесс считывает числа из файла построчно.

1. buddy.c: Аллокатор памяти на основе блоков-близнецов (buddy system)
Этот файл реализует аллокатор памяти, использующий стратегию выделения блоков, размер которых является степенью двойки (2^n).

Инициализация: Вся доступная память при запуске делится на блоки, размер которых кратен степени двойки.
Управление свободными блоками: Все свободные блоки хранятся в списках, отсортированных по размеру, где каждый блок содержит указатель на следующий свободный блок в своем списке.

Освобождение памяти: При освобождении блока он возвращается в соответствующий список свободных блоков в правильной позиции, в зависимости от его размера. Также, если это возможно, соседние свободные блоки объединяются, чтобы сформировать блок большего размера.
Выделение памяти: Для удовлетворения запроса на память аллокатор выбирает наименьший подходящий блок, размер которого равен или больше запрошенного (N[log2(size)]). Затем возвращается указатель на начало этого блока, а блок помечается как занятый.

2. ffit.c: Аллокатор памяти First-Fit
Этот файл реализует аллокатор памяти, использующий стратегию First-Fit (первый подходящий).

Управление свободными блоками: Аллокатор поддерживает список свободных блоков, отсортированных по их адресам в памяти.
Поиск подходящего блока: Когда поступает запрос на выделение памяти, аллокатор просматривает список свободных блоков последовательно, начиная с первого, в поисках блока, достаточного для удовлетворения запроса.
Выделение памяти
Проверка блока: Аллокатор перебирает блоки в списке, пока не найдет первый свободный блок, размер которого больше или равен запрошенному.
Точное совпадение: Если размер найденного блока точно равен запрошенному, то весь блок целиком выделяется и исключается из списка свободных блоков.
Разделение блока: Если размер найденного блока больше запрошенного, то этот блок разделяется на два:
Занятая часть: Первый блок размером с запрос выделяется и возвращается пользователю.
Свободный остаток: Остальная часть блока (если есть) остается свободной, ее размер и адрес корректируются, и блок возвращается в список свободных блоков, сохраняя сортировку по адресу.
Нет подходящего блока: Если ни один из блоков не подходит по размеру, запрос на выделение не выполняется, и возвращается сигнал об ошибке или NULL указатель.
Освобождение памяти: При освобождении блока, он добавляется в список свободных блоков. При этом аллокатор проверяет, есть ли соседние свободные блоки, и, если таковые есть, сливает их в один блок для уменьшения фрагментации памяти.

Сравнение сортировок

```
===== ./liballocator_firstfit.so =====
Average Memory Efficiency: 95.456765
Allocation speed test (Different sizes, full drain): 0.000195
Free speed test (One size, 10K elements): 0.000037
===== === =====
supertos@DESKTOP-77RBNCB:~/bb/src$ ./main ./liballocator_buddy.so
===== ./liballocator_buddy.so =====
Average Memory Efficiency: 50.000000
Allocation speed test (Different sizes, full drain): 0.000745
Free speed test (One size, 10K elements): 0.000025
===== === =====
```

Более того, First-Fit алгоритм более удобен, так как работает быстро при любых размерах выделяемого пространства, в то время как алгоритм выделения 2^n требует знания самого ходового размера.

Код программы

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <dlfcn.h>
#include <string.h>
#include <sys/time.h>
#include <sys/mman.h>
typedef struct Allocator Allocator;
typedef Allocator* (*allocator_create_fn)(void *memory, size_t size);
typedef void (*allocator_destroy_fn)(Allocator *allocator);
typedef void* (*allocator_alloc_fn)(Allocator *allocator, size_t size);
typedef void (*allocator_free_fn)(Allocator *allocator, void *memory);
Allocator *standard_allocator_create(void *memory, size_t size) {
   (void)size;
   (void)memory;
   return memory;
}
void *standard_allocator_alloc(Allocator *allocator, size_t size) {
   (void)allocator;
   size_t *memory = mmap(NULL, size + sizeof(size_t), PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);
   if (memory == MAP_FAILED) {
      return NULL;
   }
   *memory = (size_t)(size + sizeof(size_t));
   return memory + 1;
}
void standard_allocator_free(Allocator *allocator, void *memory) {
   (void)allocator;
   if (memory == NULL)
      return;
   size_t *mem = (size_t *)memory - 1;
   munmap(mem, *mem);
}
void standard_allocator_destroy(Allocator *allocator) { (void)allocator; }
void testAllocator( allocator_create_fn create, allocator_destroy_fn destroy, allocator_alloc_fn cmalloc, allocator_free_fn cfree,
char* name ) {
        printf( "===== %s =====\n", name );
        void* memory;

        double avgMemoryUsable = 0;
        double probes = 0;


        struct timeval begin;
        gettimeofday( &begin, NULL );

        double time_taken;


        int def = !strcmp( name, "DEFAULT" );

        for( size_t i = 128; i < 32768 * 8; i *= 2 ) {
                memory = mmap(NULL, i, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
                Allocator* alloc = create( memory, i );

                double bytesAllocated = 0;

                while( (!def || bytesAllocated < 8192 ) && cmalloc( alloc, 16 ) ) bytesAllocated += 16;
                // printf( "DEBUG: %zu %.2f \n", i, bytesAllocated );
```

```
                avgMemoryUsable += bytesAllocated / i;


                destroy( alloc );
                munmap(memory, i);

                probes++;
        }
        struct timeval end;
        gettimeofday( &end, NULL );

   time_taken = (end.tv_sec - begin.tv_sec);
   time_taken += (end.tv_usec - begin.tv_usec) * 1e-6;

        printf( "Average Memory Efficiency: %f\n", avgMemoryUsable / probes * 100 );
        printf( "Allocation speed test (Different sizes, full drain): %f\n", time_taken );




        memory = mmap(NULL, 32768 * 1024 * 4, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -
1, 0);
        if( !memory ) printf( "No mem :(\n" );
        Allocator* alloc = create( memory, 32768 * 1024 * 4 );

        void* allocated[10000];
        size_t i = 0;
        while( i < 10000 && (allocated[i++]=cmalloc( alloc, 16 )) );

        gettimeofday( &begin, NULL );
        for( size_t i = 0; i < 10000; i++ ) {

                cfree( alloc, allocated[i] );
        }
        gettimeofday( &end, NULL );

   time_taken = (end.tv_sec - begin.tv_sec);
   time_taken += (end.tv_usec - begin.tv_usec) * 1e-6;
        printf( "Free speed test (One size, 10K elements): %f\n", time_taken );
        printf( "===== === =====\n" );


        munmap(memory, 32768 * 1024 * 4);
}
int main(int argc, char *argv[]) {
   if (argc < 2) {
                testAllocator( standard_allocator_create, standard_allocator_destroy, standard_allocator_alloc,
standard_allocator_free, "DEFAULT" );

   }else{
                void *handle = dlopen(argv[1], RTLD_NOW);
                if (!handle) {
                        testAllocator( standard_allocator_create, standard_allocator_destroy, standard_allocator_alloc,
standard_allocator_free, "DEFAULT" );
                }else{
                        allocator_create_fn allocator_create = (allocator_create_fn)dlsym(handle, "allocator_create");
                        allocator_destroy_fn allocator_destroy = (allocator_destroy_fn)dlsym(handle, "allocator_destroy");
                        allocator_alloc_fn allocator_alloc = (allocator_alloc_fn)dlsym(handle, "allocator_alloc");
                        allocator_free_fn allocator_free = (allocator_free_fn)dlsym(handle, "allocator_free");
                        if (!allocator_create || !allocator_destroy || !allocator_alloc || !allocator_free) {
```

```
                                dlclose(handle);
                                testAllocator( standard_allocator_create, standard_allocator_destroy, standard_allocator_alloc,
standard_allocator_free, "DEFAULT" );
                                return 1;
                        }else{
                                testAllocator( allocator_create, allocator_destroy, allocator_alloc, allocator_free, argv[1] );

                                dlclose(handle);
                                return 0;

                        }

                }
        }

} #include <stdio.h>
#include <stdlib.h>
#include <sys/mman.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 TwonAllocator {
   void* memory;
   size_t total_size;
   Block* free_lists[MAX_BLOCK_INDEX];
} TwonAllocator;
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));
}
void* allocator_create(void* const memory, const size_t size) {
   if (!memory || size < MIN_BLOCK_SIZE) {
      return NULL;
   }
   TwonAllocator* allocator = (TwonAllocator*)mmap(
      NULL, sizeof(TwonAllocator), PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
   if (allocator == MAP_FAILED) {
      return NULL;
   }
   size_t total_size = round_to_power_of_two(size);
   size_t max_size = 1 << (MAX_BLOCK_INDEX - 1);
   if (total_size > max_size) {
      munmap(allocator, sizeof(TwonAllocator));
      return NULL;
   }
   allocator->memory = memory;
   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;
}
void allocator_destroy(void* const twon_allocator) {
    if (!twon_allocator) return;
    TwonAllocator* allocator = (TwonAllocator*)twon_allocator;
    munmap(allocator, sizeof(TwonAllocator));
}
void* allocator_alloc(void* const twon_allocator, const size_t size) {
    if (!twon_allocator || size == 0) return NULL;
    TwonAllocator* allocator = (TwonAllocator*)twon_allocator;
    size_t block_size = round_to_power_of_two(size);
    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 < 0 || index >= MAX_BLOCK_INDEX) {
        return NULL;
    }
    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));
}
void allocator_free(void* const twon_allocator, void* const memory) {
    if (!twon_allocator || !memory) return;
    TwonAllocator* allocator = (TwonAllocator*)twon_allocator;
    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 < 0 || index >= MAX_BLOCK_INDEX) return;
    block->next = allocator->free_lists[index];
    allocator->free_lists[index] = block;
} #include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <stdint.h>
#include <math.h>
typedef struct FreeBlock {
    size_t size;
    struct FreeBlock *next;
} FreeBlock;
```

```
typedef struct Allocator {
    void *memory;
    size_t size;
    FreeBlock *free_list;
} Allocator;
Allocator* allocator_create(void *const memory, const size_t size) {
    Allocator *allocator = (Allocator *)memory;
    allocator->size = size;
    allocator->free_list = (FreeBlock *)((char *)memory + sizeof(Allocator));
    allocator->free_list->size = size - sizeof(Allocator);
    allocator->free_list->next = NULL;
    return allocator;
}
void allocator_destroy(Allocator *const allocator) {
}
void* allocator_alloc(Allocator *const allocator, const size_t size) {
    FreeBlock *prev = NULL;
    FreeBlock *curr = allocator->free_list;
    while (curr != NULL) {
        if (curr->size >= size) {
            if (curr->size > size + sizeof(FreeBlock)) {
                FreeBlock *new_block = (FreeBlock *)((char *)curr + size);
                new_block->size = curr->size - size;
                new_block->next = curr->next;
                curr->size = size;
                curr->next = new_block;
            }
            if (prev == NULL) {
                allocator->free_list = curr->next;
            } else {
                prev->next = curr->next;
            }
            return (void *)((char *)curr + sizeof(FreeBlock));
        }
        prev = curr;
        curr = curr->next;
    }
    return NULL;
}
void allocator_free(Allocator *const allocator, void *const memory) {
    FreeBlock *block = (FreeBlock *)((char *)memory - sizeof(FreeBlock));
    block->next = allocator->free_list;
    allocator->free_list = block;
}
```

Протокол работы программы

Тестирование:

```
supertos@DESKTOP-77RBNCB:~/oslab$ ./main liballocator_buddy.so
```

Strace :

```
execve("./main", ["./main", "./liballocator_buddy.so"], 0x7ffc81cef598 /* 23 vars */) = 0
brk(NULL)                               = 0x5605e7abb000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fffc6f82a70) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9ef4784000
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=17571, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 17571, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9ef477f000
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"..., 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) = 0x7f9ef4556000
mprotect(0x7f9ef457e000, 2023424, PROT_NONE) = 0
mmap(0x7f9ef457e000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x7f9ef457e000
mmap(0x7f9ef4713000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x7f9ef4713000
mmap(0x7f9ef476c000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x7f9ef476c000
mmap(0x7f9ef4772000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
0) = 0x7f9ef4772000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9ef4553000
arch_prctl(ARCH_SET_FS, 0x7f9ef4553740) = 0
set_tid_address(0x7f9ef4553a10)         = 14064
set_robust_list(0x7f9ef4553a20, 24)     = 0
rseq(0x7f9ef45540e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f9ef476c000, 16384, PROT_READ) = 0
mprotect(0x5605ba14d000, 4096, PROT_READ) = 0
mprotect(0x7f9ef47be000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f9ef477f000, 17571)           = 0
getrandom("\x9c\x58\xaf\x10\xd5\xf3\x93\x99", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5605e7abb000
brk(0x5605e7adc000)                     = 0x5605e7adc000
openat(AT_FDCWD, "./liballocator_buddy.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|0755, st_size=18872, ...}, AT_EMPTY_PATH) = 0
```

getcwd("/home/supertos/oslabb/lab4/src", 128) = 31

mmap(NULL, 16464, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f9ef477f000

mmap(0x7f9ef4780000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f9ef4780000

mmap(0x7f9ef4781000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f9ef4781000

mmap(0x7f9ef4782000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f9ef4782000

close(3) = 0

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

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=17571, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 17571, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9ef454e000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.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\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=940560, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 942344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f9ef4467000

mmap(0x7f9ef4475000, 507904, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe000) = 0x7f9ef4475000

mmap(0x7f9ef44f1000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8a000) = 0x7f9ef44f1000

mmap(0x7f9ef454c000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe4000) = 0x7f9ef454c000

close(3) = 0

mprotect(0x7f9ef454c000, 4096, PROT_READ) = 0

mprotect(0x7f9ef4782000, 4096, PROT_READ) = 0

munmap(0x7f9ef454e000, 17571) = 0

newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0

write(1, "===== ./liballocator_buddy.so =="..., 36===== ./liballocator_buddy.so =====

) = 36

mmap(NULL, 128, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0) = 0x7f9ef47bd000

mmap(NULL, 208, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9ef4552000

munmap(0x7f9ef4552000, 208) = 0

munmap(0x7f9ef47bd000, 128) = 0

write(1, "Average Memory Efficiency: 50.00"..., 37Average Memory Efficiency: 50.000000

) = 37

write(1, "Allocation speed test (Different"..., 62Allocation speed test (Different sizes, full drain): 0.001219

) = 62

mmap(NULL, 134217728, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0) = 0x7f9eec467000

mmap(NULL, 208, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9ef47bd000

munmap(0x7f9ef47bd000, 208) = 0

write(1, "Free speed test (One size, 10K e"..., 51Free speed test (One size, 10K elements): 0.000000

) = 51

write(1, "===== === =====\n", 16===== === =====

) = 16

munmap(0x7f9eec467000, 134217728) = 0

munmap(0x7f9ef477f000, 16464) = 0

munmap(0x7f9ef4467000, 942344) = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В рамках лабораторной работы была разработана программа, демонстрирующая работу аллокатора передаваемого в качестве аргумента при вызове программы. Было реализовано 2 аллокатора и проведена работа по сравнениб их работоспособности