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: Ласточкин М.В.

Преподаватель: Бахарев В.Д.

Оценка:

_____

Дата: 23.12.24

Москва, 2024

Постановка задачи

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. Эти аварийные оберточные функции должны быть реализованы внутри программы, которая загружает динамические библиотеки.

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);

Вариант 3. Списки свободных блоков (первое подходящее) и алгоритм двойников;

Общий метод и алгоритм решения

Использованные системные вызовы:

- ssize_t write(int _fd, const void *_buf, size_t _n); – Записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
- void exit(int _status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- int clock_gettime(clockid_t clk_id, struct timespec *tp); определяет разрешающую точность заданных в clk_id часов, и, если res не равно NULL, сохраняет её в struct timespec, указанную в res.
- void *dlopen(const char *filename, int flag); – загружает динамическую библиотеку, имя которой указано в строке filename, и возвращает прямой указатель на начало динамической библиотеки.
- int dlclose(void *handle); – уменьшает на единицу счетчик ссылок на указатель динамической библиотеки.
- const char *dlerror(void); - возвращает понятную человеку, строку с null в конце, описывающую последнюю ошибку, которая произошла при вызове одной из функций программного интерфейса dlopen

Аллокатор памяти на основе списка свободных блоков (первое подходящее)

Аллокатор памяти на основе списка свободных блоков использует структуру данных например, связанный список, для управления блоками памяти. Когда требуется выделить память, аллокатор ищет подходящий свободный блок в списке. После использования блоки возвращаются обратно в список.

Основные компоненты и принципы работы:

Список свободных блоков представляет собой структуру данных (обычно односвязный или двусвязный список), где каждый элемент описывает свободный участок памяти.

Основные компоненты:

- Список свободных блоков — структура данных (обычно связанный список), хранящая указатели на свободные участки памяти.
- Блоки памяти — участки памяти, которые аллокатор выделяет или освобождает. Каждый блок может содержать информацию о размере блока и указатель на следующий свободный блок.
- Функции выделения и освобождения памяти:
    1. Выделение: поиск подходящего свободного блока в списке.
    2. Освобождение: возвращение блока в список свободных.

Операции:

1. Инициализация свободного списка:
    - Аллокатор получает блок памяти размером, достаточным для управления списком.
    - Первый блок в списке представляет собой весь доступный фрагмент памяти.
    - Этот блок выделяется как единственный в списке и содержит указатель на следующий свободный блок.
2. Выделение памяти:
    - При запросе памяти, аллокатор ищет подходящий свободный блок в списке (если размер блока в списке больше или равен запрашиваемому размеру этот блок выделяется).
    - Если блок больше требуемого, то его можно разделить на два блока: один с нужным размером, второй с оставшимся.
    - В случае выделения памяти, блок удаляется из списка свободных блоков, а указатель на выделенную память возвращается в программу.
3. Освобождение памяти:
    - Если блок памяти не нужен, то он освобождается и переходит в список свободных блоков.
    - Если несколько блоков в списке свободных блоков могут быть объединены в один больший блок памяти.

Преимущества:

- Простота реализации: основные алгоритмы достаточно просты для понимания и кодирования.
- Гибкость: может работать с любыми размерами блоков.
- Минимальные требования к памяти: аллокатор требует минимального дополнительного пространства. Он использует саму область памяти для хранения информации о свободных блоках, добавляя только указатель на следующий блок и его размер.

Недостатки:

- Фрагментация: при выделении и удалении блоков в разное время, может образоваться множество блоков с размеров, не подходящим под задачи.
- Невысокая производительность: операции выделения и освобождения памяти не самые быстрые операции из-за необходимости перебора списка свободных блоков.
- Отсутствие быстрого поиска: список свободных блоков не поддерживает эффективных алгоритмов поиска, например, бинарного поиска.

Аллокатор памяти на алгоритма двойников

Это способ динамического распределения памяти, который помогает эффективно управлять памятью, минимизируя фрагментацию и улучшая производительность при выделении и освобождении блоков памяти. Данный метод один из популярных методов аллокации памяти, который находит применение в операционных системах и системах с жесткими требованиями к производительности и эффективности.

Основные компоненты и принципы работы:

Метод использует массив или список для хранения блоков памяти по их размерам (показатели порядка или степени двойки). Каждый блок в списке представляет собой указатель на свободный блок определенного размера, и в случае выделения или освобождения памяти эти списки обновляются.

Операции:

1) Выделение памяти:
    - При выделении памяти под определенный размер, алгоритм ищет минимальный блок, который подходит под требования. Если такой блок не удается найти, то алгоритм будет увеличивать блок до ближайшей степени двойки, пока блок не станет доступным.
    - Таким же методом он может разделять большие блоки на более мелкие.
2) Освобождение памяти:
    - Алгоритм проверяет, свободен ли соседний блок памяти. Если так, то 2 блока объединяются в 1 больший блок, который становится доступным.
3) Поиск и объединение блоков:
    - После того, как произошло освобождение памяти и 2 блока сливаются в один, процесс может рекурсивно повторяться, если соседние блоки так же свободны.

Преимущества:

- Эффективное использование памяти.
- Меньше фрагментации. Память, которая выделяется и освобождается, может быть эффективно объединена в более крупные блоки.
- Быстрота выделения памяти. Алгоритм позволяет быстро выделять блоки, поскольку размер каждого блока — это степень двойки.
- Гибкость. Этот метод работает хорошо с любыми размерами блоков.

Недостатки:

- Внутренняя фрагментация. Поскольку блоки всегда выделяются в степени двойки, если размер запрашиваемой памяти не является степенью двойки, алгоритм может выделить блок большего размера, чем требуется. Это может привести к неэффективному использованию памяти.
- Сложность с большими блоками. Может потребоваться достаточно много времени, если самый большой блок значительно меньше, чем нужный размер памяти.
- Возможное увеличение времени для освобождения. Алгоритм может требовать много времени на объединение блоков при освобождении.

Тестирование

В ходе тестирования были проведены следующие шаги для проверки работы двух типов аллокаторов памяти: аллокатора с обычным списком свободных блоков и аллокатора с алгоритмом двойников. Каждый из аллокаторов был проверен на выделение и освобождение блоков памяти, а также измерено время, необходимое для этих операций.

Представление данных в виде объектов: в качестве объектов были выбраны структуры данных, так как они являются хорошим представлением для моделирования реальных данных, которые могут быть динамически выделены и освобождены в процессе работы программы. Поскольку каждый объект содержит несколько полей (целое число, строку и вещественное число), это позволяет более полно протестировать выделение памяти, включая необходимость корректной работы с различными типами данных.

1) Инициализация аллокаторов: для каждого тестируемого аллокатора выделена память, соответствующая заявленному размеру. Это базируется на глобальной переменной global_memory, размер которой равен MEMORY_SIZE. Так мы проверяем, что инициализация прошла успешно.
2) Выделение памяти: для каждого аллокатора 3 объекта типа Object были выделены с помощью соответствующих функций выделения памяти. Для каждого выделенного блока измерялось время выполнения операции выделения памяти. Результаты сохранялись и выводились на экран в формате: адрес блока и время выделения. Так мы видели, какой именно блок выделяется под нужный объект.
3) Заполнение данных в объектах: после выделения памяти для каждого объекта присваивались значения его полям (id, name, value).
4) Освобождение памяти: после заполнения данных каждый объект был освобожден с помощью соответствующих функций. Так же выводилось соответствующее сообщение о том, что блок освобожден и выводился его id и данные внутри блока.
5) Завершение тестирования: по завершению всех операций тестирования аллокатор с обычным списком свободных блоков был уничтожен. Все ресурсы были освобождены.

Результаты тестирования:

Тестирование показало следующие результаты для двух типов аллокаторов:

Аллокатор с обычным списком свободных блоков:

Выделен блок: 0x55e16b1e0068, время: 0.000000073 секунд

Выделен блок: 0x55e16b1e00b4, время: 0.000000044 секунд

Выделен блок: 0x55e16b1e0100, время: 0.000000035 секунд

Освобождён блок: 0x55e16b1e0068 (id=1, name=Object 1, value=123.45), время: 0.000000035 секунд

Освобождён блок: 0x55e16b1e00b4 (id=2, name=Object 2, value=678.90), время: 0.000000033 секунд

Освобождён блок: 0x55e16b1e0100 (id=3, name=Object 3, value=135.79), время: 0.000000036

секунд

Аллокатор с алгоритмом двойников:

Выделен блок: 0x55e16b1e0108, время: 0.000009096 секунд

Выделен блок: 0x55e16b1e0188, время: 0.000000087 секунд

Выделен блок: 0x55e16b1e0208, время: 0.000000088 секунд

Освобождён блок: 0x55e16b1e0108 (id=1, name=Buddy Object 1, value=123.45), время: 0.000000059 секунд

Освобождён блок: 0x55e16b1e0188 (id=2, name=Buddy Object 2, value=678.90), время: 0.000000056 секунд

Освобождён блок: 0x55e16b1e0208 (id=3, name=Buddy Object 3, value=135.79), время: 0.000000049 секунд

Оба аллокатора эффективно справляются с выделением и освобождением памяти. Сильные различия присутствуют только при первой попытке выделения памяти с помощью алгоритма двойников. Скорее всего это связано это с тем, что первый блок был достаточно большой, и алгоритм двойников начал разделять его до тех пор, пока он не станет оптимальным для вводных данных, поэтому время могло сильно увеличится.

Код программы

allocator.c

```c
#include "allocator.h"

Allocator* allocator_create(void* memory, size_t size) {
    if (size < sizeof(FreeBlock)) return NULL;

    Allocator* allocator = (Allocator*)memory;
    allocator->memory_start = (char*)memory + sizeof(Allocator);
    allocator->memory_size = size - sizeof(Allocator);
    allocator->free_list = (FreeBlock*)allocator->memory_start;

    allocator->free_list->size = allocator->memory_size;
    allocator->free_list->next = NULL;

    return allocator;
}

void allocator_destroy(Allocator* allocator) {}

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

    FreeBlock* prev = NULL;
    FreeBlock* current = allocator->free_list;

    while (current) {
        if (current->size >= size + sizeof(FreeBlock)) {
            if (current->size > size + sizeof(FreeBlock)) {
                FreeBlock* new_block = (FreeBlock*)((char*)current +
sizeof(FreeBlock) + size);
                new_block->size = current->size - size -
sizeof(FreeBlock);
                new_block->next = current->next;
                current->next = new_block;
            }

            if (prev) {
                prev->next = current->next;
            } else {
                allocator->free_list = current->next;
            }

            current->size = size;
            return (char*)current + sizeof(FreeBlock);
        }

        prev = current;
        current = current->next;
    }

    return NULL;
}
```

```
void allocator_free(Allocator* allocator, void* memory) {
   if (!memory) return;

   FreeBlock* block_to_free = (FreeBlock*)((char*)memory -
sizeof(FreeBlock));
   block_to_free->next = allocator->free_list;
   allocator->free_list = block_to_free;
}
```

buddy_allocator.c

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

BuddyAllocator* buddy_allocator_create(void* memory, size_t size) {
   if (size < (1 << MAX_BUDDY_ORDER)) return NULL;

   BuddyAllocator* allocator = (BuddyAllocator*)memory;
   allocator->memory_start = (char*)memory + sizeof(BuddyAllocator);
   allocator->memory_size = size - sizeof(BuddyAllocator);

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

   size_t initial_order = (size_t)log2(size);
   allocator->free_lists[initial_order] =
(FreeBlock*)allocator->memory_start;
   allocator->free_lists[initial_order]->size = size;
   allocator->free_lists[initial_order]->next = NULL;

   return allocator;
}

void* buddy_allocator_alloc(BuddyAllocator* allocator, size_t size)
{
   if (size == 0) return NULL;

   size_t order = (size_t)ceil(log2(size + sizeof(FreeBlock)));

   for (size_t i = order; i <= MAX_BUDDY_ORDER; i++) {
       if (allocator->free_lists[i]) {
           FreeBlock* block = allocator->free_lists[i];
           allocator->free_lists[i] = block->next;

           while (i > order) {
               i--;
               size_t block_size = 1 << i;
               FreeBlock* buddy = (FreeBlock*)((char*)block +
block_size);
               buddy->size = block_size;
               buddy->next = allocator->free_lists[i];
               allocator->free_lists[i] = buddy;
           }

           block->size = (1 << order);
           return (char*)block + sizeof(FreeBlock);
```

```
        }
    }

    return NULL;
}

void buddy_allocator_free(BuddyAllocator* allocator, void* memory) {
    if (!memory) return;

    FreeBlock* block = (FreeBlock*)((char*)memory -
sizeof(FreeBlock));
    size_t order = (size_t)log2(block->size);

    FreeBlock** current_list = &allocator->free_lists[order];
    block->next = *current_list;
    *current_list = block;
}
```

main.c

```
#include "allocator.h"
#include "buddy_allocator.h"
#include <stdio.h>
#include <time.h>

#define MEMORY_SIZE (1 << MAX_BUDDY_ORDER)
char global_memory[MEMORY_SIZE];

typedef struct Object {
    int id;
    char name[50];
    float value;
} Object;

int main() {
    void* allocator_lib = dlopen("./liballocator.so", RTLD_LAZY);
    if (!allocator_lib) {
        fprintf(stderr, "Ошибка загрузки библиотеки liballocator.so:
%s\n", dlerror());
        return 1;
    }

    void* buddy_lib = dlopen("./libbuddy_allocator.so", RTLD_LAZY);
    if (!buddy_lib) {
        fprintf(stderr, "Ошибка загрузки библиотеки
libbuddy_allocator.so: %s\n", dlerror());
        dlclose(allocator_lib);
        return 1;
    }

    struct timespec start, end;

    printf("Тестирование аллокатора с обычным списком свободных
блоков:\n");

    Allocator* list_allocator = allocator_create(global_memory,
MEMORY_SIZE);
```

```
    clock_gettime(CLOCK_MONOTONIC, &start);
    Object* object1 = allocator_alloc(list_allocator,
sizeof(Object));
    clock_gettime(CLOCK_MONOTONIC, &end);
    double alloc_time1 = (end.tv_sec - start.tv_sec) + (end.tv_nsec -
start.tv_nsec) / 1e9;
    printf("Выделен блок: %p, время: %.9f секунд\n", object1,
alloc_time1);

    clock_gettime(CLOCK_MONOTONIC, &start);
    Object* object2 = allocator_alloc(list_allocator,
sizeof(Object));
    clock_gettime(CLOCK_MONOTONIC, &end);
    double alloc_time2 = (end.tv_sec - start.tv_sec) + (end.tv_nsec -
start.tv_nsec) / 1e9;
    printf("Выделен блок: %p, время: %.9f секунд\n", object2,
alloc_time2);

    clock_gettime(CLOCK_MONOTONIC, &start);
    Object* object3 = allocator_alloc(list_allocator,
sizeof(Object));
    clock_gettime(CLOCK_MONOTONIC, &end);
    double alloc_time3 = (end.tv_sec - start.tv_sec) + (end.tv_nsec -
start.tv_nsec) / 1e9;
    printf("Выделен блок: %p, время: %.9f секунд\n", object3,
alloc_time3);

    if (object1) {
        object1->id = 1;
        snprintf(object1->name, sizeof(object1->name), "Object 1");
        object1->value = 123.45;
    }

    if (object2) {
        object2->id = 2;
        snprintf(object2->name, sizeof(object2->name), "Object 2");
        object2->value = 678.90;
    }

    if (object3) {
        object3->id = 3;
        snprintf(object3->name, sizeof(object3->name), "Object 3");
        object3->value = 135.79;
    }

    clock_gettime(CLOCK_MONOTONIC, &start);
    allocator_free(list_allocator, object1);
    clock_gettime(CLOCK_MONOTONIC, &end);
    double free_time1 = (end.tv_sec - start.tv_sec) + (end.tv_nsec -
start.tv_nsec) / 1e9;
    printf("Освобождён блок: %p (id=%d, name=%s, value=%.2f), время:
%.9f секунд\n", object1, object1->id, object1->name, object1->value,
free_time1);

    clock_gettime(CLOCK_MONOTONIC, &start);
    allocator_free(list_allocator, object2);
    clock_gettime(CLOCK_MONOTONIC, &end);
    double free_time2 = (end.tv_sec - start.tv_sec) + (end.tv_nsec -
```

```
start.tv_nsec) / 1e9;
   printf("Освобождён блок: %p (id=%d, name=%s, value=%.2f), время:
%.9f секунд\n", object2, object2->id, object2->name, object2->value,
free_time2);

   clock_gettime(CLOCK_MONOTONIC, &start);
   allocator_free(list_allocator, object3);
   clock_gettime(CLOCK_MONOTONIC, &end);
   double free_time3 = (end.tv_sec - start.tv_sec) + (end.tv_nsec -
start.tv_nsec) / 1e9;
   printf("Освобождён блок: %p (id=%d, name=%s, value=%.2f), время:
%.9f секунд\n", object3, object3->id, object3->name, object3->value,
free_time3);

   printf("\nТестирование аллокатора с алгоритмом двойников:\n");

   BuddyAllocator* buddy_allocator =
buddy_allocator_create(global_memory, MEMORY_SIZE);

   clock_gettime(CLOCK_MONOTONIC, &start);
   Object* buddy_object1 = buddy_allocator_alloc(buddy_allocator,
sizeof(Object));
   clock_gettime(CLOCK_MONOTONIC, &end);
   double buddy_alloc_time1 = (end.tv_sec - start.tv_sec) +
(end.tv_nsec - start.tv_nsec) / 1e9;
   printf("Выделен блок: %p, время: %.9f секунд\n", buddy_object1,
buddy_alloc_time1);

   clock_gettime(CLOCK_MONOTONIC, &start);
   Object* buddy_object2 = buddy_allocator_alloc(buddy_allocator,
sizeof(Object));
   clock_gettime(CLOCK_MONOTONIC, &end);
   double buddy_alloc_time2 = (end.tv_sec - start.tv_sec) +
(end.tv_nsec - start.tv_nsec) / 1e9;
   printf("Выделен блок: %p, время: %.9f секунд\n", buddy_object2,
buddy_alloc_time2);

   clock_gettime(CLOCK_MONOTONIC, &start);
   Object* buddy_object3 = buddy_allocator_alloc(buddy_allocator,
sizeof(Object));
   clock_gettime(CLOCK_MONOTONIC, &end);
   double buddy_alloc_time3 = (end.tv_sec - start.tv_sec) +
(end.tv_nsec - start.tv_nsec) / 1e9;
   printf("Выделен блок: %p, время: %.9f секунд\n", buddy_object3,
buddy_alloc_time3);

   if (buddy_object1) {
       buddy_object1->id = 1;
       snprintf(buddy_object1->name, sizeof(buddy_object1->name),
"Buddy Object 1");
       buddy_object1->value = 123.45;
   }

   if (buddy_object2) {
       buddy_object2->id = 2;
       snprintf(buddy_object2->name, sizeof(buddy_object2->name),
"Buddy Object 2");
       buddy_object2->value = 678.90;
```

```
    }

    if (buddy_object3) {
        buddy_object3->id = 3;
        snprintf(buddy_object3->name, sizeof(buddy_object3->name),
"Buddy Object 3");
        buddy_object3->value = 135.79;
    }

    clock_gettime(CLOCK_MONOTONIC, &start);
    buddy_allocator_free(buddy_allocator, buddy_object1);
    clock_gettime(CLOCK_MONOTONIC, &end);
    double buddy_free_time1 = (end.tv_sec - start.tv_sec) +
(end.tv_nsec - start.tv_nsec) / 1e9;
    printf("Освобождён блок: %p (id=%d, name=%s, value=%.2f), время:
%.9f секунд\n", buddy_object1, buddy_object1->id,
buddy_object1->name, buddy_object1->value, buddy_free_time1);

    clock_gettime(CLOCK_MONOTONIC, &start);
    buddy_allocator_free(buddy_allocator, buddy_object2);
    clock_gettime(CLOCK_MONOTONIC, &end);
    double buddy_free_time2 = (end.tv_sec - start.tv_sec) +
(end.tv_nsec - start.tv_nsec) / 1e9;
    printf("Освобождён блок: %p (id=%d, name=%s, value=%.2f), время:
%.9f секунд\n", buddy_object2, buddy_object2->id,
buddy_object2->name, buddy_object2->value, buddy_free_time2);

    clock_gettime(CLOCK_MONOTONIC, &start);
    buddy_allocator_free(buddy_allocator, buddy_object3);
    clock_gettime(CLOCK_MONOTONIC, &end);
    double buddy_free_time3 = (end.tv_sec - start.tv_sec) +
(end.tv_nsec - start.tv_nsec) / 1e9;
    printf("Освобождён блок: %p (id=%d, name=%s, value=%.2f), время:
%.9f секунд\n", buddy_object3, buddy_object3->id,
buddy_object3->name, buddy_object3->value, buddy_free_time3);
    allocator_destroy(list_allocator);

    dlclose(allocator_lib);
    dlclose(buddy_lib);

    return 0;
}
```

Протокол работы программы

```
max@DESKTOP-L04A0IM:/mnt/c/Users/lasto/CLionProjects/Osi/laba4$ ./memory_allocator
    ./allocator.so
```

Тестирование аллокатора с обычным списком свободных блоков:

```
  Выделен блок: 0x55be7a33d068, время: 0.000000076 секунд
  Выделен блок: 0x55be7a33d0b4, время: 0.000000071 секунд
  Выделен блок: 0x55be7a33d100, время: 0.000000044 секунд
  Освобождён блок: 0x55be7a33d068 (id=1, name=Object 1, value=123.45), время: 0.000000036 секунд
  Освобождён блок: 0x55be7a33d0b4 (id=2, name=Object 2, value=678.90), время: 0.000000033 секунд
  Освобождён блок: 0x55be7a33d100 (id=3, name=Object 3, value=135.79), время: 0.000000030 секунд
  Тестирование аллокатора с алгоритмом двойников:
  Выделен блок: 0x55be7a33d108, время: 0.000008243 секунд
  Выделен блок: 0x55be7a33d188, время: 0.000000094 секунд
  Выделен блок: 0x55be7a33d208, время: 0.000000088 секунд
  Освобождён блок: 0x55be7a33d108 (id=1, name=Buddy Object 1, value=123.45), время: 0.000000059
       секунд
  Освобождён блок: 0x55be7a33d188 (id=2, name=Buddy Object 2, value=678.90), время: 0.000000060
       секунд
  Освобождён блок: 0x55be7a33d208 (id=3, name=Buddy Object 3, value=135.79), время: 0.000000052
       секунд
max@DESKTOP-L04A0IM:/mnt/c/Users/lasto/CLionProjects/Osi/laba4$ strace ./memory_allocator
./allocator.so
execve("./memory_allocator", ["./memory_allocator", "./allocator.so"], 0x7ffdad777128 /* 27 vars
*/) = 0
brk(NULL)                               = 0x55d8255e7000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd577b5b40) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff9f27df000
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=18383, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 18383, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff9f27da000
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\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) = 0x7ff9f26f3000
mmap(0x7ff9f2701000, 507904, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe000)
= 0x7ff9f2
```

```
701000
mmap(0x7ff9f277d000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8a000) =
0x7ff9f277d000
mmap(0x7ff9f27d8000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe4000)
= 0x7ff9f2
7d8000
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) = 0x7ff9f24ca000
mprotect(0x7ff9f24f2000, 2023424, PROT_NONE) = 0
mmap(0x7ff9f24f2000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x7ff9
f24f2000
mmap(0x7ff9f2687000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x7ff9f2687000
mmap(0x7ff9f26e0000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x7ff9
f26e0000
mmap(0x7ff9f26e6000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
0x7ff9f26e60
00
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff9f24c7000
arch_prctl(ARCH_SET_FS, 0x7ff9f24c7740) = 0
set_tid_address(0x7ff9f24c7a10)         = 55210
set_robust_list(0x7ff9f24c7a20, 24)     = 0
rseq(0x7ff9f24c80e0, 0x20, 0, 0x53053053) = 0
```

```
mprotect(0x7ff9f26e0000, 16384, PROT_READ) = 0
mprotect(0x7ff9f27d8000, 4096, PROT_READ) = 0
mprotect(0x55d7f8aa4000, 4096, PROT_READ) = 0
mprotect(0x7ff9f2819000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7ff9f27da000, 18383)           = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
getrandom("\xc6\x3b\x16\x86\x96\x85\x22\x82", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55d8255e7000
brk(0x55d825608000)                     = 0x55d825608000
write(1,
"\320\242\320\265\321\201\321\202\320\270\321\200\320\276\320\262\320\260\320\275\320\270\320\265
\
320\260\320\273\320\273\320"..., 112Тестирование аллокатора с обычным списком свободных блоков:
) = 112
write(1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275
\320\261\320\273\320\276\320\272: 0x55d7f
"..., 78Выделен блок: 0x55d7f8aa5068, время: 0.000000256 секунд
) = 78
write(1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275
\320\261\320\273\320\276\320\272: 0x55d7f
"..., 78Выделен блок: 0x55d7f8aa50b4, время: 0.000000114 секунд
) = 78
write(1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275
\320\261\320\273\320\276\320\272: 0x55d7f
"..., 78Выделен блок: 0x55d7f8aa5100, время: 0.000000189 секунд
) = 78
write(1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\321\221\320\275
\320\261\320\273\
320\276\320\272: 0"..., 120Освобождён блок: 0x55d7f8aa5068 (id=1, name=Object 1, value=123.45),
время: 0.000
000042 секунд
) = 120
write(1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\321\221\320\275
```

```
\320\261\320\273\
320\276\320\272: 0"..., 120Освобождён блок: 0x55d7f8aa50b4 (id=2, name=Object 2, value=678.90),
время: 0.000
000145 секунд
) = 120
write(1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\321\221\320\275
\320\261\320\273\
320\276\320\272: 0"..., 120Освобождён блок: 0x55d7f8aa5100 (id=3, name=Object 3, value=135.79),
время: 0.000
000066 секунд
) = 120
write(1, "\n", 1
)                       = 1
write(1,
"\320\242\320\265\321\201\321\202\320\270\321\200\320\276\320\262\320\260\320\275\320\270\320\265
\
320\260\320\273\320\273\320"..., 90Тестирование аллокатора с алгоритмом двойников:
) = 90
write(1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275
\320\261\320\273\320\276\320\272: 0x55d7f
"..., 78Выделен блок: 0x55d7f8aa5108, время: 0.000016715 секунд
) = 78
write(1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275
\320\261\320\273\320\276\320\272: 0x55d7f
"..., 78Выделен блок: 0x55d7f8aa5188, время: 0.000000499 секунд
) = 78
write(1, "\320\222\321\213\320\264\320\265\320\273\320\265\320\275
\320\261\320\273\320\276\320\272: 0x55d7f
"..., 78Выделен блок: 0x55d7f8aa5208, время: 0.000000270 секунд
) = 78
write(1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\321\221\320\275
\320\261\320\273\
320\276\320\272: 0"..., 126Освобождён блок: 0x55d7f8aa5108 (id=1, name=Buddy Object 1,
value=123.45), время:
```

```
 0.000000144 секунд
) = 126
write(1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\321\221\320\275
\320\261\320\273\320\276\320\272: 0"..., 126Освобождён блок: 0x55d7f8aa5188 (id=2, name=Buddy
Object 2, value=678.90), время: 0.000000528
 секунд
) = 126
write(1, "\320\236\321\201\320\262\320\276\320\261\320\276\320\266\320\264\321\221\320\275
\320\261\320\273\320\276\320\272: 0"..., 126Освобождён блок: 0x55d7f8aa5208 (id=3, name=Buddy
Object 3, value=135.79), время: 0.000000177
 секунд
) = 126
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе написания данной лабораторной работы я узнал об устройстве аллокакторов. Научился создавать, подключать и использовать динамические библиотеки. Были реализованы два алгоритма аллокации памяти, работающие через один API и подключаемые через динамические библиотеки. В ходе написания данной лабораторной работы я узнал об устройстве аллокакторов. Научился создавать, подключать и использовать динамические библиотеки. Были реализованы два алгоритма аллокации памяти, работающие через один API и подключаемые через динамические библиотеки.